Secure Computation in Heterogeneous Environments: *How to Bring Multiparty Computation Closer to Practice?*

Mariana Raykova

Submitted in partial fulfillment of the
requirements for the degree
of Doctor of Philosophy
in the Graduate School of Arts and Sciences

COLUMBIA UNIVERSITY

2012

ABSTRACT

Secure Computation in Heterogeneous Environments: *How to Bring Multiparty Computation Closer to Practice?*

Mariana Raykova

Many services that people use daily require computation that depends on the private data of multiple parties. While the utility of the final result of such interactions outweighs the privacy concerns related to output release, the inputs for such computations are much more sensitive and need to be protected. Secure multiparty computation (MPC) considers the question of constructing computation protocols that reveal nothing more about their inputs than what is inherently leaked by the output. There have been strong theoretical results that demonstrate that every functionality can be computed securely. However, these protocols remain unused in practical solutions since they introduce efficiency overhead prohibitive for most applications.

Generic multiparty computation techniques address homogeneous setups with respect to the resources available to the participants and the adversarial model. On the other hand, realistic scenarios present a wide diversity of heterogeneous environments where different participants have different available resources and different incentives to misbehave and collude. In this thesis we introduce techniques for multiparty computation that focus on heterogeneous settings. We present solutions tailored to address different types of asymmetric constraints and improve the efficiency of existing approaches in these scenarios. We tackle the question from three main directions:

- *New Computational Models for MPC* – We explore different computational models that enable us to overcome inherent inefficiencies of generic MPC solutions using circuit representation for the evaluated functionality. First, we show how we can use random access machines to construct MPC protocols that add only polylogarithmic

overhead to the running time of the insecure version of the underlying functionality. This allows to achieve MPC constructions with computational complexity sublinear in the size for their inputs, which is very important for computations that use large databases.

We also consider multivariate polynomials which yield more succinct representations for the functionalities they implement than circuits, and at the same time a large collection of problems are naturally and efficiently expressed as multivariate polynomials. We construct an MPC protocol for multivariate polynomials, which improves the communication complexity of corresponding circuit solutions, and provides currently the most efficient solution for mutiparty set intersection in the fully malicious case.

- *Outsourcing Computation* – The goal in this setting is to utilize the resources of a single powerful service provider for the work that computationally weak clients need to perform on their data. We present a new paradigm for constructing verifiable computation (VC) schemes, which enables a computationally limited client to verify efficiently the result of a large computation. Our construction is based on attribute-based encryption and avoids expensive primitives such as fully homomorphic encryption and probabilistically checkable proofs underlying existing VC schemes. Additionally our solution enjoys two new useful properties: public delegation and verification.

  We further introduce the model of server-aided computation where we utilize the computational power of an outsourcing party to assist the execution and improve the efficiency of MPC protocols. For this purpose we define a new adversarial model of non-collusion, which provides room for more efficient constructions that rely almost completely only on symmetric key operations, and at the same time captures realistic settings for adversarial behavior. In this model we propose protocols for generic secure computation that offload the work of most of the parties to the computation server. We also construct a specialized server-aided two party set intersection protocol that achieves better efficiencies for the two participants than existing solutions.

  Outsourcing in many cases concerns only data storage and while outsourcing the data

of a single party is useful, providing a way for data sharing among different clients of the service is the more interesting and useful setup. However, this scenario brings new challenges for access control since the access control rules and data accesses become private data for the clients with respect to the service provide. We propose an approach that offers trade-offs between the privacy provided for the clients and the communication overhead incurred for each data access.

- *Efficient Private Search in Practice* – We consider the question of private search from a different perspective compared to traditional settings for MPC. We start with strict efficiency requirements motivated by speeds of available hardware and what is considered acceptable overhead from practical point of view. Then we adopt relaxed definitions of privacy, which still provide meaningful security guarantees while allowing us to meet the efficiency requirements. In this setting we design a security architecture and implement a system for data sharing based on encrypted search, which achieves only 30% overhead compared to non-secure solutions on realistic workloads.

Table of Contents

List of Figures

List of Tables

Acknowledgments

Undoubtedly my biggest THANK YOU goes to my family for their endless love and unwavering support for any decision I take and any choice I make, even if that meant going across half of the world. My parents instilled in me the belief that if you work hard you can reach anywhere, and any difficulty on the way is just another challenge to overcome. Their love and care made the person I am, and their hard work gave me the opportunity to follow my dreams. And my sister is one of the dearest people to me who counterbalances me in many good ways. Having her has made everything much more fun. Without all of this I would not be writing these lines today.

My advisors, Tal Malkin and Steve Bellovin, were the people who guided me in my journey during my graduate studies. While I was determined to look for the path that bridges between the tools that theoretical cryptography provides and the real needs of practical applications, I knew this was a way full of pitfalls. I am grateful to my advisors for helping me avoid these pitfalls, for showing me their two perspectives on the problems I was trying to tackle, and for demanding that I do not lose sight of the requirements of both provable and practical security while finding my own way in research. I would also like to thank them for being great people who have shown a lot of understanding on every level, both academic and personal. It has always been fun to laugh together with Tal. And it has always been fascinating to learn about trains, the history of of cryptography and all the other interesting hobbies that Steve has.

Another person who, even though he was not my official advisor, has spent many hours collaborating with me, talking about research and advising me at various important steps in my PhD, is Moti Yung. He has taught be that many times the odds may not be on your side, but if you keep on working hard, things eventually turn around. Moti is an inspiring example for me being a well-established and respected cryptographer, who also spends his

time working at companies collaborating with people who face and need to resolve practical security issues.

Although I have not had a lot of collaboration with Sal Stolfo, I have had many conversation with him that have made me think more about the big picture of security and the contribution I would like to make with my research. He has always been one of the people who want to know how I am doing, and who has always been ready to help with any concern that I have.

I was fortunate to have many mentors and collaborators outside Columbia as well. Certainly, my mentors during my internships at MSR, Seny Kamara and Bryan Parno, have taught me a lot about doing research — about asking interesting questions and looking for innovative ideas, and also about writing and presenting these ideas. My internships at MSR have been a lot fun and have started collaborations that I hope will continue for in the long term. Payman Mohassel and Vinod Vaikuntanathan are other great collaborators that I have met through my work at MSR. I also had the chance to start working on a research project with Craig Gentry and Rosario Gennaro from the crypto group at IBM, and my collaboration with the amazing researchers and wonderful people from that group will continue during my postdoc at IBM.

Of course, this journey would not have been the same and certainly not as much fun without all my friends (and in many cases also collaborators). I would start with Sasha who "found" me on our first day at Columbia and since then she has been a great friend, who has listened to anything that moved me in happy moments and anything that bothered me in hard times. We have had great times with Natalia first in Astoria and NYC, and then in London, Frankfurt, Copenhagen, and probably many other places in the world in the future. Lida and Gabriela have started as my NY friends and have moved to my California friends who always have their homes and their hearts open for me.

And all "the Greeks" at Columbia have made a wonderful group of people with whom I had lots of great times both at school and outside school. It has always been good to have Angelika around to be the woman and friend that I can talk to. Having a great officemate makes going to work much better and this was definitely true for me thanks to my officemate Vasilis. With Maritza and Hang we shared an advisor and we walked together the path of

the PhD. Dov certainly goes in both of the lists of my friends and my collaborators. He is still one of the people that I enjoy most working with but also a friend that is always fun to hang out with.

And there are many other friends that I have met during my numerous internships and my semester at Berkeley (I was lucky to have Carla as my officemate at Berkeley and my friend even since, I hope our paths cross more times in the future). All of them, even for the short time we have spent together, have left lots of memories of good times.

This is the beginning of the list of people that I was fortunate to meet and interact with at different points of my PhD life. Since I will most likely forget someone if I try to mention everyone on this list, I will not try to, but my *thank you* goes to all of them.

Chapter 1

Introduction

We engage daily in numerous electronic interactions that have become an integral part of almost every aspect of our life; whether it is shopping online, managing finances, maintaining social connections, checking medical test results, we conduct many of our activities using the Internet. Often these interactions involve private sensitive information the unwanted exposure of which may have various negative consequences such as putting at material disadvantage, stigmatizing with social prejudice or causing other subjective personal damage. That is why providing security guarantees for such information is an important requirement for the corresponding applications. However, the question of security is often much more complicated than keeping private information isolated and protected from everyone else. Most often the value of the services that people use comes from the fact that they combine and process private data from different resources. Some scenarios that exemplify the importance of bringing together such information are the following: conducting experiments for the effectiveness of newly developed medicine that involves analysis of the results on many patients, finding places of interest near your current location, realizing business transactions such as an auction sale, receiving recommendations based on the preferences and the relevant activities of your friends in a social networking site. In these settings the security goal is to protect information privacy optimally while achieving the desired utility for the service.

Let us pause for a moment and consider the following questions, which illustrate the difference between the utility and the privacy aspects of data in various scenarios. Can we

find out whether the same person has been treated in two different hospitals and share test results from the two places without revealing the identities of other patients? Can we study the effect of a newly developed medicine on patients without revealing their individual test results? Can we analyze data from social networks in order to target better ads and provide more relevant recommendations without revealing the contacts and the online activities of the users? Can we find out whether a suspected terrorist has boarded an airplane without learning the identities of the people on the flight? Can we provide location based services without learning the exact location of the person? Can we conduct an auction without exposing the individual bid prices? Can we ask the cloud to do computation on our behalf without letting him learn our private data and getting assurance of the correctness of the returned result?

The capabilities that the above questions ask for underly many of the services and the applications that people use and expect to have available. They rely on the ability to combine and process private data from multiple sources. On the other hand, the privacy guarantees required concern input data that is necessary to obtain the final result, and yet, is not inherently part of it. In many cases the unwanted exposure of such information (in addition to the output) can be much more damaging than the final output itself. Thus the major challenge that all of these questions pose can be summarized as follows:

Can we conduct computation on private data that reveals nothing more than the desired output and what is inherently leaked by it? Can we do this in an efficient way that will be usable in practical applications?

Given the inherent tension between utilization and protection of sensitive data, the goal formulated above defines the optimal privacy guarantees we can achieve while facilitating the intended use of the private information.

Secure multiparty computation (MPC) is an area of cryptography which offers a formal approach to the above problem. It introduces cryptographic techniques that allow to compute the desired output without revealing the inputs. Formally a secure multiparty computation protocol for the evaluation of a function f on inputs $x_1, \ldots, x_n$ is a protocol that outputs $f(x_1, \ldots, x_n)$, but does not reveal anything more about the inputs $x_1, \ldots, x_n$

than what is inherently leaked by the result $f(x_1, \ldots, x_n)$. Early on work in cryptography research has demonstrated that every functionality can be computed securely [Yao, 1982; Yao, 1986; Goldreich *et al.*, 1987; Chaum *et al.*, 1988b; Ben-Or *et al.*, 1988]. However, for a long time MPC has been considered too impractical for any real application because of the efficiency overhead that it incurs. In the recent years this perception has gradually started to change and there have been multiple efforts towards implementation of MPC techniques [Malkhi *et al.*, 2004; Ben-David *et al.*, 2008; Pinkas *et al.*, 2009; Henecka *et al.*, 2010; Huang *et al.*, 2011; Huang *et al.*, 2012]. While these results have demonstrated working implementations of generic MPC protocols, which makes secure computation a much more tangible option for practical solutions, the functionalities and the inputs that these implementations handle are still far from the complexity and scale of many real world systems that need to deal with privacy preserving computation. Nevertheless, this line of work is a sign that existing generic MPC techniques can be useful for the secure implementation of moderate size building blocks in complex systems.

If we consider scenarios where we need systems that operate on the private data from multiple sources and where multiparty computation solutions would provide the necessary privacy guarantees, we observe that these scenarios present a wide diversity of heterogeneous environments. Here heterogeneity refers to the following aspects of the environment. Different parties have different computation and storage resources. We can take as an example the setting of a big company providing services to its clients. In this case the clients have at their disposal a laptop or a smart phone, which are limited in both CPU power and memory, while the service provider operates with a data center that can handle orders of magnitude more information and computation. Further, large scale computational resources often have distributed nature and can yield different efficiency performance depending on the type of computation and whether it can be easily parallelized.

Another point to consider is the fact that communication channels may not exist between all participants or might not be available at all times during the execution of the protocol. Furthermore, available communication channels might have very different bandwidths. These concerns come up, for example, in the case of multiple clients of a service provider, who do not know about each other and most often cannot communicate directly.

The visitors of a webpage can be expected to interact only once with the web server but would not be available to repeatedly come back and communicate with the website just in order to execute a secure computation protocol. Also the rate-limited data connection of a client's phone can accommodate much less information transfer than the high speed connection between the data centers at the provider's site.

While preserving the privacy of the data that is used in the computation is often an important requirement, maintaining the correctness of the final output is really the first property that we want to guarantee. Seemingly if we are not concerned with data privacy, correctness should not be an issue since each participant should be able to verify the correctness of the final output given the inputs. However, this is not quite the case when we are dealing with parties that have different computational resources such that some of the participants cannot execute the whole computation on their own. In this case achieving efficient verification even without privacy guarantees is an important question. This is quite relevant in setups of outsourced computation where a weak client delegates to a powerful party some computation over his data, and he wants to be able to confirm that the returned output is really the result of the intended evaluation.

A computation protocol prescribes the steps that each party has to follow in its execution. However, the participants may exhibit different adversarial behavior and deviate from the protocol depending on their specific incentives. While some parties may be willing to cheat in the execution if they can obtain any advantage, others may be legally bound to follow honestly all steps but can still try to learn as much as possible from the exchanged information during the execution. Some dishonest parties might choose to collude and share private information in order to get advantage in the final outcome of the protocol. But for others, who might have competing interests in the long run, the potential risks of revealing private information might outweigh the possible gain in the particular computation. Thus, they could still choose to deviate from the prescribed protocol, and yet, would not collude with other parties.

As we saw in the discussion above, once we start to examine carefully the exact settings for which we would like to construct an MPC protocol, we encounter a wide range of diverse computation, communication, and adversarial requirements. On the other hand, generic

multiparty computation techniques address much more homogeneous setups: they require all participants in the protocol to have symmetric resources, both in terms of computation and memory storage. They assume the existence of communication channels between every pair of parties and have symmetric communication patterns. Generic MPC constructions model corruptions among the parties as a monolithic adversary who controls all corrupted participants, all their information and determines the same type of behavior (semi-honest or malicious) for all of them in the protocol execution. Such protocols are not designed to address optimally heterogeneous setups for MPC such as the ones that we discussed above, which hurts the efficiency of the resulting solutions or often makes such generic protocols unusable in settings with highly asymmetric resource distribution.

In this thesis we introduce techniques for multiparty computation that focus on heterogeneous settings. We present solutions tailored to address different types of asymmetric constraints and improve the efficiency of existing approaches in these scenarios. We tackle the question from three main directions. First, we consider the computation model for the evaluated functionality that is used in MPC constructions. While existing protocols use circuits (Boolean and arithmetic) as computational models, which introduces several inherent points of inefficiency in the resulting MPC solutions, we consider new computation models and use them to construct MPC protocols that improve the efficiency guarantees of existing generic approaches. In particular we construct MPC solutions using random access machines and multivariate polynomials to represent the evaluated functionality. Second, we focus on the setting of outsourced computation, where we have one computationally powerful party and several weak clients. This is a setting of increasing importance in view of the ever growing popularization of cloud computing and the concept of providing computation resources as a service. We tackle the question of verification for the delegated computation in this setup and introduce a new paradigm for constructing efficient verifiable computation solutions. Further, we introduce a new adversarial model of non-colluding adversaries, which is weaker than a fully malicious adversarial setting but still models accurately many scenarios of computation outsourcing. We leverage this new security model to construct protocols for outsourced computation that improve the efficiency of the participants. We also consider just the setting of outsourced storage and the ensuing challenges for access

control when sharing outsourced data. Finally, we adopt a different approach to MPC for the setting of encrypted search. Starting with particular efficiency requirements dictated by what is considered usable from a practical point of view, we explore how we can relax in a meaningful way strong security definitions to meet the efficiency threshold. We architect and implement a system which provides these security guarantees and at the same time incurs acceptable efficiency overhead.

1.1 Contributions

1.1.1 New Models for Secure Computation

Traditional approaches for secure computation use circuits (Boolean or arithmetic) as a computation model for the evaluated function. Since circuits have size at least linear in the length of their inputs, this implies that any protocol for secure evaluation that uses circuit function representation will have complexity at least linear in the size of the input on which it depends. Further, the requirement for linearity of the running time for secure protocols appears *inherent* in the fact that if a computation does not "touch" every part from its input, this already leaks information that the untouched parts of the input were not used in the computation. This seems to rule out secure computation as a viable practical solution for many interesting functionalities that take as input huge databases and for which there are sublinear algorithms in the insecure setting. Such an example is *private database search* where binary search gives a solution for the insecure setting that has logarithmic complexity in the database size.

Secure Computation for Random Access Machines. We explore the question of filling in the efficiency gap between protocols in the insecure and the secure setting and propose a two party protocol that achieves sublinear amortized computation in the size of its input. We develop a generic method to compile any two party functionality that can be computed in *sublinear amortized time* in the size of its input on a *random access machine* (RAM) into a secure protocol for the same function that runs in sublinear time [Gordon *et al.*, 2011; Gordon *et al.*, 2012]. The resulting secure protocol has the following

space requirements for the two parties: $O(\log s)$ and $O(s \cdot polylog(s))$ where s is the size of the database. This general compiler makes use of any oblivious RAM protocol, which provides access pattern privacy for memory access, and any protocol for secure two-party computation, which is used for the evaluation of a small number of simple operations. We extend the general construction into an optimized protocol that looks at particular instantiations of the main building block protocols (Yao two party computation and the ORAM construction of Goldreich-Ostrovsky). In this construction we improve the efficiency beyond the asymptotic bounds looking at the exact constants. We minimized the part of the computation that would be implemented with the more expensive generic techniques and develop new primitives such as a *shared oblivious pseudorandom function* that contribute to the efficiency of the protocol.

Secure Computation for Multivariate Polynomials. A large collection of problems are naturally and efficiently expressed as multivariate polynomials over a field or a ring: for example, problems from linear algebra, statistics, logic and set operations, which makes them of practical interest. At the same time multivariate polynomials yield more succinct functionality representation than circuits. We show how to take advantage of this representation and construct a protocol that allows multiple parties to evaluate a *multivariate polynomial* that depends on their private inputs more efficiently than the corresponding generic MPC approaches [Dachman-Soled *et al.*, 2011]. It also achieves better communication complexity than the solution of [Franklin and Mohassel, 2010], the only previous work that focuses on multivariate polynomial evaluation, as well as it answers an open question in [Franklin and Mohassel, 2010] for constructing protocols for polynomials of degree higher than 3 that improve generic MPC techniques. As a special case of our general protocol we propose the first solution for the problem of multi-party set intersection in the fully malicious adversarial model that does not use generic zero knowledge techniques. This work extends the first solution that provides security against malicious adversaries for the problem of two-party set intersection, which we introduced in [Dachman-Soled *et al.*, 2009].

1.1.2 Outsourcing Computation

The recent advent of cloud computing has popularized the paradigm of providing computation resources as a service. The goal in this setting is to utilize the resources of a single powerful service provider for the work that computationally weak clients need to perform on their data with the following caveats. Computation outsourcing can be useful only if the *returned results can be trusted*, and hence we need to enable the clients to efficiently verify the correctness of the returned output. On the other hand, if the clients' input data is sensitive, we need mechanisms that allow the server to process the data while *maintaining its privacy*. While there are solutions for both of these questions based on fully homomorphic encryption (FHE) [Gentry, 2009; Brakerski and Vaikuntanathan, 2011; Brakerski *et al.*, 2012], this is a primitive that is still too expensive for most practical applications. Thus we focus on looking for trade-offs that would allow constructions for outsourced computation with better efficiency not using FHE.

Publicly Verifiable Computation. The goal of verifiable computation (VC) is to provide means for a weak client to efficiently verify the correctness of the results returned for outsourced computation jobs (i.e. doing less work than the job itself). We extend this definition with the following two properties: *public delegation* and *public verifiability*, which are useful for many practical scenarios. Public delegation allows to decouple the party who provides the function to be evaluated and the party who has the input for the computation. Public verifiability enables anyone to verify the correctness of the returned results. To illustrate the importance of these two properties, we can consider the scenario where we outsource the evaluation of an analysis function for the lab tests of hospital patients. In this case public delegation enables the doctor to specify the function that will be evaluated and the lab assistant to provide the input for the computation. The public verifiability property makes it possible that both the doctor and the patients can verify the output of the analysis function on the test results. We show how to construct a verifiable computation scheme, which satisfies both of these properties, from attribute-based encryption [Parno *et al.*, 2012]. Our solution does not use expensive primitives such as fully homomorphic encryption or probabilistically checkable proofs (PCPs), which underly existing VC solu-

tions [Gennaro *et al.*, 2010; Chung *et al.*, 2010; Chung *et al.*, 2011; Bitansky *et al.*, 2011; Goldwasser *et al.*, 2011].

Server-Aided Multiparty Computation with Non-Colluding Adversaries. Existing adversarial models used in the proofs for MPC protocols assume a *monolithic* adversary who controls all corrupted parties among the participants and sees their private inputs. However, there are many instances in practice where collusion between participants is unlikely to happen while each corrupted party may still misbehave on his own. Collusion may be infeasible because it is too costly, it is prevented by physical means, by the Law or due to conflict of interests. An important example for such a heterogeneous environment is the cloud where the users might be in completely different parts of the world and even not know about each other; or the same parties might be interacting in several different contexts and while a possible collusion may give them a short-term advantage, it will be harmful in the long term if they are competitors on the market. Motivated by this application scenario we formally define the *heterogeneous adversarial model* where some of the adversaries are not willing to collude. We present new server-aided MPC protocols providing security in this model with efficiency improvements [Kamara *et al.*, 2011], where all but one of the parties have to do work only proportional to the size of their input. In the constructions of these protocols we introduce a new technique for *oblivious cut-and-choose* that allows to outsource the major part of the verification work for correctness of the construction of Yao garbled circuits to the server facilitating the computation. In addition we show a general transformation from any *delegated computation scheme* into a two-party server-aided protocol. Finally we constructe a specialized protocol for the problem of *set intersection* in the server-aided setting that achieves better efficiency than existing solutions.

Privacy Enhanced Access Control for Outsourced Data Sharing. One of the most popular cloud services is data storage. The next logical step extending this capability is to facilitate data sharing among different clients. Constructing an access control scheme for this setting faces the following challenge: the storage server, which is the first point of access control enforcement, is not the data owner and thus the *access rules as well as the access patterns of the users are private information* that needs to be protected from

him. With these privacy requirements in mind we design a new scheme for access control built for sharing of outsourced data [Raykova *et al.*, 2012]. It divides the outsourced data into *access blocks* and combines different approaches for *coarse-grained* (at block level) and *fine-grained* (within each block) access control to offer a flexible level of trade-offs between efficiency and privacy guarantees.

1.1.3 Efficient Private Search in Practice

There are many scenarios where parties possess data of mutual interest, which they are willing to share but without revealing any other information about the rest of their data sets. Examples for such scenarios include police investigating embezzlement who needs to check banks' databases for information relevant to the case, a physician who wants to find other patients with a rare disease that he is trying to treat and methods that have worked before, institutions who want to detect attacks on their networks using the information from their logs to correlate across different domains. In these settings the parties need methods to find out whether they have data worth sharing and means to exchange such data. One approach for a solution to this problem is to provide search capabilities over the data of one party (data owner) to other parties (queriers). However, providing such capabilities needs to be accompanied by the appropriate privacy guarantees for both parties. For the data owner these are guarantees that the queriers will be able to retrieve only data relevant to their queries and further that only authorized parties will be allowed to submit queries. On the other hand, for the queriers this means keeping their queries private from the data owner and even anonymous (within the set of authorized parties) since in certain scenarios even the intent to query might already be revealing some sensitive information.

Secure Data Sharing with Encrypted Search. We explore solutions for secure search from a different perspective compared to traditional settings for MPC. We start with strict efficiency requirements motivated by speeds of available hardware and what is considered acceptable overhead from practical point of view, and we adopt relaxed definitions of privacy, which still provide meaningful security guarantees while allowing us to meet the efficiency requirements. We design a security architecture and implement a system for data sharing

based on encrypted search [Raykova *et al.*, 2009; Pappas *et al.*, 2011]. Our protocol combines ideas of Bloom filters, a construction for a new private key deterministic encryption scheme and a new primitive called re-routable encryption. We evaluate the performance of our system and its practical usability with tests over tens of gigabytes of data, in which our implementation achieves only 30% overhead compared to the running time for SQL queries on the same database. The latencies for the document retrieval in our system are of the order of the time required for file transfer over SSH. Further, we propose a modification of the protocols that trades off a relaxation of the privacy guarantees for the opportunity of another implementation optimization (bitslicing) that brings several orders of improvement in the search time.

Chapter 2

Related Work

2.0.4 Secure Multiparty Computation

Secure multi-party computation (MPC) addresses the following problem: how a set of n parties, each with a private input, can securely and jointly evaluate an n-party functionality f over their inputs. An MPC protocol guarantees that (1) the parties will not learn any information from the interaction other than their output and what is inherently leaked from it; and (2) that the functionality was computed correctly. A major factor determining the complexity of an MPC protocol is the adversarial model in which it is proven secure. This includes the adversarial behavior that is admissible for the participants: whether they follow the prescribed steps or deviate from the protocol arbitrarily, whether they try to derive additional information from the messages they receive during the execution on their own or collude with other parties. It also depends on the computational power that each participant is assumed to have: whether we would want perfect information theoretic security that provides against adversaries with unbounded computational power, or we are willing to assume limited computational power that allows the use of computational hardness assumptions. While the former provides theoretically stronger security guarantees, it limits the extent of collusion among parties that protocols can handle, allowing at most a half passive corruptions or at most a third active corruptions among the pasties [Ben-Or *et al.*, 1988; Chaum *et al.*, 1988b], and resulting in more computational and communication overhead for the protocol. At the same time for practical purposes computational security

suffices and such setting allows protocols that can be proven secure even in the case of when the majority of the parties behave maliciously and collude.

Early feasibility results in the area [Yao, 1982; Yao, 1986; Goldreich *et al.*, 1987; Ben-Or *et al.*, 1988; Chaum *et al.*, 1988b] demonstrated that any functionality can be computed securely in both the two party and the multiparty setting. There are multiple sources of inefficiency in these early works: their communication and computation complexities depends on the number of wires and gates in the Boolean circuit computing the functionality, and they use generic zero-knowledge proofs or require many rounds of computation. Following works introduce multiple directions of improvement of these initial protocols: constant round [Beaver *et al.*, 1990], black-box use of crypto pseudorandom generators [Damgard and Ishai, 2005; Ishai *et al.*, 2008] , adaptive adversary [Damgard and Ishai, 2006; Ishai *et al.*, 2008], and dishonest majority [Ishai *et al.*, 2008], cut-and-choose techniques that eliminate the need of generic zero-knowledge proofs [Lindell and Pinkas, 2007]. Certain classes of functionalities admit more efficient representation as arithmetic circuits rather than Boolean circuits. There have been corresponding constructions for MPC protocols using arithmetic circuits [Cramer *et al.*, 1999; Ishai *et al.*, 2009; Damgard and Orlandi, 2010; Cramer *et al.*, 2000].

While generic MPC techniques using Boolean or arithmetic circuits can be used for the secure evaluation of any function, such protocols do incur computational and communication overhead proportional to the size of the function circuit. An alternative approach for optimizing the efficiency of MPC protocols is to consider limited classes of functionalities that admit more efficient representation than circuits, which can be used in MPC constructions. Franklin et el. [Franklin and Mohassel, 2010] consider secure computation of a class of functionalities representable as multivariate polynomials. This work focuses on multivariate polynomials of degree 3 but points out that the proposed protocols can be generalized to higher degree polynomials, however, with communication complexity that is no longer optimal and leaves as an open question improvements of this complexity. Two functionalities that can be expressed as multivariate polynomials are oblivious polynomial evaluation and set intersection. Oblivious polynomials evaluation [Naor and Pinkas, 2006] gives a secure solution for class of two party computation functionalities where the inputs

one of the parties are coefficients of a polynomial and the inputs of the other party are the evaluation points. The problem of set intersection asks how several parties with private input sets can compute the intersection of these sets. It has been considered in several works providing secure computation solutions specifically for this problem. The two-party variant of the set intersection problem has been addressed in a series of works [Freedman *et al.*, 2004; Hazay and Lindell, 2008; Kissner and Song, 2005; Jarecki and Liu, 2009; Dachman-Soled *et al.*, 2009] providing solutions in various different adversarial models. The only work that has considered specifically the problem of multiparty set intersection in the malicious adversarial model is [Dawn and Song, 2005] giving a semi-honest protocol and suggesting to use generic zero knowledge techniques to address the malicious case, which incurs substantial complexity overhead.

Random access machines present a different computational model, where the computation can be expressed as a series of small computations and reads/write instructions into memory, which stores the inputs and intermediate state for the protocol execution. The works of [Damgard *et al.*, 2010] and [Ostrovsky and Shoup, 1997] observe that this model can be also used for secure computation. Specifically, these works consider the following scenarios: two parties share the *entire* (super-linear) memory state for the protocol in [Damgard *et al.*, 2010], and a (stateless) client storing data on *two* servers that are assumed not to collude [Ostrovsky and Shoup, 1997].

Collusion. The problem of collusion in MPC was first explicitly considered in the work Lepinski, Micali and Shelat [Lepinksi *et al.*, 2005], where they defined and gave constructions of collusion-free protocols. Roughly speaking, an MPC protocol is collusion-free if it meets all the standard security properties and, in addition, it cannot be used as a covert channel. While the protocol of [Lepinksi *et al.*, 2005] relies on physical assumption (e.g., ballot boxes and secure envelopes), recent work by Alwen, Shelat and Visconti [Alwen *et al.*, 2008] and Alwen, Katz, Lindell, Persiano, Shelat and Visconti [Alwen *et al.*, 2009] shows how to construct collusion-free protocols that rely only on a trusted mediator.

2.0.5 Delegation of Computation

The goal of a verifiable computation (VC) scheme is to provide a way to efficiently verify the correctness results of outsourced computation. Solutions for this problem have been proposed in various settings. These include interactive proofs [Lund *et al.*, 1992; Shamir, 1992; Fortnow and Lund, 1991; Goldwasser *et al.*, 2008] and interactive arguments [Kilian, 1992; Brassard *et al.*, 1988; Micali, 1994]. However, in the context of delegated computation, a non-interactive approach for verifiability is much more desirable. CS proofs [Micali, 1994] realize a non-interactive argument in the random oracle model where the verification work is logarithmic in the complexity of the computation performed by the worker. Goldwasser, Kalai and Rothblum [Goldwasser *et al.*, 2008] construct a two message (non-interactive) protocol for functions in NC, where the verifier's running time depends on the depth of the circuit for the evaluated function.

The first solutions that provide single round verifiable computation schemes secure in the standard model for any polynomial-time computable function are the works of Gennaro, Gentry, and Parno [Gennaro *et al.*, 2010] and Chung, Kalai, and Vadhan [Chung *et al.*, 2010]. Both constructions employ fully homomorphic encryption for the evaluation of the delegated function, and neither can safely provide oracle access to the verification algorithm. This problem is resolved by Chung et al. [Chung *et al.*, 2011], who consider the setting of memory delegation, where all inputs are preprocessed and given to the worker who will later execute multiple computations on them. Similar to the non-interactive solution of Goldwasser et al. [Goldwasser *et al.*, 2008], the effort required to verify results from memory delegation is proportional to the depth of the computation's circuit, which for certain functions may be proportional to the circuit size (e.g., exponentiation). The recent works of Bitansky et al. [Bitansky *et al.*, 2011] and Goldwasser et al. [Goldwasser *et al.*, 2011] also achieve reusable soundness, though they rely on non-falsifiable "knowledge of exponent" type assumptions to do this. Specifically, Bitansky et al. [Bitansky *et al.*, 2011] present a construction for succinct non-interactive arguments based on a combination of PCP and PIR techniques, while Goldwasser et al. [Goldwasser *et al.*, 2011] give a construction for designated verifier CS proofs for polynomial functions, which also employs leveled fully homomorphic encryption.

Barbosa and Farshim [Barbosa and Farshim, 2011] construct a verifiable computation protocol for arbitrary functions (without the rejection problem) from fully homomorphic encryption and functional encryption. Similar to the proposal of Applebaum, Ishai, and Kushilevitz [Applebaum *et al.*, 2010], their protocol calculates a verifiable MAC over the computation's result, allowing efficient verification. However, this approach relies on powerful functional encryption functionality (e.g., the ability to compute MACs) that are currently not known to be achievable.

The solutions of Benabbas, Gennaro, and Vahlis [Benabbas *et al.*, 2011] and Papamanthou, Tamassia, and Triandopoulos [Papamanthou *et al.*, 2011] provide verifiable computation schemes for smaller classes of functions, polynomials and set operations respectively, but using more efficient tools than FHE or PCPs. Although VC schemes with reusable soundness protect against cheating even when the worker learns the output of the verification algorithm, they do not provide public verifiability where anyone can check the correctness of the result. The only exception is the work of Papamanthou et al. [Papamanthou *et al.*, 2011] which allows anyone who receives the result of the set operation to verify its correctness.

2.0.6 Secure Search

Secure search considers the following question: there are two parties, one of which holds a database and the other has a query, and we want to enable the querier to submit his query and learn the relevant results from the database without leaking any private information about either participant where private information is defined as follows.The query is always private information that should be protected from the data owner. As far as the database is concerned there are two main types of scenarios that have different requirements: in *data outsourcing* the stored database is owned by the querier and in *data sharing* the querier and the data owner are different parties. The latter case requires that any non-matching information should be kept private from the querier. A generalization of the problem allows multiple querying parties, which introduces issues of access control and revocation of search capabilities as well as anonymity of the querier among all authorized parties for search. The problem of secure anonymous database search can be solved with general secure multiparty

techniques, which, however, will not be optimal in terms of efficiency and would not be suitable for practical purposes. Since the problem is relevant to many real scenarios there are numerous protocols that offer solutions specifically for this setting.

Protocols for Private Information Retrieval (PIR) [Chor *et al.*, 1998] and Symmetric Private Information Retrieval (SPIR) [Gertner *et al.*, 2000] provide a limited type of privacy preserving search. The scenario that PIR addresses is between two parties: server and client, where the server has a database of n items and the client wants to obtain the item at position i without the server learning the value of i. In the case of SPIR, it is additionally required that the user does not learn any other item except the one that was requested. These protocols have sub-linear communication and polynomial computational complexity, already improving on generic multiparty computation protocols, but still remain inefficient for many practical uses. Additionally, these protocols typically support only simple selection, rather than general query capability (a notable exception being [Chor *et al.*, 1997]).

Many papers address the scenario of database outsourcing [Song *et al.*, 2000; Boneh *et al.*, 2004; Boneh and Waters, 2007; Boneh *et al.*, 2007; Williams and Sion, 2008; Williams *et al.*, 2008; Curtmola *et al.*, 2006; cheng Chang and Mitzenmacher, 2005; Aviv *et al.*, 2007]. In this setting one party possesses data but does not have enough resources to store it. He keeps the data on an untrusted storage server, but maintains the ability to search the data without leaking any information to the server. The approaches of [Song *et al.*, 2000; Boneh *et al.*, 2004; Boneh and Waters, 2007; Boneh *et al.*, 2007] use encryption systems that allow matching of ciphertexts of the same encrypted word and enable search over the encrypted content of documents. Thus the running time of the search in these approaches is linear in the number of all searchable tokens. Bellare et al. [M. Bellare and O'Neill, 2007] show that in order to achieve better than linear complexity of search the mechanism for computing the searchable tags needs to be deterministic, which affects the security guarantees that can be proven for the protocol. This pinpoints the issue for tradeoff between efficiency and strong privacy guarantees. Curtmola et al. [Curtmola *et al.*, 2006] use the idea of inverted indices for efficiency gain and suggest the querier preprocess the data by computing inverted indices on search words. An inherent leakage in this case is the search pattern over multiple queries. The works of [Bellovin and Cheswick, 2007] and [Goh, 2004]

use Bloom filters as a basis for their search structures, which allow efficiency improvement but weaker privacy definitions.

In data sharing an important question relevant to the leakage of information is how and at what granularity the search capability is granted. The works of [Waters *et al.*, 2004] and [Shi *et al.*, 2007] assume the existence of an authorization party that can provide search tokens for words that the querier is allowed to decrypt. The approach of [Bellovin and Cheswick, 2007] allows that search capabilities are granted for a collection of documents as opposed to separate words, which will be more relevant in cases where data sharing should be enabled for the whole content of a particular set of documents.

2.0.7 Oblivious Random Access Memory (ORAM)

Oblivious RAM was introduced by Goldreich and Ostrosky [Goldreich and Ostrovsky, 1996] as a solution that allows storage and data access on an untrusted server while hiding the access pattern and avoiding computational overhead linear in the database size. The main idea behind the construction is that the entries in the database are associated with virtual addresses, which serve as their searchable tags, and these virtual addresses change their actual physical location in memory each time they are accessed (read or written). For this purpose the database with n entries is preprocessed to be stored encrypted in a multilevel structure with $\log n$ levels. Using this structure each operation (read or write) accesses all elements in the first level (the cache) and a constant number of elements in each of the rest of the levels of the ORAM. After a certain number of data operations some part of the stored encrypted database needs to be reshuffled in order to maintain the hiding property for the data access pattern.

Pinkas et al. [Pinkas and Reinman, 2010] suggested an optimization for the construction of [Goldreich and Ostrovsky, 1996] that uses Cuckoo hash tabled for data storage at each level which results in smaller space and computation overhead. However, an attack [Goodrich and Mitzenmacher, 2011; Kushilevitz *et al.*, 2012; Gordon *et al.*, 2011] on the scheme of [Pinkas and Reinman, 2010] have demonstrated that additional care is required when using Cuckoo hash. The idea of the attack is related to the fact that even if a Cuckoo hash can hold all the elements assigned to a particular level, there still might be search

sequences of elements that result in collisions that are incompatible with the structure of the Cuckoo hash tables. The fix for the issue [Goodrich and Mitzenmacher, 2011] is a new construction for a Cuckoo hash table in which all possible collisions are allocated into a separate stash. Several following works have adopted with some modifications the approach relying on Cuckoo hashing [Goodrich and Mitzenmacher, 2011; Goodrich *et al.*, 2011; Kushilevitz *et al.*, 2012]. The work of [Kushilevitz *et al.*, 2012] improves the computational complexity of accesses to $O(\log^2 n/\log\log N)$. An alternative construction for oblivious RAM, which does not use the hierarchical memory structure but rather a series of recursive binary trees and avoids the need of oblivious shuffles, is suggested by Shi et al. [Shi *et al.*, 2011]. The works of [Ajtai, 2010] and [Damgard *et al.*, 2010] provide ORAM construction that avoid the use of pseudorandom functions and achieve information theoretic security.

Part I

New Computational Models

Chapter 3

Secure Computation with Sublinear Amortized Work

3.1 Motivation and Contributions

Consider the natural task of searching over a sorted database of n elements. Using binary search, this can be done in time $O(\log n)$. Next consider a secure version of this problem where a client holds an item and wants to learn whether this item is present in a database held by a server, with neither party learning anything else. Applying standard protocols for secure computation to this task, we would find that they begin by expressing the computation as a (binary or arithmetic) circuit of size at least n, resulting in protocols of complexity $\Omega(n)$. Moreover, it is well known that this is inherent. Namely, in any secure protocol for this problem the server must "touch" every bit of its database; otherwise, the server learns some information about the client's input from the portions of its database that were never touched.

One may notice two opportunities for improvement:

- Any circuit computing a non-trivial function f on inputs of length n must have size $\Omega(n)$. On the other hand, many interesting functions can be computed in *sublinear* time on a random-access machine (RAM). Thus, it would be desirable to have protocols for generic secure computation that use RAMs — rather than circuits — as

their starting point.

- The fact that linear work (or more) is inherent for secure computation of any non-trivial function f only applies when f is computed *once*. However, it does not rule out the possibility of doing better, in an amortized sense, when the parties compute the function *several* times.

Inspired by the above, we explore scenarios where secure computation with *sublinear* amortized work is possible. We focus on a setting where a client and server repeatedly evaluate a function f, maintaining state across these executions, with the server's (huge) input D given at the outset and the client's (small) input x chosen anew each time f is evaluated. Our main result is:

Theorem 1 (Informal)**.** *Say $f(x, D)$ can be computed in time t and space s in the RAM model of computation. Then there is a secure two-party protocol computing f in which the client and server run in amortized time $O(t) \cdot \mathsf{polylog}(s)$, the client uses space $O(\log(s))$, and the server uses space $O(s \cdot \mathsf{polylog}(s))$.*

We show a generic protocol achieving the above bounds based on any oblivious RAM (ORAM) construction and any secure two-party computation protocol, following an idea of Ostrovsky and Shoup [Ostrovsky and Shoup, 1997]. The resulting protocol demonstrates the feasibility of sublinear-complexity secure computation, and serves as a useful template for our second, optimized construction. Here we use a specific ORAM construction, and design the protocol so that generic secure computation is utilized only for a small number of simple operations. The resulting protocol is much more efficient.

3.2 Solution Overview

Our starting point is the ORAM primitive [Goldreich and Ostrovsky, 1996], which allows a client (with small memory) to perform RAM computations using the (large) memory of a remote untrusted server. At a high level, the client stores its encrypted memory cells on the server and then emulates a RAM computation of some function f by replacing each read/write access of the original RAM computation with a series of read/write accesses

of the remote data, such that the client's actual access pattern remains hidden from the server. Results of Goldreich and Ostrovsky [Goldreich and Ostrovsky, 1996], since improved by others (see Section 2.0.7), show that if f can be computed on a RAM in t steps and space s (see Section 3.3.1 for our formal model of RAM algorithms), then it can be computed on an ORAM in $t \cdot \mathsf{polylog}(s)$ steps while using $s \cdot \mathsf{polylog}(s)$ space at the server.

In our setting, ORAM suggests a candidate protocol for computing f with sublinear amortized overhead. Say the server starts with input D, and the client wants to compute $f(x_i, D)$ for a sequence of inputs $x_1, x_2, \ldots$. The client and server (interactively) pre-process D as required for the ORAM construction. This pre-processing step will take (at least) time linear in $|D|$, but will be amortized over several computations of f. In each computation, the client and server run the ORAM protocol until the client learns the output. If f can be evaluated in t steps on a RAM, then each such evaluation can be done in time $t \cdot \mathsf{polylog}(|D|)$.

The above protocol provides "one-sided security," in that it ensures privacy of the client's input against a semi-honest server. (Though, in fact, it was already shown by Goldreich and Ostrovsky [Goldreich and Ostrovsky, 1996] how malicious behavior by the server can be addressed.) However, it provides no security guarantees for the server! We can address this by running each ORAM instruction inside a (standard) secure two-party computation protocol, with intermediate states being shared between the client and server. This is the basic idea behind our generic construction, as described in detail in Section 3.4. We note that this idea can be traced to the work of Ostrovsky and Shoup [Ostrovsky and Shoup, 1997] from 1997 (see Section 2.0.4).

In our second construction, we optimize the efficiency of the above by building on the *specific* ORAM construction of Goldreich and Ostrovsky [Goldreich and Ostrovsky, 1996] and aiming to minimize our reliance on generic secure computation. In particular, we make sure that generic secure computation is applied only to very small circuits. To reduce our reliance on generic secure computation, we design a protocol for obliviously evaluation of a pseudorandom function (PRF) where both the key and the input/output are shared; this may be of independent interest. With careful attention to detail, and several important changes to the underlying ORAM protocol, we end up with a much more efficient protocol. We describe this in detail in Section 3.5.

In the course of proving security of our protocol, we identified a security issue with some previous ORAM constructions; our observations impact the security of the Pinkas-Reinman protocol [Pinkas and Reinman, 2010] as well as the analysis (and the security for some parameter settings) of the Goldreich-Ostrovsky protocol [Goldreich and Ostrovsky, 1996].[1] See Section 3.5.4 for further discussion.

3.3 Preliminaries

3.3.1 Random Access Machines

In this work, we focus on RAM programs for computing a function $f(x, D)$, where x is a "small" input that can be read in its entirety and D is a larger array that is accessed via a sequence of read and write instructions. Any such instruction $I \in (\{\mathsf{read}, \mathsf{write}\} \times \mathbb{N} \times \{0,1\}^*)$ takes the form (write, v, d) ("write data element d in location/address v") or $(\mathsf{read}, v, \perp)$ ("read the data element stored at location v"). We also assume a designated "stop" instruction of the form (stop, z) that indicates termination of the RAM protocol with output z.

Formally, a RAM program is defined by a "next instruction" function Π which, given its current state and a value d (that will always be equal to the last-read element), outputs the next instruction and an updated state. Thus if D is an array of n entries, each ℓ bits long, we can view execution of a RAM program as follows:

- Set $\mathsf{state}_\Pi = (1^{\log n}, 1^\ell, \mathsf{start}, x)$ and $d = 0^\ell$. Then until termination do:
  1. Compute $(I, \mathsf{state}'_\Pi) = \Pi(\mathsf{state}_\Pi, d)$. Set $\mathsf{state}_\Pi = \mathsf{state}'_\Pi$.
  2. If $I = (\mathsf{stop}, z)$ then terminate with output z.
  3. If $I = (\mathsf{write}, v, d')$ then set $D[v] = d'$.
  4. If $I = (\mathsf{read}, v, \perp)$ then set $d = D[v]$.

(We stress that the contents of D change during the course of the execution.) To make things non-trivial, we require that the size of state_Π, and the space required to compute Π,

[1] Independent of (but prior to) our work, similar flaws have been pointed out by others [Goodrich and Mitzenmacher, 2011; Kushilevitz *et al.*, 2011].

is polynomial in $\log n, \ell$, and $|x|$. (Thus, if we view a client running Π and issuing instructions to a server storing D, the space used by the client is small.)

We allow the possibility for D to grow beyond n entries, so the RAM program may issue write (and then read) instructions for indices greater than n. The space complexity of a RAM program on inputs x, D is the maximum number of entries used by D during the course of the execution. The time complexity of a RAM program on the same inputs is the number of instructions issued in the execution as described above. For our application, we do not want the running time of a RAM program to reveal anything about the inputs. Thus, we will assume that any RAM program has associated with it a polynomial t such that the running time on x, D is exactly $t(n, \ell, |x|)$.

3.3.2 Oblivious RAM

We view an oblivious-RAM (ORAM) construction as a mechanism that simulates read/write access to an underlying (virtual) array D via accesses to some (real) array $\tilde{D}$; "obliviousness" means that no information about the virtual accesses to D is leaked by observation of the real accesses to $\tilde{D}$. An ORAM construction can be used to compile any RAM program into an oblivious version of that program.

An ORAM construction consists of two algorithms $\mathsf{ORAMInit}$ and $\mathsf{ORAMEval}$ for initialization and execution, respectively. $\mathsf{ORAMInit}$ initializes some state $\mathsf{state}_{\mathsf{oram}}$ that is used (and updated by) $\mathsf{ORAMEval}$. The second algorithm, $\mathsf{ORAMEval}$, is used to compile a single read/write instruction I (on the virtual array D) into a sequence of read/write instructions $\tilde{I}_1, \tilde{I}_2, \ldots$ to be executed on (the real array) $\tilde{D}$. The compilation of an instruction I into $\tilde{I}_1, \tilde{I}_2, \ldots$, can be adaptive; i.e., instruction $\tilde{I}_j$ may depend on the values read in some prior instructions. To capture this, we define an iterative procedure called $\mathsf{doInstruction}$ that makes repeated use of $\mathsf{ORAMEval}$. Given a read/write instruction I, we define $\mathsf{doInstruction}(\mathsf{state}_{\mathsf{oram}}, I)$ as follows:

- Set $d = 0^\ell$. Then until termination do:
  1. Compute $(\tilde{I}, \mathsf{state}'_{\mathsf{oram}}) \leftarrow \mathsf{ORAMEval}(\mathsf{state}_{\mathsf{oram}}, I, d)$, and set $\mathsf{state}_{\mathsf{oram}} = \mathsf{state}'_{\mathsf{oram}}$.
  2. If $\tilde{I} = (\mathsf{done}, z)$ then terminate with output z.

3. If $\tilde{I} = (\mathsf{write}, v, d')$ then set $\tilde{D}[v] = d'$.
4. If $\tilde{I} = (\mathsf{read}, v, \perp)$ then set $d = \tilde{D}[v]$.

If I was a read instruction with $I = (\mathsf{read}, v, \perp)$, then the final output z should be the value "written" at $D[v]$. (See below, when we define correctness.)

Correctness. We define correctness of an ORAM construction in the natural way. Let $I_1, \ldots, I_k$ be any sequence of instructions with $I_k = (\mathsf{read}, v, \perp)$, and $I_j = (\mathsf{write}, v, d)$ the last instruction that writes to address v. If we start with $\tilde{D}$ initialized to empty and then run $\mathsf{state}_{\mathsf{oram}} \leftarrow \mathsf{ORAMInit}(1^\kappa)$ followed by $\mathsf{doInstruction}(I_1), \ldots, \mathsf{doInstruction}(I_k)$, then the final output will be equal to d with all but negligible probability.

Security. Intuitively, the security requirement is that for any two equal-length sequences of RAM instructions, the (real) access patterns generated by those instructions will be indistinguishable. We will use the standard definition from the literature, which assumes the two instruction sequences are chosen in advance.[2] Formally, let $\mathcal{ORAM} = \langle \mathsf{ORAMInit}, \mathsf{ORAMEval} \rangle$ be an ORAM construction and consider the following experiment:

Experiment $\mathsf{ExpAPH}_{\mathcal{ORAM},\mathsf{Adv}}(\kappa, b)$:

1. The adversary Adv outputs two sequences of queries $(\mathbf{I}^0, \mathbf{I}^1)$, where $\mathbf{I}^0 = \{I_1^0, \ldots, I_k^0\}$ and $\mathbf{I}^1 = \{I_1^1, \ldots, I_k^1\}$ for arbitrary k.
2. Run $\mathsf{state}_{\mathsf{oram}} \leftarrow \mathsf{ORAMInit}(1^\kappa)$; initialize $\tilde{D}$ to empty; and then execute $\mathsf{doInstruction}(\mathsf{state}_{\mathsf{oram}}, I_1^b)$, $\ldots$, $\mathsf{doInstruction}(\mathsf{state}_{\mathsf{oram}}, I_k^b)$ (note that $\mathsf{state}_{\mathsf{oram}}$ is updated each time $\mathsf{doInstruction}$ is run). The adversary is allowed to observe $\tilde{D}$ the entire time.
3. Finally, the adversary outputs a guess $b' \in \{0, 1\}$. The experiment evaluates to 1 iff $b' = b$.

[2] It appears that existing ORAM constructions are secure even if the adversary is allowed to adaptively choose the next instruction after observing the access pattern on $\tilde{D}$ caused by the previous instruction. Since this has not been claimed by any ORAM construction in the literature, we do not define it.

Definition 1. *An ORAM construction* $\mathcal{ORAM} = \langle \mathsf{ORAMInit}, \mathsf{ORAMEval} \rangle$ *is* access-pattern hiding *against honest-but-curious adversaries if for every* PPT *adversary* Adv *the following probability, taken over the randomness of the experiment and* $b \in_R \{0,1\}$*, is negligible:*

$$\left| \Pr\left[\mathsf{ExpAPH}_{\mathcal{ORAM},\mathsf{Adv}}(1^\kappa, b) = 1 \right] - \frac{1}{2} \right|.$$

3.3.3 Secure Computation

We focus on the setting where a server holds a (large) database D and a client wants to repeatedly compute $f(x, D)$ for different inputs x; moreover, f may also change the contents of D itself. We allow the client to keep (short) state between executions, and the server will keep state that reflects the (updated) contents of D.

For simplicity, we focus only on the two-party (client/server) setting in the semi-honest model but it is clear that our definitions can be extended to the multi-party case with malicious adversaries.

Definition of security. We use a standard simulation-based definition of secure computation [Goldreich, 2001], comparing a real execution to that of an ideal (reactive) functionality F. In the ideal execution, the functionality maintains the updated state of D on behalf of the server. We also allow F to take a description of f as input (which allows us to consider a single ideal functionality).

The real-world execution proceeds as follows. An environment Z initially gives the server a database $D = D^{(1)}$, and the client and server then run protocol Π_f (with the client using input init and the server using input D) that ends with the client and server each storing some state that they will maintain (and update) throughout the subsequent execution. In the ith iteration ($i = 1, \ldots$), the environment gives x_i to the client; the client and server then run protocol Π_f (with the client using its state and input x_i, and the server using its state) with the client receiving output out_i. The client sends out_i to Z, thus allowing adaptivity in Z's next input selection x_{i+1}. At some point, Z terminates execution by sending a special end message to the players. At this time, an honest player simply terminates execution; a corrupted player sends its entire view to Z.

For a given environment Z and some fixed value κ for the security parameter, we let

$\text{REAL}_{\Pi_f,\text{Z}}(\kappa)$ be the random variable denoting the output of Z following the specified execution in the real world.

In the ideal world, we let F be a trusted functionality that maintains state throughout the execution. An environment Z initially gives the server a database $D = D^{(1)}$, which the server in turn sends to F. In the ith iteration ($i = 1, \ldots$), the environment gives x_i to the client who sends this value to F. The trusted functionality then computes

$$(\mathsf{out}_i, D^{(i+1)}) \leftarrow f(x_i, D^{(i)}),$$

and sends out_i to the client. (Note the server does not learn anything from the execution, neither about out_i nor about the updated contents of D.) The client ends out_i to Z. At some point, Z terminates execution by sending a special end message to the players. The honest player simply terminates execution; the corrupted player may send an arbitrary function of its entire view to Z.

For a given environment Z, some fixed value κ for the security parameter, and some algorithm $\mathcal{S}$ being run by the corrupted party, we let $\text{IDEAL}_{F,\mathcal{S},\text{Z}}(\kappa)$ be the random variable denoting the output of Z following the specified execution.

Definition 2. *We say that* protocol Π_f securely computes f *if there exists a probabilistic polynomial-time ideal-world adversary* $\mathcal{S}$ *(run by the corrupted player) such that for all non-uniform, polynomial-time environments* Z *there exists a negligible function* negl *such that*

$$\left|\Pr\left[\text{REAL}_{\Pi_f,\text{Z}}(\kappa) = 1\right] - \Pr\left[\text{IDEAL}_{F,\mathcal{S},\text{Z}}(\kappa) = 1\right]\right| \leq \mathsf{negl}(\kappa).$$

Remark: "adaptivity" in the choice of the $\{x_i\}$. In the "standard" ideal-world definition of reactive computation, a corrupted player (either client or server) would give its entire view to the environment each time the functionality F is accessed. Here, however, we allow the players to give its view to the environment only at the end of the entire execution. This seems to be reasonable in the semi-honest setting we consider, where a subsequent input x_{i+1} should have no dependence on the view of the ith protocol execution. (On the other hand, we do allow x_{i+1} to depend on $\mathsf{out}_1, \ldots, \mathsf{out}_i$, a dependence that is realistic.) In fact, our protocols satisfy the stronger notion (where a corrupted server gives its view to Z after each execution of the protocol in the real world, and gives an arbitrary function

Secure initialization protocol

Input: The server has input D of length n, and the client does not use its input in this stage.

Protocol:

1. The participants run a secure computation of $\mathsf{ORAMInit}(1^\kappa)$, which results in each party receiving a secret share of the initial ORAM state. We denote this by $[\mathsf{state}_{\mathsf{oram}}]$.
2. For $i = 1, \ldots, n$ do
   (a) The server creates instruction $I = (\mathsf{write}, v, D[v])$ and secret shares it with the client. We denote the resulting sharing by $[I]$.
   (b) The parties execute $([\mathsf{state}'_{\mathsf{oram}}], [\perp]) \leftarrow \mathsf{doInstruction}([\mathsf{state}_{\mathsf{oram}}], [I])$ (see Figure 3.3), and set $[\mathsf{state}_{\mathsf{oram}}] \leftarrow [\mathsf{state}'_{\mathsf{oram}}]$.

Figure 3.1: Secure initialization protocol π_{Init}.

of its view to Z after each iteration in the ideal world) as long as the underlying ORAM construction they use satisfies the adaptive notion of security discussed in footnote 2. (To the best of our knowledge, this property has not been considered in any prior work on ORAM. Nevertheless, we conjecture that all known constructions are secure even under adaptive choice of instructions.)

3.4 Generic Construction

In this section we present our generic solution to the amortized sublinear secure computation problem. The construction is based in a black-box manner on any ORAM scheme and any secure two-party computation protocol. While our second protocol, which we present in Section 3.5, is substantially more efficient than any specific instantiation of the protocol in this section, this generic protocol is conceptually simple and clean, demonstrates theoretical feasibility, and provides a good overview of our overall approach.

Our first observation is that the server can store his *own* data in his *own* ORAM structure: the security definition of ORAM in Section 3.3 guarantees security against a semi-honest server even when he knows the data content completely. This allows us to give

Secure evaluation protocol π_f

Inputs: The server has array $\tilde{D}$ and the client has input $1^{\log n}, 1^\ell$, and x. Also the server and the client have secret shares of an ORAM state, denoted $[\mathsf{state}_{\mathsf{oram}}]$.

Shared input: A RAM program defined by the "next-instruction function" Π.

Protocol:

1. The client sets $\mathsf{state}_\Pi = (1^{\log n}, 1^\ell, \mathsf{start}, x)$ and $d = 0^\ell$ and secret shares both values with the server; we denote the shared values by $[\mathsf{state}_\Pi]$ and $[d]$, respectively.
2. Do:
   (a) The parties securely compute $([I], [\mathsf{state}'_\Pi]) \leftarrow \Pi([\mathsf{state}_\Pi], [d])$, and set $[\mathsf{state}_\Pi] = [\mathsf{state}'_\Pi]$.
   (b) The parties perform a secure computation to check whether $\mathsf{state}_\Pi = (\mathsf{stop}, z)$. If so, break.
   (c) The parties execute $([\mathsf{state}'_{\mathsf{oram}}], [d']) \leftarrow \mathsf{doInstruction}([\mathsf{state}_{\mathsf{oram}}], [I])$. They set $[\mathsf{state}_{\mathsf{oram}}] = [\mathsf{state}'_{\mathsf{oram}}]$ and $[d] = [d']$.
3. The server sends its share of $[\mathsf{state}_\Pi]$ and $[d]$ to the client, who recovers the output z.

Output: The client outputs z.

Figure 3.2: Secure evaluation of a RAM program.

the client access to the server's data without violating client privacy, and without requiring him to store a secret-sharing of the entire database. We now only need to ensure that the client does not learn any information either. Our second observation is that this can be achieved, at a cost *independent* of the size of D, by always secret-sharing the client's state with the server (this keeps the client oblivious), and by facilitating the ORAM operations using standard secure computation techniques on their joint state. More specifically, we will use MPC to compute each next-instruction function in RAM, and then further to compile each RAM instruction into a sequence of ORAM instructions. ORAM instructions are then reconstructed and executed by the server (they can safely be shown to the server), and the result of the RAM instructions is secret-shared with the client to form part of the updated client's state. The players then use the updated state to continue with the evaluation of

the next RAM instruction.

In more technical detail, let f, encoded as a RAM next-instruction function Π, be the computed function. Notation-wise, for value v, let $[v]$ denote a bitwise secret-sharing of v between the two parties. Our secure ORAM protocol proceeds as follows:

1. The parties run a secure computation of $\mathsf{ORAMInit}$ (Figure 3.1). This initializes the ORAM structure, and securely populates it with Server's data D.
2. The parties securely generate and evaluate the RAM program (Figure 3.2). That is, the following is repeated until the RAM protocol terminates:
   (a) The server and the client use MPC to evaluate Π and obtain shares of the next instruction I.
   (b) I is then compiled, through repeated secure computations of $\mathsf{ORAMEval}$, into a

The $\mathsf{doInstruction}$ subroutine

Inputs: The server has array $\tilde{D}$, and the server and the client have secret shares of an ORAM state (denoted $[\mathsf{state}_{\mathsf{oram}}]$) and a RAM instruction (denoted $[I]$).

1. The server sets $d = 0^\ell$ and secret shares this value with the client; we denote the shared value by $[d]$.
2. Do:
   (a) The parties securely compute $([\hat{I}], [\mathsf{state}'_{\mathsf{oram}}]) \leftarrow \mathsf{ORAMEval}([\mathsf{state}_{\mathsf{oram}}], [I], [d])$, and set $[\mathsf{state}_{\mathsf{oram}}] = [\mathsf{state}'_{\mathsf{oram}}]$.
   (b) The parties perform a secure computation to check if $[\hat{I}] = (\mathsf{done}, z)$. If so, set $[d] = [z]$ and break.
   (c) The client sends its share of $[\hat{I}]$ to the server, who reconstructs $[\hat{I}]$. Then:
      i. If $\hat{I} = (\mathsf{write}, v, d')$ then the server sets $\tilde{D}[v] = d'$ and sets $d = d'$.
      ii. If $\hat{I} = (\mathsf{read}, v, \perp)$ then the server sets $d = \tilde{D}[v]$.
   (d) The server secret shares d with the client.

Output: Each player outputs his share of $\mathsf{state}_{\mathsf{oram}}$ and his share of $[d]$.

Figure 3.3: Subroutine for executing one RAM instruction.

sequence of sub-queries, $\hat{I}_1, \ldots \hat{I}_\ell$, where $\ell = O(\log n)$.

(c) After the server executes each of the sub-queries, instruction I is complete. (In case of a read instruction, the resulting data item is secret shared between the server and the client.)

Again, we stress that, although we do use generic MPC, it is independent of the size of D, and depends only on the RAM representation of f (i.e., Π), and on the complexity of the ORAMEval function. Of course, using generic MPC creates significant overhead. In Section 3.5 we present several tailored MPC protocols for computing the ORAM steps, which greatly improve the performance of our approach.

3.4.1 Proof of Security

We now prove that the construction presented in the previous section is a secure MPC protocol according to Definition 2.

At the very high level, security against the client holds because he only manipulates the data protected by secret sharing and MPC; the server additionally sees plaintext ORAM instructions – but they do not reveal anything by the ORAM guarantee. (ORAM security [Goldreich and Ostrovsky, 1996] is proven in the non-adaptive setting only. However, as we will show, our security simulation goes through, since the adaptive input and function selection by Z does not depend on protocol message view, and hence the simulators can query the ORAM functions *after* Z had completed the adaptive selection.)

We start with the descriptions of the Client simulator S_{cl}, who interacts with Z. In i-th computation, S_{cl} receives x_i and $y_i = f(x_i, D^{(i-1)})$, stores them, and postpones it simulation until he receives the special end symbol from Z.

At this point, S_{cl} outputs entire simulation, as follows:

Pre-processing. S_{cl} simulates pre-processing by generating an appropriate number of random ORAM state shares:

1. S_{cl} runs the $\mathsf{ORAMInit}(1^\kappa)$ functionality to obtain an initial state for the ORAM, and generates a uniformly random share $[\mathsf{state}_{\mathsf{oram}}]_c$ for the client.

2. Let $I_1, \ldots, I_{|D^{(0)}|}$ be instructions of the form $(\mathsf{write}, v, \bar{0})$ for $1 \leq v \leq |D^{(0)}|$. S_{cl} sequentially applies $\mathsf{ORAMEval}$ to $I_1, \ldots, I_{|D^{(0)}|}$, along with the current ORAM state. After each instruction is submitted, the $\mathsf{ORAMEval}$ functionality returns an updated state, and the simulator generates a uniformly random share $[\mathsf{state}_{\mathsf{oram}}]'_c$ of the updated state for the client.

Computation. For each RAM f to be evaluated, S_{cl} will simulate its execution evaluating the same number of instructions of the form $(\mathsf{read}, 0, \perp)$ using $\mathsf{ORAMEval}$. Denote by $|f|$ the execution length of RAM f. Then, for each functionality f:

1. S_{cl} starts with a previously generated share $[\mathsf{state}_{\mathsf{oram}}]_c$ of the ORAM state that was generated during the pre-processing, or during the last computation.
2. Let $I_1, \ldots, I_{|f|}$ be instructions of the form $(\mathsf{read}, 0, \perp)$. As in the pre-processing phase, S_{cl} sequentially runs $\mathsf{ORAMEval}$ on $I_1, \ldots, I_{|f|}$, along with the current ORAM state. After each instruction is evaluated, $\mathsf{ORAMEval}$ returns an updated state, and S_{cl} generates a new uniformly random state share $[\mathsf{state}_{\mathsf{oram}}]'_c$.
3. The output reconstruction is simulated by opening to y_i the secret sharing of the output.

The server simulator S_{serv} proceeds similarly to S_{cl}. The notable difference is that the generated view additionally contains the instructions issued by $\mathsf{ORAMEval}$. Specifically, during pre-processing, $\mathsf{ORAMEval}$ is used to evaluate instructions of the form $I_j = (\mathsf{write}, v, \bar{0})$ for $1 \leq v \leq |D^{(i-1)}|$. For each such instruction, $\mathsf{ORAMEval}$ generates a sequence of subqueries $\hat{\mathbf{I}}_j$, which are included in the generated view. Similarly, during the computation of each functionality f, each instruction is converted by $\mathsf{ORAMEval}$ into a sequence of subqueries. These subqueries are included in the generated view (in addition to the state shares).

It is not hard to see that these simulators produce views indistinguishable from real execution. The reduction to the (non-adaptive) security of ORAM is straightforward, given our prior observation that the simulators produce their output only after the entire sequence of x_i is specified by Z (and hence the adaptively chosen sequence of x_i can be fed non-adaptively into the ORAM security experiment).

This leads to the following.

Theorem 2. *Let $\mathcal{ORAM}$ be access-pattern hiding, as defined by Definition 1, and let the underlying employed MPC be secure according to standard definitions. Then, our generic construction (π_f) described above is a secure protocol (according to Definition 2) for computing f, in the presence of honest-but-curious adversaries. Furthermore, if f can be computed in time t and space s by a RAM machine, then π_f runs in amortized time $t \cdot \mathsf{polylog}(s)$, the client uses space $\log(s)$, and the server uses space $s \cdot \mathsf{polylog}(s)$.*

3.5 An Optimized Protocol

In Section 3.4 we showed that any Oblivious RAM protocol can be combined with any secure two-party computation scheme to obtain a secure computation scheme with sublinear amortized complexity. This generic solution may be appropriate in many situations, however, current instantiations of the ORAM primitive require us to evaluate complex functions, such as pseudorandom function (PRF), using a secure two-party computation protocol. For example, the ORAM construction of Goldreich and Ostrovsky [Goldreich and Ostrovsky, 1996] requires many encryptions, decryptions and executions of a PRF. In spite of the fact that recent advances in the latter provide very efficient solutions for secure joint evaluation of PRFs [Freedman *et al.*, 2005], such secure evaluation is orders of magnitude slower than simply evaluating a PRF locally.

In this section we present a far more efficient secure computation system with sublinear amortized input access. Specifically, we construct, and prove secure, a new secure computation scheme that borrows ideas from ORAM protocols (specifically [Goldreich and Ostrovsky, 1996]) and MPC protocols, in order to provide extreme efficiency. Our resulting protocol uses only a handful of garbled circuits that contain nothing more than a few multiplications, if statements, and XOR operations, and in particular does not require evaluation of PRFs inside MPC. All other computation is done locally by the parties.

3.5.1 Technical Overview

Our starting point is the construction of Goldreich and Ostrovsky [Goldreich and Ostrovsky, 1996], which we use to store the server data. We begin with an overview of their ORAM protocol.

An overview of the Goldreich-Ostrovsky (GO) construction. In the GO construction, every pair (v_i, d_i), where d_i is a data item stored at index v_i in the original RAM protocol, is encrypted under a private key held by the client. (Recall that their protocol – and the ORAM model – does not offer privacy from the client.) Each of the N pairs are stored together in a data structure that has the following properties:

- It consists of $L = \log N$ levels for data of size N, though it will grow to size $\log t$ if there are t read and write operations. Level i contains 2^i "buckets", each of which can hold up to m data elements, where $m = \max(i, \log \kappa)$ and κ is the security parameter. The extra allocation in each bucket will be filled with "dummy" items, which we will explain below. All items stored in these buckets are encrypted with a key that is held by the client.

- Each level $i > 0$ has a hash function associated with it, h^i, which is chosen by the client. If an element (v, d) is stored at level i, it will be stored in the bucket having index $h^i(v)$.

The execution of a read operation and the execution of a write operation have identical structure, in order to prevent the server from distinguishing one from the other. The client begins by scanning the entire bucket at level $i = 0$, looking for the element of interest. Specifically, he decrypts each item in the bucket, one at a time, comparing it to the target value v. He then scans exactly one bucket at each level $i > 0$: if v was not yet found, the client scans the bucket with index $h^i(v)$, and if v was already found, he simply scans a random bucket. Finally after scanning a bucket at each level, (v, d) is written to the top level (regardless of whether this is a read or a write operation).

As mentioned above, whenever an item is being read or written, it is placed in the top level of the data structure. After m operations, this level will fill up. These items are then moved down a level, and shuffled in with the items below. Similarly, after 2^i read and write

operations, the items at level i are moved down and shuffled in with the items at level $i+1$. It follows that after every 2^i read and write operations, level $i+1$ becomes half full, and after another 2^i operations, it becomes full and is immediately emptied. Every time level i is moved down to level $i+1$, a fresh hash function is chosen for level $i+1$, and the items are re-inserted in that level using the new function. This process of merging two levels is quite complex, and we describe it in the course of describing our own protocol below. We refer the interested reader to [Goldreich and Ostrovsky, 1996] for a proof that this is a secure ORAM construction.

Extending the protocol to efficient secure computation: In our setting, we are further restricted in that the content of the RAM and the access pattern must remain unseen by the client as well as the server (with the exception of the output that is revealed at the very end of the computation). We have to overcome several challenges:

1) In the protocol of Goldreich and Ostrovsky [Goldreich and Ostrovsky, 1996], while the client is reading or writing item (v, d), he has to compute $h^i(v)$ up to d times. In practice, these hash functions would be implemented by a pseudo-random function (PRF), since we require h^i to be i-wise independent. In our setting, since the client should not learn the value of v, the naive way of implementing their protocol is to compute the PRF inside a garbled circuit. The resulting protocol would be extremely inefficient. Instead, we introduce a new primitive that we call a *shared-oblivious-PRF* (soPRF). This is a PRF in which the input and secret key are each shared between two parties, and the (single) recipient of the pseudo-random output can be designated at the start of the execution.[3] Our construction builds upon the oblivious PRF described by Freedman et al. [Freedman *et al.*, 2005]. Dodis et al. [Dodis *et al.*, 2006] gave two constructions of the same primitive. Their first construction, like ours, is based on the DDH assumption. However, the round complexity of their protocol is linear in the input size, while ours is constant round. Their second protocol is constant round, but relies on the stronger, q-decisional Diffie-Hellman inversion assumption.

2) While scanning each bucket to look for v, the client has to decrypt every ciphertext to see whether he has found a match. He also has to re-encrypt the value after reading

[3]So far we have only hinted at why the input to the soPRF needs to be shared, and we have said nothing about why the secret key would need to be shared. We explain this when we give the details of the protocol.

it. We need to use an encryption scheme that can be efficiently computed inside a secure computation. We use $\mathsf{Enc}(\mathsf{m};\mathsf{r}) = (\mathsf{F}_\mathsf{K}(\mathsf{r}) \oplus \mathsf{m}, \mathsf{r})$, and to ensure that encryption and decryption can be efficiently computed inside a garbled circuit, we have the client compute $F_K(r)$ outside the secure computation. All that has to be done inside the garbled circuit is boolean XOR. However, this requires care, since the value r may reveal something about the access pattern to the client. For example, we perform several oblivious sorts on the data during the shuffle protocol. In doing this, we repeatedly decrypt two items, decide whether to swap them, and then re-encrypt them. Suppose the result of one of these operations is $(\mathsf{Enc}(\mathsf{m}_0;\mathsf{r}_0), \mathsf{Enc}(\mathsf{m}_1;\mathsf{r}_1))$ if they are not swapped, and $(\mathsf{Enc}(\mathsf{m}_1;\mathsf{r}_1), \mathsf{Enc}(\mathsf{m}_0;\mathsf{r}_0))$ if they are. Since we allow the client to choose the randomness used in re-encryption, he can easily determine whether the values were swapped if (when) he sees these ciphertexts again at a later time! The solution is to make certain that the position of the randomness is independent of the outcome of the swap: we use $(\mathsf{Enc}(\mathsf{m}_0;\mathsf{r}_0), \mathsf{Enc}(\mathsf{m}_1;\mathsf{r}_1))$ if they are not swapped, and $(\mathsf{Enc}(\mathsf{m}_1;\mathsf{r}_0), \mathsf{Enc}(\mathsf{m}_0;\mathsf{r}_1))$ if they are.

3) In the original protocol, the client randomly reassigns elements to buckets whenever two levels are shuffled together (i.e., by choosing a new hash function, and re-hashing all the values). This is a crucial step for providing privacy, but in our setting we cannot entrust this task to the client, since we must protect their locations from him as well. Actually, this issue is subtly tied to the fact that the client knows the encryption randomness. Because he is given the randomness used for decryption during a lookup, he can easily determine the bucket index of the lookup as well. This is not in itself a problem: recall that the server also learns the bucket index during lookup, even in the original ORAM protocol. However, it requires us to hide bucket assignments from the client during the shuffling. If we did not reveal the bucket index during lookup, we might have hoped to reveal more to the client during the shuffle. Instead, we use a shared-oblivious-PRF during our shuffle protocol in order to hide the bucket assignments from *both* parties.

3.5.2 Building Blocks

We start with a description of some primitives that we will use as building blocks in our construction.

3.5.2.1 Shared-Oblivious PRF

As we mentioned above, our protocol makes use of a new primitive that we call a *shared-oblivious-PRF* (soPRF). Our soPRF construction is based on the oblivious PRF of Freedman et al. [Freedman *et al.*, 2005] (see Figure 3.4). Our particular construction is a function

$$\mathsf{soPRF} : \mathbb{Z}_p^{O(logN)} \times \mathbb{Z}_p^{O(\log N)} \times \{0,1\}^{\log N} \times \{0,1\}^{\log N} \to \mathbb{G} \times \mathbb{G},$$

where $\mathbb{G}$ is a group of prime order $p = O(2^\kappa)$ for which the DDH assumption is expected to hold. The output are secret shares α and β such that α^β is pseudorandom. Next we provide the construction and the security proof for out soPRF.

Definition 3. *Let F be some pseudorandom function (PRF) with key-space $\mathcal{K}$ and input domain X. Let $[k]$ denote a 2-out-of-2 secret of $k \in \mathcal{K}$, and let $[x]$ be the same for input $x \in X$. We say that the function* soPRF *is a* shared oblivious pseudorandom function (soPRF) built on F *if for any $k \in \mathcal{K}$, and any $x \in X$,* $\mathsf{soPRF}([x],[k])$ *outputs a secret sharing of $F(k;x)$.*

We note that we can trivially build an soPRF from any PRF by using Yao's protocol. However, the goal is to give more efficient construction. Our construction of an soPRF is built on the Naor-Reingold PRF [Naor and Reingold, 2004]. We review this PRF here. Let $\mathbb{G}$ be some prime order group for which the DDH assumption is expected to hold, and let g be a generator for $\mathbb{G}$. The Naor-Reingold PRF has input domain $X = \{0,1\}^K$, key-space $\mathcal{K} = \mathbb{G}^\kappa$, and output space $\mathbb{G}$. The function is defined as $F(k;x) = g^{r_0 \Pi_i x_i r_i}$, where $r_i \in \mathbb{G}$ makeup the key, and $x_i \in \{0,1\}$ the input.

Theorem 3. *Assuming DDH is hard in $\mathbb{G}$, and secure OT exists, the protocol described in Figure 3.4 computes an soPRF, and is secure against semi-honest, polynomial time adversaries.*

Proof. Let $\mathbf{I} = \{i_1, \ldots, i_{|\mathcal{I}|}\}$ denote the set of indices such that $\mathsf{v}_c[i_j] \oplus \mathsf{v}_s[i_j] = 1$. We prove our theorem in the standard way, comparing a real execution of π (in the OT-hybrid world) with an ideal function that outputs $f_{\overline{r}_c,\overline{r}_s}(v_c, v_s) = g^{(r_c^0 \cdot r_s^0) \cdot \Pi_{i \in \mathbf{I}} (r_c^i \cdot r_s^i)}$. We define our ideal world functionality as the following randomized functionality:

Shared Oblivious PRF

Let g be a generator of a group G of prime order p for which the DDH assumption holds. Let $\overline{r}_s, \overline{r}_c$ be the shares of the PRF key.

Inputs: Server: $\mathsf{v}_s \in \{0,1\}^m, \overline{r}_s = (r_s^0, r_s^1, \ldots, r_s^m)$, where each $r_s^i \in \mathbb{Z}_p^*$

Client: $\mathsf{v}_c \in \{0,1\}^m, \overline{r}_c = (r_c^0, r_c^1, \ldots, r_c^m)$, where each $r_c^i \in \mathbb{Z}_p^*$

Protocol:

1. The server samples m values $a_1, \ldots, a_m$ in Z_p^* uniformly at random.
2. The client samples m values $b_1, \ldots, b_m$ in Z_p^* uniformly at random.
3. For each $1 \leq i \leq m$:
   1. The client and the server run an oblivious transfer protocol, with the server as sender and the client as receiver, using the following inputs:
      - If $\mathsf{v}_s[i] = 0$, the server's input is $(a_i, a_i \cdot r_s^i)$. Otherwise the server's input is $(a_i \cdot r_s^i, a_i)$.
      - The client's input is $\mathsf{v}_c[i]$.

      Let x_i be the output value that the client receives from the OT execution.
   2. The client and the server run an oblivious transfer protocol, with the client as sender and the server as receiver, using the following inputs:
      - If $\mathsf{v}_c[i] = 0$, the client's input is $(b_i \cdot x_i,\ b_i \cdot x_i \cdot r_c^i)$. Otherwise the client's input is $(b_i \cdot x_i \cdot r_c^i,\ b_i \cdot x_i)$.
      - The server's input is $\mathsf{v}_s[i]$.

      Let y_i be the output value that the server receives from the OT execution.
3. Let $\mathcal{I} \subseteq [m]$ denote the set of indices such that $\mathsf{v}_c[i_j] \oplus \mathsf{v}_s[i_j] = 1$.

   The server computes

$$\alpha = r_s^0 \left(\prod_{i=1}^{m} \frac{y_i}{a_i} \right) = r_s^0 \left(\prod_{i=1}^{m} b_i \right) \left(\prod_{i \in \mathcal{I}} r_c^i r_s^i \right).$$

   The client computes $\beta = g^{\frac{r_c^0}{\prod_{i=1}^m b_i}}$. These are secret shares of the pseudorandom value

$$\beta^\alpha = \left(g^{\frac{r_c^0}{\prod_{i=1}^m b_i}} \right)^{r_s^0 (\prod_{i=1}^m b_i)(\prod_{i \in \mathcal{I}} r_c^i r_s^i)} = g^{r_c^0 r_s^0 \prod_{i \in \mathcal{I}} r_c^i r_s^i}.$$

Outputs: The server outputs α, and the client outputs β.

Figure 3.4: A construction of a Shared Oblivious PRF

Ideal functionality F computing an soPRF:

Server input: $(r_s^0, r_s^1, \ldots, r_s^m, \mathsf{v}_s)$.

Client input: $(r_c^0, r_c^1, \ldots, r_c^m, \mathsf{v}_c)$

The functionality F chooses $b_1, \ldots, b_m$ uniformly and independently at random from $\mathbb{Z}_p^*$.

Server output: $\alpha = r_s^0(\prod_{i=1}^m b_i)(\prod_{i\in\mathcal{I}} r_c^i r_s^i)$

Client output: $\beta = g^{\frac{r_c^0}{\Pi_{i=1}^m b_i}}$ and $\Pi_{i=1}^m b_i$.

Note that in the ideal functionality, we include Πb_i in the output of the server. Although this is not obviously necessary, it turns out that it is important for the simulation; in the real world protocol, the server chooses the b_i values himself, so leaking this information is not "harmful". However, we note that this has some implications about the pseudo-randomness of the output if a player is given both shares. We will return to discuss this further below.

We begin our security proof by assuming that an adversary $\mathcal{A}$ controls the client in the protocol. We construct a simulator $\mathcal{S}$ which interacts with the adversary and simulates the execution of the protocol in the semi-honest setting.

Lemma 1. *Let F denote an ideal execution of the soPRF as described above, and let π^{OT} denote an execution of the protocol in Figure 3.4 in the OT-hybrid world. For any semi-honest, polynomial time adversary $\mathcal{A}$ with auxiliary input $z \in \{0,1\}^*$ that corrupts the server in the OT-hybrid world, there exists a semi-honest, polynomial time adversary $\mathcal{S}$ with auxiliary input $z \in \{0,1\}^*$ corrupting the server in the ideal world such that*

$$\text{IDEAL}^{(i)}_{F,\mathcal{S}(z)}(\kappa, (r_s^0, r_s^1, \ldots, r_s^m, \mathsf{v}_s)) \stackrel{c}{=} \text{REAL}^{(i)}_{\pi^{\mathsf{OT}},\mathcal{A}(z)}(\kappa, (r_s^0, r_s^1, \ldots, r_s^m, \mathsf{v}_s)).$$

Proof. The simulator $\mathcal{S}$ acts as follows:

1. $\mathcal{S}$ submits the server's input to the ideal functionality and receives output α.

2. $\mathcal{S}$ receives $\mathcal{A}$'s input for the m ideal executions of OT in Step 1. From these inputs, he computes and stores the value $\Pi_{i=1}^m a_i$.

3. For $i \in \{1, \ldots, m-1\}$, $\mathcal{S}$ simulates the output of the ith execution of the ideal OT in Step 2 by choosing $c_i \in \mathbb{Z}_p^*$ uniformly at random, and sending it to the server. He

then simulates the mth ideal OT in Step 2 by computing and sending

$$c_m = \frac{\alpha \cdot \Pi_{i=1}^{m} a_i}{r_s^0 \cdot \Pi_{i=1}^{m-1} c_i}$$

The only messages the server receives in the OT-hybrid world are the outputs of the m OTs in Step 2. Therefore, the view of $\mathcal{A}$ in the hybrid world is $\{(y_1, \ldots, y_m), \alpha\}$, where y_i is the output received in the ith execution of OT. In the ideal world, these messages are replaced by $(c_1, \ldots, c_m)$, so the view of the adversary is instead $\{(c_1, \ldots, c_m), \alpha\}$. We must argue that these distributions are indistinguishable, when taken jointly with the output of the honest client:

$$\{(y_1, \ldots, y_m), \alpha, \beta, \Pi_{i=1}^{m} b_i\} \stackrel{c}{=} \{(c_1, \ldots, c_m), \alpha, \beta, \Pi_{i=1}^{m} b_i\}$$

where the first distribution is over the random coins of the two parties in the hybrid world, and the second distribution is over the coins of the ideal party and of $\mathcal{S}$ in the ideal world. The distributions on α, β and $\Pi_{i=1}^{m} b_i$ are clearly identical in both worlds, so we are really concerned only with the distributions on $(y_1, \ldots, y_m)$ and on $(c_1, \ldots, c_m)$ given α, β and $\Pi_{i=1}^{m} b_i$. Consider first the distributions on $(y_1, \ldots, y_{m-1})$ and $(c_1, \ldots, c_{m-1})$. Since the value of $\Pi_{i=1}^{m} b_i$ does not restrict the value of any $m-1$ size subset of the b_i values, and, by extension, neither does the value of α or β, it follows that both $(y_1, \ldots, y_{m-1})$ and $(c_1, \ldots, c_{m-1})$ are uniformly distributed over $\{\mathbb{Z}_p^*\}^{m-1}$ given α, β and $\Pi_{i=1}^{m} b_i$. In the hybrid world, recall that α is computed by the server as:

$$\alpha = \frac{r_s^0 \cdot \Pi_{i=1}^{m} y_i}{\Pi_{i=1}^{m} a_i}.$$

Therefore, given $(y_1, \ldots, y_{m-1})$ and α, the value of y_m is fully determined by

$$y_m = \frac{\alpha \cdot \Pi_{i=1}^{m} a_i}{r_s^0 \cdot \Pi_{i=1}^{m-1} y_i}.$$

Since the simulator chooses c_m in precisely this manner, we conclude that the distributions are identical.

□

Lemma 2. *Let F denote an ideal execution of the soPRF as described above, and let π^{OT} denote an execution of the protocol in Figure 3.4 in the OT-hybrid world. For any semi-honest, polynomial time adversary $\mathcal{A}$ with auxiliary input $z \in \{0,1\}^*$ that corrupts the*

client in the OT-hybrid world, there exists a semi-honest, polynomial time adversary $\mathcal{S}$ with auxiliary input $z \in \{0,1\}^*$ corrupting the client in the ideal world such that

$$\text{IDEAL}^{(i)}_{F,\mathcal{S}(z)}(\kappa, (r_c^0, r_c^1, \ldots, r_c^m, \mathsf{v}_c)) \stackrel{c}{=} \text{REAL}^{(i)}_{\pi^{\mathsf{OT}},\mathcal{A}(z)}(\kappa, (r_c^0, r_c^1, \ldots, r_c^m, \mathsf{v}_c)).$$

Proof. The only messages received by the client are the outputs from the first m executions of OT in Step 1, $(x_1, \ldots, x_m)$. The simulator simulates these m outputs with random, independently chosen group elements from $\mathbb{Z}_p^*$: $(d_1, \ldots, d_m)$. As before, we need to prove that the distributions

$$\{(y_1, \ldots, y_m), \alpha, \beta, \Pi_{i=1}^m b_i\} \stackrel{c}{=} \{(d_1, \ldots, d_m), \alpha, \beta, \Pi_{i=1}^m b_i\}$$

Here the proof is immediate, since the values of a_i are never known to the distinguisher. It follows that the x_i are each independent, random values in $\mathbb{Z}_p^*$, even when the output of each party is given. Therefore the simulated distribution, $(d_1, \ldots, d_m)$, and the hybrid world distribution are identically distributed.

□

We now argue that our construction has an additional property: given the output $\gamma = f_{\overline{r}_c, \overline{r}_s}(v_c, v_s)$, and the server's share of the secret key, $\overline{r}_s = (r_s^0, r_s^1, \ldots, r_c^m)$, we can generate shares α and β from the appropriate (random) distribution such that $\beta^\alpha = \gamma$. The implication is that the client can safely send β to the server, who holds α, while still ensuring that β^α looks random. We note that this property does not have to hold, because our definition of an soPRF allows α and β to contain information about the shares of the secret key. To make this issue more explicit, consider an soPRF that includes player i's share of the secret key (entirely) within i's output share. The output could still be pseudorandom given one of the two shares, but it is certainly not pseudorandom when one player holds both shares. We remark that we did not require this property in our definition because an soPRF may be useful even without it. For example, in our shuffle protocol, neither player ever obtains both output shares. Instead, the shares are used directly as input to a secure computation, which yields encrypted output.

We argue that our protocol remains secure if $\beta = g^{r_c^0 / \prod_{i=1}^m b_i}$ is sent to the player holding $\alpha = r_s^0 (\prod_{i=1}^m b_i)(\prod_{i \in \mathcal{I}} r_c^i r_s^i)$, but we note that this is *not* true if the share α is sent to the

player holding $(\beta, \Pi_{i=1}^{m} b_i)$. To see that it is secure when β is sent to the player holding α, note that given a pseudo-random value γ and the secret key $(r_s^0, \ldots, r_s^i)$, we can easily simulate shares α and β such that $\beta^\alpha = \gamma$, even without knowing $r_c^0, \ldots, r_c^m$. The simulator simply chooses a random α, and then computes $\beta = \gamma^{-\alpha}$. Since Πb_i is not yet fixed, the simulated shares are always consistent with the correct values of r_c^i for *some* choice of Πb_i. However, note that this simulation fails when Πb_i is already known and fixed. More specifically, without knowing $r_s^0, \ldots, r_s^m$, given $(\gamma, \Pi b_i, r_c^0, \ldots r_c^m)$, β is already well defined, and finding α that is consistent with β and γ requires solving an instance of the discrete log problem. Due to this discussion, in our protocol we will always assign β to the client, since he will occasionally send his share of the soPRF output to the server. We leave it as an open problem to find an efficient soPRF that allows either party to send their secret share to the other.

3.5.2.2 A Secret Re-Sharing Scheme

For our purposes, this is insufficient, because subsequent use of these shares in a Yao garbled circuit will be quite impractical; even a single exponentiation is more costly than computing AES. We therefore provide a second, efficient protocol that takes shares of this form, and outputs new *multiplicative* shares: $\alpha' \cdot \beta' = \alpha^\beta$. This appears in Figure 3.5. When we use the soPRF in our protocol, we leave the re-sharing protocol implicit. In our scheme, the input to the soPRF are secret shares of a virtual address, and the output are shares of a pseudorandom value, which is used for determining the location of the virtual address.

We describe a secret re-sharing protocol between two parties C and S. The protocol implements (under the DDH assumption) an ideal functionality which, given group elements α, β that are the inputs of S, C respectively, outputs uniformly distributed group elements u_S, u_C such that $u_c u_s = \alpha^\beta$.

Recall that we use this protocol to reshare an soPRF value that was shared through exponentiation, obtaining multiplicative shares instead. However, we note that the protocol works for any α, β, and does not rely on the pseudorandomness of α^β.

The protocol. The protocol starts with S choosing random α_1, α_2 such that their product is α, and C choosing random β_1, β_2 such that their sum is β. S sends α_1 to C, who can now compute his output, $\alpha_1^{\beta_1}$. The rest of the protocol is designed to allow S to compute his output, $\alpha_2^{\beta}\alpha_1^{\beta_2}$, without revealing any extra information. This is done by using blinding (raising to a random power, or multiplying by a random number), and by El-Gamal encryption (and its multiplicative homomorphic properties). The details are described in Figure 3.5.

Multiplicative Re-Sharing

Let G be a group of prime order p for which the DDH assumption holds.

Inputs: Server (S): α; Client (C): β.

Outputs: Server: u_s; Client: u_c; where u_s, u_c are uniformly distributed subject to $u_c u_s = \alpha^{\beta}$.

Protocol:

1. The server chooses random $\alpha_1, \alpha_2 \leftarrow G$ such that $\alpha_1 \cdot \alpha_2 = \alpha$. The server also chooses random $r \leftarrow \mathbb{Z}_p$. The server sends α_1, α_2^r to the client.
2. The client chooses random $\beta_1, \beta_2, x \leftarrow \mathrm{Z}_p$ such that $\beta_1 + \beta_2 = \beta$. The client also chooses random $g \leftarrow G$ and sets $h = g^x$. We let $[\![W]\!]$ denote an El Gamal encryption of W with respect to the public key (g, h). The client sends g, h, $[\![\alpha_2^{r\beta}]\!]$, and $[\![\alpha_1^{\beta_2}]\!]$ to the server.
3. The server chooses random $r_2 \leftarrow \mathrm{Z}_p$. Using the homomorphic properties of El Gamal encryption, the server computes $[\![\alpha_2^{\beta}\alpha_1^{\beta_2} g^{r_2}]\!]$ and sends it to the client.
4. The client decrypts the ciphertext sent in the last round, and sends the result $\alpha_2^{\beta}\alpha_1^{\beta_2} g^{r_2}$ back to the server. The client outputs $u_c = \alpha_1^{\beta_1}$.
5. The server divides by g^{r_2} and outputs $u_s = \alpha_2^{\beta}\alpha_1^{\beta_2}$.

Figure 3.5: A protocol for converting exponentiation-based shares to multiplicative shares

Analysis. Correctness follows straightforwardly: $u_c u_s = \alpha_1^{\beta_1}\alpha_2^{\beta}\alpha_1^{\beta_2} = \alpha_1^{\beta_1+\beta 2}\alpha_2^{\beta} = \alpha_1^{\beta}\alpha_2^{\beta} = \alpha^{\beta}$.

Security for one side is information theoretic, and for the other side is based on DDH assumption. A simulator that gets only the input and output of one of the parties, can

simulate the entire view by using random values for the incoming messages, subject to the correct output being computed. Specifically, a simulator getting only the input β and output u_c of the client can simulate the entire view by choosing a random β_1, β_2 that sum up to β, then choosing $\alpha_1 = u_c^{\beta_1^{-1}}$ when simulating the first message. All the other values on incoming messages are chosen randomly (and all values sent out are computed honestly). It is easy to see that this simulated view is statistically close to the real view (as all messages sent to the client in the protocol are blinded by a random number, and since the output u_c is random). A simulator getting only the input α and output u_s of the server can simulate the entire view by using random values for the incoming message in step 2 (while computing step 1 and 3 honestly), and then sending $u_s g^{r_2}$ in step 4 (for the corresponding values g, r_2 from steps $2, 3$). Using the semantic security of the El Gamal encryption (based on the DDH assumption), it is easy to see that this view is indistinguishable from the view in the real protocol.

□

3.5.2.3 Encryption Scheme with Small Encryption and Decryption Circuits

In the protocol that follows, all elements, (v, d), are encrypted using semantically secure, symmetric key encryption; the key K for a (standard) PRF is stored by the client and never changes. We will frequently perform secure computations that involve decrypting a ciphertext, performing an operation, and the re-encrypting the resulting value. As described above, before performing any such computation, the server first sends the random values r and r' to the client, where r is the randomness currently being used in the relevant ciphertext, and r' is chosen randomly for the re-encryption. The client computes $F_K(r)$ and $F_K(r')$ locally, and uses both values as input to the secure computation. Then, inside the garbled circuit, both decryption and re-encryption can be achieved with simple XOR operations. The server stores r', and sends it again to the client the next time the same value is needed in a secure computation. (This saves the client from having to store the randomness.) Below, when we describe the protocol, we leave this step implicit.

3.5.3 Our Construction

Our construction follows along the lines of the general protocol we described in Section 3.4. The two difficulties are to build an efficient implementation of the doInstruction function, and to make the shuffle protocol from [Goldreich and Ostrovsky, 1996] secure against the client while maintaining it's efficiency. We focus on those two tasks here, and do not repeat the remainder of the protocol. We find the prose that follows easier to read than would be precise pseudo-code. Although we reference the functionalities that will be implemented using Yao, and we give precise pseudo-code for each of those functionalities in Appendix A.1, we expect the reader will find those descriptions helpful mainly to verify the simplicity of the garbled circuits that we rely upon. We note that in the proof of security (Section 3.6), the interested reader will find a more precise listing of the messages sent to and from each party.

Preprocessing: The players insert the server's data into the ORAM using a sequence of write instructions. We describe the process for a write instruction next.

Read/Write: We assume the players each hold a secret share of the instruction being performed, which includes the virtual address, $v \in [N]$, being sought.

1. **The players scan the top layer looking for data item d stored at address v.** They do this by repeatedly performing a secure computation in which they decrypt an element, compare its virtual address to v, and then re-encrypt it before storing it back. However, the computation should not reveal to either player whether or not v was found, nor what the value of d is. Therefore, the output of the secure computation includes a secret sharing of a state variable that indicates whether v was found, along with shares of d in case v is found. Both players remain unaware of the values of these variables. They scan the entire level, even if v is found mid-way through the scan. The secure computation for this step is found in Figure A.1.

2. **The players scan exactly one bucket at each level.** The index of the bucket scanned at level i is chosen as follows:

   (a) First, the players engage in a secure computation in which the output is:

- a secret sharing of v if v was not yet found, and
- a secret sharing of the string "$\mathsf{dummy} \circ t$" if v was already found.

Here t is a counter stored by the server, and incremented after every read or write operation. The secure computation for this step is described in Figure A.2.

(b) They then compute the soPRF on their shares of v (or on their shares of the dummy address). The key for the soPRF will have been created and shared when elements were last inserted into this level. This is described in the shuffle protocol below. The client sends his share of the soPRF output to the server.

(c) The server maps the output to the integers using the universal hash function h^i associated with level i. He then fetches the corresponding bucket from memory. The players scan this bucket searching for (v, d), precisely the way they scanned the top layer. They do this even if v was already found. If (v, d) is found in this bucket, they store the value in their state (again, unaware that they have done so), and instead of re-encrypting (v, d) and storing it back in the same location, they replace it with an encryption of an empty item. We use the same secure computation as above, described in Figure A.1.

3. **The players write the element back to the top layer.** They do this by scanning the layer as before, using a sequence of secure computations to decrypt, compare, and re-encrypt. If they come across the previous version of (v, d), they overwrite it with the (possibly) newer value. (This will happen when (v, d) was first found in the top layer.) If they come across an empty spot in the layer, they simply store the newly encrypted value there. Either way, they continue the scan until they have re-encrypted the entire level. The secure computation for this step is described in Figure A.3.

Shuffling. As in the work of Goldreich and Ostrovsky [Goldreich and Ostrovsky, 1996], we must occasionally merge a level with the one below, shuffle together the items, and reinsert them into the data structure. As in their protocol, we merge level i with level $i + 1$ after 2^i read or write instructions. We let $n = 2^i$ denote the maximum number of elements in level i. Recall that there are also n buckets at level i, each of size m. We note that there are at most n elements in level $i + 1$ at the time of the shuffle; the capacity is $2n$, but as soon

as it fills, they are all moved down to level $i+2$. Although there are at most $2n$ items in these two levels, there is enough space allocated for $3nm$ words. The remaining spaces are filled with encryptions of empty elements, of the form ("empty, empty"), which help hide how many real elements are currently contained in the level. The goal in the shuffle is to exactly fill every bucket, while ensuring that neither player learns anything about how the real items are distributed.

1. **The players choose new keys and setup a buffer.** They each choose a shared key for the soPRF. Each player will store their share of the key until the next time these two levels are merged; the same key will be used while reading and writing elements (as described above). The server also chooses and stores a universal hash function h^{i+1}, which will be used to map the output of the soPRF to a bucket index in $\{1, \ldots, 2n\}$. He stores this along with his share of the soPRF key. Finally, the server creates a buffer big enough to hold $3nm$ data elements. He places all nm elements from level i and all $2nm$ elements from level $i+1$ in this buffer.

2. **The players assign the real items to buckets.** There are $2n$ (or fewer) items to be put in the $2n$ empty buckets of level $i+1$, and each bucket is of size m. (The remaining $2nm - 2n$ empty spaces of the will later be filled with (encrypted) empty items.)

   (a) They begin this process by (obliviously) sorting the elements, giving priority to real items. This is done by jointly implementing an oblivious sort over the virtual addresses. For each comparison of the sort, the players compute a secure computation that recovers the value of the addresses, compares them, chooses whether to swap them, and finally re-encrypts both the address and the data element. The secure computation to be performed is described in Figure A.5.

   (b) The players then do the following for each of the first $2n$ elements in the buffer:

   - They perform a secure computation of the functionality GetHashInput (Figure A.2), which outputs a secret sharing of v. For empty items (if there are any), the secure computation simply outputs a random string $r \in \{0,1\}^\kappa$.
   - They compute the soPRF on this value.

- They perform another secure computation in which both players use the shares they received from the soPRF as input, and the server uses as additional input the description of the universal hash function h^{i+1}. Inside the secure computation, the output shares of the soPRF are reconstructed and then mapped to a bucket index using the hash function h^{i+1}. The bucket index is encrypted and output to the server, who stores it with the element (still kept in the buffer). In Figure A.4 we describe the hash function of Mansour et al. [Mansour *et al.*, 1993], which is very simple to compute inside a Yao circuit.

3. **The players assign the empty items to buckets.**

   (a) They scan the last $2nm$ elements in the buffer, which are all guaranteed to be empty (since $2n < mn$). The client encrypts a bucket index for each one: the first m items are mapped to bucket 1, the next m to bucket 2, and so on until exactly m of these empty elements have been assigned to each of the $2n$ buckets.

   (b) They perform another oblivious sort (again using a sorting network), this time sorting by bucket index, with priority given to real items. They then scan the items, using a secure computation to decrypt, increase the count for the current bucket and re-encrypt. If the counter has exceeded m elements for the current bucket, then the element's index is replaced with the symbol $\perp$ before re-encryption. The count is kept private (again by use of encryption). We note that the probability of removing a real element here is negligible, since the probability that more than m real items fall into one bucket is negligible. (See Lemma 5.) In case this does occur, we let the output of this secure computation be a special abort symbol, and the players abort the protocol. The necessary secure computation is described by the functionality in Figure A.6.

   (c) Finally, they perform one more oblivious sort on the bucket index, treating $\perp$ as the largest index. In the end, the buffer contains m items per bucket, ordered by bucket index. The server simply copies these directly back into level $i + 1$ in their current order, ignoring the leftover items labeled with $\perp$. They again use

a secure computation for the functionality described in Figure A.5.

3.5.4 Discussion: Bucket Size

As we will see below, the size of each bucket will play an important role in the proof of security. In particular, we claimed above in Step 3b that if we map n items to n buckets, the probability of overflowing a bucket of size m is negligible in the security parameter. Suppose this were not the case: that we instead use a smaller bucket size, and that we simply sample a new hash function if our elements overflow a bucket during insertion. This admits the following attack: consider a server that is trying to distinguish between two different search sequences by the client, $X = (x_1, \ldots, x_n)$ and $Y = (y_1, \ldots, y_n)$. Assume further that he knows all elements in search pattern X are found in level i, while all elements in Y are found at (say) level $i + 1$. We note that security must hold even in such a situation. With non-negligible probability, while the client is searching for the elements of Y, he will query the same bucket at level i at least $m+1$ times. On the other hand, if his search was for items in list X, this could not happen, because, by our assumption, the hash function assigned to level i was chosen to map no more than m items to any one bucket. Put another way, we can ensure that the hash function for level i never overflows when mapping elements in level i, but we cannot ensure that it doesn't overflow when mapping elements that are *not* in level i. Therefore, we choose bucket sizes that are large enough to guarantee a negligible probability of overflow for any 2^i elements.

This issue affects security of previous ORAM constructions as well, including [Pinkas and Reinman, 2010], and [Goldreich and Ostrovsky, 1996] when the buckets are small ([Goldreich and Ostrovsky, 1996] say that any bucket size will do). Indeed, [Pinkas and Reinman, 2010] suffers from this issue (and a related one that stems from its use of cuckoo hashing) in a way that we were not able to repair (unless we increase their $\log^2$ overhead to $\log^3$).[4] However, for the [Goldreich and Ostrovsky, 1996] protocol, security *is* maintained for the parameters they suggest as most practical (logarithmic bucket size).

[4] As mentioned in Section 2.0.7, these insecurities were independently discovered by others, and potential fixes as well as alternative schemes have since been suggested.

3.6 Security Proof

The security of the protocol that we presented in the previous section is captured by the following theorem (the performance is analyzed in detail in Section 3.7).

Theorem 4. *Assuming that the Decisional Diffie-Hellman (DDH) assumption holds, the protocol described in Section 3.5.3 (Π_F) is a secure protocol for computing F in the presence of honest-but-curious adversaries. Furthermore, if F can be computed in time t and space s by a RAM machine, then Π_F runs in amortized time $t \cdot \mathsf{polylog}(s)$, the client uses space $\log(s)$, and the server uses space $s \cdot \mathsf{polylog}(s)$.*

Proof. To prove security of the protocol against honest-but-curious players, we analyze our protocol in the *hybrid model*, in which we replace all secure computations used during the read and write operations with ideal executions of their corresponding functionalities. It follows from a well known result of Canetti [Canetti, 2000a] that if the resulting protocol is secure in this hybrid world, then the protocol remains secure in the real world as well.

We start with the following lemma which will be an important part of our proof.

Lemma 3. *In an honest-but-curious execution of protocol Π_F (as described in Section 3.5), for all soPRF keys k and for all inputs v, the probability that the players compute $\mathsf{soPRF}(k, v)$ in Step 2b more than one time is less than $\mathsf{negl}(\kappa)$, where negl is some negligible function.*

Proof. We consider two types of inputs: $v \in [N]$, and dummy inputs of the form $\mathsf{dummy} \circ t$. For inputs of the latter form, note that this particular input can only be used in the tth operation, since t is incremented with every operation. The only way $\mathsf{soPRF}(k, \mathsf{dummy} \circ t)$ can be computed more than once, therefore, is if the same key k is assigned to two different levels at the same time. Since the number of levels is bound by some polynomial (in κ), and there are an exponential number of keys, this is negligibly likely to occur. Consider a pair (k, v) where v is of the first form. In this case, the proof follows from three properties of the protocol: a) elements are moved to the top layer once they are found. b) Whenever we have found an element at some level i, we query $\mathsf{dummy} \circ t$ at all levels $j > i$, where t is uniquely chosen in each operation. c) Whenever an item is moved down to a lower level, a new soPRF key is assigned to the lower level. If we assume that keys are chosen

without replacement (i.e. that we never choose the same key more than once), then the lemma clearly follows from these three properties. Since the total number of keys chosen is bounded by some polynomial in κ, the probability that the same key is chosen more than once is negligible. □

Lemma 4. *For every non-uniform, polynomial time adversary $\mathcal{A}$ corrupting the server in a hybrid-world execution of the secure computation described in Section 3.5.3, there exists a non-uniform, polynomial time adversary $\mathcal{S}$ corrupting the server in the ideal world execution F, and a negligible function $neg(\cdot)$, such that for all $\kappa \in \mathbb{N}$:*

$$|\Pr[\mathsf{Real}(1^\kappa, \Pi, \mathcal{A}) = 1] - \Pr[\mathsf{Ideal}(1^\kappa, F, \mathcal{S}) = 1]| \leq neg(\kappa)$$

Proof. We begin by describing the simulation of the shuffle protocol. We will then describe the simulation of a single read/write operation. In the end we will put these together to argue that sequences of read/write operations and shuffles remain secure.

In simulating the shuffle protocol, recall that $\mathcal{S}$ is simulating the hybrid world in which the players have access to ideal executions of the functionalities described in Figures A.5, A.2, 3.4, 3.5, A.4, and A.6. We outline the hybrid world protocol in Figure 3.6.

We note that in this hybrid world, almost the entire shuffle protocol proceeds through a sequence of ideal function calls; the players rarely interact outside of these ideal executions. With a few exceptions (described below), the simulator only needs to simulate the output of each of these ideal functionalities. Furthermore, the output from these ideal functionalities is always either a ciphertext, or a random secret sharing. The simulation is therefore quite straightforward: the output of each functionality is simulated with a random string. When the output is supposed to be a ciphertext, the indistinguishability of the simulation follows from the security of the PRF used in the encryption scheme. When the output is supposed to be a secret share, the simulation is distributed identically to the output of the hybrid world. As we will see, the messages that are sent from the client (i.e. that are not communicated through an ideal function call) are always either the random string used in some ciphertext, or a complete ciphertext. Both can again be simulated with random strings.

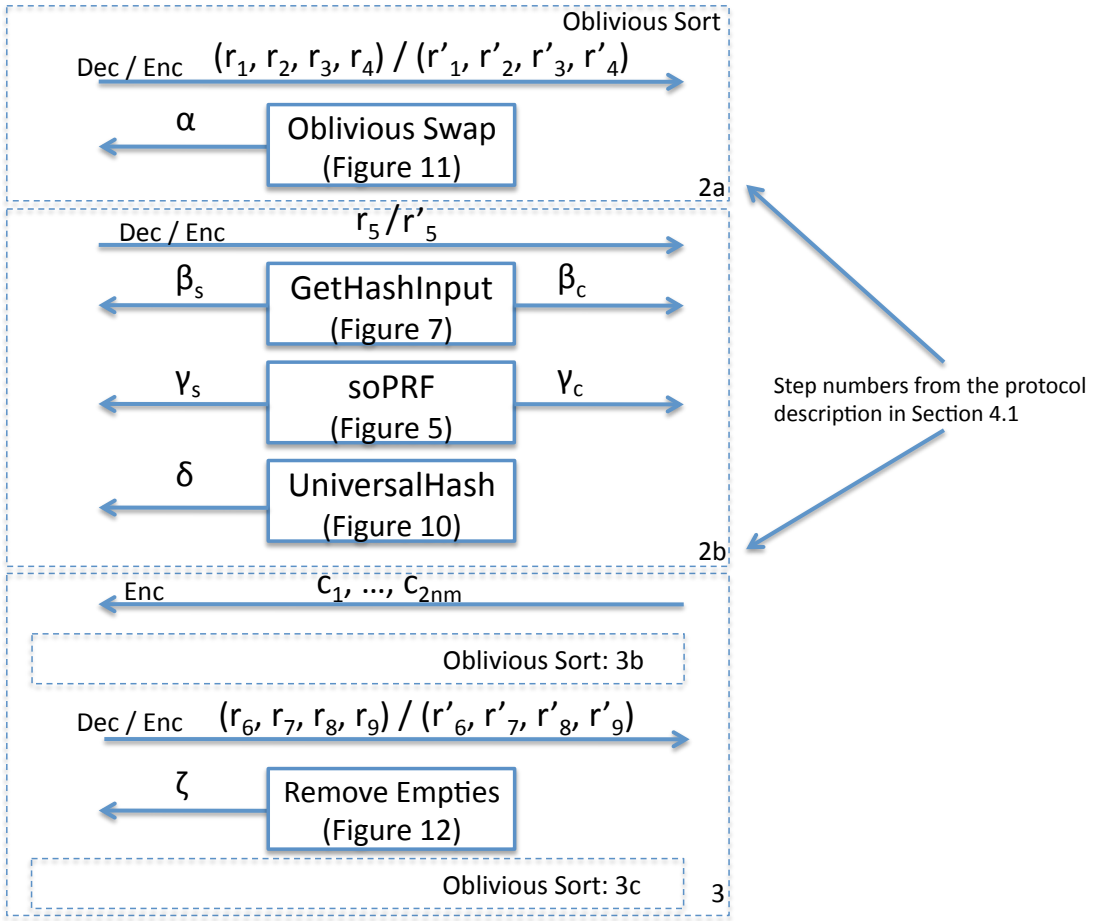


Figure 3.6: The shuffle protocol in the hybrid world.

Simulating oblivious sort: Since oblivious sort is performed three different times, we describe the simulation separately. We note that each oblivious sort in the hybrid world proceeds through a sequence of calls to the ideal instance of the swap functionality (Figure A.5), each preceded by the exchange of four encryption and decryption strings. (In Figure 3.6, we have only depicted a single call in order to save space.) As described above, we simulate (r'_1, r'_2, r'_3, r'_4), which are sent from the client before each ideal function call, with random strings. We let α denote the output of the Oblivious Swap functionality; α is a group of four ciphertexts, which are also simulated with random strings. The security of the PRF used in the encryption scheme guarantees that the simulated ciphertexts are indistinguishable from the actual output of the trusted swap functionality.

Step 2a only contains an oblivious sort. In Step 2b, the players make three calls to ideal functionalities (again, engaging in no other communication). To simulate β_s, the output of the ideal functionality of Figure A.2, and γ_s, the output of the ideal soPRF, the simulator simply outputs a random string. In the hybrid world, β_s and γ_s are random secret shares, so the simulated output is identically distributed to the hybrid world output. (We note that even if the input to the soPRF is identical in two sequential executions, the *output* is still a fresh pair of secret shares. Therefore, the simulator does not need to do any "book keeping" regarding previous inputs to the soPRF.) The random value r'_5 is used in constructing the ciphertext δ, and can be simulated with a random string. Finally, the last ideal call in this step is to an ideal functionality that takes the output of the soPRF, along with the description of a hash function, and outputs an encryption, δ of the bucket index that results from applying h^i to the output of the PRF (Figure A.4). Since the output is a ciphertext, the simulation simply proceeds as above, outputting a random string.

Step 3a proceeds without any ideal function calls. Here, the client sends exactly $2nm$ ciphertexts to the server, each an encryption of a random bucket index, which is to be assigned to an empty element. The simulator simply sends random strings, which are again indistinguishable from the messages sent by the client in the hybrid world, due to the security of the PRF used in encryption. In Step 3b, the simulator has to simulate the output of the ideal function described in Figure A.6. In the hybrid world, when fewer than m real items are mapped to each bucket, the output of this ideal function call is a pair of ciphertexts, which can be simulated as usual. However, the simulator will not attempt to simulate the bad event in which more than m real items are mapped to a particular bucket. Instead, we rely on Lemma 5, which proves that this is negligibly likely to occur, and we allow our simulation to fail with his negligible probability. The final step of the shuffle contains another oblivious sort, which is handled as described above.

Simulating read/write: We proceed now to describe the simulation of a single read or write operation. With one important exception, the simulation of the server view during read and write executions is very similar to the simulation used in the shuffle protocol. The only message sent directly from the client to the server is the client's secret share of the soPRF output, sent in Step 2b (message γ_c in Figure 3.7). The rest of the protocol proceeds

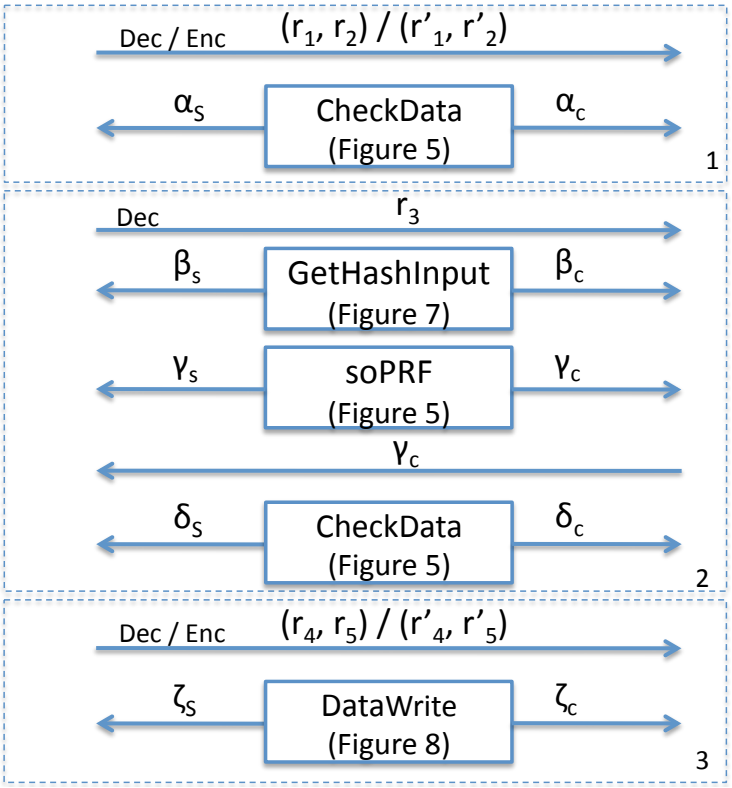


Figure 3.7: The read and write protocol in the hybrid world.

through the execution of ideal function calls, and, as before, simulating the output of these functions is straightforward. This is because the outputs of these computations are all either random secret shares, or ciphertexts, both of which can be simulated with random strings. The output from CheckData (messages α_S and δ_S) contain two secret shares which are uniformly distributed, and one ciphertext. β_S is a secret sharing, and ζ_S contains one secret sharing and one ciphertext.

The most difficult part of the security proof is to simulate the secret share of the soPRF that is sent from the client to the server (γ_c) in Step 2b. In the shuffle protocol, the players keep their output from the soPRF private, using them as input to another secure computation. This simplified the simulation, since it allowed us to simulate just one share of the pseudorandom value, which is distributed uniformly. Here, since the client sends his share in the clear, we have to simulate the reconstructed output of the PRF, and not just

a secret sharing of that output.[5]

To simulate the client's share of the soPRF output, we simply send a random group element. We stress that the simulator does *not* try to simulate a random function, because he has no idea what the input to the soPRF is; in particular, if the same input were used twice, he is very unlikely to output the same value both times. To claim that this simulation is indistinguishable from the real execution, therefore, we rely on Lemma 3, where we demonstrated that the players never reuse the same input to the soPRF in between two shuffles. (Recall that when a level is shuffled, the players refresh the key for the soPRF, so at that point it is irrelevant if they reuse an input that was used prior to the shuffle.) If they did reuse the same input, the simulation of the soPRF would be easily distinguished from the real execution.

As mentioned in the discussion of Section 3.5, another key lemma in the proof demonstrates that the buckets are large enough that they are negligibly likely to overflow during the shuffle. If this were not the case, the simulation of the soPRF *would* be distinguishable from the hybrid-world execution of the soPRF. Note that in the hybrid world protocol, we abort if $m + 1$ inputs from the same level collide under the choice of soPRF key (See Figure A.6). Therefore, the output of the soPRF in the real execution is pseudorandom, *conditioned on the fact that buckets do not overflow.* In contrast, the simulated output of the soPRF is truly random. Put more formally, for any N inputs, $v_1, \ldots, v_N$, let good_k denote the set of soPRF keys that map no more than m elements in $v_1, \ldots, v_N$ to the same value. Then we require that the following two distributions are indistinguishable:

$$\{\mathsf{soPRF}(k, v_1), \ldots, \mathsf{soPRF}(k, v_N) \mid k \xleftarrow{R} \mathcal{K}\} \stackrel{c}{=} \{\mathsf{soPRF}(k, v_1), \ldots, \mathsf{soPRF}(k, v_N) \mid k \xleftarrow{R} \mathsf{good}_k\}$$

It follows that these distributions are indistinguishable if the probability of overflow is negligible (i.e. if the set good_k contains most of the keyspace). We prove now that the buckets are unlikely to overflow if the soPRF is replaced with a random function. It follows

[5] We note that we *could* change the protocol to match the shuffling protocol, having the players use their output from the soPRF in a secure computation of the functionality depicted in Figure A.4. However, it is much more efficient to send the value in the clear. The reason we don't do the same thing when shuffling is that the protocol becomes insecure if the server sees the reconstructed output of the soPRF both during shuffling *and* during lookup.

from a hybrid argument that the simulation of message γ_c is indistinguishable from the hybrid-world soPRF.

Lemma 5 ([Mitzenmacher and Upfal, 2005])**.** *Let Y be a Poisson random variable with parameter (and mean) μ. If $y > \mu$ then* $\Pr[Y \geq y] \leq \frac{e^{-\mu}(e\mu)^y}{y^y}$.

Lemma 6 ([Mitzenmacher and Upfal, 2005])**.** *Let $X_i^{(m)}$, $1 \leq i \leq n$, be a random variable representing the number of balls in the ith bin when m balls are randomly thrown into n bins. Let $Y_1^{(m)}, \ldots, Y_n^{(m)}$ be Poisson random variables with mean m/n. Then, any event that takes place with probability p in the Poisson case, takes place with probability at most $pe\sqrt{m}$ in the exact case.*

Corollary 5. *Suppose that n balls are thrown into n bins. Then, the probability that in the end there is a bin that contains more than $z = \max(\log(n), \log(\kappa))$ balls is at most $\frac{n^{1/2}\max^{\frac{1}{\ln(2)}}(n,\kappa)}{z^z}$.*

Proof. Let $z = \max(\log(n), \log(\kappa))$. When n balls are thrown into n bins, the expected number of balls in each bin is 1. Now consider Poisson variables $Y_1, \ldots, Y_n$ with parameter $\mu = 1$. From Lemmas 5 and 6 we obtain that for all $1 \leq i \leq n$:

$$\Pr[Y_i \geq z] \leq \frac{e^{z-1}}{z^z} \implies \Pr[X_i \geq z] \leq \frac{e^z n^{1/2}}{z^z}$$

$$\frac{n^{1/2}\max^{\frac{1}{\ln(2)}}(n,\kappa)}{z^z}$$

□

We now put together the pieces and consider the full simulation of the server. Since we are in a semi-honest setting, the simulator can begin by submitting the server's input to the trusted party. The output of the honest player is always correct, and is distributed identically to the hybrid world execution. (We leave the correctness of the hybrid world protocol for the reader to verify.) Since the server receives no output, it remains only to argue that the complete view of the server in the ideal world is indistinguishable from that in the hybrid world. We have already argued above that the simulation of a *single* shuffle, or a single read write execution, is indistinguishable from the hybrid-world execution of the same protocol. We now need to argue that a sequence of such simulations remains

indistinguishable. Again, the only subtlety arises with the soPRF output; all other messages are clearly independent of one another, even across multiple shuffles, reads and writes. Given Corollary 5 and Lemma 3, it follows that the output of the soPRF in the hybrid world is indistinguishable from a random sequence of group elements, each chosen independently. This is precisely how we have simulated the soPRF, so this concludes the proof of Lemma 4.

□

Lemma 7. *For every non-uniform, polynomial time adversary $\mathcal{A}$ corrupting the client in a hybrid-world execution of the secure computation described in Section 3.5.3, there exists a non-uniform, polynomial time adversary $\mathcal{S}$ corrupting the client in the ideal world execution F, and a negligible function $neg(\cdot)$, such that for all $\kappa \in \mathbb{N}$:*

$$|\Pr[\mathsf{Real}(1^\kappa, \Pi, \mathcal{A}) = 1] - \Pr[\mathsf{Ideal}(1^\kappa, F, \mathcal{S}) = 1]| \leq neg(\kappa)$$

Proof. Simulating encryption and decryption: In many of the ideal function calls, the server provides input $F_K(r) \oplus v$, while the client provides $F_K(r)$ for decryption, and $F_K(r')$ for re-encryption (see the discussion in Section 3.5.3). In every such case, the server sends r, r' to the client before they call the ideal function. We need to describe how to simulate these random strings sent by the server. The simulator has to do some book-keeping to be sure the appropriate random strings are sent. (Recall, the client saw the random string needed for decryption at some prior time when it was used for encryption, so the randomness sent by the simulator must remain consistent with those values.) Specifically, the simulator keeps an array of size N to store random strings. He keeps track of the randomness currently being used to encrypt each item at each location in the ORAM structure. We stress that this book-keeping succeeds only because our functionalities all maintain the ordering of the randomness provided by the client. For example, in Figure A.5, note that regardless of whether the items are swapped, the ordering of the randomness provided by the client remains fixed. If this were not the case, not only would the simulation fail (because it could not know what order to place the random strings in), but the client would easily learn

something about the outcome of the oblivious sort by observing the final ordering of the randomness he provided.

In the hybrid world protocol, the only messages the client receives from the server are the random strings that we have just finished discussing (see Figure 3.6). Everything else is done through the execution of ideal functionalities. In the shuffle protocol, the simulation of these functionalities is actually a bit simpler than it was in the case of the server, because the client does not receive any output from the swap functionality used in oblivious sort (Figure A.5), from the functionality that computes an encryption of the bucket index (Figure A.4), or from the functionality that skims empty items from overfull buckets (Figure A.6). The outputs that the client receives from the remaining ideal functionalities are random secret shares of various values: β_c from GetHashInput, and γ_c from the soPRF. The simulator simply outputs random strings to simulate each of these ideal function calls. In the read/write protocol, the same is true: the reader can verify that client output from all functionalities are secert-shares of state variables (i.e. messages α_c, β_c, γ_c, δ_c and ζ_c). All can be easily simulated with random secret shares.

To put these pieces together, when the simulator receives x_i from the environment, he immediately submits them to the trusted functionality and stores the output. He simulates the view of the client through the appropriate number of read/write executions and shuffles, until the RAM protocol terminates.[6] He then sends a second secret share of the output to the client, allowing him to reconstruct the correct output value, sends the output value to the environment, and waits for the next input to arrive from the environment. We note that the server has no output, so we do not need to worry about the joint distribution over the client's view and the server's output. When we consider the simulation of a sequence of read and write executions, we have to describe how the simulator chooses which random values to send to the client (i.e. which buckets should be read). We claim that choosing a random bucket at each level suffices. This follows from two facts: a) the client's view during the shuffle protocol reveals no information about how items have been mapped to buckets, and b) as proven in Lemma 3, with overwhelming probability, the same input is never used

[6] As mentioned in Section 3.3.1, we assume that the runtime of the RAM protocol is independent of the inputs.

twice in the soPRF. □

This completes the proof of Theorem 4. □

3.7 Performance

We evaluate the performance of our protocol and compare it with the state-of-the-art solutions. First, recall our notation: N is the number of records in the database (each of length m), k is the security parameter for underlying DDH groups ($k \approx 256$). While we use big-O notation in our analysis, we stress that we do not hide any large constants; in fact we keep the constants of the higher-order terms.

Each ORAM read/write has complexity $O(4m \log^2 N + m \log N + 3 \log^3 N)$. Additionally, amortized per read/write shuffling has complexity $O((\log \log N)^3 + k^{\log_2 3} \log \log N)$. The amortized shuffling terms are of low order and may be dropped in further analysis, possibly except for the term $k^{\log_2 3} \log \log N$, (here $k^{\log_2 3} \approx k^{1.6}$ is the size of boolean circuit implementing Karatsuba multiplication [Karatsuba and Ofman, 1962]).

While each step of ORAM is relatively costly, our solution is orders of magnitude faster than existing solutions (which are all linear in input size) for important functions such as binary search on large DB (or its derivative, location-based service provision). To illustrate the costs relationship for today's medium-size DB of 10^7 records, each of size 10^5, our solution performs $\approx 5 \cdot 10^9$ basic operations (comparable to Yao-gate evaluations), vs standard solution's cost of $\approx 10^{12}$ of same operations. In general, our approach will likely be advantageous when server's input is large, and ORAM program length is short (it is logarithmic for search).

Most importantly, as DB sizes grow with time, the performance advantage of our approach will increase (as much as exponentially, for binary search).

Chapter 4

Secure Multiparty Computation for Multivariate Polynomials

4.1 Motivation and Contributions

As we already observed generic MPC approaches have inherent inefficiency related to the fact that they use circuits as their computation model and the circuit size representation of a functionality may be very large. Thus, an important open problem in MPC is designing highly efficient protocols for smaller, yet large enough to be interesting, sets of functionalities, taking advantage of the domain specific mathematical structure (election problems is a narrow example, while linear algebraic problems is a more generic example). One such class of functions are multivariate polynomials, which can be used to express many problems from linear algebra, statistics, set operations.

We consider the problem of secure multiparty computation of the class of functions that can be represented by *polynomial-size multivariate polynomials*. The multivariate polynomial is defined over the inputs of the participating parties so that each party contributes its inputs as values for some subset of the variables in the polynomial representation. There is a designated party receiving output that learns only the output of the polynomial evaluation while all other parties receive no output. [1] We assume a broadcast channel and that the

[1] We note that our protocol can be generalized to allow any subset of the parties to receive output.

private keys for the threshold encryption scheme distributed in a preprocessing stage.

4.1.1 Our Contributions

General Protocol. We present a protocol that allows multiple parties to compute the above functionalities, assuring security against a dishonest majority and robustness (detection misbehavior). Our protocol is fully black-box assuming any threshold additive homomorphic encryption with a natural property that we specify later, (instantiated by Paillier scheme, say). The protocol utilizes a "round table" structure where parties are nodes in a ring network (which means that frequently a party only communicates with the consecutive parties around the table). This structure (employed already in past protocols) has two benefits: first, it allows each party to be offline for the majority of the execution of the protocol and to be involved only when it needs to contribute its inputs at its turn. Second, it allows a division of the communication complexity into two types: "round table" communication complexity including messages exchanged between two neighboring parties, and broadcast communication complexity including messages sent simultaneously to all parties. We give simulation-based proofs of security in the Ideal/Real (standard) Model as per definitions in [Goldreich, 2005]. To the best of our knowledge, the only paper that has considered secure computation of multivariate polynomials is [Franklin and Mohassel, 2010]. This recent independent work has focused on multivariate polynomials of degree 3 but points out that the proposed protocols can be generalized to higher degree polynomials, however, with communication complexity that is no longer optimal, leaving as an open question improvements of this complexity. Their protocol is based on the compiler of [Ishai *et al.*, 2008], but with the difference being that the outer and the inner protocols inhere are instantiated with efficient constructions tailored for multivariate polynomials. The communication complexity of their protocol is (sub)-exponential in the number of variables t: $O(poly(k)d^{\lfloor t/2 \rfloor})$ for polynomials of degree d and security parameter k. Our work, in turn, improves their communication complexity to be fully polynomial (i.e., polynomial in all parameters of the problem). Obviously one can take a poly-size multivariate polynomial and translate it to a circuit with poly time secure computation solultion, but this will have a huge polynomial factor expansion and will lose the structure enabling the special-purpose

speedups. We achieve "round-table" complexity $10kDn(m-1)$ and broadcast complexity $k(10D+1)(\sum_{j=1}^{m}\sum_{t=1}^{l_j}\log\alpha_{j,t}+1)$ for m parties where party i has l_i inputs of degrees $\alpha_{i,1},\ldots,\alpha_{i,l_i}$, D being the sum of the logarithms of the variable degrees for polynomials consisting of n monomials. Next, recall that every polynomial can be easily converted into an arithmetic circuit, our protocol can be viewed as a protocol for MPC of a subclass of all arithmetic circuits. From this point of view, the work of [Ishai *et al.*, 2009] addresses a comparable problem to ours (constructing a MPC protocol for all poly-size arithmetic circuits, using a black-box construction and assuming no honest majority). The work of [Franklin and Mohassel, 2010] already improves in the worst case the complexity results of [Ishai *et al.*, 2009] (for proper set of multivariate polynomials), and as we noted above we bring additional improvement (intuitively our amortized broadcast complexity is linear in the size of the representation of the largest term of the polynomial, and does not depend on the number of terms in the representation, which contributes to the size of the arithmetic circuit). Further, the protocol of [Ishai *et al.*, 2009] requires as many rounds (involving all the parties) as the depth of the circuit and communication complexity depending on the size of the circuit. In contrast, we achieve a number of rounds independent of depth of the size of the arithmetic circuit of the polynomial (which is constant when either counting a round-table round as one round or when considering only a constant number of parties).

Special Cases. The class of polynomial size multivariate polynomials contains a wide range of efficiently representable functionalities with special structure that enables further optimizations. Most of the commonly used statistics functions can either be represented as polynomials or approximated with polynomials using Taylor series approximation for trigonometric functions, logarithms, exponents, square, etc. Examples include average, standard deviation, variance, chi-square test, Pearson's correlation coefficients, and central moment of statistical distributions. Matrix operations (i.e., linear algebra) can also be translated to polynomial evaluations.

In particular, as a special case of the general protocol, we implement *secure multiparty set intersection* against a malicious adversary controlling a majority of the parties; we note that the set intersection question in the two party case has been addressed in many

works [Freedman *et al.*, 2004; Hazay and Lindell, 2008; Kissner and Song, 2005; Jarecki and Liu, 2009; Dachman-Soled *et al.*, 2009; Cristofaro *et al.*, 2010] while there are fewer works that have considered the multiparty version. Two works adress the issue in the computational protocol setting. First, Kissner et al. [Kissner and Song, 2005] present a semi-honest protocol and suggests using generic zero knowledge techniques (ZK) to address the malicious case which requires communication complexity $O(m^2d^2)$ for m parties with input sets of size d. The work of [Sang and Shen, 2009] improves this complexity by a factor of $O(m)$ for m party protocols, using more efficient ZK based on pairings. In addition, relatively inefficient information theoretic solutions are presented in [Patra *et al.*, 2009a; Patra *et al.*, 2009b]). Our protocol achieves communication complexity $O(md + 10d \log^2 d)$ improving the existing works. We note that we achieve linear complexity in the number of parties m due to the round table communication paradigm, whereas even the recent unpublished work [Cheon *et al.*, 2010] is quadratic in the number of parties. We note that our scheme extends the classical approach of representing a set as the zeroes of a polynomial as in [Freedman *et al.*, 2004; Camenisch and Zaverucha, 2009; Agrawal *et al.*, 2003; Dawn and Song, 2005].

Finally, if we view the polynomial's coefficients as the input of the designated output receiver, we obtain a *multi-party oblivious multivariate polynomial evaluation*, a generalization of the problem of oblivious polynomials evaluation [Naor and Pinkas, 2006] to inputs from multiple parties.

Techniques. Multivariate polynomials have a "nice structure" and in our protocol we utilize a number of techniques exploiting the structure and various interactions of this structure with structures of other algebraic and cryptographic primitives.

We crucially utilize the fact that multivariate polynomials can be viewed as linear operators when combined with additive homomorphic encryption and polynomial secret sharing. We formalize this property by presenting a commutativity property between the evaluation of multivariate polynomials and reconstruction of Shamir's secret sharing [Shamir, 1979]. Intuitively, this allows us to evaluate a given polynomial on multiple (modified) Shamir secret shares in parallel and obtain the final evaluation of the polynomial by re-

constructing the secret shares. This technique is useful since it allows us to apply generic (black box) "cut-and-choose" techniques to verify the correctness of the evaluation for malicious parties, without revealing information about the shared inputs or outputs. We note that analogous techniques were used in a different context by [Choi *et al.*, 2008; Dachman-Soled *et al.*, 2009].

A second property of multivariate polynomials is that they can be viewed as a collection of monomials which can be computed under additive homomorphic encryption non-interactively in a round-table type protocol where each participant incrementally contributes his inputs to the encryption of the partial monomial evaluation done by the previous participants (note that each participant's contribution is a multiplication by a scalar).

We additionally use the polynomial structure of a variant of Shamir's threshold sharing in zero knowledge protocols proving that inputs were shared correctly and committed under homomorphic encryption. We utilize Lagrange interpolation combined with what we call vector homomorphic encryption (where the homomorphic properties hold for both the plaintexts and the encryption randomness; which is true for many of the known homomorphic encryption schemes [Paillier, 1999; Fouque *et al.*, 2001; ElGamal, 1985; Goldwasser and Micali, 1982]) to verify that inputs were shared correctly by interpolating over encrypted values. This verifies that inputs were shared and encrypted correctly, provided that the randomness for the encryptions was chosen in a specific way. This encrypted interpolation technique combined with the large minimum distance of Reed-Solomon codes allows us to guarantee the correctness of an entire computation on encrypted codewords based on the verification that a small random subset of shares were computed correctly. Finally, we use the linear operator properties of the sharing polynomials for share re-randomization under additive homomorphic encryption.

We note that when we perform our protocol with homomorphic encryption over a ring, then we use the technique initiated by Feldman ([Feldman, 1987]) and also used, e.g., in Fouque et al. ([Fouque *et al.*, 2001]) for Paillier sharing that transforms interpolation over an RSA-composite ring and similar structures to an interpolation over (a range of) the integers (where computing inverses, i.e., division, is avoided and finding uninvertible elements is hard, assuming factoring is hard).

4.2 Solution Overview

Semi-honest structure. As described above, we view multivariate polynomials as a collection of monomials that can be computed under homomorphic encryption in a round-table type protocol. We employ this to construct the underlying semi-honest evaluation protocol.

Robustness idea. To achieve security against malicious adversaries, in turn, we employ the commutativity between evaluation of multivariate polynomials and Shamir's secret sharing reconstruction described above. Consider the following very simplified example that illustrates a few of the basic techniques we utilize. Let us say that we have m parties that wish to evaluate the univariate polynomial $\mathbf{Q}(x) = x^5 + 10x^3 + 6x + 9$, at point x, where x is the input of Party 1. Letting Party 1 execute the whole computation will not handle the case the it is a malicious party. One approach will be to use techniques based on threshold fully homomorphic encryption schemes [Myers *et al.*, 2011; Asharov *et al.*, 2012], which will be more computationally expensive than approaches using circuit representation for the function.

Instead, we take the following approach: Party 1 computes a Shamir secret-sharing of its input x by choosing a polynomial P_x of degree k uniformly at random conditioned on $P_x(0) = x$. Now, instead of committing to the value x, Party 1 commits to, say, $20k$ input shares of P_x : $P_x(1), \ldots, P_x(20k)$. Next, Party 1 commits to $20k$ output shares of $\mathbf{Q} \circ P_x(i)$: $\mathbf{Q}(P_x(1)), \ldots, \mathbf{Q}(P_x(20k))$. Notice that $\mathbf{Q} \circ P_x(i)$ is a polynomial of degree $5k$ and that $\mathbf{Q} \circ P_x(0) = Q(P_x(0)) = Q(x)$. Thus, by reconstructing $\mathbf{Q} \circ P_x(0)$ we obtain the output value $\mathbf{Q}(x)$. After Party 1 sends the input and output commitments, the parties verify efficiently that the input and output shares indeed lie on a polynomial of degree k and $5k$ respectively using an interpolation algorithm we define below. Now, the parties run a cut-and-choose step where a set $I \subset [20k]$ of size k is chosen at random. For each index $i \in I$, Party 1 must open the commitments to reveal $P_x(i)$ and $\mathbf{Q} \circ P_x(i)$. The remaining parties now verify privately that for each index, $\mathbf{Q} \circ P_x(i)$ was computed correctly. Note that due to the secret-sharing properties of the commitment scheme, the cut-and-choose step reveals no information about $P_x(0) = x$ or $\mathbf{Q} \circ P_x(0) = \mathbf{Q}(x)$.

Now, let's assume that Party 1 acted maliciously. Since the set I was chosen at random, we have that if Party 1 is able to open all the shares corresponding to I correctly, then with very high probability Party 1 must have computed *all* of the output shares correctly. We note that the above description leaves out important **re-randomization techniques** (that are described in the full protocol) whose goal is to prevent parties from learning during the incremental evaluation and robustness checking. When the polynomial has inputs from more than one party, for example $\mathbf{Q}(x, y) = x^3y^2 + x$ where Party 1 contributes x and Party 2 contributes y the evaluation is broken down at the monomial level $M_1(x, y) = x^3y^2$ and $M_2(x, y) = x$. Party 1 evaluates $M_1^1(x) = x^3$ and $M_2^1 = x$ on its shares and passes the encrypted values to Party 2 $\mathsf{Enc}(\mathsf{M}_1^1(\mathsf{P_x}(1))), \ldots, \mathsf{Enc}(\mathsf{M}_1^1(\mathsf{P_x}(20)))$ and $\mathsf{Enc}(\mathsf{M}_2^1(\mathsf{P_x}(1))), \ldots, \mathsf{Enc}(\mathsf{M}_2^1(\mathsf{P_x}(20)))$. Party 2 evaluates $M_1^2(y) = y^2$ and $M_2^2(y) = 1$ on its shares and contributes then to the received encryption using the homomorphic properties of the encryption $(\mathsf{Enc}(\mathsf{M}_\mathsf{i}^1(\mathsf{P_x(j)})))^{\mathsf{M}_\mathsf{i}^2(\mathsf{P_y(j)})} = \mathsf{Enc}(\mathsf{M}_\mathsf{i}^1(\mathsf{P_x(j)}) * \mathsf{M}_\mathsf{i}^2(\mathsf{P_y(j)})) = \mathsf{Enc}(\mathsf{M_i}(\mathsf{P_x(j)}, \mathsf{P_y(j)}))$ for $i = 1, 2$ and $1 \leq j \leq 20$. Now the verification proceeds as above where both Party 1 and Party 2 will open the shares chosen in the cut-and-choose protocol.

Efficient Robustness. Although the technique described above is sufficient to ensure that the parties behave honestly, it causes a huge blow-up in the number of required shares. This is because in order to reconstruct the zero coefficient of a polynomial of degree deg, we must have at least $deg + 1$ secret shares. Thus, when evaluating a polynomial such as $\mathbf{Q} = x^{2^n}$, we would require an exponential number of shares. To prevent this blow-up, we utilize an input preprocessing step (described in Section **??**).

Secure output reconstruction. Finally, we use an additive homomorphic encryption scheme with a threshold decryption algorithm to ensure that no subset of the parties can decrypt any of the intermediate messages exchanged between the parties in the protocol. The threshold decryption is needed in the case that more than one party contributes its inputs to the polynomial (and is actually not necessary in our example above). Although any additive homomorphic threshold encryption scheme (with one additional natural property, which we describe later) would suffice for the correctness of our protocol, the only such schemes that we are aware of are the El Gamal threshold encryption scheme [Gennaro *et*

al., 2007] and the Paillier threshold encryption scheme [Fouque *et al.*, 2001]. Additive El Gamal does not allow efficient decryption over a large domain (field), but it suffices for our Set Intersection applications. We use the Paillier threshold encryption scheme (over a ring) to instantiate our general polynomial evaluation protocols. To obtain the final output, the designated party receiving output decrypts reconstructs the encryption of the final output value using Lagrange interpolation over encrypted values and decrypts with the help of the other parties.

4.3 Definitions and Techniques

4.3.1 Definitions

We use a standard simulation-based definition of security see [Canetti, 2000b], and follow the definitions of zero knowledge proofs of knowledge and commitment schemes presented in [Goldreich, 2005]. We denote by Com_B a perfectly binding commitment scheme and by Com_H a perfectly hiding commitment scheme.

Lagrange interpolation allows reconstruction of a polynomial of degree d given $d+1$ evaluation points in the following way:

Definition 4. *Let $(x_0, y_0), \ldots (x_d, y_d)$ be $d+1$ evaluation point of a polynomial of degree d. We can reconstruct the evaluation $L(x)$ of the polynomial at point x as $L_{x_0,\ldots,x_d}(y_0, \ldots, y_d, x) = \sum_{j=0}^{d} y_j l_j(x)$*
using the Langrange basis polynomials $l_j(x_0, \ldots, x_d, x) = \prod_{i=0, i\neq j}^{d} \frac{x-x_i}{x_j-x_i}$ for $0 \leq j \leq d$.

In most cases where we will use Lagrange interpolation the points $x_0, \ldots, x_d$ will be $1, \ldots, d$ and we will omit them and use the notation $L(y_0, \ldots, y_d, x)$ and $l_j(x)$.

We utilize the following notation for Shamir's secret sharing scheme [Shamir, 1979], which can also be viewed as a Reed-Solomon encoding of x:

Definition 5. *Let R be a ring and let $x \in R$. Let $P_x \in R[x]$ be a random polynomial of degree t (threshold decryption parameter) such that $P_x(0) = x$, let $z_1, \ldots, z_k$ be points different from 0. Then we say that the values $P_x(z_1), \ldots, P_x(z_k)$, are* shares *of x. We say that*

x is reconstructed *from the k shares $P_x(z_1), \ldots, P_x(z_k)$ for $k > t$ when the value $x = P_x(0)$ is computed via Lagrange interpolation, which we denote by $L_{z_1,\ldots,z_k}(P_x(z_1), \ldots, P_x(z_k), 0)$.*

We require threshold additive homomorphic encryption scheme with the following additional property, capturing the fact that the homomorphism applies also to the randomness.

Property 1 (Vector Homomorphic Encryption)**.** *Let $E = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ be an encryption scheme where the plaintexts come from a ring R_1 with operations $(+, \cdot)$, the randomness comes from a ring R_2 with operations $(\oplus, \odot)$, and the ciphertexts come from a ring R_3 with operations $(\otimes, \hat{})$. We say that E is* vector homomorphic *if the following holds:* $\mathsf{Enc}(\mathsf{m}_1; \mathsf{r}_1) \otimes \mathsf{Enc}(\mathsf{m}_2; \mathsf{r}_2) = \mathsf{Enc}(\mathsf{m}_1 + \mathsf{m}_2; \mathsf{r}_1 \oplus \mathsf{r}_2)$ *and* $\mathsf{Enc}(\mathsf{m}; \mathsf{r})^{\mathsf{c}} = \mathsf{Enc}(\mathsf{c} \cdot \mathsf{m}; \mathsf{r} \odot \mathsf{c})$.

Such a property is satisfied by most known homomorphic encryption schemes, such as Paillier [Paillier, 1999] and threshold Paillier [Fouque *et al.*, 2001], ElGamal [ElGamal, 1985], and Goldwasser-Micali [Goldwasser and Micali, 1982] encryption schemes. In the case of Paillier encryption, which we will use in our protocols, we have the following operations: addition ($+$); multiplication ($\cdot$, $\oplus$, $\otimes$); exponentiation ($\odot$, $\hat{}$).

4.3.2 Techniques

In this section we describe the main techniques that we employ in our protocols.

4.3.2.1 Polynomial Code Commutativity.

Shamir secret sharing/Reed-Solomon codes are commutative with respect to polynomial evaluations, which we formalize as follows:

Property 2 (Polynomial Code Commutativity)**.** *Let $\mathbf{Q}(x_1, \ldots, x_m)$ be a multivariate polynomial, t be the threshold for Shamir secret-sharing and L be the secret reconstruction algorithm for Shamir secret-sharing. The evaluation of $\mathbf{Q}$ commutes with L in the sense that we can compute the value $\mathbf{Q}(x_1, \ldots, x_m)$ with either of the following algorithms starting from shares of the inputs $x_1, \ldots, x_m$:*

$$(\mathbf{Q} * L)(P_{x_1}(1), \ldots P_{x_1}(t+1), \ldots P_{x_m}(1), \ldots P_{x_m}(t+1), 0) =$$
$$= \quad \mathbf{Q}((L(P_{x_1}(1), \ldots P_{x_1}(t+1), 0), \ldots, L(P_{x_m}(1), \ldots P_{x_m}(t+1), 0))) = \mathbf{Q}(x_1, \ldots x_m),$$

where we first use L to retrieve the secrets and then evaluate $\mathbf{Q}$ on those secrets and

$$\begin{aligned}
&(L * \mathbf{Q})(P_{x_1}(1), \ldots P_{x_1}(t+1), \ldots P_{x_m}(1), \ldots P_{x_m}(t+1), 0) = \\
= \quad & L(\mathbf{Q}(P_{x_1}(1), \ldots, P_{x_m}(1)), \ldots, \mathbf{Q}(P_{x_1}(t+1), \ldots, P_{x_m}(t+1)), 0) = \\
= \quad & L(w_1, \ldots, w_{t+1}, 0) = \mathbf{Q}(x_1, \ldots x_m),
\end{aligned}$$

where we evaluate $\mathbf{Q}$ on each set of shares of $x_1, \ldots, x_m$ to obtain shares of $\mathbf{Q}(x_1, \ldots, x_m)$ and then use L to reconstruct the final secret.

The above property allows us to evaluate $\mathbf{Q}$ in parallel on shares of the inputs to yield shares of the output.

4.3.2.2 Incremental Encrypted Polynomial Evaluation.

We will use homomorphic encryption to allow multiple parties to evaluate a multivariate polynomial depending on their inputs by incrementally contributing their inputs to partial encrypted evaluations of its monomials. This is facilitated by the following property:

Property 3 (Incremental Encrypted Polynomial Evaluation)**.** *Let m be the number of parties evaluating a multivariate polynomial $\mathbf{Q}$ defined by*

$$\mathbf{Q}(x_{1,1}, \ldots, x_{1,l_1}, \ldots, x_{m,1}, \ldots, x_{m,l_m}) = \sum_{s=1}^{n} c_s (\prod_{j=1}^{m} h_{j,s}(x_{j,1}, \ldots, x_{j,l_j})),$$

where $h_{j,s}$ represents the inputs of party j to the s-th monomial of $\mathbf{Q}$. Let $E = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ be an additive homomorphic encryption. We define the partial evaluations $b_{j,s}$ (including the contributions of parties 1,..., j) of the monomials $s, 1 \leq s \leq n$ of $\mathbf{Q}$ as follows:

$$b_{0,s} = \mathsf{Enc}(\mathsf{c_j}) \text{ for } 1 \leq j \leq n, \text{ and } b_{j,s} = b_{j-1,s}^{h_{j,s}(x_{j,1}, \ldots, x_{j,l_j})} \text{ for } 1 \leq j \leq m$$

Polynomial Interpolation Over Encrypted Values In Figure 4.1 We present a protocol that allows a verifier to verify (without help from the prover) that the prover's encrypted points lie on a polynomial of low degree, assuming the prover constructed the encryptions in a predetermined manner. Recall that Lagrange interpolation allows us, given $d+1$ points, to reconstruct the polynomial of degree d that interpolates the given points. In the following, we use the fact that Lagrange interpolation can, in fact, be carried out over

encrypted points when the known encryption used possesses the vector homomorphic Property 1. Since the encryption is over a ring we use Feldman's technique for shift interpolation by factorial [Feldman, 1987].

Lagrange Interpolation Protocol Over Encrypted Values (LIPEV)

Input: $(1, \mathsf{Enc_{pk}}(\mathsf{y_1}, \mathsf{r_1})), \ldots (\mathsf{A}, \mathsf{Enc_{pk}}(\mathsf{y_A}, \mathsf{r_A}))$, d where $d + 1 < A$,

Output: Verifier outputs Accept if there are polynomials $P_1 \in R_1[x], P_2 \in R_2[x]$ of degree at most d such that $y_j = P_1(j)$ for $1 \leq j \leq A$ and $r_j = P_2(j)$ (P_1 and P_2 are defined with respect to the operations in the respective rings) for $1 \leq j \leq A$.

Protocol:

1. Let $\Delta = A!$.
2. Let $l_j(x) = \Delta \cdot \prod_{i=1, i \neq j}^{d+1} \frac{x-i}{j-i}$ for $1 \leq j \leq d+1$.
3. Verifier checks whether $\mathsf{Enc_{pk}}(\mathsf{y_i}, \mathsf{r_i})^{\Delta} = \prod_{\mathsf{j}=1}^{\mathsf{d}+1} (\mathsf{Enc_{pk}}(\mathsf{y_j}, \mathsf{r_j}))^{\mathsf{l_j(i)}}$, and rejects otherwise.

Figure 4.1: Encrypted Interpolation

Using the LIPEV protocol, a prover can prove to a verifier that A encrypted points lie on one polynomial of degree d, provided that the randomness for the encryptions was chosen in a specific way; namely, the random values chosen must also lie on a polynomial of degree d. For completeness, we describe in Figure 4.2 how to compute the random values for the encryptions so that they lie on a polynomial $P_2 \in R_2[x]$ of degree d. We note that even though the randomness for all A encrypted points are not chosen uniformly at random, semantic security is still preserved since the randomness for $d + 1$ of the points is chosen uniformly at random and the remaining $A - d - 1$ encryptions can be computed given only the first $d + 1$ encryptions due to Property 1.

4.4 Building Block Protocols

In this section we introduce several subprotocols that we will use in our main construction.

Generation of Encryption Randomness for Shamir Shares

Input: $(1, y_1), \ldots, (A, y_A), r_1, \ldots, r_{d+1}$ for $d + 1 < A$

Output: $r_{d+2}^{\Delta}, \ldots, r_A^{\Delta}$ such that the Lagrange Interpolation Protocol Over Encrypted Values outputs accept on the input: $[i, \mathsf{Enc_{pk}}(\mathsf{y_i}, \mathsf{r_i})^{\Delta}]_{1 \leq \mathsf{i} \leq \mathsf{A}}, \mathsf{d}$.

Protocol:

- Compute $l_j(x) = \Delta \prod_{i=0, i \neq j}^{d+1} \frac{x-i}{j-i}$ for $1 \leq j \leq d + 1$.
- Compute $r_i^{\Delta} = \prod_{j=0}^{d+1} (r_j)^{l_j(i)}$ for $d + 2 \leq i \leq A$.

Figure 4.2: Interpolation Randomness

4.4.1 Multiparty Homomorphic Encryption Proof of Knowledge and Plaintext Verification (HEPKPV)

In several places in our main protocol we would need to enable a party to prove statements to the other participants in the computation about plaintext values given the encryptions of their shares. For this purpose we introduce a protocol multiparty homomorphic encryption proof of knowledge and verification that allows a prover to show that the plaintexts underlying a set of homomorphic encryptions belong to a particular language.

To illustrate the main idea for the proof in this protocol we consider the example where a prover (P) needs to prove to a verifier (V) that a ciphertext $c = \mathsf{Enc}(0; \mathsf{r})$ encrypts zero. In this case the two parties execute the following protocol:

- Prover sends to Verifier $c' = \mathsf{Enc}(0; \mathsf{r}')$.
- Verifier sends to Prover $b \in \{0, 1\}$.
- If $b = 0$, Prover reveals r' and Verifier checks that $c' = \mathsf{Enc}(0; \mathsf{r}')$.
- If $b = 1$, Prover reveals $r \cdot r'$ and Verifier checks that $c \cdot c' = \mathsf{Enc}(0; \mathsf{r} \cdot \mathsf{r}')$.

In the following full protocol we boost the soundness of the above proof from 1/2 to $1 - \text{negl}$ with a polynomial number of parallel executions. To provide security for the parallel executions we utilize the modification of [Goldwasser and Micali, 1982] which requires the verifier to first commit to its challenge and extend this modified protocol to the multiparty

setting. We use the fact that a slight variation of the above protocol can be used to prove that the plaintexts belong to any language that is closed under addition/subtraction to be able to instantiate the protocols with any of the following languages:

- *Description*: Language consisting of a vector of encryptions of *val*:

$$L^u_{val} = \{[c_{0,j}]_{1\leq j\leq u} \mid \text{ for } 1 \leq j \leq u, c_{0,j} = \mathsf{Enc_{pk}}(\mathsf{val};\mathsf{r}) \text{ for some } \mathsf{r}\}$$

- *Description*: Language consisting of a vector of pairs of encryptions that encrypt the same value.

$$\begin{aligned} L^u_{=} = \quad \{[c^0_{0,j}, c^1_{0,j}]_{1\leq j\leq u} \mid \quad & \text{for } 1 \leq j \leq u, c^0_{0,j} = \mathsf{Enc_{pk}}(\mathsf{m_0};\mathsf{r_0}) \text{ and} \\ & c^1_{0,j} = \mathsf{Enc_{pk}}(\mathsf{m_1};\mathsf{r_1}) \text{ and } \mathsf{m_0} = \mathsf{m_1} \text{ for some } \mathsf{r_0}, \mathsf{r_1}\} \end{aligned}$$

  We note that this language will be used subsequently with pairs of the form $(c^0_{0,j}, c^1_{0,j})$ such that $c^0_{0,j}$ and $c^1_{0,j}$ have been interpolated from encrypted shares of polynomials of degree k and $2k$ respectively.

Lemma 8. *Assume that* $E = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ *is a CPA-secure Vector Homomorphic encryption scheme. Then protocol* Π_{POK} *presented in Figure 4.3 is a zero knowledge proof of knowledge for* L*.*

See Appendix B.1 for the proof of the lemma.

4.4.2 Multiparty Coin Tossing

Following ideas from [Lindell, 2001; Barak and Lindell, 2002] and using the HEPKPV protocol as an efficient zero knowledge proof we present a protocol that implements constant round multiparty coin tossing in Figure 4.4. The goal of the protocol is to allow s parties $T_1, \ldots, T_s$ to jointly compute a random number in the interval $[1, max]$. We assume the existence of broadcast channel that the parties use for communication. All parties know a homomorphic encryption scheme $E = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$.

Homomorphic Encryption Proof of Knowledge and Proof of Verification

Input:

$P_1 \leftarrow \overline{C} = (pk, c_1, \ldots, c_u)$, $(x_1, \ldots x_u)$, $(r_1, \ldots, r_u)$, k, L where $c_i = \mathsf{Enc_{pk}}(\mathsf{x_i}; \mathsf{r_i})$ for $1 \leq i \leq u$;
$P_2, \ldots P_s \leftarrow \overline{C} = (pk, c_1, \ldots, c_u)$, L

Output:

$P_2, \ldots P_s$ output *ACCEPT* if $(c_1, \ldots, c_u) \in L$, and *REJECT* otherwise.

Protocol:

1. For $1 \leq i \leq k$, P_1 computes random encryptions $(c_{i1}, \ldots, c_{iu}) = (\mathsf{Enc_{pk}}(\mathsf{e_{i1}}, \mathsf{r_{i1}}), \ldots, \mathsf{Enc_{pk}}(\mathsf{e_{iu}}, \mathsf{r_{iu}}))$ such that $(c_{i1} \ldots, c_{iu}) \in L$. for $1 \leq i \leq k$ and sends them on the broadcast channel.
2. For $2 \leq j \leq s$ P_j chooses a sequence of k bits $b_{j1} \ldots b_{jk}$ and sends a commitment to those bits: $Com_H(b_{j1} \ldots b_{jk})$ on the broadcast channel.
3. P_1 chooses a sequence of k bits $b_{11} \ldots b_{1k}$ and sends them on the broadcast channel.
4. For $2 \leq j \leq s$ P_j decommits the value $b_{j1} \ldots b_{jk}$.
5. All parties verify that the decommitted values correspond to the commitments that were sent earlier and if the check fails abort.
6. The parties compute $b_1 \ldots b_k = \oplus_{j=1}^{s}(b_{j1} \ldots b_{jk})$ and for each $1 \leq i \leq k$:
   (a) if $b_i = 0$, P_1 sends to on the broadcast channel $\overline{M} = ((e_{i1}, r_{i1}), \ldots, (e_{iu}, r_{iu}))$.
   (b) if $b_i = 1$, P_1 sends on the broadcast channel $\overline{M} = ((x_1 + e_{i1}, r_1 r_{i1}), \ldots, (x_\ell + e_{iu}, r_\ell r_{iu}))$.
7. For each $1 \leq i \leq k$:
   (a) if $b_i = 0$ each party P_j for $2 \leq j \leq s$ verifies that $(c_{i1}, \ldots, c_{iu}) = (\mathsf{Enc_{pk}}(\mathsf{e_{i1}}, \mathsf{r_{i1}}), \cdots, \mathsf{Enc_{pk}}(\mathsf{x_{iu}}, \mathsf{r_{iu}}))$ and $(c_{i1}, \ldots, c_{iu}) \in L$. If the verification fails the protocol aborts.
   (b) if $b_i = 1$ each party P_j for $2 \leq j \leq s$ verifies that $(c_1 c_{i1}, \ldots, c_u c_{iu}) = (\mathsf{Enc_{pk}}(\mathsf{x_1} + \mathsf{e_{i1}}, \mathsf{r_1 r_{i1}}), \cdots, \mathsf{Enc_{pk}}(\mathsf{x_u} + \mathsf{e_{iu}}, \mathsf{r_u r_{iu}}))$ and that $(c_1 c_{i1}, \ldots, c_u c_{iu}) \in L$. If the verification fails the protocol aborts.
8. If any of the verifications steps of P_1 fail, abort the protocol. Otherwise, accept.

Figure 4.3: HEPKPV

Lemma 9. *If* $E = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ *is semantically secure homomorphic encryption scheme, the protocol* Π_{coin} *presented in Figure 4.4 is a secure multiparty protocol when there is at*

Multiparty Coin Tossing

Input: $E = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$, max, k
Output: R
Protocol:

1. For each $1 \leq i \leq s$ party T_i runs Gen to obtain a pair of public and private keys (pk_i, sk_i).
2. For each $1 \leq i \leq s$ party T_i chooses a random $R_i \in [0, max]$ and commits to it sending $C_i = \mathsf{Enc}_{\mathsf{pk_i}}(\mathsf{R_i}, \mathsf{r_i})$ to all other parties on the broadcast channel, where r_i is the randomness used for encryption.
3. For each $1 \leq i \leq s-1$ party T_i sends the value R_i on the broadcast channel.
4. All parties run in parallel s executions of the multiparty HEPKPV protocol. In the j-th execution for $1 \leq i \leq s$ party T_j proves that $C_j \in L^1_{R_j}$, i.e. C_j is an encryption of R_j.
5. Party T_s computes $R = \oplus_{i=1}^{s} R_i$ and runs the multiparty HEPKPV protocol to prove that $C_s \in L^1_{R-R'}$, i.e. $R = R' + R_s$ where C_s was the commitment of R_s and $R' = \oplus_{i=1}^{s-1} R_i$.

Figure 4.4: Choosing Joint Randomness

least one honest party.

See Appendix B.2 for the proof of the lemma.

4.4.3 Input Preprocessing and Verification

One of the ideas that we employ in our main protocol is to share function evaluation by secret sharing the arguments of the function via polynomials of degree k and evaluating the function on the corresponding shares in order to obtain shares of the final value of the function. The above can be implemented straightforwardly using Shamir's secret sharing ([Shamir, 1979]) and evaluating the polynomial on corresponding shares. The problem with this approach is that if the degree of the output function is Deg, then we require at least $k \cdot Deg$ shares to reconstruct the output value. We avoid this blow-up in number of required shares by applying the following transformation on the inputs.

We consider a multivariate polynomial $\mathbf{Q}(x_1, \ldots, x_\ell)$ over the input set $\overline{X} = \{x_1, \ldots, x_\ell\}$ which we would like to evaluate on shares of the input variables in order to obtain shares of the output value. Since the number of output shares that we will need to compute will

depend on the degree of $\mathbf{Q}$ and the degrees of the sharing polynomials P_{x_i} that we use to share each of the input variable, we employ techniques that allow us to decrease the degree of the final polynomial [Dachman-Soled *et al.*, 2009]. The main idea is to introduce new variables that represent higher degrees of the current variables. For each variable x_i that has maximum degree d_i in $\mathbf{Q}$ we substitute each power $x_i^{2^j}$ with a new variable for $0 \leq j \leq \lfloor \log d_i \rfloor$. Note that if we view the original polynomial $\mathbf{Q}$ as a polynomial over these new variables, we have that each variable has degree at most one. Let M_t be the t-th monomial in $\mathbf{Q}$ and let $d_{M_t,i} \in M_t$ be the degrees of the variables in monomial M_t. Thus, the original degree of $\mathbf{Q}$ was $\max_t\{\sum_{i \in M_t} d_{M_t,i}\}$, whereas the degree of the transformed polynomial over the new variables is only $\max_t\{\sum_{i \in M_t} \log d_{M_t,i}\}$. In our protocol, each party will pre-process its inputs to compute its new input variables and their shares and will prove that the new shares are consistent with the initial inputs.

We realize the above functionality through two protocols: *Efficient Preprocessing of Input*, presented in Figure 4.5, which starts with the input variables of a party and produces commitments to shares of the new variables computed according to the above intuition, and *Preprocessing Verification*, presented in Figure 4.6, that takes the commitments from the efficient preprocessing and verifies their correctness by opening and verifying the committed values in a random subset of those. For the purposes of these protocols let J be the ordered set of all elements of $\{0,1\}^{10kD}$ that contain exactly k ones. Note that given R, an index of a string in the set J, we can efficiently reconstruct the R-th string, j_R. Let $J_R = \{i | j_R[i] = 1\}$, where $j_R[i]$ denotes the i-th position of the string j_R.

Claim 1. *Assume R was chosen randomly after T committed to its inputs through the Efficient Preprocessing protocol. If the parties run the Preprocessing Verification protocol and do not abort, then with all but negligible probability, the committed input shares of T are valid encryptions of $k+1$-sharing polynomials of the inputs $x_\ell, x_\ell^2, \ldots, x_\ell^{2^{\lfloor log d_\ell \rfloor}}$.*

See Appendix B.3 for the proof of the claim.

Lemma 10. *For all sets X_1, X_2 where $|X_1| = |X_2| = poly(k)$, we have that the output distributions of the Verifier in consecutive executions of the Efficient Preprocessing protocol and Preprocessing Verification protocol with inputs X_1 and X_2 (respectively) are computa-*

Efficient Input Preprocessing

Let k be a security parameter and $10kD$ be the number of required shares. Let T be one of the parties with input set X of size $|X|$. For each input $x_j \in X$, we assume that the highest power of x_j required is $x_j^{2^{\alpha_j}}$.

1. For each $x_j \in X$, T chooses a random polynomial P_{x_j} of degree k such that $P_{x_j}(0) = x_j$, and computes shares of the form $P_{x_j}(i)$ for $1 \leq i \leq 10kD$. T chooses randomly values $r_{j,1}, \ldots, r_{j,k+1}$ and computes according to the Randomness generation protocol the values $r^{\Delta}_{j,k+2}, \ldots r^{\Delta}_{j,10kD}$. T computes the encryptions $\mathsf{Enc}(\mathsf{P}_{\mathsf{x_j}}(\mathsf{i}), \mathsf{r_{j,i}})^{\Delta}$.
2. For each x_j, for $\ell = 0$ to α_j, for $i = 1$ to $10kD$, T chooses randomly values $r_{j,\ell,1}, \ldots, r_{j,\ell,k+1}$ and $r'_{j,\ell,1}, \ldots, r'_{j,\ell,k+1}$ and computes according to the Randomness generation protocol the values $r^{\Delta}_{j,\ell,k+2}, \ldots r^{\Delta}_{j,\ell,10kD}$ and $(r'_{j,\ell,k+2})^{\Delta}, \ldots (r'_{j,\ell,10kD})^{\Delta}$. S computes the following:
   - Local Computation on Shares: a polynomial $P^2_{x_j^{2^\ell}}$ of degree $2k$ such that $P^2_{x_j^{2^\ell}}(i) = (P_{x_j^{2^\ell}}(i))^2$
   - Degree Reduction Step: a random polynomial $P_{x_j^{2^{\ell+1}}}$ of degree k such that $P_{x_j^{2^{\ell+1}}}(0) = P^2_{x_j^{2^\ell}}(0)$.
3. For each x_j, for $\ell = 0$ to α_j, for $i = 1$ to $10kD$, T computes the following commitments:
   - New input shares: $\mathsf{Enc_{pk}}(\mathsf{P}_{\mathsf{x_j^{2^{\ell+1}}}}(\mathsf{i}), \mathsf{r_{j,\ell,i}})^{\Delta} = \mathsf{Enc_{pk}}(\Delta \cdot \mathsf{P}_{\mathsf{x_j^{2^{\ell+1}}}}(\mathsf{i}), \mathsf{r}^{\Delta}_{\mathsf{j,\ell,i}})$ and
   - Intermediate shares: $\mathsf{Enc_{pk}}(\mathsf{P}^2_{\mathsf{x_j^{2^\ell}}}(\mathsf{i}), \mathsf{r'_{j,\ell,i}})^{\Delta} = \mathsf{Enc_{pk}}(\Delta \cdot \mathsf{P}^2_{\mathsf{x_j^{2^\ell}}}(\mathsf{i}), (\mathsf{r'_{j,\ell,i}})^{\Delta})$

   where the randomness $r^{\Delta}_{j,\ell,i}$ and $(r'_{j,\ell,i})^{\Delta}$ are chosen using the Randomness generation protocol and sends them on the broadcast channel.

Figure 4.5: Input Preprocessing

tionally indistinguishable.

See Appendix B.3 for the proof of the lemma.

4.5 Main Protocol

The multiparty polynomial evaluation has the following setup:

- Each party T_j has l_j inputs $\overline{X}_j = \{x_{j,1}, \ldots, x_{j,l_j}\}$ for $1 \leq j \leq m$.
- A polynomial $\mathbf{Q}(x_{1,1}, \ldots, x_{1,l_1}, \ldots, x_{m,1}, \ldots, x_{m,l_m})$, which depends on the inputs of all parties.

Input Preprocessing Verification

Common Inputs:

Commitments: $\mathsf{Enc}_{\mathsf{pk}}(\mathsf{P}_{\mathsf{x}_j^{2^{\ell+1}}}(\mathsf{i}), \mathsf{r}_{\mathsf{j},\ell,\mathsf{i}})^{\Delta}$, $\mathsf{Enc}_{\mathsf{pk}}(\mathsf{P}^2_{\mathsf{x}_j^{2^{\ell}}}(\mathsf{i}), \mathsf{r}'_{\mathsf{j},\ell,\mathsf{i}})^{\Delta}$ for $1 \leq j \leq s, 0 \leq \ell \leq \alpha_j, 1 \leq i \leq 10kD$

A number number $R \in [|J|]$ chosen by all parties using the Coin-Tossing protocol after T committed to its inputs.

Private Inputs of T: Decommitments to the above values.

Protocol:

1. For all $i \in J_R, 1 \leq j \leq s, 0 \leq \ell \leq \alpha_j, 1 \leq i \leq 10kD$ S opens the commitment $\mathsf{Enc}_{\mathsf{pk}}(\mathsf{P}_{\mathsf{x}_j^{2^{\ell+1}}}(\mathsf{i}), \mathsf{r}_{\mathsf{j},\ell,\mathsf{i}})^{\Delta}$ and $\mathsf{Enc}_{\mathsf{pk}}(\mathsf{P}^2_{\mathsf{x}_j^{2^{\ell}}}(\mathsf{i}), \mathsf{r}'_{\mathsf{j},\ell,\mathsf{i}})^{\Delta}$ sending to the other parties $P_{x_j^{2^{\ell+1}}}(i), r_{j,\ell,i}$ and $P^2_{x_j^{2^{\ell}}}(i), r'_{j,\ell,i}$. All parties verify the committed encryptions.
2. For all $i \in J_R, 1 \leq j \leq s, 0 \leq \ell \leq \alpha_j, 1 \leq i \leq 10kD$ all parties check that $P^2_{x_j^{2^{\ell}}}(i) = (P_{x_j^{2^{\ell}}}(i))^2$.
3. For each $1 \leq j \leq s$ the parties run the LIPEV protocol to interpolate the values $c^{0,j}_{0,\ell}$ from the shares $c^{0,j}_{i,\ell} = (\Delta \cdot P_{x_j^{2^{\ell+1}}}(i), r^{\Delta}_{j,\ell,i})$ and $c^{1,j}_{0,\ell}$ from the shares $c^{1,j}_{i,\ell} = (\Delta \cdot P^2_{x_j^{2^{\ell}}}(i), (r'_{j,\ell,i})^{\Delta})$ and to check that the shares indeed lie on polynomials of degree k and $2k$, respectively, for $1 \leq i \leq 10kD$ and then HEPKPV for the language $L^{\alpha_j}_{=} = \{[c^{0,j}_{0,\ell}, c^{1,j}_{0,\ell}]_{0 \leq \ell \leq \alpha_j}\}$.

Figure 4.6: Preprocessing Verification

- A public parameter $\Delta = 10kD!$ and a security parameter k.

We use the following representation of the polynomial $\mathbf{Q}$:

$$\mathbf{Q}(x_{1,1}, \ldots, x_{1,l_1}, \ldots, x_{m,1}, \ldots, x_{m,l_m}) = \sum_{s=1}^{n} c_s x_{1,1}^{\alpha_{1,1,s}} \ldots x_{1,l_1}^{\alpha_{1,l_1,s}} \ldots x_{m,1}^{\alpha_{m,1,s}} \ldots x_{m,l_m}^{\alpha_{m,l_m,s}}$$

where $\alpha_{j,v,s} = 0$ if $x_{j,v}$ does not participate in the s-th monomial of $\mathbf{Q}$ and c_s values are known coefficients. We define $h_{j,s}(x_{j,1}, \ldots, x_{j,l_j}) = x_{j,1}^{\alpha_{j,1,s}} x_{j,2}^{\alpha_{j,2,s}} \ldots x_{j,l_j}^{\alpha_{j,l_j,s}}$ for $1 \leq j \leq m$, $1 \leq s \leq n$. Thus

$$\mathbf{P}(x_{1,1}, \ldots, x_{1,l_1}, \ldots, x_{m,1}, \ldots, x_{m,l_m}) = \sum_{s=1}^{n} c_s (\prod_{j=1}^{m} h_{j,s}).$$

Alternatively, we view $h_{j,s}$ for $1 \leq j \leq m$, $1 \leq s \leq n$ in the following way:

$$h_{j,s}(x_{j,1}, x_{j,1}^2, \ldots, x_{j,1}^{2^{\lfloor \log \alpha_{j,1,s} \rfloor}}, \ldots, x_{j,l_j}, x_{j,l_j}^2, \ldots, x_{j,l_j}^{2^{\lfloor \log \alpha_{j,l_j,s} \rfloor}})$$

in which each variable is of degree at most one.

A designated party receives the output of the polynomial evaluation while all other parties learn nothing. We denote the designated output receiver by T^*. For $1 \leq s \leq n$, let $D_{h,s} = \sum_{j=1}^{m} deg(h_{j,s})$ and for $1 \leq j \leq m$, let $D_{h,j,s} = k \sum_{v=1}^{j} deg(h_{v,s})$, where $h_{j,s}$ is defined as above as a polynomials with variables of degree at most 1. Let $D = \max_{s=1}^{n} D_{h,s}$.

Protocol Intuition. The protocol consists of four phases: *Input Preprocessing, Round-Table Step, Re-randomization, Verification and Reconstruction.* During the *Input Preprocessing* phase, each party commits to shares of its inputs via the Efficient Preprocessing protocol. In the *Round Table Step* the parties compute the encrypted evaluations of the monomials in $\mathbf{Q}$ in a round-table fashion. Next, in the *Re-Randomization* phase, each party helps re-randomize the output shares. Honest behavior of the parties is checked during the *Verification* step via cut-and-choose. If the verification passes, the parties jointly decrypt the output shares and the output receiver reconstructs the final polynomial evaluation result in the *Reconstruction* phase. We now present the detailed protocol and state our main theorem [2]. In the following, let $E = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ be a threshold encryption scheme that possesses Property 1.

4.5.1 Security analysis

We state the security guarantees of the multivariate polynomial evaluation protocol presented in Section 4.5 in following theorem:

Theorem 6. *If the Decisional Composite Residuosity problem is hard in $\mathbf{Z}^*_{n^2}$, where n is a product of two strong primes, and protocol Π_{poly_eval} is instantiated with the threshold Paillier encryption scheme TP^m_{enc} such that $E = TP^m_{enc}$, then Π_{poly_eval} securely computes the Polynomial Evaluation functionality in the presence of malicious adversaries.*

[2] We note that for each intermediate monomial $h_{j,s}$ passed between the parties in the round-table step, each Party j needs to transmit only $D_{h,j,s} + 1$ shares to Party $j + 1$ since the rest of the shares may be constructed by the receiving party via Lagrange interpolation over committed values. This may yield significant savings in the communication complexity, which we assumed in our discussion in the introduction.

Multiparty Polynomial Evaluation Protocol Π_{poly_eval}

Input: $T_1 \leftarrow \{\overline{X}_1, pk, sk_1, k, \mathbf{Q}\}, \ldots, T_m \leftarrow \{\overline{X}_m, pk, sk_m, k, \mathbf{Q}\}$
Output: $T^* \rightarrow \mathbf{Q}(x_{1,1}, \ldots, x_{1,l_1}, \ldots, x_{m,1}, \ldots, x_{m,l_m}), \{T_1, \ldots, T_m\} \backslash T^* \rightarrow \perp$
Protocol:
Input Preprocessing:

1. For $1 \leq j \leq m$, $1 \leq s \leq n$ Party T_j converts each $h_{j,s}$ from a function over $x_{j,1}, \ldots, x_{j,l_j}$ to a function in which each variable is of degree at most one:

$$h_{j,s}(x_{j,1}, x_{j,1}^2, \ldots, x_{j,1}^{2^{\lfloor \log \alpha_{j,1,s} \rfloor}}, \ldots, x_{j,l_j}, x_{j,l_j}^2, \ldots, x_{j,l_j}^{2^{\lfloor \log \alpha_{j,l_j,s} \rfloor}})$$

   Let $D_{h,j,s} = k \sum_{v=1}^{j} deg(h_{v,s})$.

2. For $1 \leq j \leq m$, Party T_j runs the *Efficient Input Preprocessing* protocol to generate $10kD$ shares for each of its new inputs

$$x_{j,1}, x_{j,1}^2, \ldots, x_{j,1}^{2^{\lfloor \log \alpha_{j,1} \rfloor}}, \ldots, x_{j,l_j}, x_{j,l_j}^2, \ldots, x_{j,l_j}^{2^{\lfloor \log \alpha_{j,l_j} \rfloor}}$$

   where $\alpha_{j,t} = \max_{i=1}^{n} \alpha_{j,t,i}$ for $1 \leq t \leq l_j$ and commits to these shares.

Round-Table Step:

3. For $1 \leq s \leq n$ party T_1 encrypts the polynomial coefficients $b_{0,i} = \mathsf{Enc_{pk}}(\mathsf{c_s}; 1)$.

4. For $1 \leq s \leq n$, T_1 generates $10kD$ numbers that will be used for the encryptions of his shares as follows:

   (a) Generate $D_{h,1,s} + 1$ random values $r_1^{1,s}, \ldots, r_{D_{h,1,s}+1}^{1,s}$.

   (b) Run the protocol for *Generation for Encryption Randomness for Shamir Shares* with inputs $(1, 0), \ldots, (10kD, 0), r_1^{j,s}, \ldots, r_{D_{h,1,s}+1}^{j,s}$ to compute $(r_{D_{h,1,s}+2}^{j,s})^{\Delta}, \ldots, (r_{10kD}^{j,s})^{\Delta}$.

5. For $1 \leq i \leq 10kD$, $1 \leq s \leq n$ T_1 uses the values $r_i^{1,s}$ from the previous step to compute $b_{1,s,i} = b_{0,j}^{\Delta \cdot h_{1,s}(i)} \cdot \mathsf{Encpk}(0; \mathsf{r_i^1})^{\Delta}$, where

$$h_{1,s}(i) = h_{1,s}\big(P_{x_{1,1}}(i), \ldots, P_{x_{1,1}^{2^{\lfloor \log \alpha_{1,1,j} \rfloor}}}(i), \ldots, P_{x_{1,l_1}}(i), \ldots, P_{x_{1,l_1}^{2^{\lfloor \log \alpha_{1,l_1,s} \rfloor}}}(i)\big)$$

   and sends $b_{1,s,i}$ to party T_2.

6. For each $2 \leq j \leq m$:

   (a) T_j receives from T_{j-1} coefficients $b_{j-1,1,i}, \ldots, b_{j-1,n,i}$ for $1 \leq i \leq 10kD$.

Figure 4.7: Polynomial Evaluation

6. (b) For $1 \leq s \leq n$, T_j generates $10kD$ numbers that will be used for the encryptions of his shares as follows:
   i. Generate $D_{h,j,s} + 1$ random numbers $r_1^{j,s}, \ldots, r_{D_{h,j,s}+1}^{j,s}$.
   ii. Run the protocol for *Generation for Encryption Randomness for Shamir Shares* with inputs $(1,0), \ldots, (10kD, 0), r_1^{j,s}, \ldots, r_{D_{h,j,s}+1}^{j,s}$ to compute $(r_{D_{h,j,s}+2}^{j,s})^{\Delta}, \ldots, (r_{10kD}^{j,s})^{\Delta}$.

7. (a) For $1 \leq s \leq n$ T_j uses the values $r_i^{j,s}$ from the previous step to compute $b_{j,s,i} = b_{j-1,s,i}^{\Delta \cdot h_{j,s}(i)} \cdot \mathsf{Enc}_{\mathsf{pk}}(0; \mathsf{r}_{\mathsf{i}}^{\mathsf{j,s}})^{\Delta}$ where

$$h_{j,s}(i) = h_{j,s}(P_{x_{j,1}}(i), \ldots, P_{x_{j,1}^{2^{\lfloor \log \alpha_{j,1} \rfloor}}}(i), \ldots, P_{x_{j,l_j}}(i), \ldots, P_{x_{j,l_j}^{2^{\lfloor \log \alpha_{j,l_j} \rfloor}}}(i))$$

   and $h_{j,s}(i)$ denotes evaluation of $h_{j,s}$ on the i-th shares of its inputs .

   (b) If $j < m$, T_j sends all $b_{j,s,i}$ to P_{j+1}.

   (c) If $j = m$, for each $1 \leq i \leq 10kD$ T_m computes $S_i' = \Pi_{s=1}^{n} b_{m,s,i}$ and sends them to all parties on the broadcast channel.

Re-Randomization Step:

8. For $1 \leq j \leq m$, Party T_j computes polynomial $P_{j,0}$ of degrees kD such that $P_{j,0}(0) = 0$.

9. For $1 \leq j \leq m$, Party T_j generates $10kD$ values that it will used for share encryptions as follows:
   (a) Generate $kD + 1$ random values $r_{j,1}, \ldots, r_{j,kD+1}$.
   (b) the protocol for *Generation for Encryption Randomness for Shamir Shares* with inputs $(1, P_{j,0}(1)), \ldots, (10kD, P_{j,0}(10kD)), r_{j,1}, \ldots, r_{j,kD+1}$ to compute $r_{j,kD+2}^{\Delta}, \ldots, r_{j,10kD}^{\Delta}$.

10. T_j uses the $r_{j,i}$ values computed in the previous step to commit to shares $Z_{j,0} = \mathsf{Enc}_{\mathsf{pk}}(\mathsf{P_{j,0}(i)}; \mathsf{r_{j,i}})^{\Delta}$ for $1 \leq i \leq 10kD$

11. All parties run the *LIPEV* and *HEPKPV* protocols in parallel to ensure that each vector $[Z_{j,i}]_{1 \leq i \leq 10kD}$ is an encryption of a polynomial with constant coefficient 0.

12. The final encryptions are calculated as $S_i = S_i' \cdot \Pi_{j=1}^{m} Z_{j,0}$ for $1 \leq i \leq 10kD$.

Verification:

13. All parties verify independently using the *LIPEV* protocol that the encryptions S_i lie on a polynomial of degree kD. Otherwise reject.

Figure 4.8: Polynomial Evaluation Continued

14. All parties run a multi-party coin-tossing protocol (Figure 4.4) to choose a random subset I of size k from $[1, 10kD]$.

15. For each $i \in I$ parties $T_1, \ldots, T_m$ decommit their corresponding shares from the Efficient Input Preprocessing.

16. All parties run the Preprocessing Verification for their inputs.

17. For each $i \in I$ each party T_j decommits the i-th shares of its inputs as well as the i-th share of the polynomials $P_{j,0}$. Additionally, each party T_j reveals the randomness $r_i^{j,s}$ for $1 \leq s \leq n$ and $r_{j,i}$ used for the corresponding shares. To verify, each party recomputes the entire share S_i^*, using the inputs and randomness revealed and checks that $S_i = S_i^*$. If any verification fails the protocol is aborted.

Reconstruction:

18. For each $1 \leq i \leq 10kD$ each party computes its partial decryption $s_{i,j}$ of S_i and sends it to the designated output receiver T^*.

19. Party T^* uses the partial decryptions $s_{i,j}$ for $1 \leq j \leq m$ received in the previous step to completely decrypt S_i. T^* reconstructs the value of $\mathbf{Q}(x_{1,1}, \ldots, x_{1,l_1}, \ldots, x_{m,1}, \ldots, x_{m,l_m})$ using interpolation and division by Δ^m.

Figure 4.9: Polynomial Evaluation Continued

Proof. The correctness of the protocol in the case that all parties are honest follows from Property 2, Property 3, and the correctness of the building-block protocols.

Proof Intuition. We start with a sketch that gives the intuition of the proof of security for the protocol. Assume there is a fixed set B, $|B| \leq m$, chosen at the outset of the protocol and that a non-uniform probabilistic polynomial-time real adversary A_B controls the parties T_j such that $j \in B$. We construct a non-uniform probabilistic expected polynomial-time ideal model adversary simulator S_B.

The idea behind how S_B works is that it extracts the inputs of the parties T_j such that $j \in B$. using the extractor for the HEPKPV protocol. Additionally, S_B simulates the messages of the honest parties T_i, for $i \in [m] \setminus B$ using dummy inputs and outputs.

When proving knowledge and validity of the parties' inputs, S_B uses the simulator for the HEPKPV protocol. At the outset of the protocol, S_B chooses a random subset I' of size k such that $I' \subset [10kD]$. When committing to the secret-sharing of the honest parties, it places random values in the positions indexed by I'. S_B computes correctly all calculations that will be verified in the cut-and-choose step for elements in the subset I'. Then, S_C uses the simulator for the Coin-Tossing protocol to guarantee that the outcome of the Coin-Tossing is $I = I'$. After the cut-and-choose step and all verifications have executed without aborting, the simulator calls the Trusted Party with the inputs it has previously extracted and receives the final output of the protocol. Note that the Simulator has the secret keys of the malicious parties and thus is able to find out their partial decryption of the dummy ciphertext. To ensure that the final output sent to party receiving output is correct, the Simulator uses the share reconstruction protocol for threshold Paillier [Fouque *et al.*, 2001] to reconstruct shares for the honest parties so that the final reconstructed decryption is equal to the output of the trusted party. S_B uses the simulator for the threshold decryption scheme to simulate the proof that the shares of the honest parties were decrypted correctly.

Intuitively, because the Simulator is able to choose the set $I = I'$ ahead of time, he can run the protocol using the challenge ciphertext from a CPA-IND experiment as the constant coefficient of the shared inputs of all uncorrupted parties, thereby reducing indistinguishability of the views to the semantic security of the encryption scheme TP^m_{enc}. Therefore, we have that A_B's view is indistinguishable in the Ideal Model when interacting with a Simulator that chooses all 0 values as the constant coefficients of the shared inputs of the uncorrupted parties and its view in the Real model when each uncorrupted party uses its actual input. This is due to the information-theoretic secrecy of the secret-sharing scheme and the semantic security of the encryption scheme. The detailed descriptions if the simulators and the distinguishability of the views are given in the full version of the paper.

Full Proof. We now describe in detail the Simulator S_B

- S_B chooses a random subset $I' \subset [10kD]$ of size k, $I' = \{j_1, \ldots, j_k\}$.
- For each of the corrupt parties, T_j for $j \in B$, S_B extracts the input of party T_j using the extractor for the HEPKPV protocol.

- **Efficient Preprocessing:** For each of the Honest Parties, T_j for $j \notin B$, S_B simulates the efficient preprocessing step as described below.

  1. S_B chooses a random value $r_{i,v,w}$ and sets $P_{x_{j,v}^{2^i}}(w) = r_{i,v,w}$ for $1 \leq v \leq l_j, 0 \leq i \leq \alpha_{j,v}, w \in I'$. S_B computes a random encryption $\mathsf{Enc_{pk}}(\mathsf{P}_{\mathsf{x_{j,v}^{2^i}}}(\mathsf{w}))$.
  2. S_B chooses a random subset $M \subseteq [10kD] \setminus I'$ of size k.
  3. S_B sets $P_{x_{j,v}^{2^i}}(0) = 0$ for $1 \leq v \leq l_j, 0 \leq i \leq \alpha_{j,v}$ and computes a random indexed encryption $(0, \mathsf{Enc_{pk}}(\mathsf{P}_{\mathsf{x_{j,v}^{2^i}}}(\mathsf{0})))$.
  4. S_B uses Lagrange interpolation over encrypted values and the above encryptions to compute the indexed encryptions $(w, \mathsf{Enc_{pk}}(\mathsf{P}_{\mathsf{x_{j,v}^{2^i}}}(\mathsf{w}))^{\Delta})$ for $1 \leq v \leq l_j, 0 \leq i \leq \alpha_{j,v}, w \in [10kD]$.
  5. S_B sets $P^2_{x_{j,v}^{2^i}}(w) = (P_{x_{j,v}^{2^i}}(w))^2$ for $1 \leq v \leq l_j, 0 \leq i \leq D_{x_{j,v}} - 1, w \in I'$ and computes a random indexed encryption $(w, \mathsf{Enc_{pk}}((\mathsf{P}_{\mathsf{x_{j,v}^{2^i}}}(\mathsf{w}))^2))$.
  6. S_B sets $P^2_{x_{j,v}^{2^i}}(w) = 0$ for $1 \leq v \leq l_j, 0 \leq i \leq \alpha_{j,v} - 1, w = 0, w \in M$ and computes a random indexed encryption $(w, \mathsf{Enc_{pk}}((\mathsf{P}_{\mathsf{x_{j,v}^{2^i}}}(\mathsf{w}))^2))$.
  7. S_B uses Lagrange interpolation over encrypted values and the above encryptions to compute the indexed encryptions $(w, \mathsf{Enc_{pk}}((\mathsf{P}_{\mathsf{x_{j,v}^{2^i}}}(\mathsf{w}))^2)^{\Delta})$ for $1 \leq v \leq l_j, 0 \leq i \leq \alpha_{j,v}, w \in [10kD]$.
  8. S_B broadcasts the encryptions computed above.

- **Round-Table Step:** For each of the Honest Parties, T_j for $j \notin B$, S_B simulates the round-table step as described below:

  1. For $1 \leq s \leq n, i \in I'$, S_B computes the indexed encryption $(i, b'_{j,s,i}) = (i, b_{j-1,s,i}^{h_{j,s}(i)} \cdot \mathsf{Enc_{pk}}(\mathsf{0}; \mathsf{r_i^{j,s}}))$ honestly using the values committed above and a randomly chosen $r_i^{j,s}$.
  2. For $1 \leq s \leq n$, S_B chooses a set $N_s \subseteq [10kD] \setminus I'$ of size $D_{h,j,s} - k$.
  3. For each s, $1 \leq s \leq n$, S_B computes the random indexed encryptions $(i, \mathsf{Enc_{pk}}(\mathsf{0}))$ for $i = 0, i \in N_s$.
  4. For $1 \leq s \leq n$, S_B uses Lagrange interpolation over encrypted values and the above encryptions to compute $(i, (b'_{j,s,i})^{\Delta}) = (i, b_{j,s,i})$ for $i \in [10kD]$.

- For each of the Honest Parties, T_j for $j \notin B$, S_B follows steps $5(d)$ and $5(e)$ of the protocol.
- **Re-Randomization Step:** For each of the Honest Parties, S_B chooses the encrypted shares of $P_{j,0}$ in the following way:
  1. For all but one of the honest parties: S_B chooses the random indexed encryptions of 0, $(i, \mathsf{Enc}_{\mathsf{pk}}(0))$ for $1 \leq i \leq kD+1$.
  2. S_B uses Lagrange interpolation over encrypted values to compute the rest of the indexed encryptions (raised to Δ).
  3. For the final honest party P_h: S_B chooses a random degree kD polynomial $P'_{h,0}$ and computes the following random indexed encryptions: $E_i = (i, \mathsf{Enc}_{\mathsf{pk}}(\mathsf{P}'_{\mathsf{h},0}(\mathsf{i})))$ for $1 \leq i \leq kD+1$.
  4. S_B uses Lagrange interpolation over encrypted values to compute the rest of the indexed encryptions (raised to Δ).
  5. For $1 \leq i \leq 10kD$, S_B calculates the shares $E_i \cdot S'_i$ and uses these as the encrypted shares of $P_{h,0}$.
  6. S_B uses the HEPKPV protocol to prove that all the 0-shares were computed correctly.
- S_B plays the role of the honest parties as specified in steps $10, 11$ of the protocol.
- **Verification:**
  1. S_B simulates a run of the Coin-Tossing protocol to ensure the outcome is the set I' using the simulator for the Coin-Tossing protocol.
  2. S_B plays the role of the honest parties as specified in steps $14, 15, 16$.
- **Reconstruction:**
  1. If all the corrupt parties pass the cut-and-choose step, S_B queries the Trusted Party with the previously extracted inputs and receives back Out.
  2. S_B now shares Out via a random degree kD polynomial P_{Out} such that $P_{Out}(0) = Out \cdot \Delta^m$.

3. For each of the Corrupt Parties, T_j for $j \in B$, S_B calculates the partial decryption of $S_{j,i}$, obtaining $s_{j,i}$, using T_j's secret key.
4. S_B uses the share reconstruction protocol for threshold Paillier to construct simulated shares $s_{j,i}$ for each value $P_{Out}^{sim}(i)$ for each of the Honest Parties and follows step 1 of the protocol honestly using the simulated shares.
5. For each of the Honest Parties, T_j for $j \notin B$, S_B uses the simulator for the threshold encryption scheme to simulate the proof that s_j is the correct decryption of S_j.

We show that the view of A_B in the simulation with S_B is indistinguishable from its view in a real execution.

In the case that one of the parties controlled by A_B does not answer the challenges in the Efficient Preprocessing protocol correctly, the output distributions are computationally indistinguishable in the real and ideal executions. In the case that all the parties controlled by A_B do answer the challenges correctly, the probability that S_B fails to extract the parties' inputs is negligible.

We now consider the case that A_B answers the challenge correctly and S_B successfully extracts the parties' inputs. We consider the following hybrid distributions: The first hybrid distribution consists of the joint ouput of A_B and the honest parties in a real execution of Π_{poly_eval}. The second hybrid distribution is the same as the first, except instead of a real execution of Π_{poly_eval}, the proofs of correct decryption of the threshold encryption scheme are replaced with simulated proofs for each of the honest parties. The third hybrid distribution is the same as the second, except we additionally replace the real Coin-Tossing protocol with a simulation. The fourth hybrid distribution is the same as the third, except for each honest party T_j, we replace the real Efficient Preprocessing protocol and computation step with a simulation (as above). The fifth hybrid distribution is the same as the fourth, except we replace the computation of the Re-Randomization step with the one given above. The sixth hybrid distribution is the same as the fifth, except we replace all intermediate outputs of the honest parties' the encryptions described above in the simulator's execution of the Round-Table step. The seventh hybrid distribution is the same as the sixth, except we replace all committed input shares of the honest parties' as described above in the simu-

lator's execution of the Efficient Preprocessing step. The eighth hybrid distribution is the same as the seventh, except we replace the final decrypted output by the simulated output as described above. The eighth hybrid distribution is distributed identically to the joint output of S_B and the honest parties in an Ideal execution.

Indistinguishability follows due to the indistinguishability of the simulated runs of the Proof of Correct Decryption, Coin-Tossing protocols, and Efficient Preprocessing protocols from real runs, the information-theoretic security of the secret-sharing scheme, and the semantic security of the threshold encryption scheme TP^m_{enc}.

We give some intuition for the indistinguishability of the seventh and eighth hybrid distribution. Given that the honest parties do not abort after the input preprocessing verification and cut-and-choose step, we have that with all but negligible probability, at least a .9-fraction of the final encrypted output shares were computed correctly. If the honest parties do not abort in the LIPEV protocol, then the decryptions of the output shares must all lie on some degree kD polynomial. Since any 2 degree kD polynomials must disagree on at least a .9-fraction of the $10kD$ shares, we must have that the output shares are actually all exactly correct. Due to the re-randomization steps, this degree kD polynomial is distributed uniformly conditioned on the fact that its constant coefficient is equal to the final output value.

In the case that party T^* is an honest party, we show that the joint output distribution of A_B and P^* in the Real model is indistinguishable from the joint output distribution of S_B and P^* in the Ideal model. Similarly to the analysis above, we note that if the honest party reaches the end of the protocol, then with all but negligible probability all the shares were computed exactly correctly. Thus, when the honest party in the Real model reconstructs the final output, it will be the same value outputted by the honest party in the Ideal model.

We note that the Simulator exactly as specified does not run in expected polynomial time. This is a subtlety that concerns the probability that the adversary will open the commitment in the coin-tossing protocol correctly after rewinding, since after rewinding, the commitment is no longer random, but is correlated with the input; all indices in I (the set chosen by the coin-tossing protocol) are computed honestly, while all indices not in I are incorrect. This technicality was pointed out by [Goldreich and Kahan, 1996] in their

work on constant-round zero knowledge. However, as in [Lindell, 2008] we slightly modify the simulator so it will run in expected polynomial time. Very briefly, the simulator first estimates p, the probability that all parties controlled by A_B decommit correctly when the commitment they received was randomly chosen, and then bounds the number of rewinding attempts by $poly(k)p$, where $poly(k)$ is a fixed polynomial.

□

4.6 Communication and Computation Complexity

Our protocol computes the polynomial functionality in a constant number of rounds (counting round-table rounds as one, or with a constant number of players). The communication complexity of the protocol can be divided into two types: messages that are broadcast to all parties and the "round-table" communication that is passed between two consecutive parties. We note that the "round-table" communication can be done off-line. The broadcast communication consists of the commitments of the inputs shares, the decommitments used in the final verification phase, the encrypted and decrypted output shares as well as the messages used in the coin tossing and HEPKPV protocols. These messages add up to $k(10D+1)(\sum_{j=1}^{m}\sum_{t=1}^{l_j}\log\alpha_{j,t}+1)$. Note that the communication complexity may be much smaller than the size of the polynomial representation. For example, if party P_j with input $x_{j,1}$ must contribute $\alpha_{j,t}$ consecutive powers of x_i: $x_i^1,\ldots,x_i^{\alpha_{j,t}}$ to $\alpha_{j,t}$ different terms, the broadcast communication complexity for this party will still only be $k(10D+1)\log\alpha_{j,t}+1$. The round-table messages passed between consecutive parties include all intermediate messages in the computation that are sent by the all the parties except the last one, which in total are $10kDn(m-1)$. The computational complexity (where we count number of exponentiations) for all m parties in total is $O(kDnm)$. Further, if we apply the share packing optimization from Section 4.7.1 over k executions of the protocol, we can drop k factor for the new amortized complexities.

4.7 Multiparty Set Intersection and Other Applications

4.7.1 Optimizations

We apply several optimizations to the protocol given in Section 4.5 for polynomials with specific structures. First, if we have a monomial that is computed only from the inputs of a subset of the parties, then clearly, we can evaluate it in a round-table fashion that only includes parties in this subset and proceed to the Re-Randomization Step.

Additionally, in some cases, we can remove the requirement of a party to share all of its inputs. Recall that we require the input-sharing in order to enable the cut-and-choose verification of honest behavior of the parties. In the case when an input is used only once in the polynomial, this type of proof may not be necessary. We can avoid sharing an input if it belongs to the first party in the round table computation of the corresponding monomial as long as we can verify that the encryption itself is valid with a ZKPOK and extract the encrypted value. We notice that the requirements imposed on the structure of the polynomial in order to be able to apply this optimization substantially limit the range of the possible polynomials. However, in the next section we will see how the problem of multiparty set intersection can be reduced to the evaluation of exactly this type of polynomials.

Finally, we use the approach of multi-secret sharing from [Franklin and Yung, 1992] that allows us to use the same polynomials to share the input values for multiple parallel executions of the protocol, which lowers the amortized communication complexity of our protocol. Intuitively, we choose a set of points on the sharing polynomials to represent the input values for each of the different executions of the protocol, say points 1 to k for each of k different executions. The shares that will be used in the computation will be those corresponding to points not in this set. As a result, the final output polynomial will evaluate to each of the different output values corresponding to each execution at the points 1 to k respectively.

4.7.2 Multiparty Set Intersection as Polynomial Evaluation

The setting for the multiparty set intersection problem that we consider is as follows: there are m parties $T_1, \ldots, T_m$ who have input sets $X_1, \ldots, X_m$ and wish to jointly compute $X_1 \bigcap \ldots \bigcap X_m$; one of the parties T_m is the designated output receiver that will learn the elements in the set intersection [3]. The first step is our approach is to translate the problem into a problem for secure multivariate polynomial evaluation that depends on the inputs of all participants. Recall that a set $X = \{x_1, \ldots, x_d\}$ can be represented as a polynomial $P(x) = (x - x_1) \ldots (x - x_d)$. Now if we consider the polynomial $P'(x) = r \cdot P(x) + x$, where r is random, we have that if $x' \in X$ then $P'(x') = x'$ and if $x' \notin X$ then $P'(x')$ is uniformly distributed (see [Freedman *et al.*, 2004]). In the multiparty case we have m parties with input sets $X_1, \ldots, X_m$, represented by polynomials $P_{X_1}(x), \ldots, P_{X_m}(x)$. Thus the polynomial $\mathbf{P}(x) = \mathbf{r} \cdot \sum_{i=1}^{m-1} P_{X_i}(x) + x$, where $\mathbf{r} = r_1 + r_2 + \cdots + r_m$ and each r_i is a randomly chosen input contributed by Party i, will have the same property mentioned above: if $x' \in X_1 \bigcap \ldots \bigcap X_m$ then $\mathbf{P}(x') = x'$ and if $x' \notin X_1 \bigcap \ldots \bigcap X_m$ then $\mathbf{P}(x')$ is uniformly distributed. Now to compute the intersection of the sets $X_1, \ldots, X_m$, the parties $T_1, \ldots, T_{m-1}$ would construct the multivariate polynomial that represents the intersection of their sets as above and the parties will run the polynomial evaluation algorithm to evaluate this polynomial on all inputs of T_m. We treat the multivariate polynomial constructed by $T_1, \ldots, T_{m-1}$ as polynomial $\mathbf{P'}$ that has coefficients 1 and its real coefficients being input variables for the parties.

Applying the optimizations described in Section 4.7.1 we obtain or final protocol for computing multiparty set intersection for which we have the following theorem and complexities:

Theorem 7. *If the Decisional Composite Residuosity problem is hard in $\mathbf{Z}^*_{n^2}$, where n is a product of two strong primes, protocol Π_{poly_eval} is instantiated with the threshold Paillier encryption scheme TP^m_{enc} such that $E = TP^m_{enc}$, and $\mathbf{Q} = \mathbf{P'}$, then Π_{poly_eval} securely computes the Set Intersection functionality [4] in the presence of malicious adversaries.*

[3] We note that our protocol can be generalized to allow any subset of the parties to receive output.

[4] We consider here a slight variant of the Set Intersection functionality where Party i for $1 \leq i \leq m - 1$

We have that the broadcast communication complexity of the Set Intersection protocol is $O(md+d\log^2 d)$ (there is no round-table communication) and the computational complexity is $O(md^2 \log d)$, where $d >> k$ is the maximum input set size of each party.

Multiparty Oblivious Polynomial Evaluation. In the protocol for multiparty set intersection the coefficients of the evaluated polynomial are inputs for different parties. If we assume that the polynomial coefficients are the inputs of one of the parties, we reduce the problem to oblivious multivariate polynomial evaluation (introduced by [Naor and Pinkas, 2006] in the single-variable case) for a small class of multivariate polynomials.

---

submits the polynomial P_{X_i} to the Trusted Party, Party m submits X_m and the Trusted Party returns the intersection of $X_1, \ldots, X_m$. In order to compute the standard Set Intersection functionality, we must use the threshold El Gamal encryption scheme.

Part II

Outsourced Computation

Chapter 5

How to Delegate and Verify in Public: Verifiable Computation from Attribute-based Encryption

5.1 Motivation and Contributions

In the modern age of cloud computing and smartphones, asymmetry in computing power seems to be the norm. Computationally weak devices such as smartphones gather information, and when they need to store the voluminous data they collect or perform expensive computations on their data, they outsource the storage and computation to a large and powerful server (a "cloud", in modern parlance). Typically, the clients have a pay-per-use arrangement with the cloud, where the cloud charges the client proportional to the "effort" involved in the computation.

One of the main security issues that arises in this setting is – *how can the clients trust that the cloud performed the computation correctly?* After all, the cloud has the financial incentive to run (occasionally, perhaps) an extremely fast but incorrect computation, freeing up valuable compute time for other transactions. Is there a way to *verifiably outsource* computations, where the client can, without much computational effort, check the correctness of the results provided by the cloud? Furthermore, can this be done without requiring much

interaction between the client and the cloud? This is the problem of *non-interactive verifiable computation*, which was considered implicitly in the early work on efficient arguments by Kilian [Kilian, 1995] and computationally sound proofs (CS proofs) by Micali [Micali, 1994], and which has been the subject of much attention lately [Goldwasser *et al.*, 2008; Gennaro *et al.*, 2010; Chung *et al.*, 2010; Applebaum *et al.*, 2010; Benabbas *et al.*, 2011; Barbosa and Farshim, 2011; Bitansky *et al.*, 2011; Goldwasser *et al.*, 2011].

Our starting point is that while the recent solutions consider and solve the bare-bones verifiable computation problem in its simplest form, there are a number of desirable features that they fail to achieve. We consider two such properties – namely, *public delegatability* and *public verifiability*.

Public Delegatability. In a nutshell, public delegatability says that everyone should be able to delegate computations to the cloud. In some protocols [Gennaro *et al.*, 2010; Chung *et al.*, 2010; Applebaum *et al.*, 2010; Benabbas *et al.*, 2011], a client who wishes to delegate computation of a function F is required to first run an expensive pre-processing phase (wherein her computation is linear in the size of the circuit for F) to generate a (small) secret key SK_F and a (large) evaluation key EK_F. This large initial cost is then amortized over multiple executions of the protocol to compute $F(x_i)$ for different inputs x_i, but the client needs the secret key SK_F in order to initiate each such execution. In other words, *clients can delegate computation to the cloud only if they put in a large initial computational investment*. This makes sense only if the client wishes to run the same computation on many different inputs. Can clients delegate computation without making such a large initial commitment of resources?

As an example of a scenario where this might come in handy, consider a clinic with a doctor and a number of lab assistants, which wishes to delegate the computation of a certain expensive data analysis function F to a cloud service. Although the doctor determines the structure and specifics of F, it is in reality the lab assistants who come up with inputs to the function and perform the delegation. In this scenario, we would like to ask the doctor to run the (expensive) pre-processing phase once and for all, and generate a (small) *public key* PK_F and an evaluation key EK_F. The public key lets anyone, including the lab assistants,

delegate the computation of F to the cloud and verify the results. Thus, once the doctor makes the initial investment, any of the lab assistants can delegate computations to the cloud without the slightest involvement of the doctor. Needless to say, the cloud should not be able to cheat even given PK_F and EK_F.

Public Verifiability. In a similar vein, the delegator should be able to produce a (public) "verification key" that enables anyone to check the cloud's work. In the context of the example above, when the lab assistants delegate a computation on input x, they can also produce a verification key VK_x that will let the patients, for example, obtain the answer from the cloud and check its correctness. Neither the lab assistants nor the doctor need to be involved in the verification process. Needless to say, the cloud cannot cheat even if it knows the verification key VK_x.

Put together, we call a verifiable computation protocol that is both publicly delegatable and publicly verifiable a *public verifiable computation* protocol. We are not aware of any such protocol (for a general class of functions) that is non-interactive and secure in the standard model. Note that we still require the party who performs the initial function preprocessing (the doctor in the example above) to be trusted by those delegating inputs and verifying outputs.

As a bonus, a public verifiable computation protocol is immune to the "rejection problem" that affects several previous constructions [Gennaro *et al.*, 2010; Chung *et al.*, 2010; Applebaum *et al.*, 2010]. Essentially, the problem is that these protocols do not provide reusable soundness; i.e., a malicious cloud that is able to observe the result of the verification procedure (namely, the accept/reject decision) on polynomially many inputs can eventually break the soundness of the protocol. It is an easy observation that public verifiable computation protocols do not suffer from the rejection problem. Roughly speaking, verification in such protocols depends only on the public key and some (instance-specific) randomness generated by the delegator, and not on any long-term secret state. Thus, obtaining the result of the verification procedure on one instance does not help break the soundness on a different instance.[1]

[1]In fact, this observation applies also to any protocol that is publicly delegatable and not necessarily

5.1.1 Our Contributions

Verifiable Computation from Attribute-Based Encryption. Our main result is a (somewhat surprising) connection between the notions of attribute-based encryption (ABE) and verifiable computation (VC). In a nutshell, we show that a *public* verifiable computation protocol for a class of functions $\mathcal{F}$ can be constructed from any attribute-based encryption scheme for a related class of functions – namely, $\mathcal{F} \cup \overline{\mathcal{F}}$. Recall that attribute-based encryption (ABE) [Sahai and Waters, 2005; Goyal *et al.*, 2006] is a rich class of encryption schemes where secret keys $\mathsf{ABE.SK}_F$ are associated with functions F, and can decrypt ciphertexts that encrypt a message m under an "attribute" x if and only if $F(x) = 1$.

For simplicity, we state all our results for the case of Boolean *functions*, namely functions with one-bit output. For functions with many output bits, we simply run independent copies of the verifiable computation protocol for each output bit.

Theorem 8 (Main Theorem, Informal)**.** *Let $\mathcal{F}$ be a class of Boolean functions, and let $\overline{\mathcal{F}} = \{\overline{F} \mid F \in \mathcal{F}\}$ where $\bar{F}$ denotes the complement of the function F. If there is a key-policy ABE scheme for $\mathcal{F} \cup \overline{\mathcal{F}}$, then there is a public verifiable computation protocol for $\mathcal{F}$.*

Some remarks about this theorem are in order.

1. First, our construction is in the pre-processing model, where we aim to outsource the computation of the same function F on polynomially many inputs x_i with the goal of achieving an amortized notion of efficiency. This is the same as the notion considered in [Gennaro *et al.*, 2010; Chung *et al.*, 2010], and different from the one in [Goldwasser *et al.*, 2008]. See Definition 6.

2. Secondly, since the motivation for verifiable computation is outsourcing computational effort, efficiency for the client is obviously a key concern. Our protocol will be efficient for the client, as long as computing an ABE encryption (on input a message m and attribute x) takes less time than evaluating the function F on x. We will further address the efficiency issue in the context of concrete instantiations below (as well as in Section 5.3.2).

---

publicly verifiable.

3. Third, we only need a *weak* form of security for attribute-based encryption which we will refer to as *one-key security*. Roughly speaking, this requires that an adversary, given a single key $\mathsf{ABE.SK}_F$ for any function F of its choice, cannot break the semantic security of a ciphertext under any attribute x such that $F(x) = 0$. Much research effort on ABE has been dedicated to achieving the much stronger form of security against collusion, namely when the adversary obtains secret keys for not just one function, but polynomially many functions of its choice. We will not require the strength of these results for our purposes. On the same note, constructing one-key secure ABE schemes is likely to be much easier than full-fledged ABE schemes.

We consider attribute-based encryption (ABE) schemes to be ones in which each secret key $\mathsf{ABE.SK}_F$ is associated with a function F. We provide further discussion about the difference and the parallel between the notions of attribute-based encryption and predicate encryption in Appendix C.1.

Let us now describe an outline of our construction. The core idea of our construction is simple: attribute-based encryption schemes naturally provide a way to "prove" that $F(x) = 1$. Say the server is given the secret key $\mathsf{ABE.SK}_F$ for a function F, and a ciphertext that encrypts a *random message* m under the attribute x. The server will succeed in decrypting the ciphertext and recovering m if and only if $F(x) = 1$. If $F(x) = 0$, he fares no better at finding the message than a random guess. The server can then prove that $F(x) = 1$ by returning the decrypted message.

More precisely, this gives an effective way for the server to convince the client that $F(x) = 1$. The pre-processing phase for the function F generates a master public key $\mathsf{ABE.MPK}$ for the ABE scheme (which acts as the public key for the verifiable computation protocol) and the secret key $\mathsf{ABE.SK}_F$ for the function F (which acts as the evaluation key for the verifiable computation protocol). Given the public key and an input x, the delegator encrypts a random message m under the attribute x and sends it to the server. If $F(x) = 1$, the server manages to decrypt and return m, but otherwise, he returns $\perp$. Now,

- If the client gets back the same message that she encrypted, she is convinced beyond doubt that $F(x) = 1$. This is because, if $F(x)$ were 0, the server could not have found

m (except with negligible probability, assuming the message is long enough).

- However, if she receives no answer from the server, it could have been because $F(x) = 0$ and the server is truly unable to decrypt, or because $F(x) = 1$ but the server intentionally refuses to decrypt.

Thus, we have a protocol with one-sided error – if $F(x) = 0$, the server can never cheat, but if $F(x) = 1$, he can.

A verifiable computation protocol with no error can be obtained from this by two independent repetitions of the above protocol – once for the function F and once for its complement $\bar{F}$. A verifiable computation protocol for functions with many output bits can be obtained by repeating the one-bit protocol above for each of the output bits. Intuitively, since the preprocessing phase does not create any secret state, the protocol provides public verifiable computation. Furthermore, the verifier performs as much computation as is required to compute two ABE encryptions.

Perspective: Signatures on Computation. Just as digital signatures authenticate messages, the server's proof in a non-interactive verifiable computation protocol can be viewed as a "signature on computation", namely a way to authenticate that the computation was performed correctly. Moni Naor has observed that identity-based encryption schemes give us digital signature schemes, rather directly [Boneh and Franklin, 2003]. Given our perspective, one way to view our result is as a logical extension of Naor's observation to say that just as IBE schemes give us digital signatures, ABE schemes give us signatures on computation or, in other words, non-interactive verifiable computation schemes.

Multi-Function Verifiability and ABE with Outsourcing. The definition of verifiable computation focuses on the evaluation of a single function over multiple inputs. In many constructions [Gennaro *et al.*, 2010; Chung *et al.*, 2010; Benabbas *et al.*, 2011] the evaluated function is embedded in the parameters for the VC scheme that are used for the input processing for the computation. Thus evaluations of multiple functions on the same input would require repeated invocation for the ProbGen algorithm. A notable difference are approaches based on PCPs [Goldwasser *et al.*, 2008; Bitansky *et al.*, 2011;

Goldwasser *et al.*, 2011] that may require a single offline stage for input processing and then allow multiple function evaluations. However, such approaches inherently require verification work proportional to the depth of the circuit, which is at least logarithmic in the size of the function and for some functions can be also proportional to the size of the circuit. Further these approaches employ either fully homomorphic encryption or private information retrieval schemes to achieve their security properties.

Using the recently introduced definition of ABE with outsourcing [Green *et al.*, 2011], we achieve a multi-function verifiable computation scheme that decouples the evaluated function from the parameters of the scheme necessary for the input preparation. This VC scheme provides separate algorithms for input and function preparation, which subsequently can be combined for multiple evaluations. When instantiated with an existing ABE scheme with outsourcing [Green *et al.*, 2011], the verification algorithm for the scheme is very efficient: its complexity is linear in the output size but independent of the input length and the complexity of the computation. Multi-function VC provides significant efficiency improvements whenever multiple functions are evaluated on the same input, since a traditional VC scheme would need to invoke ProbGen for every function.

Attribute-Based Encryption from Verifiable Computation. We also consider the opposite direction of the ABE-VC relation: can we construct an ABE scheme from a VC scheme? We are able to show how to construct an ABE scheme from a *very special* class of VC schemes with a particular structure in Appendix C.2. Unfortunately, this does not seem to result in any new ABE constructions.

5.2 Definitions

5.2.1 Public Verifiable Computation

We propose two new properties of verifiable computation schemes, namely

- *Public Delegation*, which allows arbitrary parties to submit inputs for delegation, and
- *Public Verifiability*, which allows arbitrary parties (and not just the delegator) to verify the correctness of the results returned by the worker.

Together, a verifiable computation protocol that satisfies both properties is called a *public verifiable computation* protocol. The following definition captures these two properties.

Definition 6 (Public Verifiable Computation)**.** *A* public verifiable computation scheme (with preprocessing) $\mathcal{VC}$ *is a four-tuple of polynomial-time algorithms* $(\mathsf{KeyGen}, \mathsf{ProbGen}, \mathsf{Compute}, \mathsf{Verify})$ *which work as follows:*

- $(PK_F, EK_F) \leftarrow \mathsf{KeyGen}(F, 1^\lambda)$*: The randomized* key generation *algorithm takes as input a security parameter* λ *and the function* F*, and outputs a public key* PK_F *and an evaluation key* EK_F*.*

- $(\sigma_x, VK_x) \leftarrow \mathsf{ProbGen}(PK_F, x)$*: The randomized* problem generation *algorithm uses the public key* PK_F *to encode an input* x *into public values* σ_x *and* VK_x*. The value* σ_x *is given to the worker to compute with, whereas* VK_x *is made public, and later used for verification.*

- $\sigma_{\mathsf{out}} \leftarrow \mathsf{Compute}(EK_F, \sigma_x)$*: The deterministic* worker *algorithm uses the evaluation key* EK_F *together with the value* σ_x *to compute a value* σ_{out}*.*

- $y \leftarrow \mathsf{Verify}(VK_x, \sigma_{\mathsf{out}})$*: The deterministic* verification *algorithm uses the verification key* VK_x *and the worker's output* σ_{out} *to compute a string* $y \in \{0,1\}^* \cup \{\perp\}$*. Here, the special symbol* $\perp$ *signifies that the verification algorithm rejects the worker's answer* σ_{out}*.*

A number of remarks on the definition are in order.

First, in some instantiations, the size of the public key (but not the evaluation key) will be independent of the function F, whereas in others, both the public key and the evaluation key will be as long as the description length of F. For full generality, we refrain from making the length of the public key a part of the syntactic requirement of a verifiable computation protocol, and instead rely on the definition of efficiency to enforce this (see Definition 9 below).

Secondly, our definition can be viewed as a "public-key version" of the earlier VC definition [Gennaro *et al.*, 2010; Chung *et al.*, 2010]. In the earlier definition, KeyGen produces

a secret key that was used as an input to ProbGen and, in turn, ProbGen produces a secret verification value needed for Verify (neither of these can be shared with the worker without losing security). Indeed, the "secret-key" nature of these definitions means that the schemes could be attacked given just oracle access to the verification function (and indeed, there are concrete attacks of this nature against the schemes in [Chung *et al.*, 2010; Gennaro *et al.*, 2010; Applebaum *et al.*, 2010]). Our definition, in contrast, is stronger in that it allows any party holding the public key PK_F to delegate and verify computation of the function F on any input x, even if the party who originally ran ProbGen is no longer online. This, in turn, automatically protects against attacks that use the verification oracle.

Definition 7 (Correctness)**.** *A verifiable computation protocol $\mathcal{VC}$ is* correct *for a class of functions $\mathcal{F}$ if for any $F \in \mathcal{F}$, any pair of keys $(PK_F, EK_F) \leftarrow \mathsf{KeyGen}(F, 1^\lambda)$, any $x \in \mathtt{Domain}(F)$, any $(\sigma_x, VK_x) \leftarrow \mathsf{ProbGen}(PK_F, x)$, and any $\sigma_{\mathsf{out}} \leftarrow \mathsf{Compute}(EK_F, \sigma_x)$, the verification algorithm* Verify *on input VK_x and σ_{out} outputs $y = F(x)$.*

Providing public delegation and verification introduces a new threat model in which the worker knows both the public key PK_F (which allows him to delegate computations) and the verification key VK_x for the challenge input x (which allows him to check whether his answers will pass the verification).

Definition 8 (Security)**.** *Let $\mathcal{VC}$ be a public verifiable computation scheme for a class of functions $\mathcal{F}$, and let $A = (A_1, A_2)$ be any pair of probabilistic polynomial time machines. Consider the experiment $\mathbf{Exp}_A^{PubVerif}[\mathcal{VC}, F, \lambda]$ for any $F \in \mathcal{F}$ below:*

Experiment $\mathbf{Exp}_A^{PubVerif}[\mathcal{VC}, F, \lambda]$

$(PK_F, EK_F) \leftarrow \mathsf{KeyGen}(F, 1^\lambda)$;

$(x^*, \mathsf{state}) \leftarrow A_1(PK_F, EK_F)$;

$(\sigma_{x^*}, VK_{x^*}) \leftarrow \mathsf{ProbGen}(PK_F, x^*)$;

$\sigma^*_{\mathsf{out}} \leftarrow A_2(\mathsf{state}, \sigma_{x^*}, VK_{x^*})$;

$y^* \leftarrow \mathsf{Verify}(VK_{x^*}, \sigma^*_{\mathsf{out}})$

If $y^* \neq \perp$ and $y^* \neq F(x^*)$, output '1', else output '0';

A public verifiable computation scheme $\mathcal{VC}$ is secure *for a class of functions $\mathcal{F}$, if for every*

function $F \in \mathcal{F}$ and every p.p.t. adversary $A = (A_1, A_2)$:

$$Pr[\mathbf{Exp}_A^{PubVerif}[\mathcal{VC}, F, \lambda] = 1] \leq \text{NEGL}\lambda. \tag{5.1}$$

where negl *denotes a negligible function of its input.*

Later, we will also briefly consider a weaker notion of "selective security" which requires the adversary to declare the challenge input x^* before it sees PK_F.

For verifiable outsourcing of a function to make sense, the client must use "less resources" than what is required to compute the function. "Resources" here could mean the running time, the randomness complexity, space, or the depth of the computation. We retain the earlier efficiency requirements [Gennaro *et al.*, 2010] – namely, we require the complexity of ProbGen and Verify combined to be less than that of F. However, for KeyGen, we ask only that the complexity be poly($|F|$). Thus, we employ an *amortized* complexity model, in which the client invests a larger amount of computational work in an "offline" phase in order to obtain efficiency during the "online" phase. We provide two strong definitions of efficiency – one that talks about the running time and a second that talks about computation depth.

Definition 9 (Efficiency)**.** *A verifiable computation protocol $\mathcal{VC}$ is* efficient *for a class of functions $\mathcal{F}$ that act on $n = n(\lambda)$ bits if there is a polynomial p s.t.:* [2]

- *the running time of* ProbGen *and* Verify *together is at most $p(n, \lambda)$, the rest of the algorithms are probabilistic polynomial-time, and*
- *there exists a function $F \in \mathcal{F}$ whose running time is $\omega(p(n, \lambda))$.* [3]

In a similar vein, $\mathcal{VC}$ is depth-efficient *if the computation depth of* ProbGen *and* Verify *combined (written as Boolean circuits) is at most $p(n, \lambda)$, whereas there is a function $F \in \mathcal{F}$ whose computation depth is $\omega(p(n, \lambda))$.*

We now define the notion of unbounded circuit families which will be helpful in quantifying the efficiency of our verifiable computation protocols.

[2] To be completely precise, one has to talk about a family $\mathcal{F} = \{\mathcal{F}_n\}_{n \in \mathbb{N}}$ parameterized by the input length n. We simply speak of $\mathcal{F}$ to implicitly mean $\mathcal{F}_n$ whenever there is no cause for confusion.

[3] This condition is to rule out trivial protocols, e.g., for a class of functions that can be computed in time less than $p(\lambda)$.

Definition 10. *We define a family of circuits* $\{\mathcal{C}_n\}_{n\in\mathbb{N}}$ *to be* unbounded *if for every polynomial* p *and all but finitely many* n*, there is a circuit* $C \in \mathcal{C}_n$ *of size at least* $p(n)$*. We call the family* depth-unbounded *if for every polynomial* p *and all but finitely many* n*, there is a circuit* $C \in \mathcal{C}_n$ *of depth at least* $p(n)$*.*

5.2.2 Key-Policy Attribute-Based Encryption

Introduced by Goyal, Pandey, Sahai and Waters [Goyal *et al.*, 2006], Key-Policy Attribute-Based Encryption (KP-ABE) is a special type of encryption scheme where a Boolean function F is associated with each user's key, and a set of attributes (denoted as a string $x \in \{0,1\}^n$) with each ciphertext. A key SK_F for a function F will decrypt a ciphertext corresponding to attributes x if and only if $F(x) = 1$. KP-ABE can be thought of as a special-case of predicate encryption [Katz *et al.*, 2008] or functional encryption [Boneh *et al.*, 2011], although we note that a KP-ABE ciphertext need not hide the associated policy or attributes. We will refer to KP-ABE simply as ABE from now on. We state the formal definition below, adapted from [Goyal *et al.*, 2006; Lewko *et al.*, 2010].

Definition 11 (Attribute-Based Encryption)**.** *An attribute-based encryption scheme* $\mathcal{ABE}$ *for a class of functions* $\mathcal{F} = \{\mathcal{F}_n\}_{n\in\mathbb{N}}$ *(where functions in* $\mathcal{F}_n$ *take* n *bits as input) is a tuple of algorithms* $(\mathsf{Setup}, \mathsf{Enc}, \mathsf{KeyGen}, \mathsf{Dec})$ *that work as follows:*

- $(PK, MSK) \leftarrow \mathsf{Setup}(1^\lambda, 1^n)$ *: Given a security parameter* λ *and an index* n *for the family* $\mathcal{F}_n$*, output a public key* PK *and a master secret key* MSK*.*
- $C \leftarrow \mathsf{Enc}(\mathsf{PK}, \mathsf{M}, \mathsf{x})$*: Given a public key* PK*, a message* M *in the message space* MsgSp*, and attributes* $x \in \{0,1\}^n$*, output a ciphertext* C*.*
- $SK_F \leftarrow \mathsf{KeyGen}(MSK, F)$*: Given a function* F *and the master secret key* MSK*, output a decryption key* SK_F *associated with* F*.*
- $\mu \leftarrow \mathsf{Dec}(\mathsf{SK_F}, \mathsf{C})$*: Given a ciphertext* $C \in \mathsf{Enc}(\mathsf{PK}, \mathsf{M}, \mathsf{x})$ *and a secret key* SK_F *for function* F*, output a message* $\mu \in \mathsf{MsgSp}$ *or* $\mu = \perp$*.*

Definition 12 (ABE Correctness)**.** *Correctness of the ABE scheme requires that for all* $(PK, MSK) \leftarrow \mathsf{Setup}(1^\lambda, 1^n)$*, all* $M \in \mathsf{MsgSp}$*,* $x \in \{0,1\}^n$*, all ciphertexts* $C \leftarrow \mathsf{Enc}(\mathsf{PK}, \mathsf{M}, \mathsf{x})$

and all secret keys $SK_F \leftarrow \mathsf{KeyGen}(MSK, F)$*, the decryption algorithm* $\mathsf{Dec}(\mathsf{SK_F}, \mathsf{C})$ *outputs* M *if* $F(x) = 1$ *and* $\perp$ *if* $F(x) = 0$*. (This definition could be relaxed to hold with high probability over the keys* (PK, MSK)*, which suffices for our purposes).*

We define a natural, yet relaxed, notion of security for ABE schemes which we refer to as "one-key security". Roughly speaking, we require that adversaries who obtain a single secret key SK_F for any function F of their choice and a ciphertext $C \leftarrow \mathsf{Enc}(\mathsf{PK}, \mathsf{M}, \mathsf{x})$ associated with any attributes x such that $F(x) = 0$ should not be able to violate the semantic security of C. We note that much work in the ABE literature has been devoted to achieving a strong form of security against collusion, where the adversary obtains not just a single secret key, but polynomially many of them for functions of its choice. We do not require such a strong notion for our purposes.

Definition 13 (One-Key Security for ABE)**.** *Let* $\mathcal{ABE}$ *be a key-policy attribute-based encryption scheme for a class of functions* $\mathcal{F} = \{\mathcal{F}_n\}_{n\in\mathbb{N}}$*, and let* $A = (A_0, A_1, A_2)$ *be a three-tuple of probabilistic polynomial-time machines. We define security via the following experiment.*

Experiment $\mathbf{Exp}_A^{ABE}[\mathcal{ABE}, n, \lambda]$

$(PK, MSK) \leftarrow \mathsf{Setup}(1^\lambda, 1^n)$*;*

$(F, \mathsf{state}_1) \leftarrow A_0(PK)$*;*

$SK_F \leftarrow \mathsf{KeyGen}(MSK, F)$*;*

$(M_0, M_1, x^*, \mathsf{state}_2) \leftarrow A_1(\mathsf{state}_1, SK_F)$*;*

$b \leftarrow \{0, 1\}$*;* $C \leftarrow \mathsf{Enc}(\mathsf{PK}, \mathsf{M_b}, \mathsf{x}^*)$*;*

$\hat{b} \leftarrow A_2(\mathsf{state}_2, C)$*;*

If $b = \hat{b}$*, output '1', else '0';*

The experiment is valid if $M_0, M_1 \in \mathsf{MsgSp}$ *and* $|M_0| = |M_1|$*. We define the advantage of the adversary in all valid experiments as*

$$Adv_A(\mathcal{ABE}, n, \lambda) = |Pr[b = b'] - 1/2|.$$

We say that $\mathcal{ABE}$ *is a one-key secure ABE scheme if* $Adv_A(\mathcal{ABE}, n, \lambda) \leq \mathrm{NEGL}(\lambda)$*.*

5.2.3 Multi-Function Verifiable Computation

The original definition of verifiable computation [Gennaro *et al.*, 2010] assumed that multiple inputs would be prepared for a single function; here, we expand this definition to efficiently allow workers to verifiably apply multiple functions to a single input. In other words, previously, to evaluate $F(x)$ and $G(x)$, the client needed to run KeyGen for F, KeyGen for G, and then run ProbGen on x twice, once for F and once for G (since the public key PK used for the input preprocessing in ProbGen depends on the function that is evaluated). Our new definition only requires the client to run ProbGen once, and yet still allows the client to verify that a particular output was the output of a particular function on a particular input.

We present the multi-function property in the secret key setting of the original definition of verifiable computation [Gennaro *et al.*, 2010], but note that it is orthogonal to the public delegation and verification defined in Section 5.2.1, and hence a scheme may have both properties, none, or one but not the other.

Since the original definition embeds the function to be computed in the scheme's parameters, we separate the generation of the parameters for the scheme, which will be used in ProbGen, into a Setup stage, and the generation of tokens for the evaluation of different functions into a KeyGen routine, which could be executed multiple times using the same parameters for the scheme. This allows the evaluation of multiple functions on the same instance produced by ProbGen.

Definition 14 (Multi-Function Verifiable Computation)**.** *A VC scheme* $\mathcal{VC}$ = (Setup, KeyGen,ProbGen, Compute, Verify) *is a multi-function verifiable computation scheme if it has the following properties:*

- $\mathsf{Setup}(\lambda) \rightarrow (PK_{param}, SK_{param})$*: Produces the public and private parameters that do not depend on the functions to be evaluated.*
- $\mathsf{KeyGen}_{PK_{param},SK_{param}}(F) \rightarrow (PK_F, SK_F)$*: Produces a keypair for evaluating and verifying a specific function* F*.*
- $\mathsf{ProbGen}_{PK_{param},SK_{param}}(x) \rightarrow (\sigma_x, \tau_x)$*: The algorithm requires the secret* SK_{param}*,*

which is independent of the function that will be computed. It generates both the encoding σ_x for the input, and the secret verification key τ_x.

- $\mathsf{Compute}_{PK_{param},PK_F}(\sigma_x) \rightarrow \sigma_y$*: The computation algorithm uses both parts of the public key to produce an encoding of the output $y = F(x)$.*

- $\mathsf{Verify}_{SK_F,\tau_x}(\sigma_y) \rightarrow y \cup \perp$*: Using the private, function-specific key SK_F and the private, input-specific value τ_x, the verification algorithm converts the worker's output into $y = F(x)$, or outputs $\perp$ to indicate that σ_y does not represent a valid output of F on x.*

Definition 15 (Multi-Function Verifiable Computation Security)**.** *Let $\mathcal{VC} =$ (Setup, KeyGen, $\mathsf{ProbGen}$, $\mathsf{Compute}$, Verify) be a multi-function verifiable computation scheme. We define security via the following experiment.*

Experiment $\mathbf{Exp}_A^{MultVerif}[\mathcal{VC}, \lambda]$

$(PK_{param}, SK_{param}) \xleftarrow{R} \mathsf{Setup}(\lambda)$;

$(x, F, \hat{\sigma_y}) \leftarrow A^{\mathcal{O}_{\mathsf{KeyGen}}(\cdot),\mathcal{O}_{\mathsf{ProbGen}}(\cdot)}(PK_{param})$;

$\hat{y} \leftarrow \mathsf{Verify}_{SK_F,\tau_x}(\hat{\sigma}_y)$

If $\hat{y} \neq \perp$ and $\hat{y} \neq F(x)$, output '1', else '0';

We define the adversary's advantage and the scheme's security in the same fashion as Definition 8.

In the experiment, the adversary has oracle access to $\mathcal{O}_{\mathsf{KeyGen}}(F)$, which calls $\mathsf{KeyGen}_{PK_{param},SK_{param}}(F)$, returns PK_F, and stores SK_F. Similarly, the adversary can access the $\mathcal{O}_{\mathsf{ProbGen}}(\cdot)$ oracle, which calls $\mathsf{ProbGen}_{SK_{param}}(x)$, returns σ_x, and stores τ_x. Eventually, the adversary returns an encoding $\hat{\sigma_y}$ which purports to be an output of F applied to x. The challenger runs Verify with the corresponding values of τ_x and SK_F, and the adversary wins if this check passes.

5.2.3.1 KP-ABE With Outsourcing

Green, Hohenberger, and Waters define a notion of ABE with outsourcing in which the party performing the decryption can offload most of the decryption work to an untrusted third

party [Green *et al.*, 2011]; the third party learns nothing about the underlying plaintext, and the party holding the secret key can complete the decryption very efficiently, in time independent of the size of the formula associated with the key. Although they define and construct ABE with outsourcing for both CP-ABE and KP-ABE, below, we focus on the definitions for KP-ABE, which will be relevant for our work. We also give an IND-CPA security definition, since we do not require the stronger RCCA they defined. Note that Green et al.'s construction [Green *et al.*, 2011] is selectively secure, but they provide a sketch, based on Lewko et al.'s work [Lewko *et al.*, 2010], to show that their scheme can also be made adaptively secure.

Note that in this context the outsourcing done is for the very specific function of ABE partial decryption. The definitions also do not include a notion of integrity or verification, as in verifiable computation, but instead are concerned with the secrecy of the underlying plaintext.

Definition 16 (Key-Policy Attribute-Based Encryption With Outsourcing [Green *et al.*, 2011])**.** *A KP-ABE scheme with outsourcing, $\mathcal{ABE}$, is a tuple of algorithms* $(\mathsf{Setup}, \mathsf{Enc}, \mathsf{KeyGen}, \mathsf{Transform}, \mathsf{Dec})$ *defined as follows:*

- $\mathsf{Setup}(\lambda, U) \rightarrow (PK, MSK)$ *: Given a security parameter λ and the set of all possible attributes U, output a public key PK and a master secret key MSK.*
- $\mathsf{Enc}_{\mathsf{PK}}(\mathsf{M}, \gamma) \rightarrow \mathsf{C}$*: Given a public key PK, a message M, and a set of attributes γ, output ciphertext C.*
- $\mathsf{KeyGen}_{MSK}(F) \rightarrow (SK_F, TK_F)$*: Given a function F and the master secret key MSK, output a decryption key SK_F and a transformation key TK_F associated with that function.*
- $\mathsf{Transform}_{TK_F}(C) \rightarrow C' \cup \perp$*: Given a ciphertext $C = \mathsf{Enc}_{\mathsf{PK}}(\mathsf{M}, \gamma)$ and a transformation key TK_F for function F, output a partially decrypted ciphertext C' if $F(\gamma) = 1$, or $\perp$, otherwise.*
- $\mathsf{Dec}_{\mathsf{SK_F}}(\mathsf{C}') \rightarrow \mathsf{M} \cup \perp$*: Given a partially decrypted ciphertext $C' = \mathsf{Transform}_{TK_F}(\mathsf{Enc}_{\mathsf{PK}}(\mathsf{M}, \gamma))$ and a secret key SK_F for function F, output M if $F(\gamma) = 1$, or $\perp$, otherwise.*

Definition 17 (KP-ABE With Outsourcing IND-CPA Security)**.** *Let* $\mathcal{ABE} = (\mathsf{Setup}, \mathsf{Enc}, \mathsf{KeyGen}, \mathsf{Transform}, \mathsf{Dec})$ *be a key-policy attribute-based encryption scheme with outsourcing. We define security via the following experiment.*

Experiment $\mathbf{Exp}_A^{ABE-Out}[\mathcal{ABE}, F, U, \lambda]$

$(PK, MSK) \xleftarrow{R} \mathsf{Setup}(\lambda, U)$;

$(M_0, M_1, \gamma) \leftarrow A^{\mathcal{O}_{\mathsf{KeyGen}}(\cdot), \mathcal{O}_{\mathbf{Corrupt}}(\cdot)}(PK)$;

$b \leftarrow \{0, 1\}$;

$\hat{b} \leftarrow A^{\mathcal{O}_{\mathsf{KeyGen}}(\cdot), \mathcal{O}_{\mathbf{Corrupt}}(\cdot)}(PK, \mathsf{Enc}_{\mathsf{PK}}(\mathsf{M_b}, \gamma))$;

If $b = \hat{b}$*, output '1', else '0';*

In the experiment, the adversary has access to two oracles. $\mathcal{O}_{\mathsf{KeyGen}}(F)$ *invokes* $(SK_F, TK_F) \leftarrow \mathsf{KeyGen}_{MSK}(F)$*, stores* SK_F *and returns* TK_F*.* $\mathcal{O}_{\mathbf{Corrupt}}(F)$ *returns* SK_F *if the adversary previously invoked* $\mathcal{O}_{\mathsf{KeyGen}}(F)$ *and returns* $\perp$ *otherwise. Eventually,* $\mathcal{A}$ *chooses two messages* M_0, M_1 *of equal length and a set of challenge attributes* γ*, and he receives the encryption of one of two messages. Ultimately, he must decide which of the two plaintext messages was encrypted.*

We consider the experiment valid *if* $\forall SK_F \in \mathcal{R} : F(\gamma) \neq 1$*, where* $\mathcal{R} = \{SK_F\}$ *is the set of valid responses to the* $\mathcal{O}_{\mathbf{Corrupt}}(F)$ *oracle. In other words, the adversary cannot hold a key that trivially decrypts messages encrypted under the challenge attribute* γ*.*

We define the advantage of the adversary in all valid experiments as

$$Adv_{\mathcal{A}}(\mathcal{ABE}, U, \lambda) = \left| Pr[b = b'] - \frac{1}{2} \right|.$$

We say that $\mathcal{ABE}$ *is a secure key-policy attribute-based encryption scheme with outsourcing if* $Adv_{\mathcal{A}}(\mathcal{ABE}, U, \lambda) < negl(\lambda)$*.*

5.3 Verifiable Computation from Attribute-Based Encryption

In Section 5.3.1, we present our main construction and proof, while Section 5.3.2 contains the various instantiations of our main construction and the concrete verifiable computation protocols that we obtain as a result.

5.3.1 Main Construction

Theorem 9. *Let $\mathcal{F}$ be a class of Boolean functions (implemented by a family of circuits $\mathcal{C}$), and let $\overline{\mathcal{F}} = \{\overline{F} \mid F \in \mathcal{F}\}$ where $\bar{F}$ denotes the complement of the function F. Let $\mathcal{ABE}$ be an attribute-based encryption scheme that is one-key secure (see Definition 13) for $\mathcal{F} \cup \overline{\mathcal{F}}$, and let g be any one-way function.*

Then, there is a verifiable computation protocol $\mathcal{VC}$ (secure under Definition 8) for $\mathcal{F}$. If the circuit family $\mathcal{C}$ is unbounded (resp. depth-unbounded), then the protocol $\mathcal{VC}$ is efficient (resp. depth-efficient) in the sense of Definition 9.

We first present our verifiable computation protocol.

Let $\mathcal{ABE} = (\mathsf{ABE.Setup}, \mathsf{ABE.KeyGen}, \mathsf{ABE.Enc}, \mathsf{ABE.Dec})$ be an attribute-based encryption scheme for the class of functions $\mathcal{F} \cup \overline{\mathcal{F}}$. Then, the verifiable computation protocol $\mathcal{VC} = (\mathsf{VCKeyGen}, \mathsf{ProbGen}, \mathsf{Compute}, \mathsf{Verify})$ for $\mathcal{F}$ works as follows.[4] We assume, without loss of generality, that the message space $\mathcal{M}$ of the ABE scheme has size 2^λ.

Key Generation $\mathsf{VCKeyGen}$**:** The client, on input a function $F \in \mathcal{F}$ with input length n, runs the ABE setup algorithm twice, to generate two independent key-pairs

$$(\mathsf{msk}_0, \mathsf{mpk}_0) \leftarrow \mathsf{ABE.Setup}(1^n, 1^\lambda) \quad \text{and} \quad (\mathsf{msk}_1, \mathsf{mpk}_1) \leftarrow \mathsf{ABE.Setup}(1^n, 1^\lambda)$$

Generate two secret keys $\mathsf{sk}_{\overline{F}} \leftarrow \mathsf{ABE.KeyGen}(\mathsf{msk}_0, \overline{F})$ (corresponding to $\overline{F}$) and $\mathsf{sk}_F \leftarrow \mathsf{ABE.KeyGen}(\mathsf{msk}_1, F)$ (corresponding to F).

Output the pair $(\mathsf{sk}_{\overline{F}}, \mathsf{sk}_F)$ as the evaluation key and $(\mathsf{mpk}_0, \mathsf{mpk}_1)$ as the public key.

Delegation $\mathsf{ProbGen}$**:** The client, on input x and the public key PK_F, samples two uniformly random messages $m_0, m_1 \xleftarrow{R} \mathcal{M}$, computes the ciphertexts

$$\mathsf{CT}_0 \leftarrow \mathsf{ABE.Enc}(\mathsf{mpk}_0, m_0) \quad \text{and} \quad \mathsf{CT}_1 \leftarrow \mathsf{ABE.Enc}(\mathsf{mpk}_1, m_1)$$

Output the message $\sigma_x = (\mathsf{CT}_0, \mathsf{CT}_1)$ (to be sent to the server), and the verification key $VK_x = (g(m_0), g(m_1))$, where g is the one-way function.

[4] We denote the VC key generation algorithm as $\mathsf{VCKeyGen}$ in order to avoid confusion with the ABE key generation algorithm.

Computation Compute**:** The server, on receiving the ciphertexts $(\mathsf{CT}_0, \mathsf{CT}_1)$ and the evaluation key $EK_F = (\mathsf{sk}_{\overline{F}}, \mathsf{sk}_F)$ computes

$$\mu_0 \leftarrow \mathsf{ABE.Dec}(\mathsf{sk}_{\overline{F}}, \mathsf{CT}_0) \quad \text{and} \quad \mu_1 \leftarrow \mathsf{ABE.Dec}(\mathsf{sk}_F, \mathsf{CT}_1)$$

and send $\sigma_{\mathsf{out}} = (\mu_0, \mu_1)$ to the client.

Verification Verify**:** On receiving $VK_x = (v_0, v_1)$ and $\sigma_{\mathsf{out}} = (\mu_0, \mu_1)$, output [5]

$$y = \begin{cases} 0 & \text{if } g(\mu_0) = v_0 \text{ and } g(\mu_1) \neq v_1 \\ 1 & \text{if } g(\mu_1) = v_1 \text{ and } g(\mu_0) \neq v_0 \\ \perp & \text{otherwise} \end{cases}$$

Remark 1. *Whereas our main construction requires only an ABE scheme, using an attribute-hiding ABE scheme (a notion often associated with predicate encryption schemes [Katz* et al.*, 2008; Boneh* et al.*, 2011]) would also give us input privacy, since we encode the function's input in the attribute corresponding to a ciphertext.*

Remark 2. *To obtain a VC protocol for functions with multi-bit output, we repeat this protocol (including the key generation algorithm) independently for every output bit. To achieve better efficiency, if the ABE scheme supports attribute hiding for a class of functions that includes message authentication codes (MAC), then we can define $F'(x) = MAC_K(F(x))$ and verify F' instead, similar to the constructions suggested by Applebaum, Ishai, and Kushilevitz [Applebaum* et al.*, 2010], and Barbosa and Farshim [Barbosa and Farshim, 2011].*

Remark 3. *The construction above requires the verifier to trust the party that ran* ProbGen*. This can be remedied by having* ProbGen *produce a non-interactive zero-knowledge proof of correctness [Blum* et al.*, 1988] of the verification key VK_x. While theoretically efficient, the practicality of this approach depends on the particular ABE scheme and the NP language in question.*

Proof. Correctness The correctness of the VC scheme above follows from:

- If $F(x) = 0$, then $\overline{F}(x) = 1$ and thus, the algorithm Compute outputs $\mu_0 = m_0$ and $\mu_1 = \perp$. The algorithm Verify outputs $y = 0$ since $g(\mu_0) = g(m_0)$ but $g(\mu_1) = \perp \neq g(m_1)$, as expected.

[5] As a convention, we assume that $g(\perp) = \perp$.

- Similarly, if $F(x) = 1$, then $\overline{F}(x) = 0$ and thus, the algorithm Compute outputs $\mu_1 = m_1$ and $\mu_0 = \perp$. The algorithm Verify outputs $y = 1$ since $g(\mu_1) = g(m_1)$ but $g(\mu_0) = \perp \neq g(m_0)$, as expected.

□

We now consider the relation between the efficiency of the algorithms for the underlying ABE scheme and the efficiency for the resulting VC scheme. Since the algorithms Compute and Verify can potentially be executed by different parties, we consider their efficiency separately. It is easily seen that:

- The running time of the VC key generation algorithm VCKeyGen is twice that of ABE.Setup plus ABE.KeyGen.
- The running time of Compute is twice that of ABE.Dec.
- The running time of ProbGen is twice that of ABE.Enc, and the running time of Verify is the same as that of computing the one-way function.

In short, the combined running times of ProbGen and Verify is polynomial in their input lengths, namely $p(n, \lambda)$, where p is a fixed polynomial, n is the length of the input to the functions, and λ is the security parameter. Assuming that $\mathcal{F}$ is an *unbounded* class of functions (according to Definition 10), it contains functions that take longer than $p(n, \lambda)$ to compute, and thus our VC scheme is efficient in the sense of Definition 9. (Similar considerations apply to depth-efficiency).

We now turn to showing the security of the VC scheme under Definition 8. We show that an attacker against the VC protocol must either break the security of the one-way function g or the one-key security of the ABE scheme.

Proof. Security Let $A = (A_1, A_2)$ be an adversary against the VC scheme for a function $F \in \mathcal{F}$. We construct an adversary $B = (B_0, B_1, B_2)$ that breaks the one-key security of the ABE, working as follows. (For notational simplicity, given a function F, we let $F_0 = \overline{F}$, and $F_1 = F$.)

1. B_0 first tosses a coin to obtain a bit $b \in \{0,1\}$. (Informally, the bit b corresponds to B's guess of whether the adversary A will cheat by producing an input x such that $F(x) = 1$ or $F(x) = 0$, respectively.)

   B_0 outputs the function F_b, as well as the bit b as part of the state.

2. B_1 obtains the master public key mpk of the ABE scheme and the secret key sk_{F_b} for the function F_b. Set $\mathsf{mpk}_b = \mathsf{mpk}$.

   Run the ABE setup and key generation algorithms to generate a master public key mpk' and a secret key $\mathsf{sk}_{F_{1-b}}$ for the function F_{1-b} under mpk'. Set $\mathsf{mpk}_{1-b} = \mathsf{mpk}'$.

   Let $(\mathsf{mpk}_0, \mathsf{mpk}_1)$ be the public key for the VC scheme and $(\mathsf{sk}_{F_0}, \mathsf{sk}_{F_1})$ be the evaluation key. Run the algorithm A_1 on input the public and evaluation keys and obtain a challenge input x^* as a result.

   If $F(x^*) = b$, output a uniformly random bit and stop. Otherwise, B_1 now chooses two uniformly random messages $M^{(b)}, \rho \leftarrow \mathcal{M}$ and outputs $(M^{(b)}, \rho, x^*)$ together with its internal state.

3. B_2 obtains a ciphertext $C^{(b)}$ (which is an encryption of either $M^{(b)}$ or ρ under the public key mpk_b and attribute x^*).

   B_2 constructs an encryption $C^{(1-b)}$ of a uniformly random message $M^{(1-b)}$ under the public key mpk_{1-b} and attribute x^*.

   Run A_2 on input $\sigma_{x^*} = (C^{(0)}, C^{(1)})$ and $VK_{x^*} = (g(M^{(0)}), g(M^{(1)}))$, where g is the one-way function. As a result, A_2 returns σ_{out}.

   If $\mathsf{Verify}(VK_{x^*}, \sigma_{\mathsf{out}}) = b$, output 0 and stop.

We now claim the algorithms (B_0, B_1, B_2) described above distinguish between the encryption of $M^{(b)}$ and the encryption of ρ in the ABE security game with non-negligible advantage.

We consider two cases.

Case 1: $C^{(b)}$ is an encryption of $M^{(b)}$. In this case, B presents to A a perfect view of the execution of the VC protocol, meaning that A will cheat with probability $1/p(\lambda)$ for some polynomial p.

Cheating means one of two things. Either $F(x^*) = b$ and the adversary produced an inverse of $g(M^{(1-b)})$ (causing the Verify algorithm to output $1-b$), or $F(x^*) = 1-b$ and the adversary produced an inverse of $g(M^{(b)})$ (causing the Verify algorithm to output b).

In the former case, B outputs a uniformly random bit, and in the latter case, it outputs 0, the correct guess as to which message was encrypted. Thus, the overall probability that B outputs 0 is $1/2 + 1/p(\lambda)$.

Case 2: $C^{(b)}$ is an encryption of the message ρ. In this case, as above, B outputs a random bit if $F(x^*) = b$. Otherwise, the adversary A has to produce σ_{out} that makes the verifier output b, namely a string σ_{out} such that $g(\sigma_{\mathsf{out}}) = g(M^{(b)})$, while given only $g(M^{(b)})$ (and some other information that is independent of $M^{(b)}$).

This amounts to inverting the one-way function which A can only do with a negligible probability. (Formally, if the adversary wins in this game with non-negligible probability, then we can construct an inverter for the one-way function g).

The bottom line is that the adversary outputs 0 in this case with probability $1/2 + \mathrm{NEGL}(\lambda)$.

This shows that B breaks the one-key security of the ABE scheme with a non-negligible advantage $1/p(\lambda) - \mathrm{NEGL}(\lambda)$. □

Remark 4. *If we employ an ABE scheme that is selectively secure, then the construction and proof above still go through if we adopt a notion of "selectively-secure" verifiable computation in which the VC adversary commits in advance to the input on which he plans to cheat.*

5.3.2 Instantiations

We describe two different instantiations of our main construction.

Efficient Selectively Secure VC Scheme for Formulas. The first instantiation uses the (selectively secure) ABE scheme of Ostrovsky, Sahai and Waters [Ostrovsky *et al.*, 2007]

for the class of (not necessarily monotone) polynomial-size Boolean formulas (which itself is an adaptation of the scheme of Goyal et al. [Goyal *et al.*, 2006] which only supports monotone formulas[6]). This results in a selectively secure public VC scheme for the same class of functions, by invoking Theorem 9. Recall that selective security in the context of verifiable computation means that the adversary has to declare the input on which she cheats at the outset, before she sees the public key and the evaluation key.

The efficiency of the resulting VC scheme for Boolean formulas is as follows: for a boolean formula C, KeyGen runs in time $|C| \cdot \mathsf{poly}\lambda$; $\mathsf{ProbGen}$ runs in time $|x| \cdot \mathsf{poly}\lambda$, where $|x|$ is the length of the input to the formula; $\mathsf{Compute}$ runs in time $|C| \cdot \mathsf{poly}\lambda$; and Verify runs in time $O(\lambda)$. In other words, the total work for delegation and verification is $|x| \cdot \mathsf{poly}\lambda$ which is, in general, more efficient than the work required to evaluate the circuit C. Thus, the scheme is efficient in the sense of Definition 9. The drawback of this instantiation is that it is only selectively secure.

Recently, there have been constructions of fully secure ABE for formulas starting from the work of Lewko et al. [Lewko *et al.*, 2010] which, one might hope, leads to a fully secure VC scheme. Unfortunately, all known constructions of fully secure ABE work for *bounded* classes of functions. For example, in the construction of Lewko et al., once a bound B is fixed, one can design the parameters of the scheme so that it works for any formula of size at most B. Furthermore, implicit in the work of Sahai and Seyalioglu [Sahai and Seyalioglu, 2010] is a construction of an (attribute-hiding, one-key secure) ABE scheme for *bounded* polynomial-size circuits (as opposed to formulas).

These constructions, unfortunately, do not give us efficient VC protocols. The reason is simply this: the encryption algorithm in these schemes run in time polynomial (certainly, at least linear) in B. Translated to a VC protocol using Theorem 9, this results in the worker running for time $\Omega(B)$ which is useless, since given that much time, he could have computed any circuit of size at most B by himself!

[6]Goyal et al.'s scheme [Goyal *et al.*, 2006] can also be made to work if we use DeMorgan's law to transform f and $\bar{f}$ into equivalent monotone formulas in which some variables may be negated. We then double the number of variables, so that for each variable v, we have one variable representing v and one representing its negation $\bar{v}$. Given an input x, we choose an attribute such that all of these variables are set correctly.

Essentially, the VC protocol that emerges from Theorem 9 is non-trivial if the encryption algorithm of the ABE scheme for the function family $\mathcal{F}$ is (in general) more efficient than computing functions in $\mathcal{F}$.

Depth-Efficient Adaptively Secure VC Scheme for Arbitrary Functions. Although the (attribute-hiding, one-key secure) ABE construction of Sahai and Seyalioglu [Sahai and Seyalioglu, 2010] mentioned above does not give us an efficient VC scheme, it does result in a *depth-efficient VC scheme* for the class of polynomial-size circuits. Roughly speaking, the construction is based on Yao's Garbled Circuits, and involves an ABE encryption algorithm that constructs a garbled circuit for the function F in question. Even though this computation takes at least as much time as computing the circuit for F, the key observation is that it can be done in parallel. In short, going through the VC construction in Theorem 9, one can see that both the Compute and Verify algorithms can be implemented in constant depth (for appropriate encryption schemes and one-way functions, e.g., the ones that result from the AIK transformation [Applebaum *et al.*, 2004]), which is much faster in parallel than computing F, in general.

Interestingly, the VC protocol thus derived is very similar to the protocol of Applebaum, Ishai and Kushilevitz [Applebaum *et al.*, 2010]. We refer the reader to [Sahai and Seyalioglu, 2010; Applebaum *et al.*, 2010] for details.

We believe that this scheme also illuminates an interesting point: unlike other ABE schemes [Goyal *et al.*, 2006; Ostrovsky *et al.*, 2007; Lewko *et al.*, 2010], this ABE scheme is only *one-key secure*, which suffices for verifiable computation. This relaxation may point the way towards an ABE-based VC construction that achieves generality, efficiency, and adaptive security.

■☞**Mariana: Add something about new ABE from Lewko and Waters**

5.4 Multi-Function Verifiable Computation from KP-ABE With Outsourcing

The original definition of KP-ABE does not readily lend itself to multi-function verifiable computation. Specifically, it does not allow the client an easy way to verify which function was used to compute an answer. For example, suppose the client gives out keys SK_F and SK_G for functions F and G. Following the ABE to VC construction from Section 5.3 to outsource computation on input x, the client gives out (among other things) a ciphertext $\mathsf{Enc}_{\mathsf{PK}}(\mathsf{M}_0, \mathsf{x})$. Now, suppose $F(x) = 1$, but $G(x) \neq 1$. The worker can use SK_F to obtain M_0, but claim that this output corresponds to a computation of G. In essence, the construction from Section 5.3 gives us a way to verify that an output corresponds to a particular input, but if we give out more than one secret key, it cannot distinguish between functions. One remedy would be to run two parallel instances of the ABE to VC construction, but then we need to run $\mathsf{ProbGen}$ for each function we wish to compute on a given input.

A more elegant solution is to use an ABE scheme that requires an extra step to decrypt a ciphertext. Thus, we show how to build multi-function verifiable computation from KP-ABE with outsourcing [Green *et al.*, 2011] (see Section 5.2.3.1). We use the transformation key to allow the worker to compute, and then use the secret key as a verification key for the function. This allows us to verify both the input and specific function used to compute a particular result returned by the worker.

Interestingly, a similar scheme can be constructed from Chase's multi-authority ABE [Chase, 2007], by using function identifiers (e.g., a hash of the function description, or a unique ID assigned by the client) in place of user identifiers, and using the "user key" generated by the Central Authority as a verification token for a particular function. However, since this approach does not employ the multi-authority ABE scheme in a black-box fashion, in this section, we focus on the construction from KP-ABE with outsourcing.

We specify the construction in detail below. For clarity, we only consider functions with single-bit outputs, but the construction can be generalized just as we did in Section 5.3.

Construction 1. *Let* $\mathcal{ABE} = (\mathsf{Setup}, \mathsf{Enc}, \mathsf{KeyGen}, \mathsf{TransformDec})$ *be a KP-ABE scheme*

with outsourcing with attribute universe U. We construct a multi-function verifiable computation scheme as follows:

- $\mathsf{Setup}(\lambda) \to (PK_{param}, SK_{param})$ *: Run* $\mathcal{ABE}.\mathsf{Setup}(\lambda, U)$ *twice to obtain* (PK^0, MSK^0) *and* (PK^1, MSK^1).

  Set $PK_{param} = (PK^0, PK^1)$ *and* $SK_{param} = (MSK^0, MSK^1)$.

- $\mathsf{KeyGen}_{PK_{param}, SK_{param}}(F) \to (PK_F, SK_F)$ *: Compute* $(SK^0_{\bar{F}}, TK^0_{\bar{F}}) \leftarrow \mathcal{ABE}.\mathsf{KeyGen}_{MSK^0}(\bar{F})$ *and* $(SK^1_F, TK^1_F) \leftarrow \mathcal{ABE}.\mathsf{KeyGen}_{MSK^1}(F)$*, where* $\bar{F}$ *is the complement of* F.

  Output $PK_F = (TK^0_{\bar{F}}, TK^1_F)$ *and* $SK_F = (SK^0_{\bar{F}}, SK^1_F)$*. In other words, the public key will be the transformation keys, and the secret verification key will be the "true" secret keys.*

- $\mathsf{ProbGen}_{PK_{param}, SK_{param}}(x) \to (\sigma_x, \tau_x)$*: Generate a pair of random messages* $(M_0, M_1) \xleftarrow{R} \{0,1\}^\lambda \times \{0,1\}^\lambda$*. Compute ciphertexts* $C_0 \leftarrow \mathcal{ABE}.\mathsf{Enc}_{\mathsf{PK}_0}(\mathsf{M}_0, \mathsf{x})$ *and* $C_1 \leftarrow \mathcal{ABE}.\mathsf{Enc}_{\mathsf{PK}_1}(\mathsf{M}_1, \mathsf{x})$.

  Output $\sigma_x = (C_0, C_1)$ *and* $\tau_x = (M_0, M_1)$.

- $\mathsf{Compute}_{PK_{param}, PK_F}(\sigma_x) \to \sigma_y$*: Parse* PK_F *as* $(TK^0_{\bar{F}}, TK^1_F)$*. Compute* $C'_0 = \mathcal{ABE}.\mathsf{Transform}_{TK^0_{\bar{F}}}(C_0)$ *and* $C'_1 = \mathcal{ABE}.\mathsf{Transform}_{TK^1_{\bar{F}}}(C_1)$*. Output* $\sigma_y = (C'_0, C'_1)$.

- $\mathsf{Verify}_{SK_F, \tau_x}(\sigma_y) \to y$*: Parse* SK_F *as* $(SK^0_{\bar{F}}, SK^1_F)$*,* τ_x *as* (M_0, M_1)*, and* σ_y *as* (C'_0, C'_1)*. If* $\mathcal{ABE}.\mathsf{Dec}_{\mathsf{SK}^0_{\bar{\mathsf{F}}}}(\mathsf{C}'_0) = \mathsf{M}_0$*, then output* $y = 0$*. If* $\mathcal{ABE}.\mathsf{Dec}_{\mathsf{SK}^1_{\mathsf{F}}}(\mathsf{C}'_0) = \mathsf{M}_1$*, then output* $y = 1$*. Otherwise, output* $\perp$.

The above construction will provide the efficiency property required for a VC scheme (verification that is more efficient than the delegated computation) as long as the Transform algorithm is computationally more expensive than the Enc and Dec algorithms of the ABE scheme. However, this requirement is inherent in the definition of ABE with outsourcing.

Remark 5. *Construction 1 is publicly delegatable, since* ProbGen *only makes use of* PK_{param}*; i.e., it only employs the public ABE keys to perform* ProbGen*, so anyone may do so. However, the verification function cannot be made public* while still preserving the ability to verify the specific function used. *Specifically, giving out* SK_F *in Green et al.'s ABE*

scheme [Green et al.*, 2011] would directly allow the worker to lie about the function used, i.e., claim that an output computed with F was the result of applying G. Even so, the adversary would still be unable to lie about the output's* value*. Thus, if we only care about the integrity of the output value and the fact that it was produced by* some *function submitted to* KeyGen*, then this construction can be made publicly verifiable as well.*

Proof Intuition. The proof of security looks very similar to Proof 5.3.1. The intuition for input security is the same as before, i.e., the revelation of one of the two random messages associated with a ProbGen invocation demonstrates that the computation was performed on that particular input. Unlike with a regular ABE scheme, we can also verify the function used, since decrypting with a key that does not match the transformation key used will not produce the expected message. This is provided by the security of the ABE with outsourced decryption, which guarantees semantic security of the encrypted messages even when the adversary sees the transformation keys used for the outsourced portion of the decryption.

Theorem 10. *Let* $\mathcal{ABE} = (\mathsf{Setup}, \mathsf{Enc}, \mathsf{KeyGen}, \mathsf{TransformDec})$ *be a secure (according to Definition 17) KP-ABE scheme with outsourcing with attribute universe* U*. Let* $\mathcal{VC}_{ABE} = (\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{ProbGen}, \mathsf{Compute}, \mathsf{Verify})$ *be a multi-function verifiable computation scheme obtained from* $\mathcal{ABE}$ *using Construction 1. Then* $\mathcal{VC}_{ABE}$ *is secure according to Definition 15.*

Proof. Theorem 10

Let us assume that there exists an adversary $\mathcal{A}_{VC}$ that succeeds to cheat in the security game (Definition 15) for the scheme $\mathcal{VC}_{ABE}$ with non-negligible probability μ. We show how to construct an adversary $\mathcal{A}_{ABE}$ that wins the security game from Definition 1 with non-negligible probability.

1. $\mathcal{A}_{ABE}$ chooses a random bit $r \xleftarrow{R} \{0,1\}$.
2. $\mathcal{A}_{ABE}$ receives a public key in the ABE security game. Call it PK^r_{abe}.
3. $\mathcal{A}_{ABE}$ generates a second pair of keys $(PK^{1-r}_{abe}, MSK^{1-r}_{abe}) \leftarrow \mathcal{ABE}.\mathsf{Setup}(\lambda, U)$.
4. On a call to Setup in the VC security game $\mathcal{A}_{ABE}$ provides to $\mathcal{A}_{VC}$ the keys (PK^0_{abe}, PK^1_{abe}).

5. On a call to $\mathsf{KeyGen}(f^1)$ in the VC security game $\mathcal{A}_{ABE}$ computes the complement function $f^0 \leftarrow \bar{f^1}$. $\mathcal{A}_{ABE}$ submits a query in the ABE security game for f^r and receives back the transformation key TK_{f^r}. He also generates on his own the keys $(TK_{f^{1-r}}, SK_{f^{1-r}}) \leftarrow \mathcal{ABE}.\mathsf{KeyGen}_{MSK_{abe}^{1-r}}(f^{1-r})$, and returns (TK_{f^0}, TK_{f^1}) to $\mathcal{A}_{VC}$.

6. The adversary $\mathcal{A}_{ABE}$ guesses one of the inputs x that he receives from $\mathcal{A}_{VC}$ to be the challenge input and sets $\gamma = x$. He chooses three messages m_0, m_1, m_2, sends m_0, m_1 and γ to the challenger in the ABE security game, and receives back the challenge ciphertext c^r. $\mathcal{A}_{ABE}$ computes $c^{1-r} \leftarrow \mathcal{ABE}.\mathsf{Enc}_{\mathsf{PK}_{\mathsf{abe}}^{1-\mathsf{r}}}(\mathsf{m_2}, \gamma)$ and sets $\sigma_x \leftarrow (c^0, c^1)$. For any other input y, $\mathcal{A}_{ABE}$ chooses two messages m_0, m_1 and returns $\sigma_x \leftarrow (\mathcal{ABE}.\mathsf{Enc}_{\mathsf{PK}_{\mathsf{abe}}^{0}}(\mathsf{m_0}, \gamma), \mathcal{ABE}.\mathsf{Enc}_{\mathsf{PK}_{\mathsf{abe}}^{1}}(\mathsf{m_1}, \gamma))$.

7. $\mathcal{A}_{VC}$ returns to $\mathcal{A}_{ABE}$ a cheating output $(x, f^1, \hat{\sigma_y} = (C_0', C_1'))$ for the evaluation of a function f^1 on the input x.

8. If $f^1(x) = r$, then $\mathcal{A}_{ABE}$ returns a random bit. Otherwise, he invokes his oracle $SK_{f^r} \leftarrow \mathbf{Corrupt}(f^r)$, uses SK_{f^r} to decrypt C_{1-r}' and obtain M. If $M = m_0$, he returns 0; if $M = m_1$, he returns 1, and otherwise, he returns a random bit.

To see that $\mathcal{A}_{ABE}$ succeeds with non-negligible probability, note that if $f^1(x) = 0$, then to cheat, $\mathcal{A}_{VC}$ must convince the verify function that $y = 1$; i.e., he must return $\hat{\sigma_y} = (C_0', C_1')$ such that $ABE.\mathsf{Dec}_{\mathsf{SK^1}}(\mathsf{C_1'}) = \mathsf{M_1}$. If $r = 0$, the cheat doesn't involve the challenge ciphertext c^0, so $\mathcal{A}_{ABE}$ makes a random guess. If $r = 1$, then $\mathcal{A}_{ABE}$ calls $\mathbf{Corrupt}(f^1)$. Since $f^1(x) = 0$, this will not violate the ABE security game.

The case for $f^1(x) = 1$ is symmetric.

Therefore, $\mathcal{A}_{ABE}$ succeeds in the security game with probability $\frac{\mu}{2Q}$, where Q is the number of queries made to $\mathsf{ProbGen}$ by $\mathcal{A}_{VC}$; since Q is polynomial in λ, this is non-negligible.

□

Chapter 6

Outsourcing Multi-Party Computation with Non-Colluding Adversaries

6.1 Motivation and Contributions

Early feasibility results in MPC showed that any functionality can be securely computed [Yao, 1982; Yao, 1986; Goldreich *et al.*, 1987; Chaum *et al.*, 1988a]. Since then, there has been a large number of works improving the security definitions, strengthening the adversarial model, increasing the number of malicious parties tolerated and studying the round and communication complexity (we refer the reader to [Goldreich, 2004] and the references therein). Most of these works, however, implicitly assume that the computation will be executed in a homogeneous computing environment, where all the participants play similar roles and have the same amount of resources. As such, almost all MPC protocols have a *symmetric* workload in the sense that every party is required to do the same amount of work. In practice, however, distributed computations rarely take place in such settings and, more often than not, they are carried out by a diverse set of devices that could include, e.g., high-performance servers, large-scale clusters, personal computers and even weak mobile devices. And while it is certainly possible to execute standard MPC protocols

in such environments, it is undesirable from a practical point of view since the computation will be bounded by the resources of the weakest device.

Towards making MPC more practical, it is then natural to seek protocols with asymmetric workloads. While such protocols can be designed—in the semi-honest model—based on fully-homomorphic encryption (FHE), the resulting constructions are still too computationally expensive to be of practical interest. In this work, we take a different approach and show that if some of the parties are dishonest but do not collude, then it is possible to design protocols with asymmetric workloads and avoid the use of FHE altogether. We believe security in the presence of non-colluding adversaries is a useful guarantee as there are many instances in practice where collusion is unlikely to occur. This can happen either because it is not feasible, is too costly, or because it is prevented by other means, e.g., by physical means, by the Law or due to conflicting interests. Non-collusion can also occur if the parties in the system are compromised by independent adversaries that simply do not have the capacity or the opportunity to compromise the same system (e.g., independent malware spreading through a network or hackers that are not aware of each other's presence).

An important example of a heterogeneous environment—and the main motivation for our work—is cloud computing, where a computationally powerful service provider offers possibly weaker clients access to its resources "as a service". This leads us to consider the problem of MPC in a setting where, in addition to the parties evaluating the functionality, there is an *untrusted* server that (1) does not have any input to the computation; (2) does not receive any output from the computation; but (3) has a vast (but bounded) amount of computational resources. We refer to this as *server-aided* MPC and our aim is to design protocols with workloads that are asymmetric and that minimize the computation of the parties at the expense of the server.

6.1.1 Our Contributions

We consider MPC in the server-aided setting, where the parties have access to a *single* server with a large amount of computational resources that they can use to outsource their computation. In this setting, we are concerned with designing protocols that minimize the

computation of the parties at the expense of the server. While it is possible to treat the server as one more participant in a multi-party computation and to use standard MPC constructions to obtain server-aided protocols, a naive implementation of this approach would result in a symmetric workload. We observe, however, that asymmetry can be achieved under a weaker adversarial model and, in particular, when certain parties do not collude. While our main motivation for considering non-colluding adversaries is to design server-aided protocols without making use of FHE and with better efficiency than the naive implementation discussed above, our formalization of non-collusion might be of interest in other scenarios such as the conventional MPC settings. We make several contributions:

1. We formalize and define security for server-aided MPC. Our security definition is in the ideal/real-world paradigm and guarantees that, in addition to the standard security properties of MPC, the server learns no information about the client's inputs or *outputs* and cannot affect the correctness of the computation.

2. We consider a new adversarial model for MPC in which corrupted parties do not necessarily collude. Non-collusion in heterogeneous computing environments often occurs in practice and therefore is important to consider. To address this question, we generalize the standard security definition for MPC to allow for a finer-grained specification of collusion between the parties. This requires us to introduce formal characterizations of non-colluding adversaries which may be of independent interest. Also, as we will see, by considering non-colluding adversaries we are able to obtain highly efficient protocols.

3. We explore the connection between server-aided MPC and secure delegated computation. Roughly speaking, a server-aided MPC protocol can be viewed as a (interactive) delegated computation scheme for multiple parties. We show how to transform any secure delegated computation scheme into a server-aided MPC protocol.

In addition to the theoretical contributions discussed above, we also describe two efficient general-purpose server-aided MPC protocols based on Yao's garbled circuits [Yao, 1982], and an efficient special-purpose protocol for private set intersection:

4. The first protocol we consider is (a slight variant of) the FKN protocol [Feige *et al.*, 1994]. We show that it is secure against a malicious server and semi-honest parties that do not collude with the server. It allows all but one of the parties to outsource their computation to the server; making their computation only linear in the size of their inputs (which is optimal). In addition, the protocol does not require any public-key operations (except for a one time coin-tossing protocol).

5. Our second protocol extends the FKN protocol to be secure even when all but one of the parties is malicious. Our construction uses cut-and-choose techniques from [Mohassel and Franklin, 2006; Lindell and Pinkas, 2007], but requires us to address several subtleties that do not exist in the standard two-party setting. One main problem we need to solve is how to allow an *untrusted* server to send the majority output of multiple circuits to the parties without learning the actual output or modifying the results. We achieve this by constructing a new *oblivious* cut-and-choose protocol that allows the verifier in the cut-and-choose mechanism to outsource its computation to an *untrusted* server.

6. Our third protocol is a new and efficient server-aided protocol for private set intersection. Our construction provides security against a malicious server as well as malicious participants. The bulk of computation by each participant is a number of PRF evaluations that is linear in the size of its input set. In comparison, the most efficient two-party set intersection protocol [Hazay and Nissim, 2010] with equivalent security guarantees, requires $O(m + n(\log\log m + p))$ exponentiations where m and n are the sizes of the input sets and p is the bit length of each input element, which is logarithmic in the input domain range. Protocols that provide security in weaker adversarial models (e.g., the random oracle model, the CRS model, and limited input domain) and achieve linear computational complexity in the total size of the input sets still require a linear number of public key operations.

   Our solution generalizes to the case of multi-party set intersection, preserving the same computational complexity for each participant. The best existing solution for this case [Dachman-Soled *et al.*, 2011] has computation cost of $O(Nd^2 \log d)$ in the

case of N parties with input sets of size d.

The above-mentioned protocols are significantly more efficient than the existing MPC protocols for the same problems. In order to provide a better sense of the *efficiency gain* obtained by these server-aided constructions, we initiate in Section 6.9 an informal discussion on efficiency in the server-aided model. In particular we outline different ways of quantifying the efficiency of a server-aided MPC protocol, each of which is suitable for a specific setting or application. We also provide some intuition for why any noticeable improvement in the efficiency of our general-purpose constructions is likely to yield considerably more practical secure delegated computation schemes. We note that a more formal study of efficiency in the server-aided model is an interesting research direction.

6.2 Overview of Protocols

In this overview and throughout the rest of the paper, we mostly focus on server-aided two-party computation (two parties and a server). However, as we discuss in future sections, our constructions easily extend to the multi-party case as well.

The first two protocols are based on Yao's garbled circuit construction. In both protocols, the only *interaction* between the two parties, denoted by P_1 and P_2, is to generate a set of shared random coins to be used in the rest of the protocol. This step is independent of the parties' inputs and the function being evaluated and can be performed offline (e.g., for multiple instantiations of the protocol at once). Moreover, the coin-tossing needs to be performed exactly once to share a secret key. In all future runs of the protocol, P_1 and P_2 can use their shared secret key and a pseudorandom function, to generate the necessary coins. After this step, the two parties interact directly with the *untrusted server* until they retrieve their final result.

The FKN protocol. In the (modified) FKN protocol (described in Section 6.5), after generating the shared random coins, each party sends a single message to the server and receives an encoding of his own output. The parties then individually use their local coins to recover their outputs. For P_1 and P_2, this protocol is significantly more efficient than using

a standard secure two-party computation protocol. First, none of the parties (including the server) have to perform any public-key operations, except to perform the coin-tossing which as discussed above is only performed once. This is in contrast to standard MPC where public-key operations (which are considerably more expensive than their secret-key counterparts) are a necessary. Second, one of the parties (P_2 in our case) only needs to do work that is linear in the size of his input and output, and independent of the size of the circuit being computed.

The server and P_1 will do work that is linear in the size of the circuit. If the server-aided protocol is run multiple times for the same or different functionalities, it is possible to reduce the online work of P_1 by performing the garbling of multiple circuits (for the same or different functions) in the offline phase. The online work for P_1 will then be similar to P_2 and only linear to the size of his input and output.

We prove the protocol secure against a number of corruption and non-collusion scenarios. Particularly, the protocol can handle cases where the server, or P_2 are malicious while the other players are either honest, or semi-honest but non-colluding.

Making FKN robust. The security of the modified FKN protocol breaks down when party P_1 is malicious. We address this in Section 6.6 by augmenting it with cut-and-choose techniques for Yao's two-party protocol (e.g., see [Mohassel and Franklin, 2006; Lindell and Pinkas, 2007]). First, note that the cut-and-choose procedure no longer takes place between P_1 and P_2 since this would require P_2 to do work linear in the circuit size (significantly reducing his efficiency gain). Instead, P_2 outsources his cut-and-choose verification to the server. However, a few subtleties arise that are specific to the server-aided setting and require modifications to the existing techniques. At a high level, the difficulty is that, unlike the standard setting, the server only learns the garbled outputs and therefore cannot determine the majority output on his own. One might try to resolve this by having the server send the garbled outputs to P_1 and P_2 and have them compute the majority but, unfortunately, this is also insecure.

We take the following approach. Instead of treating the garbled output values as the garbling of the real outputs (as prescribed by the translation table), we use them as evalu-

ation points on random polynomials that encode (as their constant factor) the "ultimate" garbled values corresponding to the real outputs. This encoding can be interpreted as a Reed-Solomon encoding of the ultimate garbled values using the intermediate ones returned by the circuit evaluation. Intuitively, as long as the majority of the garbled evaluation points are correct, the error correction of Reed-Solomon codes guarantees correct and unambiguous decoding of the majority output by the server. Further care is needed to ensure that the server performs the decoding obliviously, and that for each output bit he is only able to decode one Reed-Solomon codeword without learning the corresponding output. For this purpose, it turns out that we need the polynomials used for the Reed-Solomon encoding to be permutations over the finite field. To achieve this goal, we sample our polynomials uniformly at random from the family of Dickson polynomials of an appropriate degree.

Delegation-based protocol. We show how to construct a server-aided two-party computation protocol based on any secure non-interactive *delegated computation* scheme. A delegated computation scheme allows a client to securely outsource the computation of a circuit C on a private input x to an untrusted worker. The notion of secure delegated computation is closely related to that of verifiable computation [Goldwasser *et al.*, 2008; Gennaro *et al.*, 2010; Chung *et al.*, 2010], with the additional requirement that the client's input and output remains private. Our construction is interesting in the sense that, it formalizes the intuitive connection between the problems of server-aided computation and verifiable computation by interpreting server-aided protocols as a means for verifiably and privately outsourcing a secure two-party computation protocol to an untrusted worker. The resulting protocol inherits the efficiency of the underlying delegated computation scheme.

Set intersection protocol. We present a construction for outsourced computation for the problem of set intersection where two or more parties want to find the intersection of their input sets. The main idea of our protocol is that to have the server compute the set intersection of the input sets but since we want to preserve the privacy of the inputs, he is only given PRF evaluations on the elements under a key that all parties have agreed on. Then each party will be able to map the returned intersection PRF values to real elements. In order to protect against a malicious server, we augment this approach by

mapping each input value to several unlinkable PRF evaluations. Now the server will be asked to compute the set intersection on the new expanded sets and will be able to cheat without being detected only if he guesses correctly which PRF evaluations correspond to the same input value. This approach, however, introduces a new security issue in that it allows the parties to be malicious by creating inconsistent PRF values. We fix this by requiring each party to prove that he has computed correctly the multiple PRF evaluations for each of his input elements by opening them after the server has committed to the output result. In order not to lose the privacy guarantees for the inputs, however, we apply another level of PRF evaluations.

6.3 Preliminaries and Standard Definitions

Notation. We write $x \leftarrow \chi$ to represent an element x being sampled from a distribution χ, and $x \xleftarrow{\$} X$ to represent an element x being sampled uniformly from a set X. If f is a function, we refer to its domain as $\mathsf{Dom}(f)$ and to its range as $\mathsf{Ran}(f)$. The output x of an algorithm $\mathcal{A}$ is denoted by $x \leftarrow \mathcal{A}$. If $\mathcal{A}$ is a probabilistic algorithm we sometimes write $y \leftarrow \mathcal{A}(x; r)$ to make the coins r of $\mathcal{A}$ explicit. $[n]$ denotes the set $\{1, \ldots, n\}$. We refer to the ith element of a sequence $\mathbf{v}$ as either v_i or $\mathbf{v}[i]$. Throughout k will refer to the security parameter. A function $\nu : \mathbb{N} \to \mathbb{N}$ is negligible in k if for every polynomial $p(\cdot)$ and sufficiently large k, $\nu(k) < 1/p(k)$. Let $\mathsf{poly}(k)$ and $\mathrm{NEGL}(k)$ denote unspecified polynomial and negligible functions in k, respectively. We write $f(k) = \mathsf{poly}(k)$ to mean that there exists a polynomial $p(\cdot)$ such that for all sufficiently large k, $f(k) \leq p(k)$, and $f(k) = \mathrm{NEGL}(k)$ to mean that there exists a negligible function $\nu(\cdot)$ such that for all sufficiently large k, $f(k) \leq \nu(k)$.

Private key encryption. A private-key encryption scheme is a set of three polynomial-time algorithms $(\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ that work as follows. Gen is a probabilistic algorithm that takes a security parameter k in unary and returns a secret key K. Enc is a probabilistic algorithm that takes a key K and an n-bit message m and returns a ciphertext c. Dec is a deterministic algorithm that takes a key K and a ciphertext c and returns m if K was the key under which c was produced. Informally, a private-key encryption scheme is considered

secure against chosen-plaintext attacks (CPA) if the ciphertexts it outputs do not leak any useful information about the plaintext even to an adversary that can adaptively query an encryption oracle.

Functionalities. An n-party randomized functionality is a function $f : (\{0,1\}^*)^n \times \{0,1\}^* \rightarrow \{0,1\}^*$, where the first input is a sequence of n strings $\mathbf{x}$, the second input is a set of random coins and the output is a sequence of n strings $\overline{y}$. We will often omit the coins and simply write $\overline{y} \leftarrow f(\mathbf{x})$. If we do wish to make the coins explicit then we write $\overline{y} \leftarrow f(\mathbf{x}; r)$. We denote the ith party's output by $f_i(\mathbf{x})$. A functionality is deterministic if it only takes the input string $\mathbf{x}$ as input and it is symmetric if all parties receive the same output. It is known that any protocol for securely computing deterministic functionalities can be used to securely compute randomized functionalities (cf. [Goldreich, 2004] Section 7.3) so in this work we focus on the former. A basic functionality we will make use of is the coin tossing functionality $\mathcal{F}_{\mathsf{CT}}(1^\ell, 1^\ell) = (r, r)$, where $|r| = \ell$ and r is uniformly distributed.

Garbled circuits. Yao's garbled circuit construction consists of five polynomial-time algorithms $\mathsf{Garb} = (\mathsf{GarbCircuit}, \mathsf{GarbIn}, \mathsf{Eval}, \mathsf{GarbOut}, \mathsf{Translate})$ that work as follows. We present GarbCircuit, GarbIn and GarbOut as deterministic algorithms that take a set of coins r as input. GarbCircuit is a deterministic algorithm that takes as input a circuit C that evaluates a function f, and a set of coins $r \in \{0,1\}^k$ and returns a garbled circuit $\mathbb{G}(C)$. GarbIn is a deterministic algorithm that takes as input a player index $i \in \{1,2\}$, an input x , coins $r \in \{0,1\}^k$, and returns a garbled input $\mathbb{G}(x)$. Eval is a deterministic algorithm takes as input a garbled circuit $\mathbb{G}(C)$ and two garbled inputs $\mathbb{G}(x)$ and $\mathbb{G}(y)$ and returns a garbled output $\mathbb{G}(o)$. GarbOut is a deterministic algorithm that takes as input a random coins $r \in \{0,1\}^k$ and returns a translation table T. Translate is a deterministic algorithm that takes as input a garbled output $\mathbb{G}(o)$ and a translation table T and returns an output o.

We note that it is possible to arrange the above five functions in such a way that the computational complexity of GarbIn is linear in the input size, the computational complexity of GarbOut and Translate is linear in the output size, while the complexity of GarbCircuit and Eval are linear in the circuit size. We use this important property when discussing

efficiency of our protocols.

Informally, Garb is considered secure if $(\mathbb{G}(C), \mathbb{G}(x), \mathbb{G}(y))$ reveals no information about x and y. An added property possessed by the construction is *verifiability* which, roughly speaking, means that, given $(\mathbb{G}(C), \mathbb{G}(x), \mathbb{G}(y))$, no adversary can output some $\mathbb{G}(o)$ such that $\mathsf{Translate}(\mathbb{G}(o), \mathrm{T}) \neq f(x, y)$. We discuss these properties more formally in Appendix D.1.

Delegated computation. A delegated/verifiable computation scheme consists of four polynomial-time algorithms $\mathsf{Del} = (\mathsf{Gen}, \mathsf{ProbGen}, \mathsf{Compute}, \mathsf{Verify})$ that work as follows. Gen is a probabilistic algorithm that takes as input a security parameter k and a function f and outputs a public and secret key pair $(\mathrm{PK}, \mathsf{sk})$ such that the public key encodes the target function f. ProbGen is a probabilistic algorithm that takes as input a secret key sk and an input x in the domain of f and outputs a public encoding σ_x and a secret state τ_x. Compute is a deterministic algorithm that takes as input a public key PK and a public encoding σ_x and outputs a public encoding σ_y. Verify is a deterministic algorithm that takes as input a secret key sk, a secret state τ_x and a public encoding σ_y and outputs either an element y of f's range or the failure symbol $\perp$. Informally, a delegated computation scheme is private if the public encoding σ_x of x reveals no useful information about x. In addition, the scheme is verifiable if no adversary can find an encoding $\sigma y'$, for some $y' \neq f(x)$, such that $\mathsf{Verify}_{\mathsf{sk}}(\tau_x, \sigma_{y'}) \neq \perp$. We say that a delegated computation scheme is secure if it is both private and verifiable. We refer the reader to Section D.2 for a formal definition of verifiable computation.

6.4 Non-Collusion in Multi-Party Computation

The standard ideal/real world definition for MPC, proposed by Canetti [Canetti, 2000b] and building on [Beaver, 1992; Goldwasser and Levin, 1991; Micali and Rogaway, 1992], compares the real-world execution of a protocol for computing an n-party functionality f to the ideal-world evaluation of f by a trusted party. In the real-world execution, the parties run the protocol in the presence of an adversary $\mathcal{A}$ that is allowed to corrupt a subset of the parties. In the ideal execution, the parties interact with a trusted party that evaluates

f in the presence of a simulator $\mathcal{S}$ that corrupts the same subset of parties.

Typically, only a single adversary $\mathcal{A}$ is considered. This *monolithic* adversary captures the possibility of collusion between the dishonest parties. One distinguishes between passive corruptions, where the adversary only learns the state of the corrupted parties; and active corruptions where the adversary completely controls the party and, in particular, is not assumed to follow the protocol. Typically, adversaries that make passive corruptions are *semi-honest* whereas adversaries that make active corruptions are *malicious*. In this work, we will make a distinction between malicious adversaries who make active corruptions and can behave arbitrarily and *deviating* adversaries who make active corruptions but whose behavior may not be arbitrarily malicious (i.e., their behavior may be limited to a certain class of attacks). Another distinction can be made as to how the adversary chooses which parties to corrupt. If it must decide this before the execution of the protocol then we say that the adversary is *static*. On the other hand, if the adversary can decide during the execution of the protocol then we say that the adversary is *adaptive*. We only consider static adversaries in this work.

Roughly speaking, a protocol Π is considered secure if it emulates, in the real-world, an evaluation of f in the ideal-world. This is formalized by requiring that the joint distribution composed of the honest parties' outputs and of $\mathcal{A}$'s view in the real-world execution be indistinguishable from the joint distribution composed of the honest parties' outputs and the simulator $\mathcal{S}$'s view in the ideal-world execution. As mentioned above, the standard security definition for MPC models adversarial behavior using a *monolithic* adversary. This has the advantage that it captures collusion and thus provides strong security guarantees. There are, however, many instances in practice where collusion does not occur. This can happen either because it is not feasible, too costly, or because it is prevented by other means (e.g., by physical means, by the Law or due to conflicting interests). This is particularly true in the setting of cloud computing where one can think of many scenarios where the server (i.e., the cloud operator) will have little incentive to collude with any of the other parties.

This naturally leads to the two following questions: (1) how do we formalize secure computation in the presence of non-colluding adversaries? and (2) what do we gain by

weakening the security guarantee? In particular, can we design protocols that are more efficient or that remain secure even if *all* parties are corrupted? Note that while the standard definition becomes meaningless if all parties are corrupted (since the monolithic adversary then knows all the private information in the system), this is not so if the parties are corrupted by non-colluding adversaries.

6.4.1 Formalizing Non-Collusion With Respect to Semi-Honest Adversaries

Intuitively, we think of collusion between participants in a MPC protocol as an exchange of "useful information", where by useful information we mean anything that allows the colluding parties to learn something about the honest parties' inputs that is not prescribed by the protocol. For our purposes, a non-colluding adversary is therefore:

"an adversary that avoids revealing any useful information to other parties."

As we will see, formalizing this intuition is not straight-forward. In the following discussion, we divide the messages sent by a party into two types: *protocol* and *non-protocol* messages. Assuming the protocol starts with a "start" message and ends with an "end" message, a protocol message is one that comes after the start message and before the end message, and a non-protocol message is one that comes either before the start message or after the end message. To formalize our intuition we need to make two crucial changes to the standard definition which we discuss below.

Independent adversaries. First, in addition to the monolithic adversary (which corrupts multiple parties) we include a set of non-monolithic adversaries that corrupt at most one party and have access only to the view of that party. We also assume that all the adversaries are *independent* in the sense that they do not share any state [1]. Intuitively, this essentially guarantees that these adversaries do not send any non-protocol messages to each other and therefore that they can only collude using protocol messages. Throughout the rest of this work all adversaries are independent.

[1] This idea already appears in work on collusion-free protocols [Lepinksi *et al.*, 2005; Alwen *et al.*, 2008; Alwen *et al.*, 2009].

When working with independent adversaries, we will consider three possible adversarial behaviors: semi-honest, malicious and *non-cooperative*. Semi-honest and malicious behavior refer to the standard notions: a semi-honest adversary follows the protocol while a malicious adversary can deviate arbitrarily. Informally, a non-cooperative adversary is one that deviates from the protocol as long as he does not (willingly) send useful information to another party. We will discuss how to formalize this intuition in section 6.4.2, and for now we focus on semi-honest adversaries. Note that, intuitively, if an adversary is independent and semi-honest, then it is non-colluding in the sense outlined above because it will send only protocol messages (by independence) and because these messages will reveal at most what is prescribed by the protocol (due its semi-honest behavior).

Partial emulation. Our second modification is a weakening of the notion of emulation to only require that indistinguishability hold with respect to the honest parties' outputs and a *single* adversary's view. In other words, we require that for each independent adversary $\mathcal{A}_i$, the joint distribution composed of the honest parties' outputs and $\mathcal{A}_i$'s view in the real-world, be indistinguishable from the joint distribution composed of the honest parties' outputs and the simulator $\mathcal{S}_i'$'s output in the ideal world. Roughly speaking, this implies that the protocol remains private (i.e., the parties do not learn information about each other's inputs) as long as the parties do not share any information.

To see why partial emulation is needed in our setting, consider two independent adversaries $\mathcal{A}_1$ and $\mathcal{A}_2$ whose outputs are correlated based on some protocol message exchanged between them. Under the standard notion of emulation, the simulators $\mathcal{S}_1$ and $\mathcal{S}_2$ would also have to output correlated views. The problem, however, is that because $\mathcal{S}_1'$ and $\mathcal{S}_2'$ are independent they cannot exchange any messages and a-priori it is not clear how they could correlate their outputs [2].

We are now ready to introduce our security definition for non-colluding semi-honest adversaries. Like the standard definition of security with abort, we do not seek to guarantee

[2]This could potentially be addressed using a setup assumption, but here we restrict ourselves to the plain model.

fairness or guaranteed output delivery and this is captured by allowing the adversaries to abort during the ideal-world execution. In addition, however, we also allow the server to select which parties will and will not receive their outputs. This weakening of the standard definition was first proposed by Goldwasser and Lindell in [Goldwasser and Lindell, 2002] and has the advantage of removing the need for a broadcast channel.

Real-world execution. The real-world execution of protocol Π takes place between players $(P_1, \ldots, P_n)$, server P_{n+1} and adversaries $(\mathcal{A}_1, \ldots, \mathcal{A}_{m+1})$, where $m \leq n$. Let $\mathrm{H} \subseteq [n+1]$ denote the honest parties, $\mathbf{I} \subset [n+1]$ denote the set of corrupted and non-colluding parties and $\mathrm{C} \subset [n+1]$ denote the set of corrupted and colluding parties. Since we only consider static adversaries these sets are fixed once the protocol starts.

At the beginning of the execution, each party $(P_1, \ldots, P_n)$ receives its input x_i, a set of random coins r_i and an auxiliary input z_i while the server P_{n+1} receives only its coins r_{n+1} and an auxiliary input z_{n+1}. Each adversary $(\mathcal{A}_1, \ldots, \mathcal{A}_m)$ receives an index $i \in \mathbf{I}$ that indicates the party it corrupts, while adversary $\mathcal{A}_{m+1}$ receives C indicating the set of parties it corrupts.

For all $i \in \mathrm{H}$, let out_i denote the output of P_i and for $i \in \mathbf{I} \cup \mathrm{C}$, let out_i denote the view of party P_i during the execution of Π. The ith partial output of a real-world execution of Π between players $(P_1, \ldots, P_{n+1})$ in the presence of adversaries $\mathcal{A} = (\mathcal{A}_1, \ldots, \mathcal{A}_{m+1})$ is defined as

$$\mathrm{REAL}^{(i)}_{\Pi,\mathcal{A},\mathbf{I},\mathrm{C},\mathbf{z}}(k, \mathbf{x}; \overline{r}) \stackrel{\Delta}{=} \left\{\mathsf{out}_j : j \in \mathrm{H}\right\} \cup \mathsf{out}_i.$$

Ideal-world execution. In the ideal-world execution, all the parties interact with a trusted party that evaluates f. As in the real-world execution, the ideal execution begins with each party $(P_1, \ldots, P_n)$ receiving its input x_i, its coins r_i and an auxiliary input z_i, while the server P_{n+1} receives only its coins r_{n+1} and an auxiliary input z_{n+1}. Each party $(P_1, \ldots, P_n)$ sends x'_i to the trusted party, where $x'_i = x_i$ if P_i is semi-honest and x' is an arbitrary value if P_i is malicious. If any $x'_i = \perp$, then the trusted party returns $\perp$ to all parties. If this is not the case, then the trusted party asks the server to specify which of the corrupted parties should receive their outputs and which should receive $\perp$. The trusted party then returns $f_i(x'_1, \ldots, x'_n)$ to the corrupted parties P_i that are to receive

their outputs and $\perp$ to the remaining corrupted parties. The trusted party then asks the corrupted parties that received an output whether they wish to abort. If any of them does, then the trusted party returns $\perp$ to the honest parties. If not, it returns $f_i(x'_1, \ldots, x'_n)$ to honest party P_i.

For all $i \in \mathrm{H}$, let out_i denote the output returned to P_i by the trusted party, and for $i \in \mathbf{I} \cup \mathrm{C}$ let out_i be some value output by P_i. The ith partial output of an ideal-world execution between parties $(P_1, \ldots, P_{n+1})$ in the presence of independent simulators $\mathcal{S} = (\mathcal{S}_1, \ldots, \mathcal{S}_{m+1})$ is defined as

$$\textsc{ideal}^{(i)}_{f,\mathcal{S},\mathbf{I},\mathrm{C},\mathbf{z}}(k, \mathbf{x}; \overline{r}) \stackrel{\Delta}{=} \left\{\mathsf{out}_j : j \in \mathrm{H}\right\} \cup \mathsf{out}_i.$$

Security. Informally, a protocol Π is considered secure against non-colluding semi-honest adversaries if it *partially* emulates, in the real-world, an evaluation of f in the ideal-world.

Definition 18 (Security against semi-honest adversaries)**.** *Let f be a deterministic n-party functionality and Π be an n-party protocol. Furthermore, let $\mathbf{I} \subset [n+1]$ and $\mathrm{C} \subset [n+1]$ be such that $\mathbf{I} \cap \mathrm{C} = \emptyset$ and $|\mathbf{I}| = m$. We say that Π $(\mathbf{I}, \mathrm{C})$-securely computes f if there exists a set $\{\mathsf{Sim}_i\}_{i\in[m+1]}$ of* PPT *transformations such that for all semi-honest* PPT *adversaries $\mathcal{A} = (\mathcal{A}_1, \ldots, \mathcal{A}_{m+1})$, for all $\mathbf{x} \in (\{0,1\}^*)^n$ and $\mathbf{z} \in (\{0,1\}^*)^{n+1}$, and for all $i \in [m+1]$,*

$$\left\{\textsc{real}^{(i)}_{\Pi,\mathcal{A},\mathbf{I},\mathrm{C},\mathbf{z}}(k, \mathbf{x}; \overline{r})\right\}_{k\in\mathbb{N}} \stackrel{c}{\equiv} \left\{\textsc{ideal}^{(i)}_{f,\mathcal{S},\mathbf{I},\mathrm{C},\mathbf{z}}(k, \mathbf{x}; \overline{r})\right\}_{k\in\mathbb{N}},$$

where $\mathcal{S} = (\mathcal{S}_1, \ldots, \mathcal{S}_{m+1})$ and $\mathcal{S}_i = \mathsf{Sim}_i(\mathcal{A}_i)$ and where $\overline{r}$ is chosen uniformly at random.

6.4.2 Formalizing Non-Collusion With Respect to Deviating Adversaries

While non-collusion in the semi-honest model can be formalized via independent adversaries and partial emulation, this is not sufficient when the adversaries are allowed to deviate from the protocol. The difficulty is that such adversaries can use protocol messages to collude since they can send arbitrary messages. Of course, collusion can be prevented through physical means (e.g., using ballot boxes as in [Lepinksi *et al.*, 2005]) or trusted communication channels (e.g., the mediator model used in [Alwen *et al.*, 2008; Alwen *et al.*, 2009]) but our goal here is not to obtain (or define) protocols that prevent adversaries from colluding but to formally characterize adversaries that do not wish to or cannot collude.

Towards formalizing our intuition, we introduce the notions of *non-cooperative* and *isolated* adversaries. Let $(\mathcal{A}_1, \ldots, \mathcal{A}_{m+1})$ be a set of independent adversaries. Informally, an adversary $\mathcal{A}_i$ is non-cooperative with respect to another adversary $\mathcal{A}_j$ (for $j \neq i$) if it does not share any useful information with $\mathcal{A}_j$. An adversary $\mathcal{A}_i$ is isolated if all adversaries $\{\mathcal{A}_j\}_{j\neq i}$ are non-cooperative with respect to $\mathcal{A}_i$. In other words, $\mathcal{A}_i$ is isolated if no one wants to share any useful information with him. We formalize this intuition in the following definitions.

Definition 19 (Non-cooperative adversary)**.** *Let f be a deterministic n-party functionality and Π be an n-party protocol. Furthermore, let* H*,* **I** *and* C *be pairwise disjoint subsets of $[n+1]$ and let $\mathcal{A} = (\mathcal{A}_1, \ldots, \mathcal{A}_{m+1})$, where $m = |\mathbf{I}|$, be a set of independent* PPT *adversaries. For any $i, j \in [m+1]$ such that $i \neq j$, we say that adversary $\mathcal{A}_j$ is non-cooperative with respect to $\mathcal{A}_i$ if there exists a* PPT *simulator $\mathrm{V}_{i,j}$ such that for all $\mathbf{x} \in (\{0,1\}^*)^n$, for all $\mathbf{z}, \overline{r} \in (\{0,1\}^*)^{n+1}$ and all $y \in \mathsf{Ran}(f_i) \cup \{\perp\}$,*

$$\left\{\mathrm{V}_{i,j}(y, z_i), r_j\right\}_{k\in\mathbb{N}} \stackrel{c}{\equiv} \left\{\mathsf{view}_{i,j}, r_j \;\middle|\; \mathsf{output}_i = y : \{\mathsf{out}_\ell\}_\ell \leftarrow \mathrm{REAL}^{(i)}_{\Pi,\mathcal{A},\mathbf{I},\mathrm{C},\mathbf{z}}(k, \mathbf{x}; \overline{r})\right\}_{k\in\mathbb{N}}$$

whenever $\Pr[\,\mathsf{output}_i = y\,] > 0$*. Here* $\mathsf{view}_{i,j}$ *denotes the messages between $\mathcal{A}_i$ and $\mathcal{A}_j$ in the real-world execution and* $\mathsf{output}_i = y$ *is the event that party P_i receives output value y.*

Note that with the notion of non-cooperation, we are restricting the behavior of the non-cooperating adversary. In particular, we are assuming it will deviate from the protocol but in such a way that it will not disclose any useful information to the isolated adversary $\mathcal{A}_i$. Therefore, one has to be careful in specifying the isolated party's behavior (i.e., whether it is semi-honest or malicious) so that the non-cooperation assumption is not so strong as to imply the security of the protocol. In particular, requiring that the simulator $\mathrm{V}_{i,j}$ work with respect to a *malicious* $\mathcal{A}_i$ seems too strong. Similarly, requiring that it work with respect to an honest $\mathcal{A}_i$ seems too weak as honest adversaries can always be simulated. A more useful and reasonable notion seems to follow from requiring that $\mathrm{V}_{i,j}$ work with respect to *semi-honest* adversaries. This can be interpreted as saying that the non-cooperative adversary does not *intentionally* disclose useful information to an isolated adversary. In particular, this means that the non-cooperative adversary will not take actions such as sending its

private input to the isolated party. It does not, however, restrict the isolated adversary from trying to "trick" the non-cooperative adversary into revealing this information.

As described above, an isolated adversary is one with which no other adversary wants to cooperate. Roughly speaking, we formalize this intuition by requiring that there exist an emulator for the isolated adversary $\mathcal{A}_i$ that, given only $\mathcal{A}_i$'s output value $f_i(\mathbf{x})$, returns $\mathcal{A}_i$'s view from a real-world execution. Intuitively, the notion of isolation restricts the behavior of the other adversaries towards $\mathcal{A}_i$ by allowing them to behave arbitrarily as long as their collective actions do not result in $\mathcal{A}_i$ learning any useful information in the sense discussed above. This informal description neglects some important subtleties that we address below.

Definition 20 (Isolated adversary)**.** *Let f be a deterministic n-party functionality and Π be an n-party protocol. Furthermore, let* H, **I** *and* C *be pairwise disjoint subsets of $[n+1]$ and let $\mathcal{A} = (\mathcal{A}_1, \ldots, \mathcal{A}_{m+1})$, where $m = |\mathbf{I}|$, be a set of independent* PPT *adversaries. For any $i, j \in [m+1]$ such that $i \neq j$, we say that a semi-honest adversary $\mathcal{A}_i$ is isolated if there there exists a* PPT *emulator $\mathcal{E}_i$ such that for all $\mathbf{x} \in (\{0,1\}^*)^n$ and $\overline{r}, \mathbf{z} \in (\{0,1\}^*)^{n+1}$, and all $y \in \mathsf{Ran}(f_i) \cup \{\perp\}$,*

$$\left\{\mathcal{E}_i\big(y, z_i\big), \{r_j\}_{j\neq i}\right\}_{k\in\mathbb{N}} \stackrel{c}{\equiv} \left\{\mathsf{out}_i, \{r_j\}_{j\neq i} \;\middle|\; \mathsf{output}_i = y : \{\mathsf{out}_\ell\}_\ell \leftarrow \mathrm{REAL}^{(i)}_{\Pi,\mathcal{A},\mathbf{I},\mathrm{C},\mathbf{z}}(k, \mathbf{x}; \overline{r})\right\}_{k\in\mathbb{N}}$$

whenever $\Pr[\,\mathsf{output}_i = y\,] > 0$*. Here $\mathsf{output}_i = y$ is the event that party P_i receives output value y.*

Towards understanding Definition 20, it is perhaps instructive to consider how we will make use of it. Recall that to prove the security of a protocol Π against independent adversaries $(\mathcal{A}_1, \ldots, \mathcal{A}_{m+1})$, we need, for all $i \in [m+1]$, to describe a simulator $\mathcal{S}_i$ whose output in an ideal execution is indistinguishable from $\mathcal{A}_i$'s output in a real execution. To achieve this, we consider a simulator $\mathcal{S}_i$ that works by simulating $\mathcal{A}_i$ so that it can recover $\mathcal{A}_i$'s output and return it as its own. Now suppose that, in the real world, Π requires $\mathcal{A}_i$ to interact with some other adversary $\mathcal{A}_j$. It follows that $\mathcal{S}_i$ will somehow have to simulate this interaction, i.e., $\mathcal{S}_i$ will have to simulate the (protocol) messages from $\mathcal{A}_j$. These messages, however, could be "colluding messages" in the sense that they could carry information that helps $\mathcal{A}_i$ in learning more that what is prescribed by the protocol and it may be impossible for $\mathcal{S}_i$ to simulate them as they could include information known only

to $\mathcal{A}_i$ and $\mathcal{A}_j$ (e.g., this information could be hardwired in $\mathcal{A}_i$ and $\mathcal{A}_j$). This is the main reason that simply requiring that the real and ideal adversaries be independent (together with partial emulation) is not enough to capture non-collusion with respect to deviating adversaries.

Our approach here will be to "strengthen" the simulator $\mathcal{S}_i$ by *assuming* that the adversaries $\{\mathcal{A}_j\}_{j\neq i}$ are non-cooperative with respect to $\mathcal{A}_i$. In Lemma 11 below, we will show that this implies that $\mathcal{A}_i$ is isolated and, therefore, there exists an emulator $\mathcal{E}_i$ that will return a view that is indistinguishable from $\mathcal{A}_i$'s view when interacting with $\{\mathcal{A}_j\}_{j\neq i}$ in a real-world execution.

Lemma 11. *Let f be a deterministic n-party functionality and Π be an n-party protocol. Furthermore, let $\mathcal{A} = (\mathcal{A}_1, \ldots, \mathcal{A}_{m+1})$ be a set of independent* PPT *adversaries. For any $i, j \in [m+1]$, if $\{\mathcal{A}_j\}_{j\neq i}$ are non-cooperative with respect to $\mathcal{A}_i$, then $\mathcal{A}_i$ is isolated.*

Proof. Consider the emulator $\mathcal{E}_i$ that, given y and z_i, computes $v_{i,j} \leftarrow \mathrm{V}_{i,j}(y, z_i)$ for all $j \neq i$ and returns the view out_i composed of $\{v_{i,j}\}_{i\neq j}$. Let $\{\mathcal{A}_{j_1}, \ldots, \mathcal{A}_{j_m}\}$ be the adversaries that are non-cooperative with respect to $\mathcal{A}_i$ and let $\{\mathrm{V}_{i,j_1}, \ldots, \mathrm{V}_{i,j_m}\}$ be their corresponding simulators (which are guaranteed to exist by the fact that they are non-cooperating with respect to $\mathcal{A}_i$). We show that $\mathcal{E}_i$'s output is indistinguishable from the view of $\mathcal{A}_i$ in a real execution using the following sequence of games. Game_0 consists of running $\{\mathsf{out}_j\}_j \leftarrow \mathrm{REAL}^{(i)}_{\Pi,\mathcal{A},\mathbf{I},\mathrm{C},\mathbf{z}}(k, \mathbf{x})$ and outputting out_i. For all $\ell \in [m]$, Game_ℓ is similar to $\mathsf{Game}_{\ell-1}$ except that the messages between $\mathcal{A}_i$ and $\mathcal{A}_{j_\ell}$ in out_i are replaced with messages generated by $\mathrm{V}_{i,j_\ell}(y, z_i)$. Note that the output of Game_m is distributed exactly as the output of $\mathcal{E}_i$.

The indistinguishability of the outputs of Game_0 and Game_1 follows directly from the non-cooperation of $\mathcal{A}_{j_1}$ with respect to $\mathcal{A}_i$ so we show that, for all $2 \leq \ell \leq m$, if there exists a PPT distinguisher $\mathbf{D}_\ell$ that distinguishes the output of Game_ℓ from that of $\mathsf{Game}_{\ell-1}$, then there exists a PPT distinguisher $\mathbf{B}_\ell$ that distinguishes the messages exchanged between $\mathcal{A}_i$ and $\mathcal{A}_{j_\ell}$ in a real-world execution from the messages generated by V_{i,j_ℓ}. Given a set of messages v_{i,j_ℓ} (generated either from a real-world execution or from V_{i,j_ℓ}), $\mathbf{B}_\ell$ works as follows. It first runs $\{\mathsf{out}_j\}_j \leftarrow \mathrm{REAL}_{\Pi,\mathcal{A},\mathbf{I},\mathrm{C},\mathbf{z}}(k, \mathbf{x})$ and, for $1 \leq t \leq \ell - 1$, it replaces the messages between $\mathcal{A}_i$ and $\mathcal{A}_{j_t}$ in out_i by $v_{i,j_t} \leftarrow \mathrm{V}_{i,j_t}(y, z_i)$. It then replaces the messages between $\mathcal{A}_i$ and $\mathcal{A}_{j_\ell}$ in out_i with v_{i,j_ℓ} and simulates $\mathbf{D}(\mathsf{out}_i)$. It returns "real" if $\mathbf{D}$ outputs

ℓ and "simulated" if $\mathbf{D}$ outputs $\ell + 1$. Notice that if v_{i,j_ℓ} was generated from a real-world execution then $\mathbf{B}$ constructs out_i exactly as in $\mathsf{Game}_{\ell-1}$ whereas if v_{i,j_ℓ} is generated from V_{i,j_ℓ} then $\mathbf{B}$ constructs out_i as in Game_ℓ. It follows then that $\mathbf{D}$'s advantage is equal to $\mathbf{B}$'s advantage in distinguishing between the outputs of $\mathsf{Game}_{\ell-1}$ and Game_ℓ which, by our initial assumption, is non-negligible. □

Since, in our setting, we will consider multiple adversaries with different adversarial behaviors it will be convenient to specify the behavior of each adversary using the following notation. If $\mathcal{A}_1$ is semi-honest we will write $\mathcal{A}_1[sh]$ and if $\mathcal{A}_1$ is malicious we write $\mathcal{A}_1[m]$. If $\mathcal{A}_1$ is non-cooperative with respect to $\mathcal{A}_2$ then we write $\mathcal{A}_1[nc_2]$. Throughout, we will often need to describe classes of adversarial behaviors. We refer to such classes as *adversary structures* and describe them as follows. Consider, for example, a three party protocol between players P_1, P_2 and P_3. A protocol with security against the adversary structure

$$\mathsf{Adv} = \left\{ \left(\mathcal{A}_1[m], \mathcal{A}_2[m], \mathcal{A}_3[sh] \right), \left(\mathcal{A}_1[sh], \mathcal{A}_2[sh], \mathcal{A}_3[nc_1, nc_2] \right) \right\},$$

is secure in the following two cases: (1) $\mathcal{A}_1$ and $\mathcal{A}_2$ are malicious while $\mathcal{A}_3$ is semi-honest; and (2) $\mathcal{A}_1$ and $\mathcal{A}_2$ are semi-honest while $\mathcal{A}_3$ is non-cooperating with respect to both $\mathcal{A}_1$ and $\mathcal{A}_2$. We stress that we require the same protocol to be secure for each of the cases in the adversarial structure. This is a stronger guarantee than having separate protocols that address each of the cases since one does not need to know in advance which parties are corrupted in order to choose which protocol to use (as long as it falls in one of the cases of the adversarial structure).

We now present our security definition for non-colluding deviating adversaries. It is based on the real- and idea-world executions defined in section 6.4.

Definition 21 (Security against deviating adversaries)**.** *Let f be a deterministic n-party functionality and Π be an n-party protocol. Furthermore, let $\mathbf{I} \subset [n+1]$ and $\mathrm{C} \subset [n+1]$ be such that $\mathbf{I} \cap \mathrm{C} = \emptyset$ and $|I| = m$ and let Adv be an adversary structure. We say that Π $(\mathbf{I}, \mathrm{C}, \mathsf{Adv})$-securely computes f if there exists a set $\{\mathsf{Sim}_i\}_{i\in[m+1]}$ of* PPT *transformations such that for all* PPT *adversaries $\mathcal{A} = (\mathcal{A}_1, \ldots, \mathcal{A}_{m+1})$ that satisfy Adv, for all $\mathbf{x} \in (\{0,1\}^*)^n$*

and $\mathbf{z} \in (\{0,1\}^*)^{n+1}$*, and for all* $i \in [m+1]$*,*

$$\left\{\text{REAL}^{(i)}_{\Pi,\mathcal{A},\mathbf{I},\text{C},\mathbf{z}}(k,\mathbf{x};\overline{r})\right\}_{k\in\mathbb{N}} \stackrel{c}{\equiv} \left\{\text{IDEAL}^{(i)}_{f,\mathcal{S},\mathbf{I},\text{C},\mathbf{z}}(k,\mathbf{x};\overline{r})\right\}_{k\in\mathbb{N}}$$

where $\mathcal{S} = (\mathcal{S}_1,\ldots,\mathcal{S}_{m+1})$ *and* $\mathcal{S}_i = \mathsf{Sim}_i(\mathcal{A}_i)$ *and where* $\overline{r}$ *is chosen uniformly at random.*

Notice that the standard security definitions of secure MPC in the presence of a semi-honest and malicious adversary can be recovered from Definition 21 by setting $\mathbf{I} = \emptyset$ and letting the adversary in Adv be semi-honest or malicious.

6.5 An Efficient Protocol for Non-Colluding Semi-Honest Parties

In this Section, we describe a slight variation of the protocol by Feige, Killian and Naor from [Feige *et al.*, 1994] (from now referred to as the FKN protocol). The protocol makes use of Yao's garbled circuit construction as a black-box. We provide a high level description of Yao's construction in Appendix D.1.

At a high level, the FKN protocol works as follows. P_1 and P_2 are assumed to share a set of random coins. P_1 then uses these coins to generate a garbling of the circuit, the translation table and a garbling of its own input. P_1 sends the garbled circuit, its garbled input and the translation table to the server. P_2 uses the same coins, but only computes its own garbled input and the translation table. P_2 sends its garbled inputs to the server. The server evaluates the garbled circuit using the garbled inputs, translates the garbled output and returns the evaluation to both parties.

We slightly modify the FKN protocol to adapt it to our setting in which the server is not allowed to learn the output and where we do not assume the parties share a set of random coins. For this purpose, it suffices that (1) the parties execute a coin tossing protocol to generate the random coins; and (2) that P_1 not send the translation table to the server. The server can still evaluate the garbled output which the parties can translate on their own. Intuitively, the privacy and verifiability properties of the garbled circuit construction (see Appendix D.1) and the coin-tossing protocol guarantee that a malicious server cannot

Inputs: P_1's input is x, P_2's input is y, and S has no inputs. All parties know the circuit C which computes the functions f.
Outputs: P_1 and P_2 learn the output $f(x, y)$.

1. P_1 and P_2 execute a coin tossing protocol to generate coins r. As a result, both players learn r.
2. P_1 computes $\mathrm{G}(C) \leftarrow \mathsf{GarbCircuit}(C; r)$, $\mathrm{G}(x) \leftarrow \mathsf{GarbIn}(C, 1, x; r)$ and $\mathrm{T} \leftarrow \mathsf{GarbOut}(r)$. It sends $\mathrm{G}(C)$ and $\mathrm{G}(x)$ to S.
3. P_2 computes $\mathrm{G}(y) \leftarrow \mathsf{GarbIn}(C, 2, y; r)$ and $\mathrm{T} \leftarrow \mathsf{GarbOut}(r)$. It then sends $\mathrm{G}(y)$ to S.
4. S computes $\mathrm{G}(o) \leftarrow \mathsf{Eval}(\mathrm{G}(C), \mathrm{G}(x), \mathrm{G}(y))$ and sends it to P_1 and P_2.
5. P_1 and P_2 separately compute $o \leftarrow \mathsf{Translate}(\mathrm{G}(o), \mathrm{T})$.

Figure 6.1: The (modified) FKN protocol.

return the wrong result or learn anything about the inputs of P_1 and P_2 if he does not collude with either party.

We formally describe this variant of the FKN protocol in Figure 6.1 and in Theorem 11 below we show that it is secure against the following adversary structure:

$$\begin{aligned}\mathsf{Adv}_1 = \Big\{ &\Big(\mathcal{A}_S\big[sh\big], \mathcal{A}_1\big[sh\big], \mathcal{A}_2\big[sh\big]\Big),\\ &\Big(\mathcal{A}_S\big[m\big], \mathcal{A}_1\big[h\big], \mathcal{A}_2\big[h\big]\Big),\\ &\Big(\mathcal{A}_S\big[nc_1, nc_2\big], \mathcal{A}_1\big[sh\big], \mathcal{A}_2\big[sh\big]\Big),\\ &\Big(\mathcal{A}_S\big[h\big], \mathcal{A}_1\big[h\big], \mathcal{A}_2\big[m\big]\Big),\\ &\Big(\mathcal{A}_S\big[sh\big], \mathcal{A}_1\big[sh\big], \mathcal{A}_2\big[nc_S\big]\Big)\Big\}.\end{aligned}$$

Before proving the security of the FKN protocol in our model, we present a simple Lemma that we will use throughout this work and that will simplify our proofs significantly.

Lemma 12. *If a multi-party protocol Π between n players $P_1, \ldots, P_n$, securely computes $f = (f_1, \ldots, f_n)$, (1) in presence of semi-honest and independent parties and (2) in presence of a malicious P_j and honest P_k for $k \in [n] - \{j\}$, then the protocol is also secure in presence of (3) a P_j who is non-cooperative with respect to all other parties who are semi-honest.*

Proof. We need to prove that protocol Π is secure when the adversary A_j corrupting P_j is non-cooperative with respect to all other parties. Since A_j is non-cooperative, and all other parties are semi-honest (and hence non-cooperative), based on Lemma 11, we can assume that A_k is isolated for $k \in [n] - \{j\}$. Hence, we can use the definition of isolation in our simulation of such A_k's.

We first need to provide a simulator Sim_j for simulating a non-cooperative P_j in the ideal world. However, since Π is secure against a malicious P_j when all other parties are honest, we already know that a simulator $\mathsf{Sim}_{j'}$ exists that simulates P_j in that case. Sim_j imitates $\mathsf{Sim}_{j'}$ completely.

We also need to describe a simulator Sim_k for $k \in [n] - \{j\}$. The simulator Sim_k runs the adversary $\mathcal{A}_k$ controlling P_k in the real world, on input x_k. It sends x_k to the trusted party and receives back $f_k(x_1, \ldots, x_n)$. Since $\mathcal{A}_k$ is isolated, according to Definition 20, there exists an emulator $\mathcal{E}_k$ that takes $f_k(x_1, \ldots, x_n)$ as input and can be used to simulate a semi-honest $\mathcal{A}_k$'s output. Sim_k feeds $f_k(x_1, \ldots, x_n)$ to $\mathcal{E}_k$ and plays the role of semi-honest P_k in interaction with it. At the end of this interaction, Sim_k outputs what the semi-honest P_k would, and halts. According to the Definition 20, Sim_k will successfully simulate the output of $\mathcal{A}_k$ in this way. □

□

Since we present all our proofs in the $\mathcal{F}_{\mathsf{CT}}$-hybrid model, we need to make sure that a coin-tossing protocol with security in all adversary structures we consider in fact exists. However, any two-party coin-tossing protocol (between P_1 and P_2) with security against malicious adversaries would be sufficient in our server-aided setting. Such a protocol can easily be proven secure when all three parties are semi-honest and independent. It is also secure, by definition, when either P_1 or P_2 are malicious. It is also secure when the server is malicious and the other two parties are honest since the server is not involved in the protocol, in any way. Finally, Lemma 12 guarantees that the same coin-tossing protocol is also secure in all adversary structures where one party is malicious and the rest are semi-honest and isolated.

We are now ready to state and prove the security of the FKN protocol with respect to Adv_1.

Theorem 11. *The (modified) FKN protocol described in Figure 6.1 securely computes any function f in the $\mathcal{F}_{\mathsf{CT}}$-hybrid model for the adversary structure Adv_1.*

Proof. We consider each case in Adv_1 separately.

Claim 2. *The protocol $\big(\mathcal{A}_S\big[sh\big], \mathcal{A}_1\big[sh\big], \mathcal{A}_2\big[sh\big]\big)$-securely computes f in the $\mathcal{F}_{\mathsf{CT}}$-hybrid model.*

We describe three independent transformations Sim_S, Sim_1 and Sim_2:

- Sim_S simulates $\mathcal{A}_S$ as follows: it computes $(st, \mathrm{G}(C), \mathrm{T}) \leftarrow \mathsf{GarbCircuit}(C)$, $\mathrm{G}(x') \leftarrow \mathsf{GarbIn}(st, C, 1, x')$ and $\mathrm{G}(y') \leftarrow \mathsf{GarbIn}(st, C, 2, y')$ for random x' and y'; and sends $\mathrm{G}(C)$, $\mathrm{G}(x')$ and $\mathrm{G}(y')$ to $\mathcal{A}_S$. If $\mathcal{A}_S$ outputs $\perp$, then Sim_S tells the trusted party to abort. In either case, Sim_S outputs $\mathcal{A}_S$'s entire view.

  The privacy property of garbled circuits (see Definition 42) guarantees that $\mathrm{G}(x)$ and $\mathrm{G}(y)$ are indistinguishable from x' and y' to $\mathcal{A}_S$ who does not know the coins r. In addition, in both the real and ideal execution the semi-honest $\mathcal{A}_S$ does not abort since he is given valid garbled inputs (in the real world this is true since the other two parties are also semi-honest). Therefore, the views of $\mathcal{A}_S$ in the real and the ideal executions are indistinguishable.

- Sim_1 receives x as input and sends it to the trusted party in order to receive $f(x, y)$. It then simulates $\mathcal{A}_1$ as follows. It answers $\mathcal{A}_1$'s $\mathcal{F}_{\mathsf{CT}}$ query by returning random coins r. Sim_1 then computes $(st, \mathrm{G}(C), \mathrm{T}) \leftarrow \mathsf{GarbCircuit}(C; r)$ and uses the translation table to find a garbling $\mathrm{G}(o)$ of $f(x, y)$. Finally, it returns $\mathrm{G}(o)$ to $\mathcal{A}_1$ and outputs $\mathcal{A}_1$'s entire view.

  The view of $\mathcal{A}_1$ consists of the garbled circuits it creates and the garbled outputs it receives. In both the real and the ideal execution he receives the garbled output values corresponding to $f(x, y)$. In the real world this is guaranteed by the fact that S

and P_2 are honest and the correctness property of garbled circuits (see Definition 41). Therefore, the views of $\mathcal{A}_1$ in the real and the ideal executions are indistinguishable.

- Sim_2 works analogously to Sim_1.

□

Claim 3. *The protocol* $\left(\mathcal{A}_S[m], \mathcal{A}_1[h], \mathcal{A}_2[h]\right)$*- securely computes* f *in the* $\mathcal{F}_{\mathsf{CT}}$*-hybrid model.*

Consider the simulator Sim_S that simulates $\mathcal{A}_S$ as follows. It chooses coins r and computes $\mathrm{G}(C) \leftarrow \mathsf{GarbCircuit}(C; r)$. Sim_S chooses random inputs x' for P_1 and y' for P_2. Then he sends $\mathrm{G}(C)$ together with garbled input labels $\mathrm{G}(x') \leftarrow \mathsf{GarbIn}(C, 1, x'; r)$ and $\mathrm{G}(y') \leftarrow \mathsf{GarbIn}(C, 2, y'; r)$ to $\mathcal{A}_S$. Sim_S receives the garbled outputs that $\mathcal{A}_S$ returns for P_1 and P_2. If any of the outputs does not correspond to the correct value, the simulator instructs the trusted party to return $\perp$ to that party. The view of $\mathcal{A}_S$ consists of the garbled circuits and the garbled input values that he receives. The garbled values that correspond to zero and one are indistinguishable for the adversary since he does not know the seed for the PRG (correctness property in definition 41). Therefore the garbled labels for the real inputs x and y in the real execution and the random values x' and y' in the ideal execution are indistinguishable for the $\mathcal{A}_S$. It follows the views of the adversary in the real and the ideal execution are also indistinguishable. The outputs of P_1 and P_2 are also indistinguishable in the real and the ideal execution. They receive the correct output, if $\mathcal{A}_S$ computes and returns the result honestly. Otherwise, in the ideal execution they receive $\perp$ from the trusted party, and in the real execution $\mathcal{A}_S$ cannot produce with all but negligible probability garbled output values for any other output but the correct evaluation of the garbled circuit by the verifiability property of garbled circuits (see Definition 43).

□

Claim 4. *The protocol* $\left(\mathcal{A}_S[nc_1, nc_2], \mathcal{A}_1[sh], \mathcal{A}_2[sh]\right)$*-securely computes* f *in the* $\mathcal{F}_{\mathsf{CT}}$*-hybrid model.*

The proof of this claim is automatically implied given the last two claims and Lemma 12.

□

Claim 5. *The protocol* $(\mathcal{A}_S[h], \mathcal{A}_1[h], \mathcal{A}_2[m])$*-securely computes* f *in the* $\mathcal{F}_{\mathsf{CT}}$*-hybrid model.*

Sim_2 simulates the view of adversary $\mathcal{A}_2$. The simulator answers $\mathcal{A}_2$'s query to $\mathcal{F}_{\mathsf{CT}}$ with random coins r. He receives the garbled values $\mathrm{G}(y) \leftarrow \mathsf{GarbIn}(C, 2, y; r)$ corresponding to the input of $\mathcal{A}_2$. Using the coins r, Sim_2 extracts $\mathcal{A}_2$'s input value y. If extraction fails he returns $\perp$ to $\mathcal{A}_2$. Otherwise, the simulator obtains the output $f(x, y)$ from the trusted party, computes the corresponding garbled output values and sends them to $\mathcal{A}_2$. The view of $\mathcal{A}_2$, which consists of the output he receives, is indistinguishable in the real and the ideal execution. If the garbled input is contructed correctly, in both cases he gets $f(x, y)$ (in the real world this is true since the other two parties are honest). If he submits invalid garbled values for his input, he receives $\perp$ from the simulator in the ideal execution, and in the real world with high probability S will fail to evaluate the circuit and will return $\perp$. Similarly, the output of P_1 will be indistinguishable in the real and the ideal execution.

□

Claim 6. *The protocol* $(\mathcal{A}_S[sh], \mathcal{A}_1[sh], \mathcal{A}_2[nc_S])$*-securely computes* f *in the* $\mathcal{F}_{\mathsf{CT}}$*-hybrid model.*

Once again the above claim is automatically implied given the proof of previous claims and Lemma 12.

□

□

Efficiency. First note that neither P_1, P_2 nor the server S have to perform any public key operations (i.e. no oblivious transfers are needed) with the exception of the initial coin-tossing between P_1 and P_2 which either requires the existence of a secure channel between them or public-key operations. Moreover, the coin-tossing needs to be performed exactly once to share a secret key. In all future runs of the protocol, P_1 and P_2 can use their shared secret key and a pseudorandom function, to generate the necessary seeds. This is a considerable improvement since public key operations are significantly more expensive

compared to secret-key ones. Second, P_2 only needs to do work that is linear in the size of his input and the output, and independent of the circuit size since he only computes the GarbIn, GarbOut and Translate algorithms and the computational cost of these algorithms do not depend on the circuit size. Finally, note that the interaction between P_1 and P_2 is minimal and only takes place in the context of coin tossing (Step 1 of the protocol). In addition, since this interaction is independent of the function f and of the parties' inputs x and y, it can be performed off-line and for many instances of the protocol at once. The server and P_1 will do work that is linear in the size of the circuit. If the server-aided protocol is run multiple times for the same or different functionalities, we can reduce the online work of P_1 by performing the garbling of multiple circuits (for the same or different functions) in the offline phase. The online work for P_1 would then be similar to P_2 and only linear to the size of his input and output.

Extending to multiple parties. Our protocol can be easily extended to multiple parties as follows. All the parties, except for the server, begin by executing a coin-tossing protocol which results in them sharing a set of coins. The coins are then used by one of parties to garble the circuit and all the parties to garble their inputs. The garbled circuit and inputs are then sent to the server who evaluates the circuit and returns the garbled outputs to the parties.

It is not difficult to show that this extended protocol is secure when either: (1) the server is malicious and the other parties are semi-honest; or (2) the server is isolated and semi-honest, the garbler is semi-honest and the remaining parties are malicious.

6.6 Protecting Against Deviating Circuit Garblers

The security of the FKN protocol critically relies on the circuit garbler being honest (or semi-honest) and breaks down completely if this is not the case. To add robustness, we augment the protocol to handle the case where player P_1 deviates from the protocol but is non-cooperative (with respect to S).

Using the semi-honest server for verification. If we assume that the server is semi-honest, there is a simple strategy for protecting against deviating but non-cooperative circuit garblers: similar to the protocol of the previous section, P_1 and P_2 run the coin-tossing protocol and as before P_1 uses the retrieved randomness to generate a garbled circuit. This time, however, P_2 also generates the garbled circuit (using the same randomness) and sends his version of the circuit to the server. The server then verifies that the two garbled circuits he receives are the same, and if so, proceeds with the rest of the computation. Note that as long as one of the players is semi-honest, dishonestly garbled circuits are detected by the honest server. This approach yields a server-aided protocol with a significantly better *average efficiency gain* (see Section 6.9 for an overview of different notions of efficiency gain) compared to the standard two-party Yao protocol with security against malicious adversaries, since it avoids the cut-and-choose steps.

Our goal, however, is to design a protocol that maintains the benefits of the previous protocol (i.e., security against a non-cooperative server) and simultaneously provides protection in cases where the circuit garbler P_1 is non-cooperative with respect to S.

To protect against a non-cooperative P_1, we use the existing cut-and-choose techniques for Yao's garbled circuit protocol (e.g. see [Mohassel and Franklin, 2006; Lindell and Pinkas, 2007]). Note that here the cut-and-choose step cannot take place between P_1 and P_2 since that would significantly increase the work of P_2 and, to a large extent, diminish our ultimate goal of gaining efficiency. Hence, we construct a cut-and-choose protocol between P_1 and an *untrusted* server instead. Note that some subtleties arise that are specific to our server-aided setting and require modifications to the way the cut-and-choose steps are performed.

The computational cost of the resulting protocol increases by a factor of λ (the number of circuits) for the players and the server. However, the new protocol inherits the two important efficiency advantages of the previous one, i.e., the computation still only consists of secret-key operations and P_2's computation is only linear in the size of his input and output and, in particular, is independent of the circuit size.

Standard cut-and-choose. In a standard cut-and-choose, P_1 sends multiple copies of the garbled circuit to S. S then asks P_1 to open the secrets related to a subset of those

circuits. S verifies the correctness of the opened circuits, evaluates the remaining circuits (called evaluation circuits) and outputs the majority result as the final output. Note that it is essential to compute the majority output and not abort immediately after seeing an inconsistent output. As discussed in previous work (e.g. see [Mohassel and Franklin, 2006; Lindell and Pinkas, 2007]), abort in this situation would reveal additional information to a deviating P_1 about P_2's input. Furthermore, to enable P_2 to compute the correct majority output, additional care is needed to make sure P_1 provides the same input to most of the circuits evaluated by S. Avoiding this extra equality-check would undermine both the correctness (by returning the wrong answer to P_2), and the privacy (by allowing P_1 to learn a different function of the inputs) of the protocol. In [Lindell and Pinkas, 2007] and [Mohassel and Franklin, 2006], additional consistency-checking mechanisms are added to the cut-and-choose step in order to guarantee the equality of inputs to most of the circuits. Since the techniques from the two papers are similar, and both would work for our server-aided construction, we give a general description of the mechanism that includes both approaches as a special case. More precisely, in addition to the garbled circuits a collection of *input-equality widgets* are also computed by P_1 and sent along with the garbled circuits to S. During the opening phase of the cut-and-choose, a subset of these input-checking widgets are also opened and verified. This step ensures that unless the majority of P_1's garbled inputs (to the evaluation circuits) are the same, his deviation from the protocol will be detected with high probability during the opening phase.

6.6.1 What goes wrong in the server-aided setting?

We need to address three issues with the above cut-and-choose strategy when it is applied in the server-aided model.

1. First, since P_1 and P_2 independently compute and send to S their garbled inputs for the evaluation circuits, and since we still want to protect against a deviating P_2, P_1 needs to generate the *input-checking widgets* for P_2's input wires as well. This modification could potentially introduce a new security problem. Particularly, for one pair of circuits, P_1 can issue a bad equality-check for a specific bit value of P_2's input wires (e.g. 0) and a correct equality-check for the other bit value (e.g. 1). In the case

that the pair of circuits are chosen for evaluation (which happens with non-negligible probability), if S aborts, P_1 concludes that P_2's input is 0 and if he does not, he concludes that P_2's input bit is a 1. However, this problem is easy to address. In fact, S need not abort if the input keys for two circuits he is evaluating do not pass the checks. He can simply evaluate the subset of remaining circuits that pass the checks and assign "invalid" outputs to the rest (without aborting). The final majority output is also computed, by taking this "invalid" outputs into consideration. In the rest of this section, whenever we talk about retrieving the majority output, we refer to this approach.

2. Second, a more subtle issue arises when the server tries to send the majority output as the final result to the parties. In the standard cut-and-choose, the circuit evaluator learns the actual outputs to all the evaluation circuits and therefore can easily determine the majority. In the server-aided setting, however, we do not allow the server to learn the actual output values. S only learns the garbled outputs and therefore cannot determine the majority output on his own.

   First attempt. Initially, one might try to resolve this issue by sending the computed garbled outputs to P_1 and P_2 and requiring them to compute the majority output on their own (note that P_1 and P_2 know the translation table). Unfortunately, this solution compromises the security of the protocol. For instance, this allows a deviating P_1 to learn "too much" information, by sending only a constant number of bad circuits (e.g. circuits that compute a function other than the agreed one), and learning multiple functions of P_2's input with a non-negligible probability of not getting caught.

   Second attempt. An alternative solution is not to reveal to the server the mapping of output keys to their actual bit values, but to map them to two random values k_0 and k_1 corresponding to 0 and 1, respectively. While this would prevent the server from learning the output of computation, it makes the protocol insecure in the scenario where the server is the dishonest party. In particular, this allows a deviating server to return either k_0 or k_1 as the correct output of computation even if it is not the right

output.

As mentioned earlier, we are interested in a solution that allows a semi-honest server to compute the majority output without learning the output itself (output privacy) and at the same time can provide a guarantee that a deviating server would not be able to modify the result of the computation.

Our oblivious cut-and-choose method. The high level idea behind our solution is as follows. After the opening phase, $\lambda/2$ unopened circuits remain. For each output wire, P_1 and P_2 generate two polynomials g_0 and g_1 of degree $\lambda/4$ over the finite field $GF(2^\lambda)$. g_0 and g_1 are chosen uniformly from the space of *permutation polynomials* of a special form which we will discuss shortly. Let $k_0 = g_0(0)$ and $k_1 = g_1(1)$. k_0 and k_1 are used as keys corresponding to 0 and 1, respectively.

P_1 and P_2 evaluate g_0 at keys corresponding to 0 in the $\lambda/2$ evaluation circuits, and g_1 at keys corresponding to 1. These evaluations along with two ciphertexts $c_0 = E_{k_0}(0^\lambda)$ and $c_1 = E_{k_1}(0^\lambda)$ are sent to S and the same process is repeated for all output wires.

These evaluations can be seen as Reed-Solomon encoding of the keys k_0 and k_1. S then evaluates the remaining $\lambda/2$ circuits and uses a Reed-Solomon decoding algorithm to recover one of k_0, or k_1 (S obliviously decodes both but only one of the keys decrypts the corresponding c_i to 0^λ). Let b be the correct output for the wire we are discussing. In case of a deviating P_1, the error correcting property of the Reed-Solomon codes ensures that as long as a "sufficiently large" fraction of the garbled outputs are correct, the decoding algorithm correctly decodes the correct key k_b (and hence this fulfils the servers' search for the majority output). On the other hand, in the adversarial scenario where the server is dishonest, we argue that he does not learn anything about k_{1-b}. This is exactly where we need g_0 and g_1 to be randomly chosen permutation polynomials (Dickson polynomials) of a special form. Intuitively, if g_{1-b} is a permutation over the finite field, the knowledge of its evaluations at uniformly random evaluation points does not reveal any information about the polynomial itself (and specifically its constant coefficient) to S since all the permutation polynomials of the same form as g_{1-b} can be evaluated to the same values given the right evaluation

points, and hence are equally likely to have been chosen. This intuition is formalized in the proof of security for the case when the server is deviating.

How to sample g_0 and g_1. A Dickson polynomial of degree n denoted by $D_n(x, \alpha)$ is given by

$$D_n(x,\alpha) = \sum_{i=0}^{\lfloor n/2 \rfloor} \frac{n}{n-i}\binom{n-i}{i}(-\alpha)^i x^{n-2i}$$

Dickson polynomials have the nice property of acting as permutations of finite fields. More accurately, we have the following lemma about them.

Lemma 13. *The Dickson polynomial $D_n(x, \alpha)$ is a permutation polynomial for a finite field of size q if n is coprime to $q^2 - 1$.*

Hence, in order for g_0 and g_1 to be permutation polynomials, we require that $\lambda/4$ is coprime to $2^{\lambda+1} - 1$. In addition, for g_0 and g_1 to have constant coefficients, $\lambda/4$ needs to be even (due to the way Dickson polynomials are defined). Finally, we require that if α is chosen uniformly at random in $GF(2^\lambda)$, the constant coefficient of $D_{\lambda/4}(x, \alpha)$ is a uniformly random element of the field too. The reason for this last requirement is so that we can use the constant coefficients as random keys k_0 and k_1 in the protocol. Since the constant coefficient of $D_{\lambda/4}(x, \alpha)$, for an even value of $\lambda/4$, is of the form $2\alpha^{\lambda/8}$, we essentially need that $\alpha^{\lambda/8}$ be a permutation over the field as well. Once again, according to Lemma 13 this is case if $\lambda/8$ is coprime to $2^{\lambda+1} - 1$ since $D_{\lambda/8}(x, 0) = x^{\lambda/8}$ is itself a Dickson polynomial.

To summarize, we sample g_0 and g_1 by choosing λ such that $\lambda/4$ is even and coprime to $2^{\lambda+1} - 1$ (this already implies that $\lambda/8$ is coprime to $2^{\lambda+1} - 1$); generating two random elements $\alpha_0, \alpha_1 \in GF(2^\lambda)$; and letting $g_0 = D_{\lambda/4}(x, \alpha_0)$ and $g_1 = D_{\lambda/4}(x, \alpha_1)$.

Finally, we note that choosing a λ that satisfies the above properties is fairly easy. For example, any $\lambda = 8p$ where p is a prime number is coprime to $2^{\lambda+1} - 1$ and hence can be used in our protocol. To obtain a λ close to 80, one could let $p = 11$. For a λ close to 128, once could let $p = 17$, and so on.

3. A third issue in our setting is that a deviating P_1 can cheat by agreeing on a seed r_i with P_2 but using a different seed r'_i to generate the garbled circuit which he sends

to S. More specifically, consider the two garbled circuits $\mathrm{G}_i(C) \leftarrow \mathsf{GarbCircuit}(C; r_i)$ and $\mathrm{G}'_i(C) \leftarrow \mathsf{GarbCircuit}(C; r'_i)$ generated by the two different seeds. A dishonest P_1 can potentially choose the seed r'_i such that for a particular input wire of P_2, the key k corresponds to a 0 in $\mathrm{G}_i(C)$ but corresponds to a 1 in $\mathrm{G}'_i(C)$. In other words, P_1 can flip P_2's input bit without P_2 or S detecting it. This issue does not arise in the standard two-party variant of Yao's protocol against malicious adversaries, due to the existence of the OTs. In the two-party case, P_1 and P_2 engage in a series of OTs (before the cut-and-choose step) as a result of which P_2 learns his garbled inputs for all circuits. In the opening phase, P_2 can verify that the OTs were performed honestly for the opened circuits, and hence gain confidence about the correctness of his garbled inputs in the majority of the unopened circuits too. Since we no longer invoke OTs in our server-aided protocols, we need a different mechanism for resolving this issue.

First attempt. One may try to solve this problem by making S ask both P_1 and P_2 to send him the seeds for the opened circuits, and verify that the two sets of seeds are equal. However, this creates a new vulnerability for the case when S is dishonest since he could ask for two different subset of circuits to be opened by each of P_1 and P_2. This allows S to learn either P_1 or P_2's input values.

The correct solution is to have S send the set of seeds he receives from P_1, to P_2 who can verify that they are equal to his own seeds and abort the protocol, otherwise (see step 5 of the protocol).

In Theorem 12 below we show that our protocol is secure against the following adversary structure:

$$\mathsf{Adv}_2 = \mathsf{Adv}_1 \bigcup \left\{ \left(\mathcal{A}_S[h], \mathcal{A}_1[m], \mathcal{A}_2[h] \right), \left(\mathcal{A}_S[sh], \mathcal{A}_1[nc_S], \mathcal{A}_2[sh] \right) \right\}.$$

Theorem 12. *The protocol in Figure 6.2 securely computes the function f in the $\mathcal{F}_{\mathsf{CT}}$-hybrid model against the adversary structure Adv_2.*

Proof. The proof for the first adversarial model of Adv_1 (where all three parties are semi-honest) is almost identical to the proof for the same model in Theorem 11 and hence is omitted here. Next, we prove security against the remaining adversarial scenarios in the

Inputs: P_1's input is x, P_2's input is y, and S has no inputs. All parties know the circuit C which computes the functions f. They also agree on an integer λ such that $\lambda/8$ is coprime to $2^{\lambda+1}-1$.
Outputs: P_1 and P_2 learn the output $f(x,y)$.

1. P_1 and P_2 execute a coin tossing protocol to generate $\lambda+1$ sets of coins $r_1,\ldots,r_\lambda$, and r'.
2. P_1 computes λ garbled circuits $(\mathrm{G}_i(C) \leftarrow \mathsf{GarbCircuit}(C;r_i)$ and garbled inputs $\mathrm{G}_i(x) \leftarrow \mathsf{GarbIn}(C,1,x;r)$ for $1 \le i \le \lambda$. λ is chosen such He sends the computed garbled circuits and inputs to S. He also sends the *input-equality widgets* corresponding to those garbled circuits to S.
3. S randomly chooses a subset $T \subset [1\ldots\lambda]$ of size $\lambda/2$ and asks P_1 to send the coins r_i for $i \in T$, and also reveal the secrets or the equality-checkers corresponding to the opened circuits.
4. S verifies that the opened circuits and input-equality widgets were generated correctly. If the verification fails for any circuit, S aborts.
5. S also sends the coins r_i for $i \in T$ to P_2 who checks if the coins are what he agreed on with P_1 in the coin-tossing phase or not. If not, he aborts the protocol.
6. P_1 and P_2 separately send their garbled inputs for the remaining circuits to S. S uses the remaining input-equality widgets to determine the subset of the circuits in S that pass the input-equality checks. In what follows, the evaluation result for those circuits that fail the checks is set to "invalid" by the server (without aborting).
7. At this point, $t = \lambda/2$ circuits remain to be evaluated. Renumber the remaining circuits as $C_1,\ldots,C_t$. Denote by ℓ_o the number of output wires in each circuit, and let $(w^i_{1,0},w^i_{1,1}),\cdots,(w^i_{t,0},w^i_{t,1})$ the keys corresponding to the output wire $1 \le i \le \ell_o$, in the t circuits. For all $1 \le i \le \ell_o$, P_1 and P_2 use the coins r' to independently generate the following

   - Two uniformly random elements $\alpha^i_0,\alpha^i_1 \in GF(2^\lambda)$, and the corresponding Dickson polynomials $g^i_0(x) = D_{\lambda/4}(x,\alpha^i_0)$, and $g^i_1(x) = D_{\lambda/4}(x,\alpha^i_1)$ over $GF(2^\lambda)$. Denote their corresponding constant coefficients by k^i_0 and k^i_1, respectively.
   - P_1 and P_2 also compute two ciphertexts $c^i_0 = E_{k^i_0}(0^\lambda)$ and $c^i_1 = E_{k^i_1}(0^\lambda)$ using a symmetric-key encryption scheme E.

Figure 6.2: A server-aided two-party protocol robust against a deviating P_1

adversary structure.

Claim 7. *The protocol $\big(\mathcal{A}_S[m],\mathcal{A}_1[h],\mathcal{A}_2[h]\big)$-securely computes f in the $\mathcal{F}_{\mathsf{CT}}$-hybrid model.*

8. P_1 and P_2 then independently compute $Y_0^i = (g_0^i(w_{1,0}^i), \ldots, g_0^i(w_{t,0}^i))$ and $Y_1^i = (g_1^i(w_{1,1}^i), \ldots, g_1^i(w_{t,1}))$ for $1 \leq i \leq \ell_o$ and send $((Y_0^i, c_0^i), (Y_1^i, c_1^i))$ to S. Each pair will be sent in a randomly permuted order so that S does not learn which one corresponds to 0 and which one corresponds to 1. The randomness for the permutation will also be derived from the coins r', and hence P_1 and P_2 will both do the same permutations.
9. S checks that the permuted $((Y_0^i, c_0^i), (Y_1^i, c_1^i))$ pairs he receives from P_1 and P_2 are in fact the same. If not, he aborts. Else, S evaluates the remaining t circuits and for each output wire $1 \leq i \leq \ell_o$, retrieves the keys $X_b^i = (w_{1,b}^i, \ldots, w_{t,b}^i)$ for a bit $b \in \{0, 1\}$ where b is the actual output value for wire i. (With high probability, only a *small* fraction of these keys will be corrupted or "invalid" or else S would catch them in the opening phase).
10. For every $1 \leq i \leq \ell_o$, S runs the Reed-Solomon decoding algorithm on both pairs (X_b^i, Y_0^i) and (X_b^i, Y_1^i) to recover the decodings d_0^i and d_1^i. S uses d_0^i to decrypt c_0^i and uses d_1^i to decrypt c_1^i. With high probability, only the decryption with d_b^i returns the message 0^λ. S returns d_b^i for $1 \leq i \leq \ell_o$ to P_1 and P_2.
11. P_1 and P_2 use their translation tables to separately recover the actual output values.

Figure 6.3: Figure 6.2 Continued

Consider the simulator Sim_S that simulates $\mathcal{A}_S$ as follows. It chooses coins $r_1, \ldots, r_\lambda, r'$ and computes $\mathrm{G}_i(C) \leftarrow \mathsf{GarbCircuit}(C; r_i)$ for $1 \leq i \leq \lambda$. Sim_S then sends $\mathrm{G}_i(C)$ for all $1 \leq i \leq \lambda$ to $\mathcal{A}_S$. $\mathcal{A}_S$ returns a subset T of size $\lambda/2$. For each $i \in T$, Sim_S returns r_i to $\mathcal{A}_S$.

Sim_S then chooses random inputs x' for P_1 and y' for P_2. He then sends the garbled input labels $\mathrm{G}_i(x') \leftarrow \mathsf{GarbIn}(C, 1, x'; r_i)$ and $\mathrm{G}_i(y') \leftarrow \mathsf{GarbIn}(C, 2, y'; r_i)$ to $\mathcal{A}_S$ for all $i \in S - T$. Denote by ℓ_0, the number of output wires in each circuit. For each $1 \leq i \leq \ell_o$, Sim_S use the seed r' to generate two random Dickson polynomials g_0^i and g_1^i of degree $\lambda/4$, and evaluates them at the $\lambda/2$ keys for output wire i. Denote the constant coefficients of g_0^i and g_1^i by k_0^i and k_1^i respectively. Sim_S then sends the evaluations along with the ciphertexts $c_0^i = E_{k_0^i}(0^\lambda)$ and $c_1^i = E_{k_1^i}(0^\lambda)$ on behalf of P_1 and P_2 to $\mathcal{A}_S$. For each $1 \leq i \leq \ell_o$, Sim_S receives a key k_b^i. If k_b^i is not a valid key, the simulator instructs the trusted party to return $\perp$ to P_1 and P_2. Sim_S then outputs whatever $\mathcal{A}_S$ does and halts.

The view of $\mathcal{A}_S$ consists of the garbled circuits, the opened seeds, the garbled input values, and evaluations of the Dickson polynomials at the output keys. The garbled input values that correspond to zero and one are indistinguishable for the adversary since he

does not know the seed for the unopened circuits (correctness property in definition 41). Therefore the garbled labels for the real inputs x and y in the real execution and the random values x' and y' in the ideal execution are indistinguishable for $\mathcal{A}_S$. All the other messages $\mathcal{A}_S$ receives are the same things he would see in the real protocol and hence it follows the views of the adversary in the real and the ideal execution are therefore indistinguishable.

The remaining issue is to prove that the outputs of P_1 and P_2 are also indistinguishable in the real and the ideal execution. Note that they receive the correct output, if $\mathcal{A}_S$ computes and returns the result, honestly. Otherwise, in the ideal execution they receive $\perp$ from the trusted party. In the real execution, for $\mathcal{A}_S$ to return an incorrect output key k^i_{1-b} for a specific output wire i, he needs to guess the constant coefficient of the corresponding polynomial g^i_{1-b}. But, the only knowledge $\mathcal{A}_S$ has of this polynomial is the fact that it is a uniformly random Dickson polynomial of degree $\lambda/4$ that evaluates to values $y_1, \ldots, y_{\lambda/2}$ at $\lambda/2$ uniformly random points that are unknown to him. The fact that these evaluation points are unknown to $\mathcal{A}_S$ is implied by the verifiability property of Yao's garbled circuit (see Definition 43).

It remains for us to show that seeing $y_1, \ldots, y_{\lambda/2}$ does note leak any information about the underlying polynomial g^i_{1-b} (and particularly its constant coefficient). However, since g^i_{1-b} is a permutation and the evaluation points are uniformly random values from $GF(2^\lambda$, y_i's essentially constitute $\lambda/2$ uniformly random values in $GF(2^\lambda)$, and hence do not contain any information about g^i_{1-b}. Put differently, for any Dickson polynomials of degree $\lambda/4$ $p(x)$, there exist a unique set of evaluation points $x_1, \ldots, x_{\lambda/2}$ such that $p(x_i) = y_i$ for $1 \leq i \leq \lambda/2$. This completes our argument.

□

Claim 8. *The protocol $\big(\mathcal{A}_S[h], \mathcal{A}_1[m], \mathcal{A}_2[h]\big)$-securely computes f in the $\mathcal{F}_{\mathsf{CT}}$-hybrid model.*

We construct a simulator Sim_1 for $\mathcal{A}_1$. Before describing the simulation, however, we give a Lemma implied by the security of the cut-and-choose variants of Yao's protocol (we refer the reader to [Lindell and Pinkas, 2007] where the Lemma is implicitly proved). We point out, however, that in our setting, the role of the honest verifying party is divided between an honest server and an honest P_2, and hence the Lemma goes through since their

verification mechanisms combined is equivalent to the verification mechanism performed by the single honest party in the standard two-party case of malicious Yao. In particular, it is essential for both the server to check the correctness of the opened circuits and the input-equality widgets in steps 3 and 4, and for the honest P_2 to check the correctness of the revealed seeds in step 5. This point is also dircetly related to the third issue we explored in the discussion above.

Lemma 14 ([Lindell and Pinkas, 2007])**.** *There exists an expected polynomial time extractor* Ext*, that takes* P_1*'s input* x *and runs and rewinds* $\mathcal{A}_1$*. If* $\mathcal{A}_1$ *creates more than* $3/4$ *of the evaluation circuits honestly (a total of* $3\lambda/8$ *circuits), and provides the same input* x' *for all of those correct circuits, w.h.p.,* Ext *extracts and outputs* x'*. Else,* Ext *will output* $\perp$ *with all but negligible probability. Furthermore, the probability of* Ext *outputting* $\perp$ *when interacting with* $\mathcal{A}_1$ *is exactly the same as the probability of an honest server outputting* $\perp$ *in the real protocol.*

The fraction 3/4 in the above Lemma is adjustable to any constant fraction greater than 1/2. Changing the constant fraction affects the extractor's probability of error in the Lemma, but that probability still remains negligible in the security parameter. In order to ensure correct decoding, we need a constant fraction of 3/4 or higher in our proofs.

Our transformation Sim_1 works as follows:

1. it makes a query to $\mathcal{F}_{\mathsf{CT}}$, and $\mathcal{A}_1$ answers back with $r_1, \ldots, r_\lambda$, and r'.

2. it runs the extractor Ext from Lemma 14 to either obtain the input x' that $\mathcal{A}_1$ has provided for the majority of the circuits, or to receive the abort signal $\perp$. For the latter, Sim_1 simulates the server aborting and outputs whatever $\mathcal{A}_1$ does.

3. it uses r' to generate the permutation polynomials $g_0^i(x)$ and $g_1^i(x)$ with constant coefficients $k_0^i, k_1^i \in \{0,1\}^\lambda$, for $1 \leq i \leq \ell_o$.

4. it sends x' to the trusted party and obtains $z = f(x', y)$. For bits b_i of the output z, he selects the corresponding key $k_{b_i}^i$, (for $1 \leq i \leq \ell_o$) and returns them to P_1.

We show that $\mathcal{A}_1$ cannot distinguish his interaction with an honest S and P_2 in the real world from his interaction with Sim_1 in the ideal world. The view of the adversary consists

of the garbled circuits he submits and the output he receives. Therefore, it is enough to show that the output that $\mathcal{A}_1$ receives in the real execution is the same as what he receives in the above simulation. Based on the existence of the extractor Ext from Lemma 14, we are assured that in case of an abort, the view of $\mathcal{A}_1$ in the simulation is indistinguihsable from when he interacts with the honest server. Furthermore, in the case that Ext extracts an input x', we know that with high probability 3/4 of the evaluation circuits (i.e., $3\lambda/8$ circuits) constructed by $\mathcal{A}_1$ are correct and use the same input x'. Let $z = f(x', y)$. We now have that for every bit b_i of z, the honest server in the real execution, receives $\lambda/2$ evaluation points for polynomial g_{b_i} of degree $\lambda/4$ and that $3\lambda/8$ of those are correct. In other words, the error in the Reed-Solomon codeword is less than $\lambda/8 < (\lambda/2 - \lambda/4)/2$. Hence, the Reed-Solomon decoding algorithm run by the honest server unambiguously recovers the garbled inputs k_{b_i} for all $1 \leq i \leq \ell_o$. Since the server runs the decoding obliviously on both polynomials, however, we also need to show that for polynomials g_{1-b_i}, the decoding will always fail to return a valid key k_{1-b_i}. However, this is the case since for each polynomial g_{1-b_i} at most $\lambda/8 = \lambda/2 - 3\lambda/8$ of the evaluation points are correct while the degree of the polynomial is $\lambda/4$. Hence, the RS decoding algorithm either fails or returns a key $k \neq k_{1-b_i}$ for all $1 \leq i \leq \ell_o$. Consequently, S will not be able to correctly decrypt c^i_{1-b} to the message 0^λ using the key k. This completes our argument.

□

□

Claim 9. *The protocol* $\big(\mathcal{A}_S[h], \mathcal{A}_1[h], \mathcal{A}_2[m]\big)$*-securely computes* f *in the* $\mathcal{F}_{\mathsf{CT}}$*-hybrid model.*

The simulation for this case is very similar to the case of malicious P_1. In fact, note that P_2 can only perform a subset of cheating strategies of P_1, by sending bad or inconsistent garbled inputs (but not garbled circuits). Hence, a simpler variant of the extractor Ext in Lemma 14 can be used to simulate $\mathcal{A}_2$ in the ideal world. Given Ext, the remainder of Sim_2's strategy will be identical to that of Sim_1 described above, and the same analysis goes through.

□

Claim 10. *The protocol* $\left(\mathcal{A}_S\left[sh\right], \mathcal{A}_1\left[nc_S\right], \mathcal{A}_2\left[sh\right]\right)$*-securely computes* f *in the* $\mathcal{F}_{\mathsf{CT}}$*-hybrid model.*

The proof of this claim is automatically implied given the proofs of the above claims and Lemma 12.

□

□

6.6.2 Extending to Multiple Parties

We describe, at a high level, how our protocol can be extended to the multi-party setting. All parties engage in the coin-tossing protocol and learn the necessary seeds. Then, the circuit garbler and the server proceed as they would in the the two-party case. All other parties (clients) who need to send their garbled inputs to the server, engage in a MPC protocol between themselves and the server, wherein their inputs is the seeds they hold and their own input keys, and the server's output is the input key for all parties. Note that this MPC protocol needs to be secure against non-cooperative parties (except for one semi-honest party and a semi-honest server), and hence, the naive solution of having each party send their input key to the server would not be sufficient. Nevertheless, due to the particular setting we work in, a very efficient construction for implementing this MPC protocol exists, but we defer a more detailed description of this step to a more complete version.

In addition to sending their garbled inputs, we require all the parties to send the server the evaluations of the polynomials g_i^0 and g_i^1 at the garbled output values of the garbled circuits used in the protocol. The server then checks that all the parties sent the same evaluations for g_i^0 and g_i^1 and if so, uses those values for the interpolation of the garbled outputs. Otherwise, the server aborts.

In addition to security against Adv_1, this protocol also provides security when all but one of the parties are non-cooperative with respect to the server and the server and a single party are semi-honest. Furthermore, if we modify the coin tossing protocol to include the server but without providing him with the result of the coin-toss, the protocol can also handle an adversarial structure where all parties (except for the server) are non-cooperative with

respect to the server. We note that an adversarial structure where *all* parties (except the server) are non-cooperative is meaningful since this corresponds to cases where each party can deviate from the protocol but does not share any private information or coordinate its behavior with other parties.

6.7 Server-Aided Computation From Delegated Computation

We describe a general construction for server-aided two-party computation based on any non-interactive delegated computation scheme and any secure two-party computation protocol. The resulting server-aided construction inherits the efficiency of the underlying protocols.

Informally, a delegated computation scheme allows a client to outsource the computation of a circuit C on a private input x to an untrusted worker such that: (1) the client's work is substantially smaller than evaluating $C(x)$ on his own; (2) the worker does not learn any information about the client's input or output; and (3) the worker cannot return an incorrect answer without being detected. The notion of secure delegated computation is closely related to that of verifiable computation [Goldwasser *et al.*, 2008; Gennaro *et al.*, 2010; Chung *et al.*, 2010], with the additional requirement that the client's input and output remain private.

Our protocol. Our protocol, described in detail in Figure 6.4, works as follows. Let $\mathsf{Del} = (\mathsf{Gen}, \mathsf{ProbGen}, \mathsf{Compute}, \mathsf{Verify})$ be a secure delegated computation scheme. The parties P_1 and P_2 use Del to outsource the evaluation of f on their combined inputs (i.e., the concatenation of x and y) to the server. However, since P_1 and P_2 do not want to reveal their inputs to each other, they use secure two-party computation to simulate the client in a delegated computation interaction. More precisely, they run the Gen, $\mathsf{ProbGen}$ and Verify algorithms of Del via secure two-party computation. The server S performs the same functionality as that of the worker in the delegated computation interaction. Put differently, we use Del to outsource a two-party computation protocol between P_1 and P_2

to the server S.

Inputs P_1's input is x and P_2's input is y. Server S does not have any input.
Output P_1 and P_2 want to learn the function F on their inputs. Without loss of generality we assume that F takes x and y as one concatenated input, i.e., the parties are computing $F(x||y)$.

1. P_1 and P_2 use the ideal functionality $\mathcal{F}_{\mathsf{2pc}}^{f}$, where f takes as input x and y, computes $(\text{PK}, \mathsf{sk}) \leftarrow \mathsf{Gen}(1^{\mathsf{k}}, \mathsf{F})$ and $(\sigma_{x||y}, \tau_{x||y}) \leftarrow \mathsf{ProbGen}_{\mathsf{sk}}(x||y)$ and outputs

$$\Big(\big(\text{PK}, \sigma_{x||y}, \tau^1_{x||y}, \mathsf{sk}^1\big), \big(\text{PK}, \sigma_{x||y}, \tau^2_{x||y}, \mathsf{sk}^2\big) \Big),$$

   where $\tau_{x||y} = \tau^1_{x||y} \oplus \tau^2_{x||y}$ and $\mathsf{sk} = \mathsf{sk}^1 \oplus \mathsf{sk}^2$. In essence P_1 and P_2 both learn PK and $\sigma_{x||y}$ while they each only learn a random share of $\tau_{x||y}$ and sk.

2. P_1 and P_2 each send $\sigma_{x||y}$ and PK to the server S but keep their shares of $\tau_{x||y}$ to themselves. Server checks to see that the two values received from the parties are the same and aborts otherwise.

3. Server S computes $\mathsf{Compute}_{\text{PK}}(\sigma_{x||y}) \rightarrow \sigma_z$ where $z = F(x||y)$ and sends σ_z to P_1 and P_2.

4. P_1 and P_2 use the ideal functionality $\mathcal{F}_{\mathsf{2pc}}^{g}$ where g takes as input $\tau^1_{x||y}, \tau^2_{x||y}, \mathsf{sk}^1$ and sk^2 and outputs (z, z) where $z = \mathsf{Verify}_{\mathsf{sk}^1 \oplus \mathsf{sk}^2}((\tau^1_{x||y} \oplus \tau^2_{x||y}), \sigma_z)$. P_1 and P_2 output z as their final output.

Figure 6.4: A server-aided two-party protocol from any delegated computation scheme

Intuitively, our protocol is secure since: (1) the verifiability of Del guarantees that a malicious server cannot change the outcome of the computation; and (2) the privacy property Del guarantees that the server cannot learn any useful information about the parties' inputs. Furthermore, even if a malicious server colludes with either P_1 or P_2, the colluding parties will not learn any information about the remaining party's input since the interaction between P_1 and P_2 is done via a secure two-party computation. We prove this intuition in Theorem 13 using the following adversary structure:

$$\mathsf{Adv}_4 = \Big\{ \Big(\mathcal{A}_S\big[sh\big], \mathcal{A}_1\big[sh\big], \mathcal{A}_2\big[sh\big] \Big), \\ \Big(\mathcal{A}_S\big[m\big], \mathcal{A}_1\big[h\big], \mathcal{A}_2\big[h\big] \Big), \\ \Big(\mathcal{A}_S\big[nc_1, nc_2\big], \mathcal{A}_1\big[sh\big], \mathcal{A}_2\big[sh\big] \Big) \Big\}.$$

Theorem 13. *If* Del *is secure, then the server-aided two-party protocol described in Figure 6.4 is secure in the* $\mathcal{F}_{\mathsf{2pc}}$*-hybrid model against the adversary structure* Adv_4*.*

Proof. We sketch the proofs for each item in Adv_4 separately.

Claim 11. *The protocol* $\big(\mathcal{A}_S\big[sh\big], \mathcal{A}_1\big[sh\big], \mathcal{A}_2\big[sh\big]\big)$*-securely computes* F *in the* $\mathcal{F}_{\mathsf{2pc}}$*-hybrid model.*

We describe three independent transformations Sim_S, Sim_1 and Sim_2:

- Sim_S runs $\mathcal{A}_S$. Sim_S then generates two arbitrary inputs x', y', computes $(\text{PK}, \mathsf{sk}) \leftarrow \mathsf{Gen}(1^{\mathsf{k}}, \mathsf{F})$ and $(\sigma_{x'||y'}, \tau_{x'||y'}) \leftarrow \mathsf{ProbGen}_{\mathsf{sk}}(x'||y')$ and sends $\sigma_{x'||y'}$ to $\mathcal{A}_S$. The privacy property of Del ensures that $\mathcal{A}_S$'s view is indistinguishable from its view in the real-world execution with semi-honest P_1 and P_2 (where they use their real inputs x and y). At some point, $\mathcal{A}_S$ sends the output σ_z. Sim_S then outputs whatever $\mathcal{A}_S$ does and halts. Since, $\mathcal{A}_S$ is semi-honest in this case, this will be the correct output.

- Sim_1 runs $\mathcal{A}_1$. Note that since we prove the security of the protocol in the $\mathcal{F}_{\mathsf{2pc}}$-hybrid model, $\mathcal{A}_1$ will send his input x to $\mathcal{F}_{\mathsf{2pc}}$. Sim_1 forwards x to the trusted party of the ideal execution and gets back $F(x, y)$. Sim_1 generates an arbitrary input y' for P_2, runs $(\text{PK}, \mathsf{sk}) \leftarrow \mathsf{Gen}(1^{\mathsf{k}}, \mathsf{F})$, and runs $\mathsf{ProbGen}_{\mathsf{sk}}(x||y')$ to compute $\sigma_{x||y'}, \tau_{x||y'}$. He then sends $\sigma_{x||y'}$ and random values $\tau^1_{x||y'}$ and sk^1 to $\mathcal{A}_1$. The privacy property of Del guarantees that in $\mathcal{A}_1$'s view, $\sigma_{x||y'}$ is indistinguishable from $\sigma_{x||y}$ for any y and y'. The same is true for $\tau^1_{x||y'}$ and sk^1 which are simply random shares. It is worth noting that the privacy property of Del requires that $\sigma_{x||y'}$ hide all *partial information* about the encoded input. Therefore, the tuple $(x, \sigma_{x||y'})$ is also indistinguishable form the tuple

$(x, \sigma_{x||y})$ for any y. Hence, we safely assume that $\mathcal{A}_1$'s view so far is indistinguishable from his view in the real execution with semi-honest P_2 and S.

Sim_1 then computes $\sigma_{z'} \leftarrow \mathsf{Compute}_{\mathrm{PK}}(\sigma_{x||y'})$ and sends $\sigma_{z'}$ to $\mathcal{A}_1$ on behalf of the server. Note that for the same reason as above and due to the privacy property of Del, $\mathcal{A}_1$ cannot distinguish $\sigma_{z'}$ from σ_z. $\mathcal{A}_1$ eventually sends sk^1 and $\tau^1_{x||y}$ as his input to the trusted party of the $\mathcal{F}_{\mathsf{2pc}}$-functionality for the second run of MPC that runs the Verify function. Since $\mathcal{A}_2$ is semi-honest, Sim_1 simply returns the value $F(x, y)$ that he received from the trusted party of the ideal-world execution to $\mathcal{A}_1$. Sim_1 then outputs whatever $\mathcal{A}_1$ does and halts.

- Sim_2's strategy is identical to Sim_1's since their roles in the protocol are symmetric.

□

Claim 12. *The protocol $\big(\mathcal{A}_S[m], \mathcal{A}_1[h], \mathcal{A}_2[h]\big)$-securely computes F in the $\mathcal{F}_{\mathsf{2pc}}$-hybrid model.*

We describe a transformation Sim_S for the adversary $\mathcal{A}_S$. Note that since S does not have any inputs to the protocol, there is no need for input extraction during the simulation. Sim_S only needs to simulate $\mathcal{A}_S$'s view correctly and make sure that in the case of an abort, or other types of cheating by $\mathcal{A}_S$, P_1 and P_2's output in the ideal execution is an abort as well.

Sim_S runs $\mathcal{A}_S$. Sim_S then generates two arbitrary inputs x', y', computes $(\mathrm{PK}, \mathsf{sk}) \leftarrow \mathsf{Gen}(1^{\mathsf{k}}, \mathsf{F})$ and $(\sigma_{x'||y'}, \tau_{x'||y'}) \leftarrow \mathsf{ProbGen}_{\mathsf{sk}}(x'||y')$ and sends $\sigma_{x'||y'}$ to $\mathcal{A}_S$. The privacy property of Del ensures that $\mathcal{A}_S$'s view is indistinguishable from his view in the real execution with honest P_1 and P_2 (where they use their real inputs). At some point, $\mathcal{A}_S$ will either abort or send the output $\sigma_{z'}$ to the two parties. Sim_S computes $z' \leftarrow \mathsf{Verify}_{\mathsf{sk}}(\tau_{x'||y'}, \sigma_{z'})$. If $z' = \perp$, Sim_S sends an *abort* message to the trusted party and simulates P_1 and P_2 aborting. Sim_S then outputs whatever $\mathcal{A}_S$ does and halts.

Note that the verifiability property of Del (see definition 45) ensures that if $z' \neq F(x', y')$ then $z' = \perp$ with high probability. Also note that the probability that $z' = \perp$ in the simulation with inputs x' and y' is (all but negligibly) close to the same probability for any

other inputs including inputs x and y of P_1 and P_2 in the real execution. If this was not the case, once again we could use $\mathcal{A}_S$ to break the privacy property of Del. These two facts combined demonstrate that the joint distribution of outputs of $\mathcal{A}_S$, P_1 and P_2 in the real execution are computationally indistinguishable from those of Sim_S, Sim_1 and Sim_2 in the ideal execution.

□

Claim 13. *The protocol* $\left(\mathcal{A}_S\left[nc_1, nc_2\right], \mathcal{A}_1\left[sh\right], \mathcal{A}_2\left[sh\right]\right)$*-securely computes* F *in the* $\mathcal{F}_{\mathsf{2pc}}$*-hybrid model.*

The proof of this claim follows from the previous two claims and Lemma 12.

□

Efficiency. Note that in the above protocol P_1 and P_2 only simulate the client in the delegated computation interaction and not the server. Using general-purpose protocols such as Yao's protocol [Yao, 1982], the computation performed by P_1 and P_2 will be linear in that of client in the delegated computation interaction. Given the efficiency properties of delegated computation, this will be significantly lower than P_1 and P_2 running their own secure two-party computation protocol to compute F on their private inputs. The server's computation is identical to that of the worker in the delegated computation interaction.

We know of two non-interactive delegated computation schemes in the literature. The first is the construction of [Gennaro *et al.*, 2010] based on Yao's garbled circuits and fully homomorphic encryption. When instantiated using their protocol, P_1 and P_2 would need perform $O(k \cdot |C|)$ computation in the preprocessing stage, where C is the circuit that computes the function F and k is the security parameter. In the online stage however, the work of P_1 and P_2 is $O(n + m)$ where n is the size of their inputs and m the size of their outputs. Hence, the amortized complexity of the work by the players is $O(n + m)$. The server's work will be $O(|C|)$.

The second instantiation is based on the construction of [Chung *et al.*, 2010] which uses non-interactive proofs with soundness amplification, as well as a fully homomorphic encryption scheme. When instantiated using their protocol, the offline cost of computa-

tion for P_1 and P_2 would be $\mathsf{poly}(k, |C|)$ while the online and hence the amortized cost is $\mathsf{poly}(k, \log(|C|))$. The servers computation is $\mathsf{poly}(k, |C|)$. The main advantage of the latter instantiation is that the public key of the scheme will be significantly smaller.

6.8 Server-Aided Private Set Intersection

The setting for the problem of set intersection includes two parties that have private input sets and wish to compute the intersection of the elements in their sets. This problem has numerous applications in practice and has been considered in a series of works [Freedman *et al.*, 2004; Hazay and Lindell, 2008; Kissner and Song, 2005; Jarecki and Liu, 2009; Dachman-Soled *et al.*, 2009; Cristofaro *et al.*, 2010; Hazay and Nissim, 2010; Dachman-Soled *et al.*, 2011]. These papers offer protocols addressing various adversarial models under different assumptions with a range of efficiency characteristics. In the semi-honest adversarial model, the protocols for set intersection have linear complexity in the size of the inputs. The goal of many of these works is to approach this complexity in stronger adversarial models. For instance, Cristofaro et al. [Cristofaro *et al.*, 2010] achieve linear complexity in the malicious case in the random oracle model. Jarecki et al. [Jarecki and Liu, 2009] propose a protocol with linear complexity secure in the standard model in the presence of malicious parties, based on the Decisional q-Diffie-Hellman Inversion assumption (in the CRS model), where a safe RSA modulus is generated by a trusted third party, and the input domain is of size polynomial in the security parameter.

We consider the problem of set intersection in the setting of outsourced computation where we would like to enable the computationally powerful server to execute the majority of the work involved comparing the values in the two input sets. Essentially we are interested in a solution where each of the two parties performs work that is linear in the size of his input set to preprocess the data and sends the results to the server who will compute the final intersection result.

A simple solution for the case when all parties are semi-honest. In the case of a semi-honest server we can obtain such a protocol as follows: the two input parties agree on a PRF key, and submit to the server the result of the evaluation of the PRF under this key on

each of the points in their input sets. Subsequently the server computes the set intersection of the two sets of PRF values he received, and sends the output to the two parties, who can map the PRF values back to the real input points. As long as the server follows the protocol honestly the two parties will receive the correct output without being able to learn anything about their private data due to the security guarantees of the PRF.

Protecting against a malicious server. The above protocol fails to guarantee correctness of the output in the case of a malicious server who can deviate from the prescribed protocol since he can return an arbitrary result without the parties being able to detect this. We adopt the following technique in order to enable the parties to detect misbehavior on the server's side: each party computes t copies of each of his inputs of the form $x|i$ for $1 \leq i \leq t$, and submits the PRF evaluations on the resulting values in a randomly permuted order. The server then computes the set intersection based on these PRF values and returns the answer. Now, we require that the set intersection contains all t copies for each element in the intersection. If it does not, then the parties will detect misbehavior on server's side and will abort. Thus in order to cheat without being detected, the server will need to guess what values correspond to the copies of the same element. The probability for this is negligible except in the following two cases (1) the server returns empty intersection (does not need to return any value) or (2) claims to each party that all elements from his/her input set are in the intersection (returns all PRF values provided by that party). To address these last issues we need to guarantee that the set intersection is neither empty nor contains all submitted elements. We achieve this in the following way: the parties agree on three elements d, e_1 and e_2 outside the range of possible input values. Then, the first party adds d and e_1 to his/her input set and the second party adds d and e_2 to his/her input set. Now the set intersection has to be non-empty since d will be in it, and at the same time cannot consist of all submitted input elements for either party since both e_1 and e_2 are not in the intersection. The protocol in Figure 6.5 presents the details of the the approach that we just outlined. We also note that for the purposes of the simulation, we need to use a pseudorandom permutation rather than any pseudorandom function.

Theorem 14. *The protocol in Figure 6.5 securely computes the* 2*-party set intersection*

Server-Aided Set Intersection

Let m and n be the sizes of the inputs sets for parties P_1 and P_2 with elements in the domain R. Let F be a pseudo-random permutation. Let t be a security parameter.

Inputs: P_1 has input set X; P_2 has input set Y

Outputs: P_1 and P_2 receive $X \bigcap Y$

Protocol:

1. P_1 and P_2 run a coin tossing protocol to choose a PRP key K.
2. P_1 and P_2 choose three elements $d, e_1, e_2 \notin R$. P_1 adds d and e_1 to his set X. P_2 adds d and e_2 to his set Y.
3. For each $x_i \in X$, P_1 computes $a_{i,j} = F_K(x_i|j)$ for $1 \leq j \leq t$. P_1 sends the set $A = \{a_{i,j}\}_{1 \leq i \leq m, 1 \leq j \leq t}$ in a randomly permuted order to S.
4. For each $y_i \in Y$, P_2 computes $b_{i,j} = F_K(y_i|j)$ for $1 \leq j \leq t$. P_2 sends the set $B = \{b_{i,j}\}_{1 \leq i \leq n, 1 \leq j \leq t}$ in a randomly permuted order to S.
5. S computes the set $A \bigcap B$ and sends it to P_1 and P_2.
6. P_1 checks that the PRP values corresponding to d are present in $A \bigcap B$ and those corresponding to e_1 are not. He also checks that if $F_K(x_i|j) \in A \bigcap B$ for some $j \in [1, t]$, then $F_K(x_i|j) \in A \bigcap B$ for all $j \in [1, t]$. If either of these checks fails, P_1 aborts the protocol. P_2 runs a similar check.
7. Using K, P_1 and P_2 recover the values in $X \cap Y$.

Figure 6.5: Security against malicious server

functionality in the $\mathcal{F}_{\mathsf{CT}}$-hybrid model for the adversary structure Adv_5 defined as follows:

$$\mathsf{Adv}_5 = \left\{ \left(\mathcal{A}_S[sh], \mathcal{A}_1[sh], \mathcal{A}_2[sh] \right), \left(\mathcal{A}_S[m], \mathcal{A}_1[h], \mathcal{A}_2[h] \right), \left(\mathcal{A}_S[nc_1, nc_2], \mathcal{A}_1[sh], \mathcal{A}_2[sh] \right) \right\}.$$

We refer the reader to Appendix D.3 for proof of the theorem.

Protecting against malicious parties. While the above protocol allows the two parties to detect a malicious server, it introduces a way for a malicious party to submit incorrectly processed input that would cause the other party to abort after receiving an invalid set intersection result, while the misbehaving party will learns the real output. Furthermore,

the fact that the honest party aborts can itself leak additional information about his inputs. In order to enable detection of misbehavior on the side of either party, we augment the protocol again. If we did not want to provide privacy of the input data from the server but still wanted to guarantee the correctness of the output result, we could solve this problem as follows: the server commits to the intersection set that he computes from the PRF values, then the parties reveal the key for the PRF to him so that he can verify the correctness of the submitted input sets and notify the two parties while not being able to change the computed output because of the commitment. In order to maintain the privacy for the input sets we can introduce an additional layer of PRF invocation where the first layer will account for the privacy guarantee and the second layer will allow for detection of the correctness of the input sets submitted by each party. We provide the details of the construction in Figure 6.6.

Theorem 15. *The protocol in Figure 6.5 securely computes the* 2*-party set intersection functionality in the* $\mathcal{F}_{\mathsf{CT}}$*-hybrid model for the adversary structure* Adv_6 *defined as follows:*

$$\mathsf{Adv}_6 = \mathsf{Adv}_5 \bigcup \Big\{ \Big(\mathcal{A}_S[h], \mathcal{A}_1[m], \mathcal{A}_2[h]\Big), \Big(\mathcal{A}_S[sh], \mathcal{A}_1[nc_S], \mathcal{A}_2[sh]\Big), \Big(\mathcal{A}_S[h], \mathcal{A}_1[h], \mathcal{A}_2[m]\Big), \Big(\mathcal{A}_S[sh], \mathcal{A}_1[sh], \mathcal{A}_2[nc_S, nc_1]\Big)\Big\}.$$

We refer the reader to Appendix D.3 for proof of the theorem.

Multiparty Set Intersection. We observe that both of our protocols can be generalized to setting where multiple parties want to find the intersection of their input sets. In this case the parties agree on common PRF key and then submit the PRF evaluations on their input sets to the server who computes the final intersection. We can apply the same techniques in order to protect against malicious server and malicious parties.

Efficiency. Our set intersection protocol requires that each party performs as many PRF evaluations as the size of his input set. The only works that give two party solutions to the set intersection problem with linear computation complexity in the total size of the input sets are [Cristofaro *et al.*, 2010] and [Jarecki and Liu, 2009], however, the former is in the random oracle model (ROM) and the latter works for input sets of limited size,

Server-Aided Set Intersection

Let m and n be the sizes of the inputs sets for parties P_1 and P_2 with elements in the domain R. Let G and F be pseudo-random permutation. Let t be a security parameter. Let *com* be a commitment scheme.

Inputs: P_1 has input set X; P_2 has input set Y

Outputs: P_1 and P_2 receive $X \bigcap Y$

Protocol:

1. P_1 and P_2 run a coin tossing protocol to choose two key K_1 and K_2.
2. P_1 and P_2 choose three elements $d, e_1, e_2 \notin R$. P_1 adds d and e_1 to his set X. P_2 adds d and e_2 to his set Y.
3. P_1 computes $X' = \{G_{K_1}(x) \mid x \in X\}$.
4. P_1 computes $Y' = \{G_{K_1}(y) \mid y \in Y\}$.
5. For each $x'_i \in X'$, $1 \leq i \leq m$ P_1 computes $a_{i,j} = F_{K_2}(x_i|j)$ for $1 \leq j \leq t$. P_1 sends the set $A = \{a_{i,j}\}$ to S.
6. For each $y'_i \in Y'$, $1 \leq i \leq n$ P_2 computes $b_{i,j} = F_{K_2}(y_i|j)$ for $1 \leq j \leq t$. P_2 sends the set $B = \{b_{i,j}\}$ to S.
7. S computes the set $A \bigcap B$ and sends a commitment $com(A \bigcap B)$ to both P_1 and P_2.
8. P_1 sends the set X' and K_2 to S and P_2 sends Y' and K_2 to S.
9. S verified that he received the same key from both parties and that the sets A and B have been computed correctly, namely contain exactly t PRF values for each input element. If the verification fails, S aborts the protocol.
10. S opens the the commitment $com(A \bigcap B)$ and sends the intersection set $A \bigcap B$ to P_1 and P_2.
11. Both P_1 and P_2 verify the open commitment. If the verification fails, they abort the protocol.
12. P_1 checks that the PRP values corresponding to d are present in $A \bigcap B$ and e_1 is not. He also checks that $A \bigcap B$ contains t corresponding PRP values for each element of X' in the intersection $A \bigcap B$. If either of these checks fails, P_1 aborts the protocol.
13. Using K_1 and K_2 P_1 and P_2 recover the values in $X \cap Y$.

Figure 6.6: Security against any one malicious party.

polynomial in the security parameter and requiring a common random string (CRS). The computation work in both of these solutions includes a linear number of exponentiations equivalent to public key operations. The solution of [Hazay and Nissim, 2010] has the best computational complexity while achieving security against malicious adversaries and it requires $O(m + n(\log\log m + p))$ exponentiations where m and n are the sizes of the input sets and p is the bit length of each input element, which is logarithmic in the input domain range. In the multiparty case computation complexity for each party remains the same which improves the computation cost of $O(Nd^2 \log d)$ in the case of N parties with input sets of size d of the most efficient existing solution [Dachman-Soled *et al.*, 2011].

6.9 Efficiency in the Server-Aided Setting

Having introduced the server-aided model for secure computation, it is natural to ask:

> *Is it possible to gain efficiency in this setting? and if so how can one quantify such a gain in efficiency?*

In this section we discuss these issues, informally. Formalizing these discussions accompanied with feasibility and impossibility results is an interesting direction for future research.

6.9.1 Evaluating the Efficiency Gain

Our main motivation for considering MPC in the server-aided setting is to allow (possibly) heterogeneous parties to outsource their computation to a server (that they do not necessarily trust). Hence, a natural way of measuring the efficiency of a server-aided protocol Π_f^{SAC} between n parties $(P_1, \ldots, P_n)$ and a server P_{n+1} that wish to compute a function f, is to compare the work performed by these parties with the work they would have to do in the most efficient "standard" MPC protocol Π_f^{MPC} (where the server is not present). Even if given a secure MPC protocol Π_f^{MPC} as a point of reference, there are multiple ways of quantifying the gain in the server-aided model each of which might be suitable for a particular computing environment.

Max/min efficiency. In cases where one party has significantly lower computational resources (e.g., a mobile phone executing a protocol with high-performance servers) we would like the weakest device to outsource as much of its computation as possible. A natural measure then is to consider only the maximum (over the parties) efficiency achieved by Π_f^{SAC}. Furthermore, when comparing to a standard MPC protocol, one should compare the total work of the party P_i that does the least amount of work in Π_f^{MPC} with the total work of the party P_j that does the least amount of work in Π_f^{SAC}. If the gap is significant enough, the weak device will see an improvement in efficiency if it plays the role of P_j in Π_f^{SAC} [3]. Similarly, one could consider the minimum efficiency, i.e., the total work of the party that does the most amount of work in Π_f^{SAC}.

We note that a sizable maximum efficiency might mean a less impressive reduction in the work of other players (or even an increase). For example, in the server-aided variant of Yao's garbled circuit protocol we design in Section 6.5, one player's computation is reduced significantly (the work is independent of the circuit size), while the second player has to do work proportional to the circuit size (though his work is still less than than what it would be in Yao's original protocol).

Average efficiency. Alternatively, one might be interested in a noticeable gain in the *total* work required by the players (excluding the server) regardless of the way this work is divided between the players. In this case one should try to optimize the *average* efficiency over by all the parties. To ensure some level of fairness, one can additionally enforce a limit on the variance in the efficiency of different parties. One example of a protocol with better average efficiency compared to standard two-party computation is briefly mentioned in Section 6.6. This protocol provides security against a malicious circuit garbler by utilizing the honest server for verification. In the resulting protocol, both players still have to compute a garbled circuit once, but can avoid the cut-and-choose and/or zero-knowledge proof techniques which would add a significant overhead.

[3] We note that in the case of special-purpose protocols, the roles that a party can play may be restricted.

Combined efficiency. Finally, there may be cases where a combination of different efficiency measures may be appropriate. Consider, for example, a computation that occurs between a mobile device and a server running "in the cloud". As discussed above, the maximum efficiency of Π_f^{SAC} is an important consideration for the weakest device (i.e., the mobile device). But the minimum efficiency of Π_f^{SAC} may be important as well since computation "in the cloud" has an economic cost and the client may have a limited budget.

More generally, taking extra costs into account (e.g., the cost of a cloud service) for some players it may only make sense (economically) to take part in a server-aided protocol if the gain in efficiency is greater than a threshold while other players may be happy with more modest gains. In such cases a combination of the maximum, minimum and average efficiency might be appropriate.

6.9.2 Comparison with Secure Delegated Computation

An alternative way of measuring efficiency is to compare the work the parties have to perform in the server-aided protocol to their work if they were to take part in an *insecure* protocol for evaluating the same function. This measure of efficiency is closely related to those considered for *secure delegated computation* (see Appendix D.2). In fact, it is not hard to show that any server-aided protocol for computing a function f that achieves reasonable efficiency compared to an insecure protocol for the same task can easily be turned into an efficient and secure delegated computation.

For example, consider our server-aided variant of Yao's protocol from Section 6.5. As mentioned above, this protocol reduces P_2's work significantly, making it independent of the circuit size for f while P_1's work is still linear in the circuit size. Now consider a construction that would improve our protocol by making P_1's work sublinear in the circuit size. One could then transform P_1 and P_2 into a single client, and let the server be the worker in a securely delegated computation scheme. This would yield a scheme that is more efficient than existing *general-purpose* secure delegation schemes in the literature [Gennaro *et al.*, 2010; Chung *et al.*, 2010] since they all take advantage of heavy machinery (e.g., fully homomorphic encryption) and only provide *amortized* improvements over insecure protocols.

This connection with secure delegated computation schemes is bi-directional. In fact in Section 6.7, we formally show how to transform any secure delegated computation scheme into an efficient server-aided secure computation protocol.

6.9.3 Why Non-Collusion Helps

Our last note on efficiency is an intuitive explanation of why non-collusion helps. In particular, consider a security definition for server-aided computation where a player (e.g., P_1) could collude with the server S. In that case, we could combine P_1 and S into a single party P_{1S} and our security definitions would reduce to the standard definitions for secure two-party computation between P_{1S} and P_2. Then, any general-purpose protocol for server-aided computation where P_2's work is sublinear in the circuit size, would automatically yield a general-purpose secure two-party computation where one of the parties performs sublinear work in the size of circuit. To the best of our knowledge, the only known way of achieving this goal (even in the case of semi-honest adversaries) is via use of a fully-homomorphic encryption scheme.

On the other hand, our non-collusion assumption between P_1 and S allows us to design a general-purpose protocol where the work of P_2 is independent of the circuit size.

Chapter 7

Privacy Enhanced Access Control for Outsourced Data Sharing

7.1 Motivation and Contributions

The emerging trend of outsourcing of data storage at third parties – "cloud storage" – has recently attracted tremendous amount of attention from both research and industry communities. Outsourced storage makes shared data and resources much more accessible as users can retrieve them anywhere from personal computers to smart phones. This alleviates data owner from the burden of data management and leaves this task to service providers with dedicated resources and more advanced techniques.

Traditional access control models focus on mediating access requests and providing confidentiality and availability guarantees for protected data and resources, often with an implicit assumption that the entity enforcing access control policies is also the owner of data. In this case the following two requirements suffice to achieve the goal: *a policy specification* that allows to define rules both syntactically and semantically that will govern the requests of data and resources from different subjects; *a mechanism* that performs request mediation based on defined rules. However, in many cases of distributed computing, the aforementioned assumption no longer holds, and we need to facilitate access control at points which should not have direct access to the data content itself, such as when outsourcing data storage to an untrusted third party. Therefore we need to store data in

encrypted form and enforce access control over the encrypted data.

We would like to facilitate data sharing in the setting of outsourced storage at a third party such as a cloud provider. The setting of cloud computing falls into the category of distributed access control scenarios discussed above. The cloud servers are considered to be *honest* but *curious*. They will follow our proposed protocol in general, but will try to find out as much information as possible based on their inputs. Therefore data confidentiality is not the only security concern in this scenario.[1]

First of all, given the fact that data storage provider (i.e., the cloud) does not own the data, access control policies defined by the data owner that govern who can have access to that date become also private information with respect to the cloud. For example, the fact that certain users share some of their data with other users or that they stop the sharing can be suggestive about their business relationships. This problem is mitigated by the use of cryptography as an enforcement mechanism. This approach translates the access control problem into the question of key management for the decryption keys. In this case, a user could obtain any encrypted data from the storage provider but access requests are answered implicitly by the fact a user can decrypt only the data that he is allowed to access (i.e, the possession of the corresponding decryption keys). Such a setup often requires that each encrypted resource is associated with an unique identifier and that the user knows the identifiers of the resources to which he wants access.

Another issue relevant to data sharing in outsourced storage is information leakage deduced from careful observations on data access patterns. Even though data is stored and transferred in an encrypted format, traffic analysis techniques can reveal privacy information about the activities of users and the underlying data. For example, access history to multiple data objects could reveal the access habits and privileges of a particular user; access to the same data object from multiple users could suggest a common interest or collaborative relationship; a ranking of data objects popularity can also be built upon access requests that the cloud receives. A trivial solution to the access pattern problem is to return all encrypted data upon any data request. While this solution may not cause any confidentiality problem with respect to the stored data, if the cryptographic mechanism guarantees that

[1]Issues of data integrity and availability in access control models are beyond the scope of this paper.

a user can decrypt only data that he has access to, it brings prohibitive costs in several directions: the data stored at a cloud provider most likely exceeds the storage capacity of any single user, the communication for transferring the data would incur enormous network usage as drastically increased latency; additionally, the user would need to spend time on cryptographic computations to find the appropriate piece of the data he needs. All these concerns rule out this obvious solution.

The question of hiding access pattern is a challenging task to achieve while avoiding work proportional to the total size of all stored files. There have been several cryptographic solutions that realize the notion of oblivious RAM and manage to achieve improved amortized complexity for queries while hiding access patterns [Goldreich and Ostrovsky, 1996; Pinkas and Reinman, 2010; Damgard *et al.*, 2010; Goodrich and Mitzenmacher, 2011]. However, such solutions are highly interactive and still require communication polylogarithmic in the size of the database, which in the setting of large storage cloud providers, weak client devices and expensive network communication will not be practical. Furthermore, they assume that the user submitting the query is the owner of all data, which does not fit our scenario where access control is enforced on data shared by multiple users, not limited to the data owner.

An equally important, but often overlooked, aspect of access control for outsourced data is to enforce the *write* access. Existing solutions often made an implicit assumption that access to shard data and resources always refer to *read* requests only. However, an inevitable requirement of data sharing is to allow authorized write access, such as in a collaborative working environment, where co-workers are allowed to contribute to the same project document. While data encryption naturally preserves authorization of the read access through key management, it says little about the authorization on writing to that data. That is the procession of a decryption key implies authorized read access but not necessarily the write. Therefore, different cryptographic schemes are mandatory to manage read and write accesses separately. On the other hand, similar to read access, access patterns on write operations should be hidden from the cloud provider to achieve better privacy. In addition, adding write operation into the current access control scheme complicates our problem as we do not assume any implications between read and write access rules. Thus

a user may have both types of access (read and write) to a protected object or only one of them (only read, only write).

We consider the scenario of a web-based data sharing service, like Dropbox, throughout the discussion of this paper. In our scenario, users are allowed to upload data, such as documents, pictures, video clips, etc., to a cloud storage provider. Parts of this data can be shared with other users of the service and regulated by access control policies specified by the data owner. For example, a user can upload vacation pictures to be viewed only by family and friends; meanwhile she may also want to share read and write access to a project document among collaborating co-workers.

We claim that a privacy enhanced access control solution for data sharing in outsourced storage, such as the cloud, needs to meet the following requirements:

1. it provides data confidentiality by implementing a cryptographic mechanism to enforce fine-grained access control, ensuring that a user can decrypt only the data that he has access to;

2. it provides a more practical and flexible data sharing scheme by supporting both read and write operations in the access control model;

3. it enhances data and user privacy by limiting the information leakage to the cloud servers from the access control rules and access patterns of users.

7.1.1 Contributions

We address the privacy requirements for the policies and users' access patterns in the setting of outsourced data storage and propose a mechanism to achieve a flexible level of privacy guarantee for the client. We introduce a two-level access control model that combines *fine-grained access control*, which supports the precise granularity for access rules, and *coarse-grained access control*, which allows the storage provider to manage access requests while learning only limited information from its inputs. This is achieved by arranging outsourced resources into units called *access blocks* and enforcing access control at the cloud only at the granularity of blocks. The fine-grained access control within each access block is enforced at the user's site and remains oblivious to the cloud. The mapping between files and access

blocks is transparent to the users in the sense that they can submit file requests without knowing in what blocks the files are contained. While most existing solutions [De Capitani di Vimercati *et al.*, 2011; Vimercati *et al.*, 2010; Yu *et al.*, 2010] focus on read request, we present a solution that provides both *read and write access control*.

7.2 Two-level Access Control Model – Solution Overview

We consider the following scenario: a set of users outsource their data to a remote storage (cloud) provider. These users further would like to be able to share selectively some of their data among themselves. This data sharing should be enabled directly at the cloud through appropriate access control rules that allow users to retrieve all data that they are authorized to access (i.e. not involving the actual data owner). Further, the access control rules governing the data sharing and the data that users access are private information of the users and our goal will be to protect this information from the cloud provider.

We distinguish the following three roles in this access control model: the *data owner* who creates data to be stored at the remote storage in an encrypted format and regulates who has what access to each part of the data; the *data user* who may have read and write access to the protected data; the *cloud provider* that stores the encrypted data and responds to access requests. While a solution that enforces access control solely through encryption of the data and appropriate decryption key distribution can achieve complete privacy for the access patterns and access control rules by allowing users to retrieve the whole encrypted database, such an approach will be completely impractical requiring an enormous amount of communication. We suggest a hybrid solution that offers a way to trade off privacy and efficiency guarantees. The basic idea behind it is to provide two levels of access control: *coarse-grained* and *fine-grained*. The coarse-grained level access control will be enforced explicitly by the cloud provider and it would also represent the granularity at which he will learn the access pattern of users. Even though the cloud provider will learn the access pattern over all user requests, he will not be able to distinguish requests from different users, which would come in the form of anonymous tokens. The fine-grained access control will be enforced obliviously to the cloud through encryption and would prevent him from

differentiating requests that result in the same coarse-grained access control decision but have different fine-grained access pattern.

We realize the above two levels of access control by introducing division of the data resources of the same owner into units called *access blocks*, which would represent the coarse-level granularity in the system. Now the cloud provider would be able to map user requests to the respective access blocks containing the relevant data only if the user has access to the requested data and without learning which part of the block is accessed. The provider would also not learn the reason for no match: missing data or no access authorization. Our solution does not require users to know the exact access blocks that would contain the data they are searching for. Files might be moved between different blocks, and the only information that users would need in order to request them will be a unique file identifier rather than the id of the current block where the file is residing. We will enable this oblivious mapping of files to blocks using techniques from predicate encryption and some extensions to the scheme [Katz *et al.*, 2008]. Once a user retrieves the content of the matching block, he would be able to decrypt only the part of the block, which he is authorized to access. We use the ideas of [Vimercati *et al.*, 2010] to minimize the decryption keys that need to be distributed for fine-grained access control within access blocks.

While the above suffices for read access control, handling write access control is a little more subtle. The main issue there is that the cloud would need to allow users to submit updates for different parts of an access block without learning which part are updated, and at the same time prevent users authorized to write to one file in the block from writing to another file. In order to facilitate this functionality the cloud provider would accept write updates for blocks only from users that provide tokens granting them write access to some part of the block (not revealing which part). These updates will be appended to the content of the block but also the cloud would obliviously tag the updates with the id of the file for which the user has been authorized, but without learning which this file is. We achieve this functionality again through a modification of the searchable ciphertexts in a predicate encryption scheme.

7.3 Access Control Model

In our access control model, we consider a scenario with a set of users U that selectively perform data sharing among themselves on a set of resources R using remote storage provided by the cloud. To discuss different activities an entity can perform, we distinguish the following three different roles in the model: the *data owner* who creates data to be stored at the remote storage in an encrypted format and regulates who determines the access rules to each unit of the data; the *data user* who may have read and write access to the protected data; the *cloud provider* that stores the encrypted data and responds to access requests enforcing the specified access rules. Note that a user can play the role of a data owner and a data user at the same time. When a data owner $o \in U$ wants to share a resource $r \in R$, the management of r is outsourced to the cloud provider. The authorization policy is defined by data owner o and enforced through cryptographic access control scheme (See details in Section 7.4 and 7.5.). Also the cloud provider is considered to be *honest-but-curious*, which can be trusted to perform the management protocol but will try to deduce as much information as possible from its inputs, such as the access pattern.

An authorization policy P_o is a set of tuples of form $\langle u, r, p \rangle$ ($p \in \{r, w\}$), which states a data user $u \in U$ is allowed access data resource $r \in R$ owned by $o \in U$ by performing *read* (r) or *write* (w) operation. An access control list (ACL) of a resource r owned by o, denoted as $acl(r)$ is then defined as a set of data users u satisfying an authorization policy. More specifically, we use $acl\_read(r)$ and $acl\_write(r)$ to represent ACL for read and write access respectively.

To provide confidentiality through data encryption while preserving privacy, we propose a two-leveled access control model illustrated in Figure 7.1. In our scheme the data resources (files) will be divided into units that we call *access blocks*. These access blocks will constitute the coarse-grained level view of the stored data. The cloud provider will be presented with this view and he will be able to enforce access control rules at this granularity. He will be able to match an authorized request to the access block that contains the file that is being accessed. In the case of a read request the cloud would provide the content of the matching block to the user. And in the case of a write request accept the cloud provider will accept only authorized updates for the content of the access block and would also obliviously match

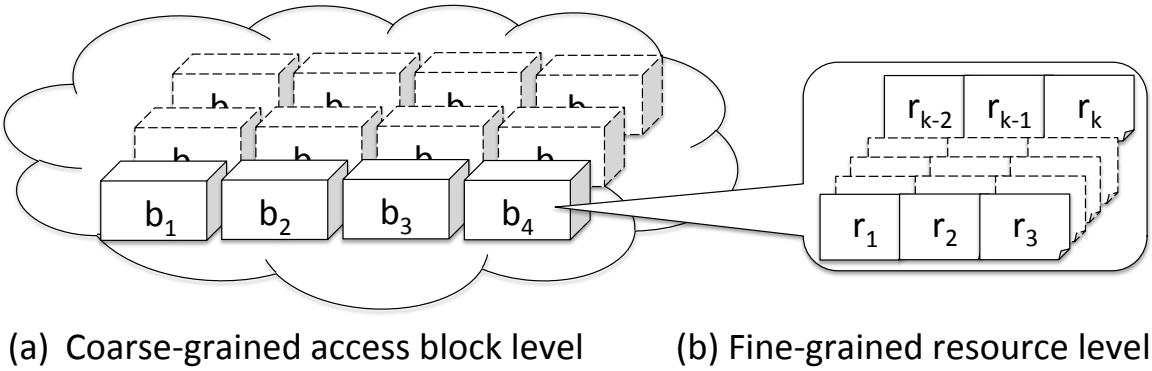


Figure 7.1: Two-leveled access control model.

them to the files for which they are submitted.

At the fine-grained level, each access block b_i consists of k_i files owned by a single party. Each data owner is responsible for distributing his resources into blocks. He would further have fine-grained access policies that would specify each user's access to the separate files stored within an access block. The fine-grained access control will be enforced at the user's side as opposed to the coarse-grained access control which is enforced at the cloud provider. For this purpose we will use encryption as implicit method for access control in the case of read requests. For write fine-grained access control we will use again appropriate encryption keys distribution together with tokens that would both authorize an update request to the cloud provider but would also implicitly bind the submitted update to the exact file within the access block. In Section 7.4 and Section 7.5 we present the exact protocols that implement these ideas.

To facilitate our discussion on the access control scheme, we present a simple yet illustrative example as follows:

Example 1: Consider a system with five users $U = \{A, B, C, D, E\}$. Let R_u denote the set of resources owner by user u, and we have $R_A = \{r_1, r_2, r_3, r_4\}$, $R_B = \{r_5, r_6, r_7\}$ and $R_C = R_D = R_E = \emptyset$. Authorization policies at the fine-grained level defined by each data owner are:

- $P_A = \{\langle A, r_1, r\rangle, \langle B, r_1, r\rangle, \langle C, r_1, r\rangle, \langle A, r_2, r\rangle, \langle B, r_2, r\rangle,$
  $\langle C, r_2, r\rangle, \langle A, r_3, r\rangle, \langle E, r_3, r\rangle, \langle A, r_4, r\rangle, \langle B, r_4, r\rangle,$

$\langle C, r_4, r\rangle, \langle E, r_4, r\rangle, \langle A, r_1, w\rangle, \langle B, r_1, w\rangle, \langle C, r_1, w\rangle,$
$\langle A, r_2, w\rangle, \langle B, r_2, w\rangle, \langle C, r_2, w\rangle, \langle A, r_3, w\rangle, \langle A, r_4, w\rangle,$
$\langle D, r_4, w\rangle\}$;

- $P_B = \{\langle A, r_5, r\rangle, \langle B, r_5, r\rangle, \langle B, r_6, r\rangle, \langle C, r_6, r\rangle, \langle D, r_6, r\rangle,$
$\langle A, r_7, r\rangle, \langle B, r_7, r\rangle, \langle C, r_7, r\rangle, \langle D, r_7, r\rangle, \langle E, r_7, r\rangle,$
$\langle A, r_5, w\rangle, \langle B, r_5, w\rangle, \langle C, r_5, w\rangle, \langle B, r_6, w\rangle, \langle D, r_6, w\rangle,$
$\langle E, r_6, w\rangle, \langle A, r_7, w\rangle, \langle B, r_7, w\rangle, \langle C, r_7, w\rangle, \langle D, r_7, w\rangle,$
$\langle E, r_7, w\rangle\}$.

Therefore, we have the following set of ACLs:

- $acl\_read(r_1) = \{A, B, C\},\ acl\_write(r_1) = \{A, B, C\}$;
- $acl\_read(r_2) = \{A, B, C\},\ acl\_write(r_2) = \{A, B, C\}$;
- $acl\_read(r_3) = \{A, E\},\ acl\_write(r_3) = \{A\}$;
- $acl\_read(r_4) = \{A, B, C, E\},\ acl\_write(r_4) = \{A, D\}$;
- $acl\_read(r_5) = \{A, B\},\ acl\_write(r_5) = \{A, B, C\}$;
- $acl\_read(r_6) = \{B, C, D\},\ acl\_write(r_6) = \{B, D, E\}$;
- $acl\_read(r_7) = \{A, B, C, D, E\},\ acl\_write(r_7) = \{A, B, C, D, E\}$.

Note that for each resource r owned by user o, we have $acl\_read(r) \cap acl\_write(r) \supseteq \{o\}$. That is the owner of a resource automatically entails both read and write access privilege. At the coarse-grained level, user A maintains two blocks $b_1 = \{r_1, r_2\}$ and $b_2 = \{r_3, r_4\}$, and user B maintains a single block $b_3 = \{r_5, r_6, r_7\}$.

7.4 Read Access Control

In this section, we present in detail the two-level access control scheme for read access only after describing the following techniques applied in our protocol.

7.4.1 Techniques

7.4.1.1 Fine-Grained Access Control

Fine-grained access control is applied to files inside each access block to explicitly enforce access control rules. While the cloud provider is able to determine whether a user submits a legitimate request for some file within a block, he should remain oblivious to the access control rules defined for that file. To guarantee this property the access control view presented to the cloud treats blocks as entities, and the cloud grants a read access by providing the content of an entire block. Fine-grained access control is enforced by encrypting files per block under different keys, and the access control problem is mitigated to appropriate key distribution. Even a user receives the encrypted content of a block, he is able to decrypt only the files that he has access to. Access revocation requires re-encryption of the resource and re-distribution of the new key to the remaining authorized users. Our goal is to minimize the amount of work and interaction between users and the system upon policy updates.

The work of [Vimercati *et al.*, 2010] proposes an encryption-based access control solution for outsourced data. They introduce a key distribution technique that allows each user to receive only one credential in the form of a public-private key pair and then later be able to derive decryption keys for all resources he has access to using a public structure, only when he needs to access these resources. This public structure helps avoid the need to explicitly re-distribute keys to each user when access policy change.

The key distribution structure used in the scheme of [Vimercati *et al.*, 2010] is a tree with the following properties:

1. Each leaf of the tree is assigned a symmetric key. The leaf keys are the private keys distributed to the users of the system when they join.

2. Each intermediate node in the tree is associated with a symmetric key, and contains tokens with encryptions of this key under the keys of the children nodes. The internal nodes' keys are encryption keys for different resources. The tokens for each node constitute the structure that is published and used by each user to derive the decryption keys for the authorized resources.

3. The tree graph contains directed edges from children nodes to their parent nodes that satisfy the following property: there is a directed path from a leaf node to an internal node if and only if the user who possesses the leaf node key has access to the resource encrypted with the keys of the internal nodes. The edges of the graph represent the access control rules for the system.

When a user wants to access a file, he derives the corresponding decryption key as follows: starting from its leaf node decrypting its content with his private key credential and then using the key obtained from the node to try to decrypt the content of the parent node. He continues this process of decryption at nodes and deriving new keys to obtain all decryption keys for the documents he can access.

We use the approach of [Vimercati *et al.*, 2010] to implement fine-grained access control in our scheme since it does not require direct interaction between data owner and users for key distribution. This property is important since we want to enable the work of our system without making any assumptions about the time when any other party except the cloud provider will be online. Since the structure that contains the encrypted keys does not need to be private, it can be stored at the cloud provider. Thus we can achieve key distribution without requiring any direct interaction between data owner and users beyond some initial set-up stage when the user establishes a private key with the data owner. We include the key distribution tree for the files in a block in the contents of the block, which the user will retrieve and then derive the appropriate decryption keys. We also need to make some modifications to the way the tree is constructed. In out case the tree structure itself can reveal certain sensitive information to the cloud. For example, a user having access to one file will have access to all the files along a directed path. So the content of each node, a pointer to next node and the token to derive next key are all protected under the current encryption key.

For the purposes of our protocols we will assume an instantiation of the scheme from [Vimercati *et al.*, 2010] that provides the following functions: **Publish**, **Access_Read**, **Find_Chain**, **Compute_Key**, **Find_Resources**, the use of which we summarize in Figure 7.2. We also extend the functionality of the underlying scheme with a protocol that allows updates in the access policies. An update in the access rules translates into a change

of the edges in the graph. If there are any internal nodes in the graph that become disconnected from any leaf node after the update, this necessitates change of the keys associated with those nodes as well as re-encryption of the corresponding resources. We describe how we instantiate the **Update** function also in Figure 7.2. Given this structure a user can obtain the keys of the internal nodes to which it is connected with a directed path, and by definition these are exactly the decryption keys for the files that he can access.

- **Publish**(r, o, e_o, acl): adds a resource r owned by o with a secret e_o and an access control list $acl = acl\_read(r)$ for read access.
- **Access_Read**(u, r, o): returns the encryption key for a resource r owned by o, if u is an authorized user.
- **Find_Chain**(u, r): finds the shortest chain of tokens from the secret key of user u to derive the decryption key for resource r.
- **Compute_Key**$(u, chain)$: derives the secret key for a user u given a chain of transition tokens.
- **Find_Resources**(u, r): finds the set of nodes that lie on any path from the user u to the node corresponding to resource r.
- **Update**(r, acl): if there is another resource with the same access control list acl, i.e., there is a node in the tree accessable exactly by a subset of users in acl, then encrypt r with the key contained in that node. Otherwise, encrypt r with a new key, add a new node containing this key to the tree and add appropriate edges to connect the new node to the users who have access to r. (Note that certain subgroups of the users in acl might already have a shared key through another node in tree, and in that case we connect to that node rather than all the users' nodes separately.)

Figure 7.2: Algorithms for key distribution and management for fine-grained AC.

7.4.1.2 Coarse-Grained Access Control

The main goal to achieve at the level of coarse-grained access control is to enable the cloud provider to obliviously match a user's request to an access block without learning which part of the block the user is authorized to access. In addition we provide *unlinkability* among multiple requests for the same resource even if coming from the same user, which further protects users' access patterns from the cloud provider. In order to achieve these goals we apply the predicate encryption scheme of [Katz *et al.*, 2008]. Observing that in

this scheme ciphertext can be re-randomized even without knowledge of the secret key, we define a re-randomization algorithm in Definition 22.

Definition 22. *A re-randomizable predicate encryption scheme consists of the following algorithms:*

- $\mathsf{Setup}(1^n)$*: produces a master secret key* SK *and public parameters;*
- $\mathsf{Enc}_{\mathsf{SK}}(x)$*: encrypts an attribute* x *using key* SK*;*
- $\text{GenKey}_{SK}(f)$*: generate a decryption key* SK_f *associated with a function* f*;*
- $\mathsf{Dec}_{\mathsf{SK_f}}(c)$*: outputs* 1 *if the attribute encrypted in* $c = Enc_{SK}(x)$ *satisfies* f*, i.e.* $f(x) = 1$*, and output a random value, otherwise;*
- $\mathsf{Rand}(c)$*: computes a new encryption* c' *of the value encrypted in* c *but with different randomness* without the secret key.

We present the predicate encryption scheme of [Katz *et al.*, 2008] and the instantiation of the function $\mathsf{Rand}(c)$ for that scheme in Appendix E.1. This scheme handles a class of functions f, which includes polynomials of bounded degree. We use polynomial functions of the type $f(x) = (x - id_1) \cdots (x - id_n)$, to implement coarse-grained access control. Figure 7.3 present a list of algorithms to enforce access control on the block level granularity without revealing the exact files that are being accessed insider a block. The algorithm **File_Access_Check** grants access if the submitted access token matches any of the files in the block without revealing the file identity. The request token produced by **File_Access_Request** is an encryption that does not leak information about the file id it contains.

7.4.2 Read Access Control

We present a read access control solution consisting of the following algorithms. Unless explicitly stated, all the actions are performed by individual data owners.

- **System Setup:** At the *fine-grained level*, files are distributed into access blocks. Generate a tree graph per block by running $\textbf{Publish}(r, o, e_o, acl)$ for each resource r

- **Block_Access_Setup:** data owner runs $Setup(1^n)$, publishes the public parameters and keeps the master secret key SK. For files $id_1, \ldots, id_n$ in each block, he computes $SK_f = \text{GenKey}_{SK}(f)$ for $f(x) = (x - id_1) \cdots (x - id_n)$ and sends SK_f to the cloud provider.
- **File_Access_Authorization:** data owner provides access to a file id by sending $c_{id} = \mathsf{Enc}_{\mathsf{SK}}(\mathsf{id})$ to an authorized user.
- **File_Access_Request:** user generates a token $t_{id} = \mathsf{Rand}(\mathsf{c_{id}})$ for file id.
- **File_Access_Check:** upon receiving a request token t, the cloud computes $\mathsf{Dec}_{\mathsf{SK_f}}(\mathsf{t})$ for each block, and returns those blocks that compute to 1.

Figure 7.3: Algorithms for enforcing coarse-grained AC at the access block level.

owned by o with initial ACLs, and encrypt resources using keys from the tree graph. At the *coarse-grained level*, each data owner computes parameters for a predicate encryption scheme. Then he constructs a separate tree graph over all resources he owns to distribute authorization tokens of the form $c_{id} = \mathsf{Enc}_{\mathsf{PK}}(\mathsf{id})$ (i.e., now tree nodes contain authorizations tokens rather than file decryption keys). Finally, data owner computes a key $SK_f = \text{GenKey}_{SK}(f)$ per block where f is the polynomial derived from the ids of the files contained in that block as described above, and gives this key to the cloud provider, which will use it to obliviously check read access on authorization tokens.

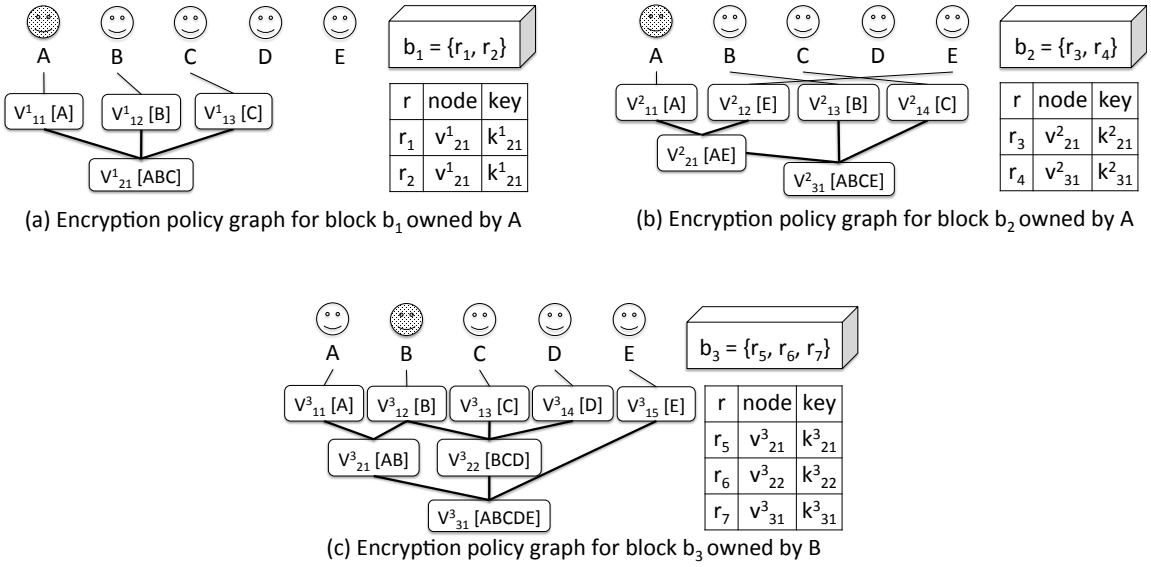
- **Access Authorization:** At the *fine-grained level*, add a leaf node containing the new user's public key to the corresponding tree graph with encryption keys. Update the graph by adding new internal nodes and appropriate edges if necessary. Update file encryptions if new internal nodes were added previously. At the *coarse-grained level*, perform similar operations with respect to the tree graph containing read access tokens.

- **Access Request:** First, at the *coarse-grained level*, the user u derives from the tree graph with access tokens the authorization token $c_{id} = \mathsf{Enc}_{\mathsf{SK}}(\mathsf{id})$ for the requested file id. For this he uses the functions **Find_Chain**(u, id), **Find_Resources** (u, id) and **Compute_Key**$(u, chain)$. Once he has the access token, he submit a randomization of it $t_{id} = \overset{\$}{\leftarrow} (c_{id})$ to the cloud.

Once the user receives the content of the matching block for his request, at the *fine-grained level*, he derives the decryption key for the specific resource he is looking for the tree graph included in the block content.

- **Access Check:** Access control is enforced as follows. At the *fine-grained level*, only authorized users can derive the correct decryption key for any file using the public tree structure. At the *coarse-grained level*, the users can obtain only tokens for the files they are authorized to access. Also the cloud provider executes **File_Access_Check** to identify the block that contains the requested file and return only that blcok.

- **Access Rule Update:** At the *fine-grained level*, changes are applied immediately upon policy updates. If the policy update involves access revocation, the data owner changes the encryption of the corresponding files. The data owner identifies the blocks affected by those files and updates their tree graphs with decryption keys. The changes at the *coarse-grained level* happen at longer intervals of time, the length of which would depend on the resources of the data owner. They involve updating of the tree graph with access tokens.



(a) Encryption policy graph for block b_1 owned by A

(b) Encryption policy graph for block b_2 owned by A

(c) Encryption policy graph for block b_3 owned by B

Figure 7.4: Tree graphs of encryption policy for fine-grained AC on read access.

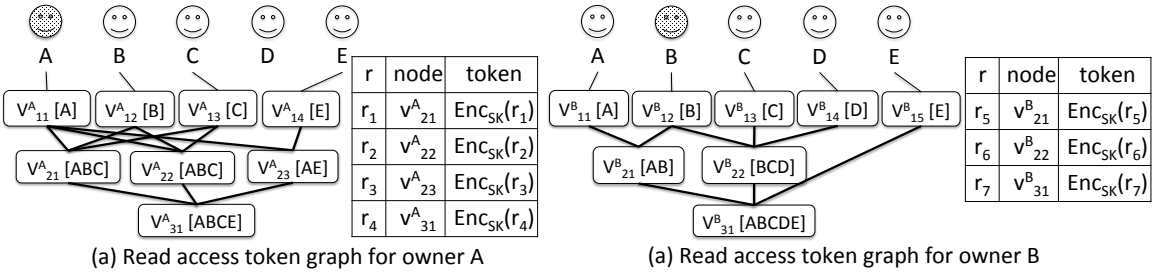


Figure 7.5: Distribution of read access tokens $Enc_{SK}(r_i)$ for coarse-grained AC.

Example Given the example in Section 7.3, we construct one tree graph per block for file encryption keys at the fine-grained level in Figure 7.4. Each block stores files owned by a single user (shaded), and each file r is encrypted under a symmetric key. Leaf nodes, indexed as v^j_{1n} with block id j, store users' initial public keys. Key derivation paths are denoted using thick links connecting leaf nodes to internal ones. Each row in the table states resource r_i in block b_j encrypted under key k^j_{mn} at vertex v^j_{mn}. The tree structure significantly reduces the number keys that each user has to maintain, and enables encryption of different files with the same ACL under the same key. For example, key k^3_{31} for encrypting r_7 can be derived by all users, since there is a directed path from each user's initial secret to vertex v^3_{31}; resources r_1 and r_2 are encrypted under the same key k^1_{21} since $acl\_read(r_1) = acl\_read(r_2) = \{A, B\}$.

Figure 7.5 depicts a tree graph per data owner to distribute read access tokens at the coarse-grained level. Each row in the table states resource r_i is associated with access token $Enc_{SK}(r_i)$ stored at vertex v^o_{mn}. Unlike in Figure 7.4, each r_i is associated with an unique read access token encrypted on its *id*. For example, r_1 and r_2 are now given different access tokens at vertexes v^A_{21} and v^A_{22} respectively.

7.5 Write Access Control

Enforcing write access control presents more challenges, mainly for the fact that access control through data encryption does not apply to cases when data can be modified. At coarse-grained level we can enforce access control similarly as in the case of reads. We can have separate write access tokens for the user and those will allow them to submit updates only to blocks in which there is at least one file they can write to. Within each block write

access to a file is granted through a public key used for the encryption of the file. These keys can be distributed similarly to the decryption keys for reading through a tree structure.

Yet, the above is still not sufficient to guarantee correct write access control enforcement. The remaining issue is related to the fact that we want to keep the fine-grained access rules oblivious to the server, i.e. he should not understand what file a user is updating. If we allow users to include encrypted identifier of the file they are updating, this opens a door for a new attack where a user without write access to a file submits an update for that file which contains garbage and just overwrites the content of the file. In order to prevent this we device a way for the server to obliviously tag the submitted update with the appropriate file identifier.

The authorization token for write that the user submits to gain access to the block contains the file identifier in an encrypted form. So using this token as an identifier for the update will provide the oblivious property. However, this solution would not be secure since any other user, who has read access to the block, would be able to obtain this token granting write access as well. To prevent this undesired situation, we take advantage of the predicate encryption ciphertexts constituting access tokens, which allows us to use part of the token as update identifier, which cannot be used as write access token.

7.5.1 Techniques

File Encryption. We use a public key rather than symmeric encryption scheme to handle all possible combinations of read and write access to a file. Since such a scheme is computationally expensive for large size of data, file content is still encrypted using a symmetric key (e.g., AES), which is further encrypted under the public key. Two trees are constructed for key distribution per block – one for the public (encryption) keys and the other for the private (decryption) keys. These two trees share the same set of internal nodes for a one-to-one correspondence between public and private key pair. Only files readable and writable by the same set of users can share the same public key pair.

Access Authorization Tokens. Two trees are constructed by each data owner for the distribution of read and write access tokens respectively.

File Identifiers for Write Updates. We observe that the write authorization token is a valid encryption for a predicate encryption that provides polynomials evaluation, and the structure of the encrypted plaintext for access to file id is a vector of the form $(1, id, id^2, \ldots, id^n)$, where n is the number of files placed in a block. The structure of the ciphertext allows it to be split into parts where one part is an encryption of the vector $(1, id, id^2, \ldots, id^k)$ ($k < n$, $n > 2$), which is no longer a valid write access token for that file, but can still be used identify file updates for users with read privilege. This can be achieved using a decryption predicate for a polynomial of degree k that has id as a zero point. (See Appendix E.1 for details.)

7.5.2 Integrated Read and Write Access Control

We realize the above proposal for the write access control enforcement to obtain an integrated solution for both read and write access. Next we describe the functionality associated with write access enforcement. The read access is the same as the construction in the previous section with the exception that once a client has retrieved a block he needs to identify both the original encryption of the file as well as all updates for that file. The latter will be achieved using an additional key (a new part in his authorization token) that will allow him to identify the valid updates submitted for that file.

- **Setup:** At the *fine-grained level*, for each block construct two copies of the same key distribution tree based on both the read and write access rules, i.e. two files can be encrypted with the same key if and only if the same set of users have read and write access to them. For each node in the tree, generate a public-private key pair (sk_n, pk_n). Store in the nodes of one of the above trees the secret keys and add edges based on the read access tules. In the nodes of the other tree store the public keys and add edges according to the write access rules. Construct another tree with the same set of nodes to store the public key pk_n, with edges determined by write access rule. For each file id generate a AES key sk_{id}^{aes} for encryption, and append to the ciphertext $\mathsf{Enc}_{\mathsf{pk_n}}(\mathsf{sk}_{\mathsf{id}}^{\mathsf{aes}})$.

  At the *coarse-grained level*, each data owner generates two sets of parameters (pk', sk') and $(pk", sk")$ for the predicate encryption. Then he constructs a tree graph, where

each node contains read access token $\mathsf{Enc}_{\mathsf{pk}'_{\mathsf{ra}}}(\mathsf{id})$ (used by the cloud provider to check the read access) and $SK_{x-id} = \text{GenKey}_{sk''_{ra}}(f)$ where $f(x) = x - id$ (used by the user to identify all updates to the file within the retrieved block). Similarly, construct another tree to distribute write access tokens $\mathsf{Enc}_{\mathsf{pk}_{\mathsf{wa}}}(\mathsf{id})$.

- **Access Authorization:** At the *coarse-grained level*, extend the trees with read and write access tokens with new leaves for the new user and update the edges according to his read and write permissions. This may involve splitting of nodes and re-encrypting files with new keys if the user has read access only to a subset of files that have been encrypted with the same key.

- **Write Access Request:** At the *coarse-grained level*, the user derives his write access authorization token for the file he wants to access from the corresponding tree structure and submits to the cloud a re-randomized copy of the token.

  Once the user has the matching block, at the *fine-grained level* he obtain the encryption key pk_n for the file to be updated from the write tree. Then he encrypts the new content for that file with key pk_n and submits it to the cloud server.

- **Write Access Check:** At the *fine-grained level*, a user can modify a file only if he has the encryption key and the write authorization token. At the *coarse-grained level*, the cloud finds if there is a block for which the authorization token grants write access. The write access token is of the form $(C_0, \{C_{1,i}, C_{2,i}\}_{i=1}^{n})$, and the cloud uses the first components $(C_0, \{C_{1,i}, C_{2,i}\}_{i=1}^{2})$ as an identifier for updates appended to a block.

- **Write Access Rule Update:** Update per-block trees for encryption keys and the tree for distributing write access tokens accordingly.

Example Following the same example in Section 7.3, we draw tree graphs of encryption policy for both read and write access at the fine-grained level within each access block in Figure 7.6. Now for each access block b_j, we construct two trees: one for read access and the other one for write access. Instead of storing AES symmetric keys in the vertexes, a public key pair (sk_{r_i}, pk_{r_i}) is generated for each resource r_i, where the private keys are stored in nodes from the read tree (v_{mn}^{Rj}) and the public keys are stored in nodes from

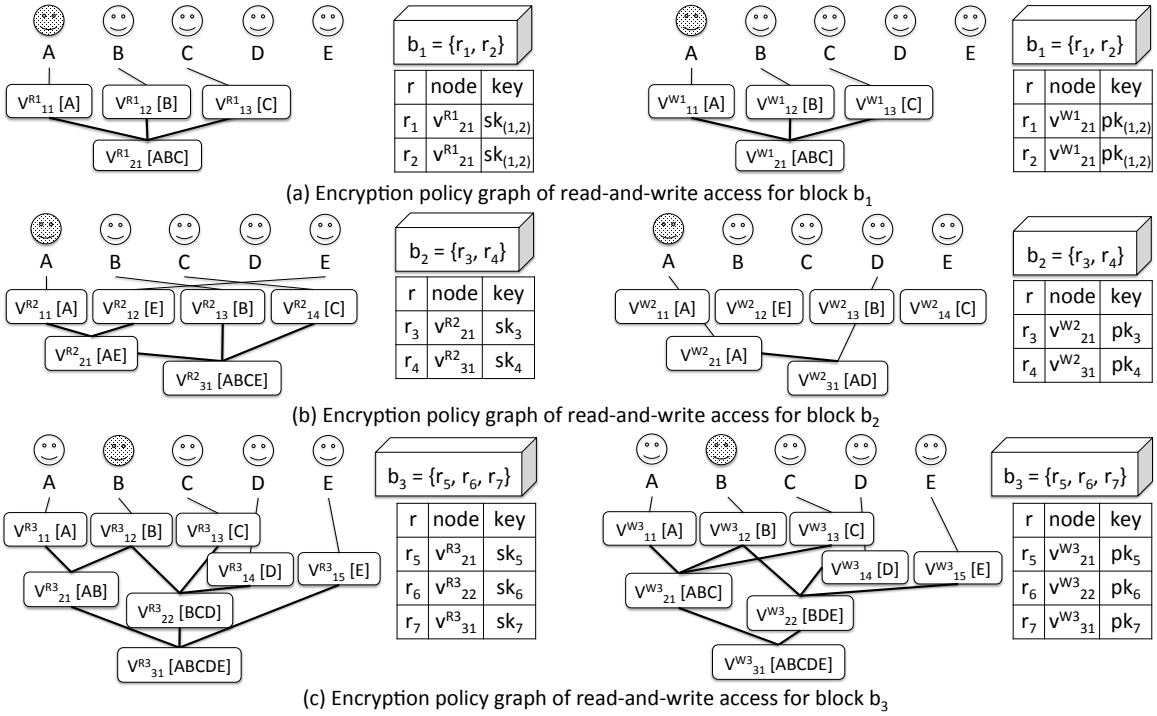
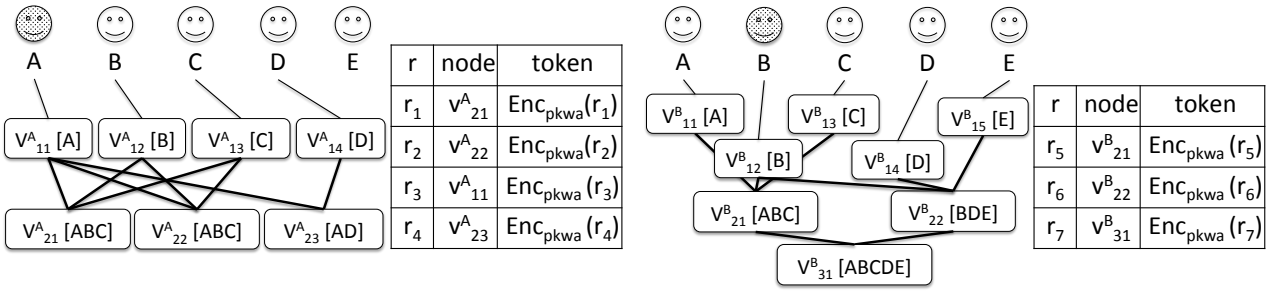


Figure 7.6: Tree graphs of encryption policy for read and write access at the fine-grained level within each access block.



(a) Write access token graph for owner A

(a) Write access token graph for owner B

Figure 7.7: Distribution of write access tokens $Enc_{pk_{wa}}(r_i)$ for each resource r_i in coarse-grained access control.

the write tree (v_{mn}^{Wj}). Every pair of read and write trees share the same set of vertexes, although the set of edges may change. For example, vertex v_{21}^{R2} is labeled as $[ABCE]$ since $acl\_read(r4) = \{A, B, C, E\}$; whereas vertex v_{21}^{W2} is labeled as $[AD]$ because $acl\_write(r4) = \{A, D\}$. Thus different ACLs on read and write for the same resource entail different labels of user list, but the one-to-one correspondence relationship between read node and write node is naturally reflected by the indexing pattern. Note that in Figure 7.6(a), both the read tree and the write tree share the same set of vertexes and edges. That is because $acl\_read(r_1) = acl\_read(r_2) = acl\_write(r_1) = acl\_write(r_2) = \{A, B, C\}$.

Moreover, Figure 7.7 depicts a tree graph for distributing write access tokens $Enc_{pk_{wa}}(r_i)$ for each resource r_i at the coarse-grained level. In addition, write access performed by authorized users requires an additional update token SK_{x-r_i} for each resource r_i. These set of tokens are distributed the same way as read access tokens demonstrated in Figure 7.5.

7.6 Analysis

7.6.1 Security Guarantees

Our two-leveled access control scheme provides the following privacy guarantees for each participant in the system:

Privacy against the cloud provider. The cloud provider does not learn any of the content of the files that he stores. He learns the frequency of access to particular blocks

and the type of request: read or write. The provider does not learn the exact files that have been accessed within a block either for reading or writing. In addition he can distinguish write requests from the data owner a block that are removing the appended updates and integrating them in the main content of the file. We formalize these guarantees in the following way.

Definition 23. *Let D be the data in the form of blocks that the cloud provider stores For any two tuples of request,we define the equivalence relation $(q_1^0, \ldots, q_\ell^0) \equiv_D (q_1^1, \ldots, q_\ell^1)$ to hold if*

- *q_i^0 and q_i^1 are the same type of requests (read or write), and if they are both write requests, they are either both from the data owner of these files or both are from users different from their data owner.*
- *$q_i^0(D) = q_i^1(D)$ where $q(D)$ denotes the matching block for a query, if any.*

Theorem 16. *Let $(q_1^0, \ldots, q_\ell^0) \equiv_D (q_1^1, \ldots, q_\ell^1)$ be two sequences of queries that are equivalent according to Definition 23.The views of the cloud provider from the execution of these two sequences are indistinguishable.*

Privacy against the users. The coarse-grained access control provides that users will be able to receive the encrypted content only of blocks in which they are authorized to read at least one file. The fined-grained access control mechanism guarantees that users will be able to decrypt only the parts of the block returned to them, which they are authorized to read. Similarly, in the case of write access users would be able to submit updates only the blocks in which they have write access at least to one file. And their updates will be associated only with the files that they are authorized to update. Users learn which of the files that they can read are allocated in the same block. This information is not revealed during writing since the user does not learn to which block his update will be appended (unless he already the location of the file from his read access).

Privacy against data owners. The data owner of a block does not learn which users have accessed the block neither for reading nor for writing. The file identifiers for the

submitted file updates are not associated with the users that have submitted them. However, he does learn the whole sequence of updates that have been submitted to a file as opposed to just the most recent version.

7.6.2 Performance Analysis

The computational overhead required by each participant in the system is described as follows:

Read Access

- **Data Owner**:
  - Setup — computation of the authorization trees with decryption keys and access tokens. The work is proportional to the number of files in the database and the number of users.
  - File access authorization of a user — update work for the tree with decryption keys and the tree with access tokens: in the worst case proportional to the depth of the trees.
  - File access revocation of a user — update work for the tree with decryption tokens: in the worst case proportional to the depth of the tree. The updates for the tree with the access tokens can be executed at larger intervals of time to achieve better amortized efficiency for updates.
- **User**: retrieving access tokens and decryption keys can be proportional to the number of files that the user is authorized to access. However, this work is enough to retrieve the access credentials for all files. Once retrieved the use can store them locally and use them directly for his requests. New credentials will have to be retrieved only when there has been an update of the access trees that has involved change of the credentials relevant for the particular user.
- **Cloud provider**: in order to map an access request to a particular block the cloud provider will have to execute the **File_Access_Check** function for the submitted

token and each block. This cost can be reduced if after the first retrieval of a file the user remembers the block id that contained the file, and the next time he need to access the same file he also submits this id. In this case the cloud provider would need to run a single check and verify that the block pointed in the access request does really contain the file of interest. Applying this optimization for efficiency reveals some additional information to the cloud, namely allows him distinguish first time access requests from repeated request, however, still without him being able to link requests to the same file. Further the user can choose whether to submit the block identifier that he has in repeated requests to weaken the additional leakage to the cloud.

Write Access

- **Data owner**: The enforcement of write access control requires duplication of the tree structures that were necessary for the read access control but this time with credential necessary for the write access. This comes as an overhead in the setup phase when these structures are computed by the data owner and also each update of the access rules will necessitate update of both types of trees since the encryption and decryption (relevant for write and read access) need to be synchronized. Also periodically the data owner would need to process the blocks and compact the updates for each file back in its initial memory location.

- **User**: The size of the blocks that the reader receives will increase depending on the frequency of the updates for a block as well as the time period at which the data owner processes the blocks and brings the updates back in place. At read access the user would need to locate both the initial place of the file he is looking for as well as all updates that have been submitted for that file, and then reconstruct the most recent version of the file.

- **Cloud provider**: The cloud provided would need to transfer larger blocks including both the original files as well as the updates. He would need to compute the identification tag for each authorized write update. This, however, would require constant

time.

We discuss further optimizations for the scheme that can improve the performance in certain case in Appendix E.2.

7.6.3 Discussion

Choosing the granularity for the access blocks in the read and write access control schemes affects the privacy guarantees for the scheme as well as its efficiency performance. The right granularity for each specific usage scenario will depend on the privacy and efficiency requirements for it, the expected patterns of access to the files and the expected frequency of access control rules' updates. Next we discuss some points that should be taken into consideration when choosing how to divide the files into access blocks:

- Each read access to a file entails a transfer of the content of the whole block containing the requested file from the cloud to the user. Thus the size of the blocks should be appropriate for the response delay that would be acceptable for the system. If we expect that the users for the system will have fairly diverse types of devices, we can create different levels of block granularity using different types of access tokens (as we discussed in the optimization section) and each user can choose which access block granularity fits best his privacy requirements as well as his communication limitations.

- If the database contains files that have "complementary" information, i.e., a user is likely to access only one of a these files (for example, if there are two files, one of which is accessed when a user wants to sell stocks, and the other one is accessed if the user wants to buy stocks), such files should be allocated to the same block. Thus even if the cloud provider manages to obtain some external information for the purpose of these two files, he still will not be able to figure out from the access block requests, which file was actually used.

- Files that are expected to be updated often should be allocated to blocks containing fewer files since the length of those blocks is expected to grow faster with the submitted updates. Alternatively the data owner should compact such blocks more often removing the appended updates and incorporating them in the main file content.

- In order to take maximal advantage of the optimization for caching read and write tokens as well as decryption keys, files that are likely to have their access rules changed often should be included in blocks with fewer files (these files might be with longer content, if we want to have blocks of approximately equal size). Most of the time the accesses to those files will require direct derivation of the decryption keys, which would have been updated, and this will be affected by the size of the tree for their respective access block.

- Since the view of the cloud provider of the access requests amounts to the frequency at which each access block is matched, files that are expected to have high access rates should be distributed across different blocks. This should prevent a certain block being accessed much more often than the rest, which might reveal information about the content of the block to the cloud provider.

Part III

Secure Data Sharing through Secure Search

Chapter 8

Practical Secure Search

8.1 Motivation and Contributions

Often, different parties possess data of mutual interest. They might wish to share portions of this data for collaborative work, but consider the leak of unrelated portions to be a privacy issue for themselves or their clients. Thus, methods that provide a well-defined and secure sharing of the data between untrusting parties can be useful tools. One such method that we introduce, is the ability for a client to search the information residing on another server without revealing to the server his identity or the content of his query; at the same time, it is desirable to guarantee that query capability is only granted to appropriate clients and that they do not learn anything unrelated to the query. Such a tool is useful in deciding and agreeing upon information-sharing between parties who do not initially know if they have data worth sharing with each other, and do not want to share information until they do. In addition the very fact that a client is interested in running certain queries is considered sensitive, and thus both his identity and the query content must be protected from the server.

The system we are proposing has many possible applications. For example, two intelligence agencies might like to search each other's data to discover if they have complementary information about the same parties. Similarly, the police may need to search the databases of different institutions, e.g., banks for information about people suspected of embezzlement. Even outside of law enforcement, this type of search might be useful to a physician who

wants to find out about other patients with the same rare disease as a patient of his own, along with treatment methods that have given good results. Or institutions might wish to protect logs containing sensitive information about the activities of their members, and yet allow restricted searches on information about suspicious behavior that, when correlated across different domains, may help detect attacks. These scenarios all present a common problem: a facility has data that legitimately could or should be shared with another party, embedded within a large amount of data that should be held confidential. Further in cases such as business acquisition research and law enforcement the identity of the querier needs to be kept anonymous to avoid causing fluctuation in share prices or tipping off investigation.

8.1.1 Our Contributions.

We address the above concerns by defining and implementing Secure Anonymous Database Search (SADS) system. Our solution achieves search time efficiency that is sufficient for real-time search by considering relaxed adversarial models that still provide security guarantees that adequately address the requirements of many practical scenarios. As in [M. Bellare and O'Neill, 2007], we consider efficiency to mean sub-linear search time in the total size of the searchable data. Protocols such as [Song *et al.*, 2000; Boneh *et al.*, 2004; Waters *et al.*, 2004; cheng Chang and Mitzenmacher, 2005] achieve linear search time; to improve this complexity, we may be willing to sacrifice strict definitions of privacy and security in a limited and measurable manner. Thus, our goal is to guarantee practical performance and achieve the maximum privacy, security and anonymity possible under the efficiency requirement. We design a security architecture that facilitates relaxed security definitions and efficiency guarantees, and at the same time corresponds to viable practical setups.

We propose secure search protocols that allow to identify and retrieve matching content. In our system we decouple the questions of search of matching document and their retrieval, and the document retrieval protocols that we introduce can be combined with any other search protocol for for the proposed security architecture. The main search functionality that we provide is keywords search.

We analyze the security properties of our protocols. In order to evaluate the practical performance of our solution we implement the system and measure its performance with

realistic working load of databases of size 50 GB. The search time that we achieve improves orders of magnitude existing solutions with stronger security guarantees and enters the scope of practicality incurring only about 30 % overhead compared to SQL queries and comparable to data transfer over SSL.

8.2 Security Architecture and Definitions

The general scenario we consider includes multiple parties who possess private sensitive data, which they are willing to share under certain very specific circumstances. Each execution of the scheme will involve a party who owns a set of documents he wishes to make available for secure anonymous keyword search by authorized parties. Any other party may be authorized to take the role of the querier, whose input is some keyword that he wishes to search for in the database. We will interchangeably refer to the first party as data owner or server, and the second party as querier or client. The goal is for the protocol to meet the following requirements:

- **Correctness:** The querier's output consists of all the matches, namely the indices of all documents containing the keyword. A tolerated probability of expected error (false positives or negatives) may be specified.
- **Client Security:** The data owner does not learn any information about the query (keyword).
- **Server Security:** The querier learns nothing about the data except for the specified output (matches) for his query.
- **Server Access Control:** Only parties authorized by the data owner can submit queries and receive outputs for this data.
- **Client Anonymity:** The data owner learns no information about the identity of the querier as chosen from amongst the pool of authorized parties. This also precludes information about linkage of two queries coming from the same client.
- **Practical efficiency:** This is a central requirement for our system, and we design our model and protocol accordingly. In particular, high communication complexity,

or per-query computation complexity that scales linearly with the number of words in a document will not be acceptable. This rules out the use of existing generic cryptographic techniques from secure multiparty computation [Yao, 1982; Yao, 1986; Goldreich *et al.*, 1987] and PIR [Chor *et al.*, 1998; Gertner *et al.*, 2000].

It is not hard to show that the security, anonymity, and efficiency requirements stipulated above are conflicting, and cannot all be achieved simultaneously without adjusting the model. For example, sublinear computation and constant communication conflict with client privacy, as they allow the server to gain information on the answers to the query, and thus on the query itself. Client anonymity seems to conflict with server access control, and obviously anonymity cannot be achieved if the server and client are the only two parties participating in the interaction. Trying to solve the latter problem by involving all parties in the system for each search is not practical (both in terms of efficiency, and since it requires a fixed and known set of parties).

Instead, we will expand our model by adding two new parties that will participate in each search, the *Index Server (IS)* and the *Query Router (QR)*. These may be viewed as neutral parties available to regulate the data sharing process without learning the participants' private inputs. The security and anonymity requirements with respect to these new parties will be reasonable, but weaker than those between the client and server; in return, they allow us to achieve practical efficiency.

Before elaborating on these requirements (which will complete our definition of SADS), we overview the general architecture of our SADS protocol, demonstrating the roles of IS and QR and their trust implications.

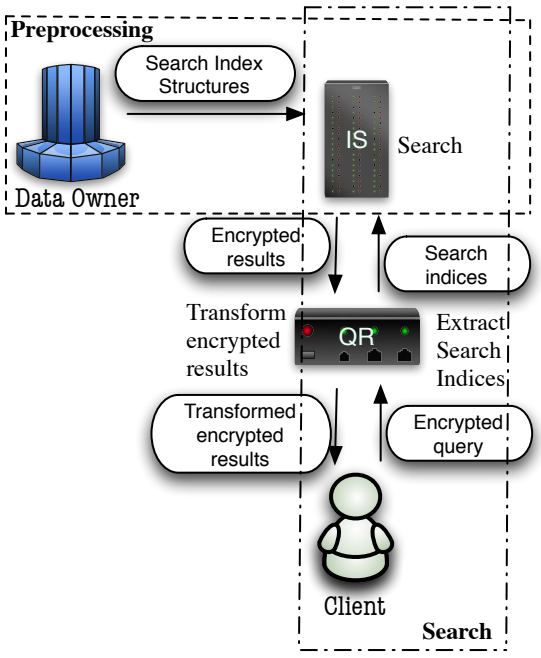


Figure 8.1: General Setup: The data owner makes its data available for search providing IS with search index structures, the client submits queries anonymously to IS via QR, IS sends back the search result though QR

Figure 8.1 illustrates the search protocol. A database owner generates a search index structure computed from (an encryption of) his data and gives it to the index server. This structure enables IS to answer (encrypted) queries but does not reveal information about the provided database. Outsourcing the search to IS prevents the data owner from finding out the results to encrypted queries. The IS sees the results, but does not know what documents they correspond to. At most, the IS will be able to tell when two submitted queries have overlapping results. This is mitigated by preserving the anonymity of the queriers with respect to the index server. However, providing such anonymity introduces a new problem: how to guarantee that only authorized users are submitting queries. This is addressed by the query router, who serves as an intermediary in the communication path between querier and IS. QR is trusted to know and protect the identities of the participants, while enforcing correct authorization before allowing queries to reach the IS. However, he is not trusted to see the content of the queries or results. Thus a client will submit his

encrypted query to the QR, who checks the authorization of the user, transforms the query and forwards it to the IS. The IS will send back search results to the QR, which will be able to forward them to the respective user. The results are encrypted so that the QR does not learn anything.

With this architecture in mind, we make the following requirements with respect to IS and QR:

- **Data Security Against IS and QR:** Both IS and QR learn no information about the data.
- **Client Anonymity Against IS:** IS learns no information about the identity of the querier. This again includes unlinkability.
- **Search Result Privacy Against IS:** Given a sequence of queries (forwarded to IS from QR, possibly by different clients), IS may learn which of the encrypted queries result in the same set of matching documents. No other information about the queries (or the client(s) who generated them) may be learned by IS.
- **Query Privacy Against QR :** Given a sequence of queries from a given client, QR may learn nothing beyond which of the encrypted queries are the same.

Thus we define secure anonymous database search as follows:

Definition 24. *A Secure Anonymous Database Search (SADS) system consists of protocols for server, client, IS and QR, satisfying all the correctness, security, anonymity, and efficiency requirements defined above between client and server and for IS and QR.*

In order to define formally the security notions behind the above properties we start with the standard simulation security notion of Canetti [Canetti, 2000b]. It guarantees that the queriers receive only the matching results, while none of the other parties in the protocol learns anything. Intuitively, what the definition captures is that a protocol is secure if the views of the participants in the real execution (namely their inputs, random inputs, outputs, and messages they receive) are indistinguishable from their views in an ideal execution where all parties send their inputs to a trusted party who computes the

results and sends them back to the receivers. We modify the definition to introduce the notion of privacy leakage as follows.

Definition 25. *A protocol π is a secure encrypted search protocol with privacy leakage $\mathcal{L}$, if for every real world adversary, there exists a simulator such that the view of the adversary in the real world execution, where he interacts with the honest parties, is indistinguishable from his view in an ideal world execution where he interacts with a simulator that takes as input $\mathcal{L}$ (and simulates the honest parties).*

Now the above privacy properties can be translated as the following types of leakage $\mathcal{L}$ in Definition 25

- *False Positive Database Leak*: a fraction of records that do not match the search criterion. This is leakage that the client learns and thus needs to be be provided to the client simulator in the security proof.
- *Search Pattern*: the equality pattern of the submitted queries. This is leakage to the IS and needs to be provided to the IS simulator in the security proof.
- *Results' Pattern*: the equality pattern among the results. This is leakage to the IS and needs to be provided to the IS simulator in the security proof.
- *Client queries linkability*: what queries were submitted by the same client. This information will be leaked to the QR and thus needs to be provided to the QR simulator as well.

8.2.0.1 Building Blocks

In this section we introduce the building blocks that we would use for our search protocol.

8.2.1 DET-CCA Deterministic Private Key Encryption Scheme.

While the standard definitions of security (e.g., [Goldwasser and Micali, 1982]) require an encryption scheme to be probabilistic, a deterministic scheme will allow us considerable efficiency gains, while still providing a level of security which is acceptable in our setting

(security-up-to-equality). This tradeoff follows the idea introduced by [M. Bellare and O'Neill, 2007], who define deterministic encryption in the public-key setting, and show how to convert a standard (probabilistic) PKE to a deterministic one. We follow the same approach, adapting it to the secret key setting.

We start by defining chosen-ciphertext (CCA) security for deterministic encryption. The adversary $\mathcal{A} = (A_1, A_2)$, defined as in [M. Bellare and O'Neill, 2007], is a pair of polynomial time algorithms that share neither coins nor state and has high min-entropy $\omega(\log(k))$ (this is the case for any adversary if the underlying plaintext domain is dense).

Definition 26 (DET-CCA)**.** *Let* $\Pi^{det} = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ *be a private key encryption scheme and* $\mathcal{A} = (A_1, A_2)$ *be an adversary against it. We conduct the following two experiments:*

$\boldsymbol{DET\text{-}EXP}^{0}_{\pi^{det},\mathcal{A}}(n)$	$\boldsymbol{DET\text{-}EXP}^{1}_{\pi^{det},\mathcal{A}}(n)$
$s \leftarrow \mathsf{Gen}(1^n)$	$s \leftarrow \mathsf{Gen}(1^n)$
$(\boldsymbol{x}_1, t_1) \leftarrow A_1(1^n)$	$(\boldsymbol{x}_0, t_0) \leftarrow A_1(1^n); (\boldsymbol{x}_1, t_1) \leftarrow A_1(1^n)$
$\boldsymbol{c} \leftarrow \mathsf{Enc}_s(\boldsymbol{x}_1)$	$\boldsymbol{c} \leftarrow \mathsf{Enc}_s(\boldsymbol{x}_0)$
$t' \leftarrow A_2^{\mathsf{Enc}_s}(1^n, \boldsymbol{c})$	$t' \leftarrow A_2^{\mathsf{Enc}_s}(1^n, \boldsymbol{c})$
$output \begin{cases} 1 \ if\ t' = t_1 \\ 0 \ else \end{cases}$	$output \begin{cases} 1 \ if\ t' = t_1 \\ 0 \ else \end{cases}$

We define the adversary advantage as $\boldsymbol{Adv}^{DET-CCA}_{\pi^{det},\mathcal{A}} = Pr[\boldsymbol{DET\text{-}EXP}^{0}_{\Pi^{det},\mathcal{A}}(n)=1] - Pr[\boldsymbol{DET\text{-}EXP}^{1}_{\Pi^{det},\mathcal{A}}(n)=1]$. *We say that* Π^{det} *is DET-CCA secure if for all adversaries* $\mathcal{A}$ *the advantage* $\boldsymbol{Adv}^{DET-CCA}_{\pi^{det},\mathcal{A}}$ *is negligible.*

Next we give a construction for converting any semantically secure private key encryption scheme into a deterministic DET-CCA secure private key encryption scheme.

Construction 17 (Deterministic Private Key Enc)**.** *Let* $\Pi = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ *be any probabilistic private key encryption scheme and let H be a hash function, which we will model as a random oracle. We define a deterministic private key encryption scheme* $\Pi^{det} = (\mathsf{Gen}', \mathsf{Enc}', \mathsf{Dec}')$ *as follows:*

- $s = \mathsf{Gen}'(1^n) = \mathsf{Gen}(1^n)$

- $c = \mathsf{Enc}'_{\mathsf{s}}(\mathsf{x}) = \mathsf{Enc}_{\mathsf{s}}(\mathsf{x}; \mathsf{H}(\mathsf{s}, \mathsf{x}))$
- $x = \mathsf{Dec}'_{\mathsf{s}}(\mathsf{c}) = \mathsf{Dec}_{\mathsf{s}}(\mathsf{c})$, $r = H(s, x)$; *return* x *if* $\mathsf{Enc}_{\mathsf{s}}(\mathsf{x}, \mathsf{r}) = \mathsf{c}$ *and* $\perp$ *otherwise.*

Theorem 18. *Let* $\Pi = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ *be any probabilistic private key encryption scheme and* Π^{det} *be the corresponding deterministic scheme according to Construction 17. Let* $\mathcal{A} = (A_1, A_2)$ *be a DET-CCA adversary with min-enthropy* μ *against* Π^{det} *that outputs vectors of size* v *and makes at most* q_h *queries to the hash oracle and* q_d *queries to the decryption oracle. Let* ms *and* mc *be the max secret key and the max-ciphertext probabilities for* Π*. The there exists an IND-CPA adversary* $\mathcal{B}$ *against* Π *such that*

$$\boldsymbol{Adv}^{DET-CCA}_{\pi^{det},\mathcal{A}} \leq \boldsymbol{Adv}^{IND-CPA}_{\pi,\mathcal{B}} + \frac{2q_h v}{2^{\mu}} + 2q_h ms + 2q_d mc,$$

where $\mathcal{B}$ *makes at most* v *queries to its oracle and its running time is within* $O(q_h(T_\epsilon + q_d))$ *and* T_ϵ *is the running time for the encryption algorithm.*

This proof follows the proof of Theorem 5 in [M. Bellare and O'Neill, 2007], replacing the (public key) encryptions with calls to an encryption oracle for the private key encryption scheme.

We instantiate the above deterministic private key encryption scheme following the construction of RSA-DOAEP in [M. Bellare and O'Neill, 2007] but with different primitives that give more security and the group property that we need. We use the Pohlig-Hellman (PH) permutation [Pohlig and Hellman, 1978] and the SAEP+ (short for Simple-OAEP) padding construction introduced in [Boneh, 2001].

Definition 27 (Pohlig-Hellman(PH) Permutation)**.** *Let* $p = 2q + 1$ *be a safe prime, and consider the prime order subgroup* G_q *of* Z_p*. Let* $k \in Z_q$ *(called a key). The Pohlig-Hellman function is defined as follows:*

$$PH_k(x) = x^k \ mod \ p$$

Note that the Pohlig-Hellman function has a commutative property for its keys:

$$PH_{k_1}(PH_{k_2}(x)) = PH_{k_1 k_2}(x).$$

As we shall see, this property will carry over to the deterministic encryption we construct, and will be used in our construction of re-routable encryption.

The Pohlig-Hellman permutation (sometimes referred to as the exponentiation cipher) was proposed over 30 years ago [Pohlig and Hellman, 1978], and is assumed to be hard to invert, as we formalize below. Like plain RSA and other classical ciphers, PH does not directly provide the standard definitions of security such as semantic security (nor does it satisfy our definitions of deterministic security). We will use this as a basis for a construction utilizing hashing, that will achieve CCA security, based on the following assumption.

Assumption 1 (The Pohlig-Hellman (PH) Assumption)**.** *Let A be a ppt, and H be a random oracle, define $\boldsymbol{Adv}_A^{PH}(n) = Pr[k \leftarrow Z_q, x \leftarrow G_q, y = PH_k(H(x)) : A^{PH_k(\cdot)}(y) = x]$. The PH assumption is that for all ppt A, $\boldsymbol{Adv}_A^{PH}(n)$ is negligible.*

Intuitively, this assumption captures the fact that even with chosen message attack, it is hard to recover the preimage of $PH(x)$ for a random x. The work of [Gjøsteen, 2008] and [Damgard *et al.*, 2006] show that the above assumption is equivalent to the DDH assumption. The PH assumption will suffice for proving CCA security of our probabilistic construction. Our resulting deterministic construction can be proven secure based on either the PH assumption, or the DDH assumption (all of these are in the random oracle model).

The SAEP+ (short for Simple-OAEP) padding scheme was introduced by Boneh in [Boneh, 2001], as a way to make trapdoor functions into CCA secure public key encryption schemes (in the random oracle model). The padding is defined as follows.

Definition 28 (SAEP+ Padding[Boneh, 2001])**.** *Let $M \in \{0,1\}^m$ be a message and $r \in \{0,1\}^{s_1}$ be a random string. Let*

$$H : \{0,1\}^{s1} \rightarrow \{0,1\}^{m+s_0} \text{ and}$$

$$G : \{0,1\}^{m+s1} \rightarrow \{0,1\}^{s_0}$$

be hash functions. We define SAEP+ as follows

$$SAEP^{+}(M,r) = ((\ M \ ||\ G(M||r)\ ) \oplus H(r)) \ ||\ r.$$

Boneh [Boneh, 2001] proves that when combined as an input to trapdoor permutations (such as RSA and Rabin functions), this yields CCA secure public key encryption schemes. It is a simpler scheme than OAEP [Bellare and Rogaway, 1995] and provides better security

guarantees. We prove an analogous theorem in the private key setting, namely that when SAEP+ is combined with a secret key function that is hard to invert, this yields a CCA secure private key encryption. In order to define things properly, since in this setting the private key function is even hard to evaluate without a secret key, we provide the adversary access to an oracle evaluating the function. If the function remains hard to invert with access to such an oracle (as in Assumption 1), we prove that combining it with SAEP+ padding yields a CCA-secure encryption scheme. We omit the general theorem and proof here, but include the construction and theorem for the specific PH based scheme that we use.

Definition 29 (PH-SAEP+ Encryption)**.** *We define PH-SAEP+* $= (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ *in the following way*

- $\mathsf{Gen}(1^n) = (p, k, k')$ *where* p *is a prime that is publicly known and* k *is a secret key and* k' *is its inverse, which is efficiently computable.*
- $c = \mathsf{Enc}_k(x) = \mathsf{PH}_k(SAEP+(M, r)) = \mathsf{PH}_k(((M \,||\, G(M||r)) \oplus H(r)) \,||\, r)$ *where* r *is chosen at random.*
- $\mathsf{Dec}(c)$*:*
  1. *Compute* $c' = c^{k'} \mod p$ *where* $c' = c'' \,||\, r$*.*
  2. *Extract* r *and compute* $H(r)$*.*
  3. *Compute* $x \,||\, g = c'' \oplus H(r)$*.*
  4. *Verify that* $g = G(x||r)$ *and return* x*.*

Theorem 19. *Assume that no algorithm with running time* t *can solve the discrete log problem with probability more than* ϵ*. Then PH-SAEP+ is chosen ciphertext secure scheme in the random oracle model satisfying the following:*

$$t' \leq t/2 - O(q_D + q_G + q_H^2)$$
$$\epsilon' \leq \epsilon^{1/2} + q_D/2^{s_0} + q_D/2^{s_1},$$

for an adversary with running time t' and advantage ϵ' that issues q_D decryption queries, q_G queries to G, q_H queries to H where s_1 is the length of the randomness used and s_0 is the length of the output of G.

Proof. In [Boneh, 2001] Boneh shows how to construct a CCA secure public key encryption from a trapdoor permutation and the SAEP+ scheme. We modify this technique to obtain a CCA secure private key encryption scheme.

The idea of public key scheme is captured in the functionality of a trapdoor permutation. The definition of a trapdoor permutation is a function f which is easy to evaluate, hard to invert on its own, but easy to invert with the knowledge of some trapdoor information. Now we consider the Pohlig-Hellman function. The value $PH_k(x)$ is hard to invert without knowledge of k because of the hardness assumption of the discrete logarithm problem. On the other hand, given k and $PH_k(x)$ it is easy to compute the inverse k^{-1} and find m. The only thing left to be able to view $PH_k(x)$ as a trapdoor permutation is to provide a way to compute easily without knowing k. We can achieve this by queries to an oracle O^{PH_k} that implements the functionality of $PH_k(x)$.

Boneh ([Boneh, 2001]) defines the set partial one-wayness problem to find a set of values that contains the inverse of a given value produced by a trapdoor function f and connects the security of the f-SAEP+ to the hardness of solving the onewayness problem. We translate this result to the case of the Pohlig-Hellman function.

Definition 30 (PH Set Partial One-wayness Problem)**.** *Let $PH_k(x)$ be the Pohlig-Hellman function that is modeled as a random oracle O^{PH_k} and k is secret. We say that an algorithm $\mathcal{A}$ solves the set partial one-wayness problem $(PH_k(x), r)$ if given $c = PH_k(x)$ it produces a set $S = \{x_1, \ldots, x_r\}$ such that $c = PH_k(x_i)$ for some $1 \leq i \leq r$.*

Lemma 15. *Let $\mathcal{A}$ be a (t, q_D, q_H, q_G) chosen ciphertext attack algorithm in the random oracle model where $\mathcal{A}$ runs in time t, makes q_H queries to the oracle H, q_G queries to the oracle G and q_D decryption queries of PH-SAEP+ and has an advantage ϵ. Then there exists an uniform algorithm $\mathcal{B}$ that solves the set partial one-wayness problem $(PH_k(x), q_H)$*

with the following time and advantage:

$$time(\mathcal{B}) \leq time(\mathcal{A}) + O(q_H + q_G + q_D)$$
$$adv(\mathcal{B}) \leq adv(\mathcal{A})(1 - q_D/2^{s_0} - q_D/2^{s_1})$$

The proof of the Lemma 15 is analogous to the proof of Theorem 5 in [Boneh, 2001] where the evaluation of the trapdoor function f are substituted with calls to the random oracle O^{PH_k} implementing the Pohlig-Hellman function.

We now have all the necessary tools for the theorem proof. Let us assume that there is a (t, q_D, q_H, q_G) chosen ciphertext adversary $\mathcal{A}$ against PH-SAEP+ with advantage ϵ. By Lemma 15 we know that there is a t'-time adversary $\mathcal{B}$ that solves the PH set partial one-wayness problem (PH_k, q_H) with advantage ϵ' for some t' and ϵ'. Fujisaki et al. in [Fujisaki *et al.*, 2004] demonstrate an algorithm that runs $\mathcal{B}$ twice on C^* and αC^* for some α and uses the resulting sets S and S_α to compute the k-th root of C^* in time $O(q_H^2)$ and hence breaks the PH with probability ϵ'^2. The theorem now follows. And Corollary 20 also follows. □

Corollary 20 (PH-DSAEP+)**.** *Let PH-DSAEP+ be the deterministic private key encryption scheme derived by applying Construction 17 to PH-SAEP+. PH-DSAEP+ is DET-CCA secure under the discrete log assumption and in the random oracle model.*

8.2.2 Re-Routable Encryption.

Re-routable encryption is a new primitive we will use in our system to protect identities, when routing (encrypted) queries from an authorized client to IS, and also when routing the (encrypted) results back to the client. Informally, re-routable encryption is a protocol to send an encrypted message, or some function of the message, from a sender to receiver through a query router QR, such that two security requirements are satisfied. We require first, security of the sender's message with respect to QR, and second, anonymity of the sender with respect to the receiver. The formal definitions of these properties are presented below. We note that this primitive may be of independent interest, e.g., in cases where the query router QR is an intermediary in a multiparty computation protocol that also has anonymity requirements for some of the participants.

Definition 31. *A re-routable encryption scheme consists of algorithms* $(\mathsf{Gen},\ \mathsf{Enc}, \mathsf{Enc}-\mathsf{QR}, \mathsf{Trans}, \mathsf{Dec}-\mathsf{R})$*:*

- $\mathsf{Gen}(1^{\mathsf{k}}, \mathsf{Sender}, \mathsf{QR}, \mathsf{Receiver})$ *outputs three keys* $(sk,\ qrk_{\{S,R\}}, rk)$ *for the sender, the QR, and the receiver.*
- $\mathsf{Enc}(\mathsf{sk}, \mathsf{m}) = \mathsf{c}$ *encrypts* m *with the sender's key.*
- $\mathsf{Trans}(\mathsf{c}, \mathsf{S}, \mathsf{st_i}) = (\mathsf{R}, \mathsf{st_{i+1}})$ *identifies the receiver of the message coming from* S *based on the inner state* st_i *of QR, and computes the new state of QR.*
- $\mathsf{Enc}-\mathsf{QR}(\mathsf{c}, \mathsf{qrk_{\{S,R\}}}, \mathsf{st_i}) = (\overline{\mathsf{c}}, \mathsf{st_{i+1}})$ *transforms the ciphertext* c *to a message* $\overline{c}$ *for the receiver* R*.*
- $\mathsf{Dec}-\mathsf{R}(\overline{\mathsf{c}}) = \overline{\mathsf{m}}$ *extracts the information that was sent to the receiver from the query router.*

Definition 32 (Message Security)**.** *Let* S *be a security definition using an adversary* A *and applicable to a general encryption scheme. Let* $R = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Enc}-\mathsf{QR},\ \mathsf{Trans}, \mathsf{Dec}-\mathsf{R})$ *be a re-routable encryption scheme. We say that* R *provides* S*-message security with respect to QR if* $(\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec}-\mathsf{R}\circ\mathsf{Enc}-\mathsf{QR})$ *meets* S *when* A *is supplemented with* $qrk_{\{S,R\}}$*.*

This definition is intentionally non-specific, and can be instantiated using different definitions of security for encryption. For our scheme, we will instantiate this definition both with a standard semantic security notion, and with deterministic encryption security notion.

Definition 33 (Sender Anonymity W.r.t. Receiver)**.** *Let* Q_0 *and* Q_1 *be two users with keys* q_0 *and* q_1 *respectively. We say that the re-routable encryption scheme* $(\mathsf{Gen}, \mathsf{Enc},\ \mathsf{Enc}-\mathsf{QR},\ \mathsf{Trans},\ \mathsf{Dec}-\mathsf{R})$ *with a security parameter* k *preserves the anonymity of the the sender with respect to the receiver if for any polynomial time adversary* $\mathcal{A}$ *that given* $\mathsf{Enc}-\mathsf{QR}(\mathsf{Enc_{q_b}}(\mathsf{m}))$ *for* $b \leftarrow_R \{0,1\}$ *outputs a guess* b'*, the following holds:* $|Pr[b = b'] - \frac{1}{2}| < negl(k)$*.*

A re-routable encryption scheme is secure if it meets both of the above definitions.

We will now show one method for constructing a re-routable encryption scheme from an encryption scheme that possesses the following group property:

Definition 34 (Encryption Group Property)**.** *Let* $\Pi = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ *be a deterministic private key encryption scheme. We say that* Π *has a group property if the keys for the encryption scheme form a group and for any message* m *and any keys* $k1$ *and* $k2$ *the following holds:* $\mathsf{Enc}_{\mathsf{k1}}(\mathsf{Enc}_{\mathsf{k2}}(\mathsf{m})) = \mathsf{Enc}_{\mathsf{k1}\cdot\mathsf{k2}}(\mathsf{m})$.

Construction 21 (Simple Re-rerouteable Encryption)**.** *Let* $\Pi = (\mathsf{Gen}', \mathsf{Enc}', \mathsf{Dec}')$ *be an encryption scheme with the group property from Definition 34. We construct a reroutable encryption scheme* $(\mathsf{Gen}, \mathsf{Enc}, \mathsf{Enc}-\mathsf{QR}, \mathsf{Trans}, \mathsf{Dec}-\mathsf{R})$ *as follows:*

- $\mathsf{Gen}(1^{\mathsf{k}})$*: If there is a single Sender and multiple Receivers, we have the following setup phase. The Sender runs* $\mathsf{Gen}'\ (1^k)$ *to obtain keys* sk *and* tk*. Each Receiver runs* $\mathsf{Gen}'\ (1^k)$ *to create a key* rk'*. Sender, Receiver, and QR then run an MPC protocol such that the Receiver learns* $rk = rk' \cdot tk^{-1}$ *and the QR gets* $qrk = \frac{sk \cdot tk}{rk}$ *secure multiparty computation with* sk*,* tk *as input from Sender,* rk *as input from Receiver, and* $qrk = \frac{sk}{rk}$ *as output for QR. In the case of single Receiver and multiple senders the setup is similar except that the Receiver chooses* tk*.*
- $\mathsf{Enc}(\mathsf{sk}, \mathsf{m})$*: Sender computes* $\mathsf{Enc}'(\mathsf{sk}, \mathsf{m}) = \mathsf{c}$*.*
- Trans*: QR chooses a Sender, Receiver pair.*
- $\mathsf{Enc}-\mathsf{QR}(\mathsf{qrk}, \mathsf{c})$*: QR computes* $\mathsf{Enc}'(\mathsf{qrk}, \mathsf{c}) = \overline{\mathsf{c}}$*.*
- $\mathsf{Dec}-\mathsf{R}(\mathsf{rk}, \overline{\mathsf{c}})$*: Receiver computes* $\mathsf{Dec}'(\mathsf{rk}, \overline{\mathsf{c}}) = \mathsf{m}$*.*

Theorem 22. *Let* $\Pi = (\mathsf{Gen}', \mathsf{Enc}', \mathsf{Dec}')$ *be an encryption scheme with a group property, satisfying a security definition* S*. The reroutable encryption* $(\mathsf{Gen}, \mathsf{Enc}, \mathsf{Enc}-\mathsf{QR}, \mathsf{Trans}, \mathsf{Dec}-\mathsf{R})$ *obtained from* Π *using Construction 21 provides* S*-message security and ensures Sender anonymity w.r.t the receiver.*

Proof. First, we show that the obtained re-routable encryption provides S-message security. We argue this in the case of a single sender and multiple receivers (the proof for multiple senders and a single receiver is similar). Assume that $(\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec}-\mathsf{R} \circ \mathsf{Enc}-\mathsf{QR})$ does not meet the security definition S. Therefore there exists an adversary $\mathcal{A}$ that can obtain information t about a message m given $\mathsf{Enc}(\mathsf{m})$. We can construct an adversary $\mathcal{A}'$ against

$\Pi = (\mathsf{Gen}', \mathsf{Enc}', \mathsf{Dec}')$ that learns the same information t as follows: $\mathcal{A}'$ acts as a challenger for $\mathcal{A}$. $\mathcal{A}'$ chooses random keys sk, tk, rk' for the sender and the receivers and computes the corresponding keys rk for the receivers. He computes the transformation keys and sends them to $\mathcal{A}$. $\mathcal{A}'$ receives encryption requests for sender and receivers from $\mathcal{A}$ forwards them to his own challenger. He transforms the ciphertexts that he receives back using tk for the sender or the corresponding rk for a receiver, and forwards the resulting ciphertexts to $\mathcal{A}$. The encryption scheme adversary receives the information t that $\mathcal{A}$ learns. The above execution interacting with the simulator $\mathcal{A}'$ will be indistinguishable from an execution where $\mathcal{A}$ interacts with real sender and receivers. Thus $\mathcal{A}'$ will have the same advantage of breaking the S message security of the re-routable encryption as the advantage of $\mathcal{A}$ against the underlying encryption scheme.

Second, we prove that the re-routable encryption ensures Sender anonymity w.r.t the receiver. Let q_0 and q_1 be the keys of two senders and $qr_0 = \frac{rk}{q_0}$ and $qr_1 = \frac{rk}{q_1}$ be the corresponding transformation keys at the third party. Let m be any message. Now using the group property of Π we have

$$\mathsf{Enc} - \mathsf{QR}(\mathsf{Enc}_{\mathsf{q_0}}(\mathsf{m})) = \mathsf{Enc}_{\mathsf{qr_0}}(\mathsf{Enc}_{\mathsf{q_0}}(\mathsf{m})) =$$
$$= \mathsf{Enc}'_{\mathsf{qr_0}}(\mathsf{Enc}'_{\mathsf{q_0}}(\mathsf{m})) = \mathsf{Enc}'_{\mathsf{qr_0 \cdot q_0}}(\mathsf{m}) = \mathsf{Enc}'_{\mathsf{rk}}(\mathsf{m})$$

$$\mathsf{Enc} - \mathsf{QR}(\mathsf{Enc}_{\mathsf{q_1}}(\mathsf{m})) = \mathsf{Enc}_{\mathsf{tp_1}}(\mathsf{Enc}_{\mathsf{q_1}}(\mathsf{m})) =$$
$$= \mathsf{Enc}'_{\mathsf{qr_1}}(\mathsf{Enc}'_{\mathsf{q_1}}(\mathsf{m})) = \mathsf{Enc}'_{\mathsf{qr_1 \cdot q_1}}(\mathsf{m}) = \mathsf{Enc}'_{\mathsf{rk}}(\mathsf{m})$$

Therefore the index server will always get the same ciphertext and cannot guess the user identity with probability non-negligibly different from 1/2. □

8.2.3 Bloom Filters.

The deterministic encryption scheme that we presented provides ciphertexts that are suitable to be used in efficient search protocols according to [M. Bellare and O'Neill, 2007]. Bellare et al. in [M. Bellare and O'Neill, 2007] suggest that the search functionality over encrypted data produced with a deterministic encryption should be realized by attaching "tags" that will be easily searchable and easily computed by both the querying party and

the server. We realize the "tagging" idea with a Bloom filters. Bloom filters were introduced in [Bloom, 1970]. They are structures that provide for efficient storage of sets for membership testing, which makes them ideal for term querying over document collections. Bloom filter search has a false positive rate which depends on the parameters of the structure; it can be made arbitrarily small by changing them. We are guaranteed there will be no false negatives.

A Bloom filter (BF) (Figure 8.2) is an m-bit array B. Initially, all bits of the filter are set to zero. There are n independent hash functions H_i where $1 \leq i \leq n$. The output of the hash functions is in the range $[0, m-1]$.

To add an entry W to the Bloom filter, we calculate the following values:

$$b_1 = H_1(W), b_2 = H_2(W), \cdots, b_n = H_n(W)$$

We add the entry W by setting $B[b_i] = 1$ for each $1 \leq i \leq n$.

To check whether W is present in the filter, we compute the indexes $b_1, \ldots, b_n$ as above. If $B[b_i] = 1$ for all $1 \leq i \leq n$, the entry is present; if any of the bits are 0, W is not present.

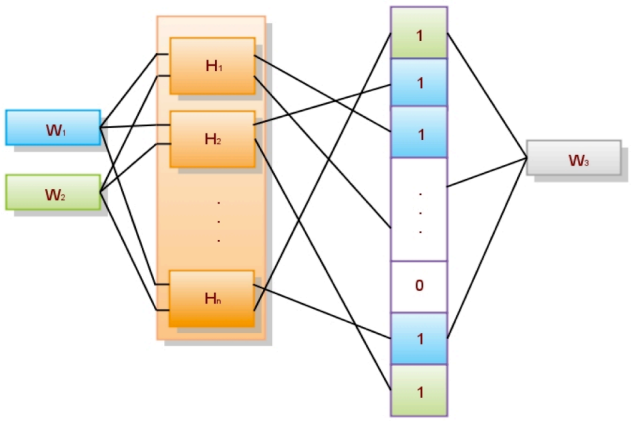


Figure 8.2: Bloom filters: w_1 and w_2 are real entries of the BF and w_3 is a false positive

It is clear that Bloom filter query will not return false negatives; the indices checked are the same as those set when the term was inserted. However, false positives are possible if the term indices happened to be set by other terms due to hash collisions. For example, if we have added entries with BF indexes $\{1, 2, 3\}$ and $\{4, 5, 6\}$, the Bloom filter will also

indicate that entries with indexes $\{1, 4, 6\}$ and $\{2, 3, 5\}$ are present, even though they have not been entered.

The size of the Bloom filter m and the number of hash functions used n depend on the number of entries that we want the filter to hold and the upper bound on the false positive rate that we are willing to accept. Let us have T entries that we insert in the Bloom filter. We assume that the output of a hash function has a uniform distribution and consequently the probability that a bit in the BF is not set to one by a hash functions is $1 - \frac{1}{m}$. Using this assumption we can compute the false positive rate of the Bloom filter as the probability of n query bits being set, which is

$$(1 - (1 - \frac{1}{m})^{nT})^n \approx (1 - e^{-\frac{nT}{m}})^n. \tag{8.1}$$

For a given m and n, the false positive rate obtained in Equation 8.1 is minimized when we set the number of hash functions to

$$n = \frac{m}{T} \ln 2. \tag{8.2}$$

One limitation of Bloom Filters is the inability to remove elements once they are added. We cannot simply zero the bits that are associated with the element, since they may have also been set by other inserted elements.

Bloom filters have had a wide variety of applications — dictionaries, databases, distributed hashing, P2P/overlay networks, routing ([Broder and Mitzenmacher, 2002]). They have been used in language modeling ([Talbot and Osborne, 2007]) and set intersection for keyword searches ([Reynolds and Vahdat, 2003]). We use Bloom filters to perform encrypted search on cipertexts produced by PH-DSAEP+.

The next two propositions formalize the information that is revealed by the Bloom filters computed from the entries of the records in a database in the case when we use different hash functions across different Bloom filters and when we use the same hash functions.

Proposition 1. *Let $\mathcal{A}$ be a distinguishing algorithm and we define his distinguishing advantage through the following experiment:*

1. *$\mathcal{A}$ chooses two data sets D_0 and D_1 such that*

(a) *D_0 and D_1 have the same number of records N.*

(b) *There is a permutation π if N elements such that record i i D_0 has the same number of entries as record $\pi(i)$ in D_1 and the same distribution of number of entries in the records.*

2. *A key K and a bit b are chosen at random, and $\mathcal{A}$ is given the BFs computed from the PH-DSAEP+$_K$ encryptions of the entries in the documents of D_b using independent set of hash functions for each document.*

3. *$\mathcal{A}$ outputs a bit b'.*

The distinguishing advantage $Adv_{\mathcal{A}} = |Pr[b' = b] - \frac{1}{2}|$ is negligible.

Proof. Since the hash functions used for each Bloom filter are chosen independently, the BF indices produces for any entry in one Bloom filter are indistinguishable from the BF indices of any entry in any other Bloom filter. Hence the Bloom filters that are generated from the same number of entries are indistinguishable. Since the records in the two database have the same distribution entries the Bloom filters corresponding to the two database are indistinguishable. □

Proposition 2. *Let $\mathcal{A}$ be a distinguishing algorithm and we define his distinguishing advantage through the following experiment:*

1. *$\mathcal{A}$ chooses two data sets D_0 and D_1 such that*

(a) *D_0 and D_1 have the same number of records N.*

(b) *There is a permutation π if N elements such that $R_i^0 \bigcap R_j^0 = R_{\pi(i)}^1 \bigcap R_{\pi(j)}^1$ for all $1 \leq i, j \leq N$ where $R_i^0, R_j^0 \in D_0$ and $R_{\pi(i)}^1, R_{\pi(j)}^1 \in D_1$.*

2. *A key K and a bit b are chosen at random, and $\mathcal{A}$ is given the BFs computed from the PH-DSAEP+$_K$ encryptions of the entries in the documents of D_b using the the same set of hash functions for each Bloom filter.*

The distinguishing advantage $Adv_{\mathcal{A}} = |Pr[b' = b] - \frac{1}{2}|$ is negligible.

Proof. From Corollary 20 we have that the PH-DSAEP+$_K(w_0)$ and PH-DSAEP+$_K(w_1)$ are indistinguishable when $w_0 \neq w_1$. Therefore the BF indices computed from PH-DSAEP+$_K(w_0)$ and PH-DSAEP+$_K(w_1)$ when $w_0 \neq w_1$ will be also indistinguishable. It follows that the Bloom filters of two records with the same number of entries computed under the same hash functions are indistinguishable.

Let us assume that there is a distinguisher $\mathcal{A}$ that can distinguish the whether he is given the Bloom filters of D_0 or D_1. We construct $D_0 = \{R_i^0\}_{i=1}^N$ and $D_1 = \{R_i^1\}_{i=1}^N$ as follows: we set R_i^0 and R_i^1 to have the same entries for $1 \leq i \leq N-1$ and R_N^0 and R_N^1 to differ in only one entry w such that $w \in R_N^0$ and $w \notin R_i^0$ for $1 \leq i \leq N-1$. Since the Bloom filters for the files R_i^0 and R_i^1, $1 \leq i \leq N-1$ are the same and $\mathcal{A}$ can distinguish the BFs of D_0 and D_1, it follows that $\mathcal{A}$ can distinguish the BFs of R_N^0 and R_N^1, which is a contradiction with the conclusion above. □

8.3 Secure Anonymous Database Search Protocol

We proceed to describe our main protocol for secure anonymous database search. First we give the intuition for the primitives of the search scheme, followed by a formal definition. The search operations are also depicted in Figure 8.3.

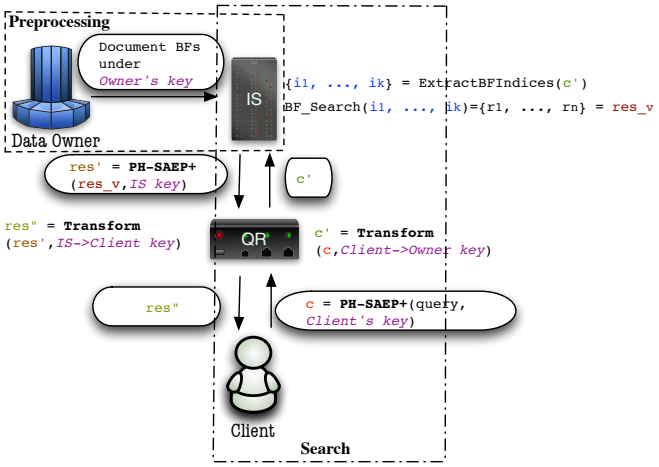


Figure 8.3: System Architecture and Data Flow.

- **Key Generation:** The data owner chooses an encryption key. IS chooses an encryption key. The client generates two keys for query submission and return result. To authorize the client to search the data owner, QR and the client run a key exchange protocol to allow QR to obtain a ratio key between the S's encryption key and C's query submission key. Also IS, the client and QR run a key exchange protocol so that QR obtains a ratio key between IS's encryption key and C's return result key.

- **Preprocessing:** The data owner generates for each of its documents a Bloom filter from the encryptions of its words under PH-DSAEP+ under his key. He sends the resulting Bloom filters to IS.

- **Query Submission:** We instantiate the re-routable encryption protocol for query submission as follows: the client encrypts his query with PH-DSAEP+ with his key and sends it to QR, QR re-encrypts the ciphertext with its transformation key from the client's to the owner's key.

- **Search:** IS extracts Bloom filter indexes from the encryption it receives from the QR and the obtained indexes to execute BF search to get the result res.

- **Query Return:** The query result is returned with a different instantiation of the re-routable encryption protocol: IS encrypts res with PH-SAEP+, sends it to QR, QR re-encrypts the ciphertext with the return result transformation key from the IS's key to the client's key and sends it to the client.

Construction 23 (Secure Anonymous Database Search)**.** *Let us have a* data owner *(**S**), a* client *(**C**), a* query router *(**QR**) and an index server (IS). Let FP be the upper bound on the false positive rate that we allow for search. Let h and m be parameters computed based on the sizes of the documents in the database of S such that a Bloom filter of size 2^m bits using h hash functions with as many entries as the largest document in the database allows at most false positive rate of FP.*

Let us have the following schemes:

- $(\mathsf{Gen}_{\mathsf{result}}, \mathsf{Enc}_{\mathsf{result}}, \mathsf{Trans}_{\mathsf{result}}, \mathsf{Enc}-\mathsf{QR}_{\mathsf{result}}, \mathsf{Dec}-\mathsf{R}_{\mathsf{result}})$ — *an instantiation of Simple re-routable encryption (Definition 21) with the encryption scheme PH-SAEP+.*

- $(\mathsf{Gen}_{\mathsf{query}}, \mathsf{Enc}_{\mathsf{query}}, \mathsf{Trans}_{\mathsf{query}}, \mathsf{Enc}-\mathsf{QR}_{\mathsf{query}}, \mathsf{Dec}-\mathsf{R}_{\mathsf{query}})$ — *an instantiation of Simple re-routable encryption (Construction 21) with the encryption scheme PH-DSAEP+, where* $\mathsf{Trans}_{\mathsf{query}}$ *identifies the IS and and* $\mathsf{Dec}-\mathsf{R}_{\mathsf{query}}$ *computes the BF indices corresponding from the query ciphertext for all documents.*

Figure 8.4 presents the scheme for secure anonymous database search that enables the client to search the database of the database owner with the help of the index server and the query router.

We can insatiate the above protocol for secure anonymous search in two ways depending on the way we construct the Bloom filters search indices for different documents. The first option is to use different hash functions for the the different BF, we call this instantiation SADSm. The second possibility, which we call SADS, is to use the same hash functions across the Bloom filter of multiple documents. These two instantiations provide different privacy guarantees as we discuss in Section 8.5 and they also allow for different types of optimizations for the implementation and hence the efficiency guarantees as we show in Section 8.6.

8.4 Document Retrieval

There exist many systems for searching databases to privately identify items of interest. An extension of obvious use is a system to then retrieve those items privately. One way to do this is with private information retrieval techniques, however these are very expensive, and can be even more expensive when fetching large numbers of records, or records of individually great size. We present a system that is much more efficient, at the cost of requiring a trusted third party, and can be modularly implemented to extend any private search system that returns handles representing matches.

Systems both with and without document retrieval have practical use. For example, a user may simply wish to establish that a server does have documents of interest to him, or may wish to determine how many are of interest, or learn about certain qualities concerning the data held there (subject to the search permissions granted by the server). Furthermore, even in systems that include document retrieval, separating this functionality from query

Preprocessing.

- For each of its documents S generates a set of hash functions that he uses the compute a Bloom filter from the PH-DSAEP+ encryptions of the entries (words) in the document.
- S sends the resulting Bloom filters to IS.

Key Generation. To authorize C for search

- S, QR and C run $\mathsf{Gen_{query}}$ to obtain keys (sk_C, $qrk_{\{C,S\}}, rk_S$) for query submission, where rk_S is the key S used in the previous step.
- IS, C and QR run $\mathsf{Gen_{result}}$ to get ($sk_{IS}, qrk_{\{IS,C\}}, rk_C$) that will be used later for the result return.

Query Submission. To submit an encrypted query for an entry (word) W:

- C computes $c_1 = \mathsf{Enc_{query}(sk_C, W)}$ and sends it to QR.
- QR computes $\mathsf{Trans_{query}(c_1, C, st'_i)} = \mathsf{(IS, st'_{i+1})}$ and sends c_1 to IS.

Search.

- IS obtains the Bloom filter search indices from $\mathsf{Dec - R_{query}(c_1)}$ that extracts the BF indices for the query for each
- IS does a BF search across the database Bloom filters. Let R be the set of matching documents.

Query Return

- IS encrypts and sends $c_2 = \mathsf{Enc_{result}(sk_{IS}, R)}$ to QR.
- QR transforms $\mathsf{Trans_{result}(c_2, IS, st''_i)} = \mathsf{(C, st''_{i+1})}$ and sends $\bar{c} = \mathsf{Enc - QR_{result}(c_2, rqk_{\{IS,C\}}, st''_i)}$ to C.
- C decrypts $\mathsf{Dec - R_{result}(rk_C, \bar{c})}$ to obtain the result R.

Figure 8.4: Secure Anonymous Database Search Scheme

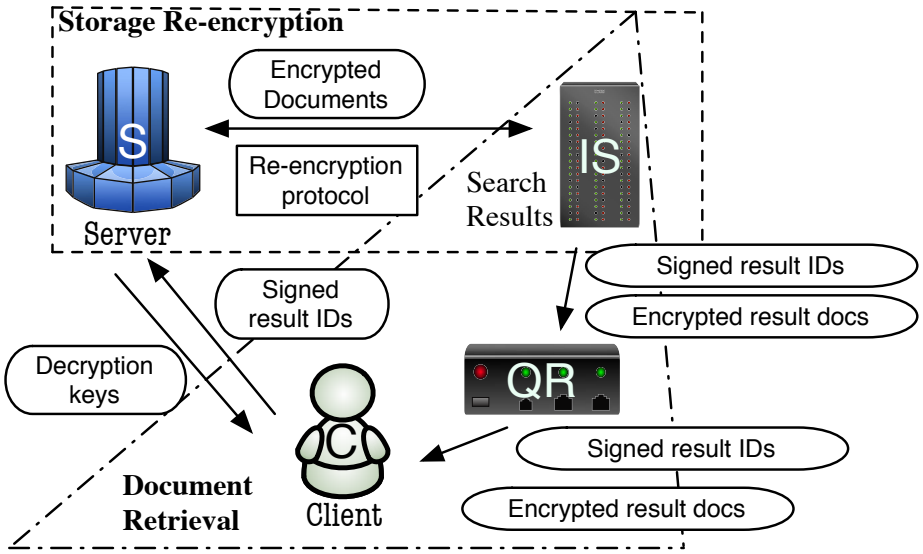


Figure 8.5: SADS with Document Retrieval.

is worthwhile. For example, the server may be running a paid service, and allow the user to operate in an initial stage wherein he determines what he wants, and a bargaining stage wherein they negotiate pricing, before purchasing the actual content.

Document retrieval poses its own challenge, especially when the data is not owned by the party retrieving it. In this scenario, returning additional data is a privacy leak for the data owner; at the same time, revealing the matching documents to the owner is a privacy leak for the retriever. Thus, the strongest security we would want to aim for would require us to touch the contents of the entire database [Chor *et al.*, 1997]. This is a prohibitively expensive cost for applications that aim to work in "real time" over a large data set. One way to avoid this cost is to relax our security definition and allow leakage of the matching documents ids. In the case of data outsourcing, this amount of privacy leakage easily suffices, since the untrusted server just searches for and returns the encrypted files that he stores to the owner who has the corresponding decryption keys [Boneh *et al.*, 2004; Curtmola *et al.*, 2006; cheng Chang and Mitzenmacher, 2005]. This approach, however, is not applicable to the case of data sharing, where leaking the matching documents to the owner reveals more than the result pattern: he also knows the content of the documents, from which he can infer information about the query.

This problem is similar to that addressed by private information retrieval protocols (PIR) [Chor *et al.*, 1998; Gertner *et al.*, 2000; Olumofin and Goldberg, 2011], wherein a server holds a set of items from which a user wishes to retrieve one without revealing which item he is requesting. It differs slightly in that we wish to retrieve multiple items (corresponding to the search results). It also differs in that we require that the selected set be certified and that the user does not learn content of documents outside of it. There are PIR schemes that address this [Gertner *et al.*, 2000], but at additional cost. Thus, our problem could be addressed by simply running an appropriate PIR scheme once for each document result. However, PIR is already quite expensive for a single document, and running them multiply would only aggravate this.

We address this by constructing a document retrieval scheme that can be used on top of any other scheme that returns document IDs. Our scheme maintains efficiency by introducing an intermediary party who stores the encrypted files of the database and provides the matching ones to the querying party. This party is given limited trust to perform the search, but he should not be able to decrypt the stored files. In this case we need to provide the querier with the decryption keys for the result documents; these are known to the data owner, who must be able to provide the correct keys obliviously without learning the search results. In Figure 16 we present a protocol that realizes the document retrieval functionality between a data owner (S) and a client (C) with the help of an intermediary party (P). For the purposed of this protocol we assume that there is a search functionality $EncSearch$ that returns the IDs of the documents matching a query from the client. For a query Q we denote $EncSearch(Q)$ the returned set of document IDs. The database of the server that is used for the protocol consists of documents $D_1, \ldots, D_n$. Our protocol also uses 1-out-of-n oblivious transfer (OT) functionality that allows two parties, one of which has input an array and the other has input an index in the array, to execute a protocol such that the latter party learns the array element at the position of his index and the former learns nothing. There are many existing instantiations of OT protocols, we use the protocol of [Gentry and Ramzan, 2005], which allows best efficiency. The last tool for our constructions is an encryption scheme with the group property from Definition 34.

Intuitively, the security of this protocol is based on the secrecy of the permutation π,

Storage Reencryption (***preprocessing*** *phase*)

Inputs:
S : $D_1, \ldots, D_n$, keys $k_1, \ldots, k_n$ and $k'_1, \ldots, k'_n$;
P : permutation π of length n ;
S, P : $(\overline{GEN}, \overline{ENC}, \overline{DEC})$ satisfying Definition 34
Outputs:
$S : \perp$; P : $\overline{ENC}_{k'_{\pi(i)}}(D_i)$ for $1 \leq i \leq n$
Protocol:

1. S sends to P $c_i = \overline{ENC}_{k_i}(D_i)$ for $1 \leq i \leq n$.
2. For each $1 \leq i \leq n$ S and P execute 1-out-of-n OT protocol that allows P to obtain $k''_i = k_i^{-1} \cdot k'_{\pi(i)}$.
3. For each $1 \leq i \leq n$ P computes $\overline{ENC}_{k''_i}(c_i) = \overline{ENC}_{k_i^{-1} \cdot k'_{\pi(i)}}(\overline{ENC}_{k_i}(D_i)) = \overline{ENC}_{k'_{\pi(i)}}(D_i)$.

Document Retrieval

Inputs:
S : keys $k'_1, \ldots k'_n$;
P : permutation π of len n, $\overline{ENC}_{k'_{\pi(i)}}(D_i)$, $1 \leq i \leq n$;
C : query Q;
S, P, C : search scheme *EncSearch* that returns IDs of matched documents to P, C.
Outputs:
S : cardinality of the output set $EncSearch(Q)$;
P : IDs of docs matching query Q from *EncSearch*;
C : the content of the docs matching Q from *EncSearch*.
Protocol:

1. S, P, C run *EncSearch* for query Q. Let $i_1, \ldots, i_L$ be the IDs of the matching documents.
2. P sends $Sign(\pi(i_1), \ldots, \pi(i_L))$ to C together with the encrypted documents $\overline{ENC}_{k'_{\pi(i_1)}}(D_{i_1}), \ldots, \overline{ENC}_{k'_{\pi(i_L)}}(D_{i_L})$.
3. C sends $Sign(\pi(i_1), \ldots, \pi(i_L))$ to S.
4. S verifies $Sign(\pi(i_1), \ldots, \pi(i_L))$ and returns $k'_{\pi(i_1)}, \ldots, k'_{\pi(i_L)}$.
5. C decrypts $\overline{ENC}_{k'_{\pi(i_1)}}(D_{i_1}), \ldots, \overline{ENC}_{k'_{\pi(i_L)}}(D_{i_L})$ to obtain the result documents.

Figure 8.6: Protocol for Document Retrieval

known only to P. Because it is not known to S, S cannot correlate the keys k'_{π_i} that are requested by C with the original indices of the matching documents. He learns only the

search pattern of the querying party. We can take two approaches to mitigate this leakage. The querying party may aggregate requests for decryption keys to the server for the search results of several queries. Another solution is to extend the scheme to include additional keys pertaining to no real documents, which P can add to the sets of requested keys so that S cannot tell how many of the keys he returns correspond to query results. Step 2 of the re-encryption can be implemented using protocols for oblivious transfer [Naor and Pinkas, 2001; Aiello *et al.*, 2001; Crescenzo *et al.*, 2000].

Lemma 16. *The re-encryption protocol in secure in the semi-honest model.*

The lemma follows from the properties of the oblivious transfer protocol.

8.5 Security Proof of the Protocol

In this section we provide proofs for the security properties of our system

8.5.1 Security Against Adversarial Client

The two instantiations SADSm and SADS provide two different security notions against a semi-honest client. The reason for this is the fact that when we use the same hash functions across different Bloom filters the false positive rates for those are no longer independent. In the case of SADSm we do not have this. Since in both cases the client learns some false positives along with the real results, we model this leakage formally through the definition of the ideal functionality for the proof, which given a false positive rate δ will return the correct search results and it will also return any other record with probability δ. A given protocol is defined to be (t, ϵ)-secure with false positive (FP) rate δ if there is a simulator in the ideal world with FP rate δ, such that no adversary running in time t can distinguish the real view from the simulated view, except with probability ϵ.

Theorem 24. *For every polynomial t there exists a negligible ϵ, such that for every δ: our SADSm scheme instantiated with BF parameter δ, provides (t, ϵ) security with false-positive rate δ against a semi-honest client. (the ideal world trusted party provides each non-matching record with independent probability δ).*

The proof follows immediately since the only thing the simulator needs to do is submit the clients queries to the trusted party and return to the client the output he gets from the TP. In the case of a malicious client we need to enable to simulator to extract the client's key in the key generation protocol so that he can decrypt the client's queries. The key generation protocol that we provide in Appendix F.1 has this property.

In the case of SADS the records returned as false positives should have the same set of indices corresponding to the query set to one in their Bloom filters. Thus documents that are similar and have greater overlap in their BFs are more likely to be returned together as false positives. This is additional leakage to the client.

We point out that when using SADS (with a single hash function), although the false positive rates across Bloom filters are not independent, the Server (who creates the BF) can calculate the exact false positives for any search word, to make sure they are acceptable. The advantage of this approach is that we can apply our slicing optimization from [Raykova *et al.*, 2009], which facilitates parallel search across multiple Bloom filters and improves the efficiency of the search protocol.

8.5.2 Security Against Adversarial Server

In our security definition we will use the equivalence relation $(q_1, \ldots, q_\ell) \equiv_D (q'_1, \ldots, q'_\ell)$ defined as follows:

Definition 35. *For any two tuples of queries,we define the equivalence relation* $(q_1^0, \ldots, q_\ell^0) \equiv_D (q_1^1, \ldots, q_\ell^1)$ *to hold if and only if* $|\cup_i q_i^0(D)| = |\cup_i q_i^1(D)|$ *where* $q(D)$ *denotes the results returned for a query* q *on database* D*.*

Our basic search part (handle retrieval) trivially does not reveal any information whatsoever to the server, as the server is not involved (thus, for this part we may take any two sequences to be equivalent). Taking our full scheme (together with the document retrieval part), our scheme hides everything but total number of matches for all queries. This is formalized as follows.

Definition 36. *Let* $\mathcal{A}$ *be a distinguisher algorithm that runs the following experiment:*

1. *$\mathcal{A}$ outputs a database D and two tuples of queries $q_1^0, \ldots, q_\ell^0$ and $q_1^1, \ldots, q_\ell^1$ such that $(q_1^0, \ldots, q_\ell^0) \equiv_D (q_1^1, \ldots, q_\ell^1)$.*

2. *A random bit b is chosen and $\mathcal{A}$ is given the $view_S(q_1^b, \ldots, q_\ell^b; D)$.*

3. *$\mathcal{A}$ outputs bit b'.*

The distinguishing advantage $Adv_{\mathcal{A}} = |Pr[b' = b] - \frac{1}{2}|$ for a polynomial time adversary $\mathcal{A}$ is negligible.

Theorem 25. *In the SADSm scheme the index server and the query router have negligible distinguishing advantage according to Definition 36.*

The proof follows from the fact that the server sees only the requests for decryption keys and by Lemma 16 he cannot link those to the real documents that were retrieved.

The above proof holds for any value of l as long as the document retrieval for all queries happens at the same time. If we want to allow that the results for each query are retrieved before the next query is issued we need to define the equivalence relation $\equiv_D$ to hold for tuples of queries that have the same search pattern, namely $|q_i^0(D) \bigcap q_j^0(D)| = |q_i^1(D) \bigcap q_j^1(D)|$ for all $1 \leq i, j \leq \ell$.

8.5.3 Security Against Honest-but-Curious Index Server and Query Router

8.5.3.1 Privacy of the Database

The SADSm instantiation of our scheme reveals nothing about the documents to either IS or QR. The SADS instantiation reveals some structural properties of the database to IB, namely document similarity. This follows from Proposition 1 for the SADSm implementation (as each record has the same number of attributes), and from Proposition 2 for SADS.

8.5.3.2 Privacy of the Client's Queries from QR and IS

The privacy definition here is again defined through an equivalence relation $\equiv_D$ on sequences of queries. It states that the scheme hides everything except "equality patterns" among matches to different queries (e.g., whether one record returned in response to a query is the

same or different from a record returned in response to another query). The content of the records should remain protected.

Definition 37. *Fix a database D. Then $(q_1^0, \ldots, q_\ell^0) \equiv_D (q_1^1, \ldots, q_\ell^1)$ if and only if there exists a permutation π on the records in D such that for all $1 \leq i \leq l$ it holds that $\pi(q_i(D)) = q_i'(D)$.*

We define the security for the search queries with respect to the index server and the query router as follows:

Definition 38. *Let $\mathcal{A}$ be a distinguisher algorithm that runs the following experiment:*

1. *$\mathcal{A}$ outputs a database D and two query sequences $q_1^0, \ldots, q_\ell^0$ and $q_1^1, \ldots, q_\ell^1$ such that $(q_1^0, \ldots, q_\ell^0) \equiv_D (q_1^1, \ldots, q_\ell^1)$ (for the minimal equivalence relation $\equiv_D$).*
2. *A random bit b is chosen and $\mathcal{A}$ is given the $view_{IB}(q_1^b, \ldots, q_\ell^b; D)$.*
3. *$\mathcal{A}$ outputs bit b'.*

The distinguishing advantage $Adv_{\mathcal{A}} = |Pr[b' = b] - \frac{1}{2}|$ for a polynomial time adversary $\mathcal{A}$ is negligible.

Theorem 26. *In the SASDm scheme both the index server and the query router have negligible distinguishing advantage according to Definition 38.*

Proof. The index server learns the matching records for each submitted query. The definition of $\equiv_D$ provides that the record identifiers for the matching results for $q_1^0, \ldots, q_\ell^0$ and $q_1^1, \ldots, q_\ell^1$ are indistinguishable for the IS since they have the same search pattern.

Theorem 20 guarantees that the QR can learn only the equality pattern of the sequence of submitted queries and he cannot distinguish PH-DSAEP+ encryptions of the two equivalent sequences of queries. This holds in the random oracle model, and assuming either DDH or PH assumption. □

8.6 Performance Evaluation

To evaluate the practicality of our proposed system we implemented it (roughly 4 Klocs of C++ code in total) and we performed a number of measurements using realistic datasets:

(i) the email dataset that was made public after the Enron scandal [Shetty and Adibi, 2004] and (ii) a synthetic dataset with personal information for 100K persons. The Enron dataset consists of about half a million emails with an average size of 900 bytes after stemming . During the preprocessing phase of SADS, a distinct Bloom filter for each email was created. Then, each of the email files was tokenized and the tokens where stored in the corresponding Bloom filter, after they were properly encrypted. The format of the second dataset is more close to a database than a collection of documents. Its schema consists of a single table with 51 attributes of three types: strings (first name, last name, etc.), numbers (height, SSN, etc.) and file links (fingerprint, private key, security image, etc.) and it is stored in a flat CSV (Comma Separated Value) file. The total size of that dataset, along with the files pointed in the records, is 51GB and the average size for a record is 512KB. During the preprocessing phase we created a distinct Bloom filter for each record and each of the attribute values was inserted after it was prefixed with the attribute name ("name_value") and properly encrypted. In both cases, we configured the BF parameters so as the false positive rate would be less than 10^{-6}.

The experimental evaluation setup was comprised by two servers and a client laptop. The servers had two four-quad Intel Xeon 2.5GHz CPUs, 16 GB of RAM, two 500 GB hard disk drives, and a 1 Gbit ethernet interface. The laptop was equipped with an Intel Core2 Duo 2.20GHz CPU, 4 GB of RAM, a 220 GB hard disk drive, and a 100 Mbit ethernet interface. All of them were connected through a Gigabit switch; they all ran a 64-bit flavor of the Ubuntu operating system. QR and IS were running on each of the servers, the queries were performed from the laptop. When Document Retrieval was enabled, the File Server was running on the same host with the IS.

8.6.1 Memory Consumption

Along with the timing measurements, we also monitored the memory consumption of the extended SADS system to determine scaling limits. We found out that the only significant factor was the type of Bloom filter storage. Bloom filters are stored either sequentially in a flat file or transposed using the slicing optimization. In the sequential storage case memory usage was constant; it grew consistently with the dataset size in the slicing case, because

the structures are kept in memory and written to files at the end. During the search phase, both the client and the QR used a small, constant amount of memory (∼2MB). On the other hand, the IS's memory usage grew with the dataset size. In the sequential storage case, the file was `mmap`'ed; the amount of memory used was the Bloom filter size in bytes times the number of BFs (e.g. 1KB * 50K = 50MB). When the slicing optimization was enabled, we saw higher memory usage, ∼109MB for the same dataset. That was most likely due to the extensive use of C++ vectors, which we can further optimize in the case of much larger databases where the available RAM may become an issue.

8.6.2 Implementation Optimizations

We performed experiments using variable-sized subsets of both datasets while changing the size of the cache. As for the Enron dataset, we show that a good cache size is 5K keywords. This gives us a ∼90% hit ratio, while reducing the preprocessing time for 50K emails from 2h to 10m. Performing the same experiments for the synthetic dataset yielded slightly worse results, as some attribute values are unique. However, using a 10K keywords cache the hit ratio was 50% on the full dataset, which still is a significant gain.

We measured the speedup of the preprocessing phase on the full datasets, while increasing the number of threads. As we expected, the speedup grew linearly until the number of threads reached the number of cores in our servers – that is eight. When the number of threads was more than the CPU cores, the speedup slightly declined, most probably due to thread scheduling overhead. Performance results for the parallelized search phase are presented in the next section.

8.6.2.1 Slicing optimizations

In the SADS instantiation of the scheme where we use the same hash functions are used across the Bloom filters for different documents we can apply a different type of optimization, which we call *bit slicing*.

To minimize the number of bits that need to be read to satisfy queries across a large number of Bloom filters, we store them in transposed order. First, they are divided into blocks of filters; within each block, all bits from a single index across the filters are stored

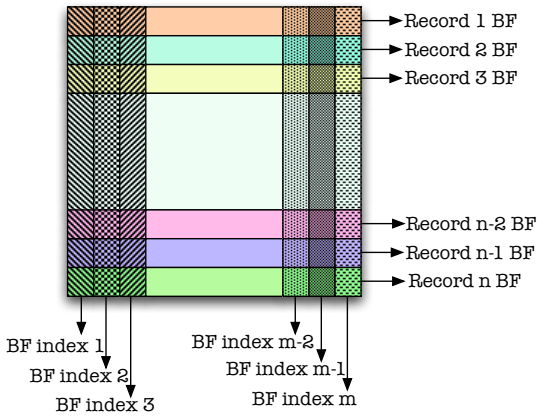


Figure 8.7: Multiple Bloom Filters Memory Storage

sequentially. Thus, each document is represented by a bit within multiple slices, one for each index of its Bloom filter representation (Fig. 8.7). To run a query, we need only fetch those slices which correspond to the indices of the query term, which is a large savings since normally we would have to read the full contents of every Bloom filter for every document for any query. This technique is referred to as bitslicing and has been studied as a method for storing signature files in database indexes [Zobel and Moffat, 1998].

By storing the Bloom filters in blocked slices, we gain the ability to avoid reading a large portion of the bits in the Bloom filter set when we run queries. We need only check those slices which correspond to an index which is present in the query term(s). Since this is very sparse, this is a large improvement over non-transposed storage; it would require us to read the entirety of every Bloom filter in order to run a query.

To run a query, we construct a result vector, which is a bit vector equal in size to the number of Bloom filters in the set. This is then "and"ed to each slice corresponding to a query index. Over time, several block-sized portions of the result vector will become zeroed out. Once this happens, as a further optimization we cease to read those portions of later indices. Our block size is chosen as the disk page size, and our end goal is thus to read the minimum number of pages necessary to answer a query. If multiple queries are being run, we keep a cache of recently viewed bitslices with a LRU replacement policy.

Because we are storing the Bloom filters in transposed order, and each filter is repre-

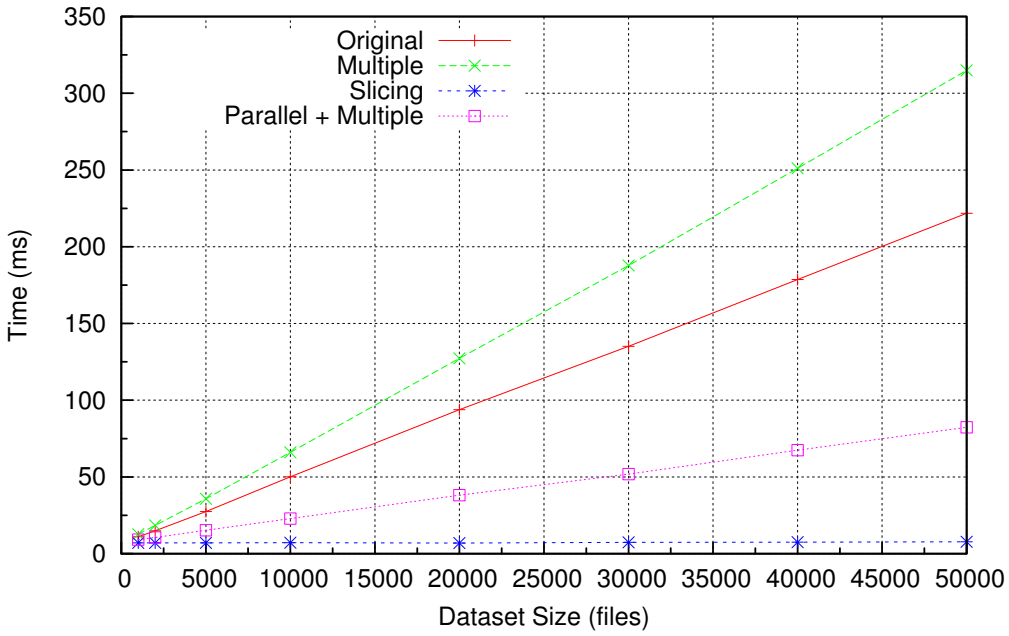


Figure 8.8: Average query time under different SADS configurations using the Enron dataset.

sented by a single bit across various slices, deletion of filters would be expensive. Thus, we implement this simply by zeroing out the indices of a filter so that it will not match future queries. As a future addition, we may support a system of periodically cleaning the slicebase by identifying "deleted" filters and compacting the remaining ones.

8.6.3 Search Performance

In this section we explore in detail the effects the performance of the two instantiations of the scheme SADS and SADSm and also how parallel search could help amortize some of the performance penalty.

Figure 8.8 shows the comparison for four different configurations of SADS: (i) same hash functions across BFs, (ii) same hash functions across BFs with the slicing optimization enabled, (iii) using multiple hash functions and (iv) using multiple hash functions and parallel searching together. The search time reported in this figure is the total time elapsed from the point when the client issues the query to the QR until it receives the set of matching document IDs if any — no document retrieval. As expected, the average query time grows

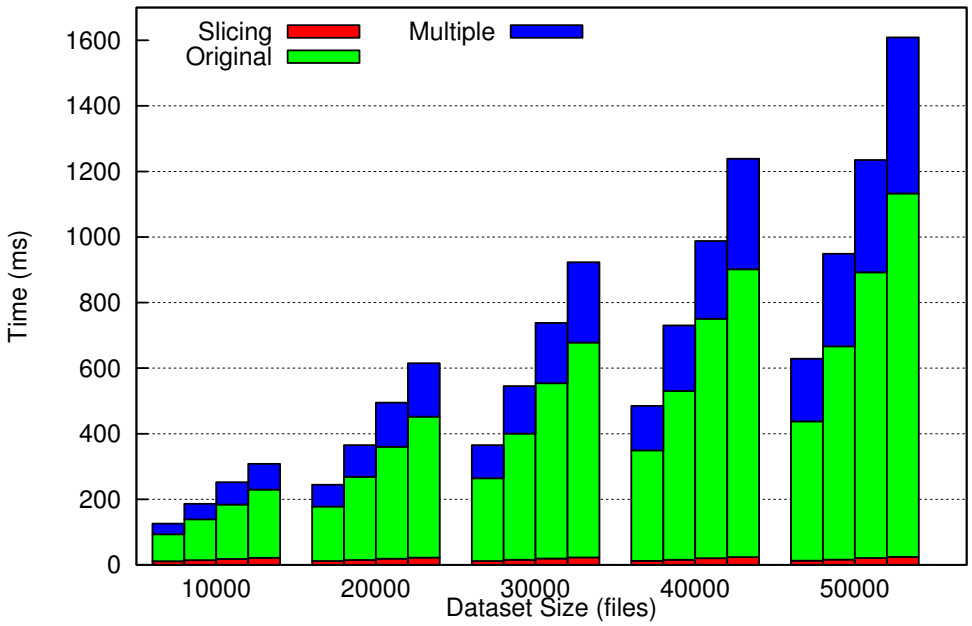


Figure 8.9: Average OR-query time under different SADS configurations using the Enron dataset. Each cluster is for a different dataset size and each bar is for a different term count (from 2 to 5).

linearly using the original SADS configuration, as the actual search is done linearly over all the Bloom filters. Next, we can see that the slicing optimization greatly reduces search time to a point that it seems almost constant across different dataset sizes. Using the multiple hash functions feature we do get better privacy guarantees, but at the cost of increased search time by another factor that is proportional to the dataset size. That is because for each document we have to recalculate the hash functions and recompute the Bloom filter indices. Finally, we see that taking advantage of the commonly used multicore architectures does increase the performance of the search in the multiple hashing scheme. More precisely, the speedup when we used 8 threads on our 8-core servers was from 1.3 to almost 4 for the dataset sizes shown in the Figure 8.8. Thus, although the multiple hash functions feature increases the computation factor, we can amortize a great part of it by executing it in parallel. It is also worth noting that the multiple hash functions plus parallel searching configuration provides better performance that the configuration with the same hash functions, while on the same time it improves the privacy guarantees.

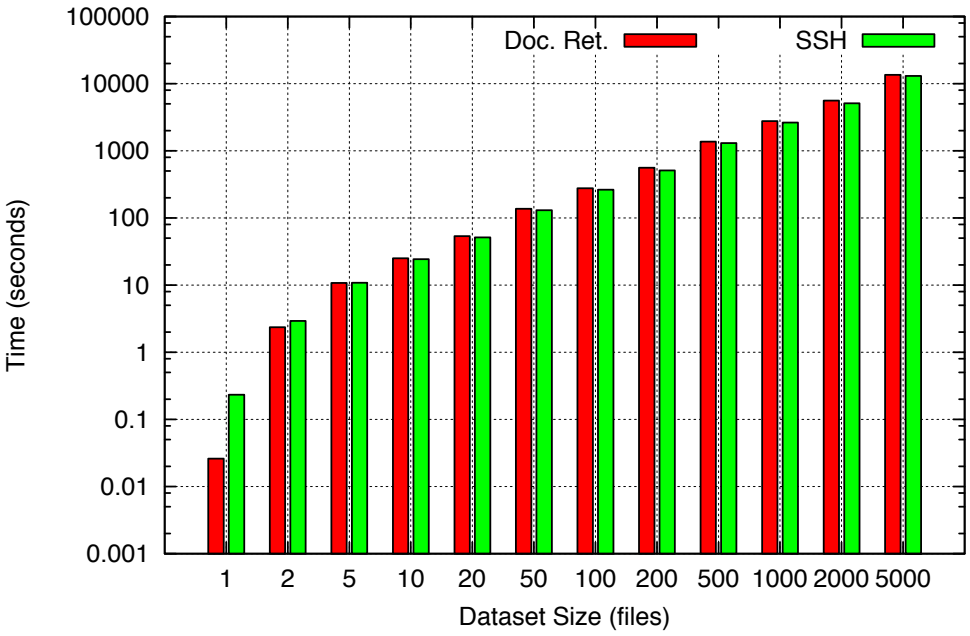


Figure 8.10: Average time for retrieving documents anonymously, compared to retrieving them non-anonymously using ssh file transfer. Average size of files being transferred was 27.8 Mb

Next, we evaluate the performance overhead of the multiple hash functions in boolean queries, and more precisely OR queries. To optimize the normal case – i.e., when the slicing optimization is not enabled – we skip BFs that already contain one of the search terms. That way we avoid searching over and over on Bloom Filters that already match the OR query thus reducing the overall searching time, especially when the search terms are frequent. Figure 8.9 shows the search time for OR queries under different SADS configurations. Each cluster of bars is for a different dataset size; each bar is for a different term count in the boolean OR query. The first bar is for two terms, the second for three, and the last two for four and five, respectively. The fact that the search time in each cluster grows sub-linearly to the number of terms clearly shows the performance gain.

8.6.4 Document Retrieval

We implemented document retrieval using PH-SAEP and standard RSA signatures to sign query results. Using PH-SAEP puts a (likely over-restrictive) limit on the length of plaintext

values. To handle this, we encrypt larger files using AES private key encryption, and store the key encrypted with PH-SAEP as a header in the encrypted file. The files can thus be read by decrypting the header with the appropriate PH-SAEP key and using the result to decrypt the content of the file. We preprocess the files in a way that provides an intermediate party with AES encrypted files under different AES keys and encryptions of these AES keys under some permutation of the keys k_1", ... k_n". The client will receive as results from the intermediary party the encrypted files, the encrypted AES keys, and the indices of the keys k" used for their encryptions. When he receives the decryption keys k" from the server, the client will first decrypt the AES keys and then use them to decrypt the remainder of the files.

Figure 8.10 shows the average time to retrieve documents using our scheme versus the number of documents being retrieved. This is shown in comparison to a non-privacy-preserving SSH-based file transfer. As we can see, our scheme adds very little overhead compared to the minimum baseline for encrypted transfer. The time also shows linear growth, suggesting that it is dominated by file encryption and transfer, rather than for the encryption and verification of the results vector itself.

As a point of comparison, Olumofin and Goldberg [Olumofin and Goldberg, 2011] present some of the best implementation performance results currently published for multi-selection single-server PIR. In their performance results, we see response times per retrieval ranging from 100 to 1000 seconds for retrievals of 5-10 documents on database sizes ranging from 1 to 28 GB. Our scheme scales strictly with number and size of documents retrieved, and not with the total database size. They do not state the sizes of the blocks retrieved in their scenario, but if we were to give a very high estimate of 1 MB per block, and assume they fetched 10 blocks every time, one could expect in our system that each query would take .7 seconds, still orders of magnitude short of the 100s fastest time they report for a 1GB database, and it would not scale up with increasing database size as theirs does, thus significantly beating the 1000s time they report for a 28GB database. Note that their system is not designed to protect privacy of the database, only of the request. The work of [De Cristofaro *et al.*, 2009] presents a protocol for privacy-preserving policy-based information transfer, which achieves privacy guarantees weaker than SPIR and similar to

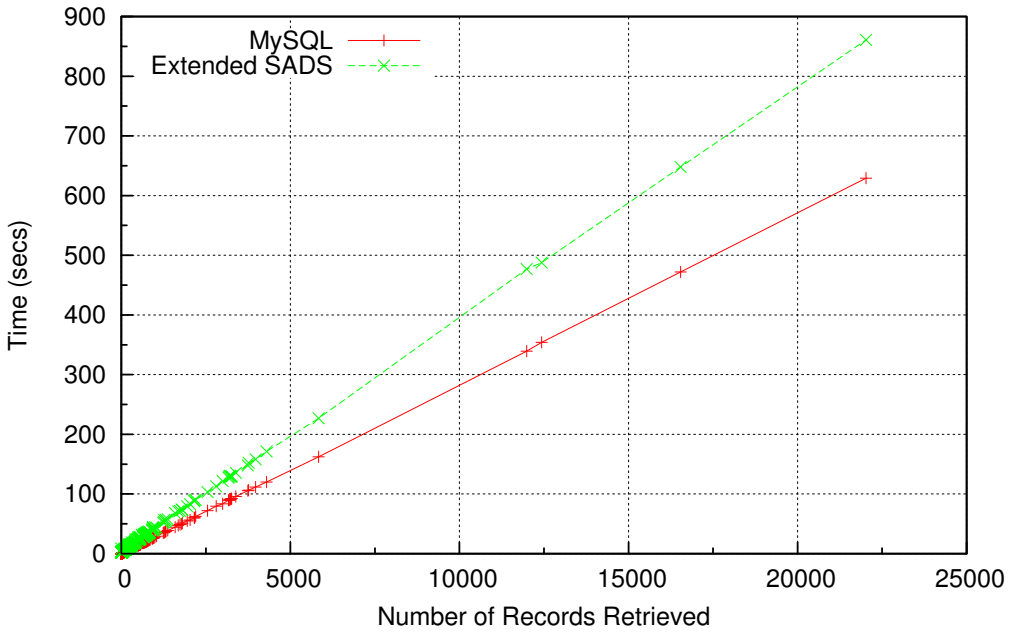


Figure 8.11: Comparison between the extended SADS and MySQL.

ours. Direct comparison between our and their performance results is hard — they present timings only for the computation time without communication, which grows linearly with the size of their database. The maximum size of their database is 900 records with 2000 ms computation per record retrieval, while for our scheme the entire record retrieval time (computation plus communication) for a database with 25000 records is about 40ms (Figure 8.11, described in the next section).

8.6.5 Overall Performance

Finally, we compare the performance of the extended SADS system with a real world DBMS, MySQL. In order to do that, we implemented a SQL front-end for SADS that could parse simple conjunctive and disjunctive queries. Then, we loaded the synthetic dataset to both systems and we executed a number of queries of variable result set size. SADS was configured to use multiple hash functions and document retrieval was enabled. Parallel searching was also disabled, which means that we compared using the less efficient version of the extended SADS. Figure 8.11 shows the total duration of both the query and the retrieval of the data

for our system and MySQL. Our scheme performs just 30% slower on average than MySQL, which is the price for the privacy guarantees it provides.

8.6.6 Case Study: Sharing of Health Records

We next examine, from a very high level, the suitability of our scheme for a hospital's admissions records database. (A database for full medical record storage is vastly more complex and is not addressed here.) A patient's health record is very sensitive information, usually kept by the patient's medical institution. There are cases, though, where such information needs to be shared among different institutions. For example, a patient may need to visit a different institution due to an emergency, or to get a second opinion. This sharing may need to be done privately for several reasons. In an emergency, a doctor may need to query all the institutions that share records with his without revealing the patient's identity, especially to the institutes that do not have information about him. If the querying is not private in that case, some institutions would learn information about a patient that has never visited them. Or, a patient may not want his institution to know which institution he visits for specialized needs, such as drug rehabilitation, so again the query for his record has to be performed privately.

A database of health records is similar to the synthetic dataset we used in our evaluation. It contains some searchable fields like name, date of birth, height, etc.; each record may be linked with several medical exam results like x-rays, electrocardiographs, magnetic tomographies, etc. In 1988, there were about ten routine tests during the hospital's admission process alone [Hubbell *et al.*, 1988]; today, about thirty individual tests are done.[1] Taking into account that some of the results can be a few tens of Mbs — for example, a head CAT scan is about 32 MB — each health record could be a couple of hundred megabytes. One major hospital complex admits about 117K inpatients per year[2]; to a first approximation, their database would thus have several hundred thousands rows and 30–40 columns.

We have already seen, though, that the extended SADS scheme we propose can successfully handle a database of this size. Our evaluation demonstrated that document retrieval

[1] Private communication with a physician.

[2] `http://nyp.org/about/facts-statistics.html`

adds only a small overhead compared to simple transfer, thus easily scaling with the size of the document retrieved. Also, searching over 100K records with 51 searchable attributes each takes less than half a second, thus meeting real-world requirements. Finally, the support for updates in health records is a requirement covered by our extended SADS scheme. We conclude that our scheme is able to handle the requirements of this hospital, while preserving patient privacy.

Chapter 9

Conclusions

Every day people use various services that take some of their private data as input. The computation underlying such services most often depends on the private data of multiple parties. Although the results of such computation do reveal some information about the private inputs of the parties, the benefits of such services outweigh the potential risk of the privacy leakage from the result. While revealing the output may be acceptable, the actual inputs needed for the computation are much more sensitive and need to be protected. In such case the desirable privacy guarantees can be formulated as follows: the computation should not reveal anything more about the inputs than what is inherently leaked by the final result.

Constructing protocols with the above properties has been the subject of secure multi-party computation. While there have been many results in the area showing how to construct secure computation protocols for any functionality, these constructions come with efficiency overhead that is prohibitive for most practical applications. In this thesis we claim that one of the reasons for the big gap between the efficiency required for practical purposes and the one provided by existing protocols is the fact that the underlying assumptions of such protocols often do not reflect closely the actual requirements of real scenarios. Existing MPC techniques assume homogeneous environments where all participants have the same computation and communication resources available. Further, the adversarial models that they use assume that all corrupted parties will exhibit the same type of misbehavior and would collude and share their private information. On the other hand, practical

setups that could benefit from MPC solution techniques often present quite heterogeneous systems, where parties have different resources and different incentives to misbehave in the execution. And if we are aiming for efficient solutions, we need to take these facts into consideration.

We explored several avenues that allow us to develop secure computation techniques more closely tailored to the requirements of different heterogeneous systems. These approaches achieve improved efficiency for the participants in a way that enables secure computation in scenarios where existing protocols would not be usable. In particular our contributions are in three main directions: we consider new computational models as representation of the evaluated function in a secure computation, which allow us to overcome inherent inefficiencies in MPC approaches relying on Boolean and arithmetic circuits. Further, we focus on the setting of outsourcing where we have one powerful party with large computational and storage resources, which it provides as a service to computationally weak and memory bounded clients. In this setting we propose solutions for verifiable delegation, server-aided computation with non-colluding adversaries and privacy enhanced sharing of outsourced data. The third perspective that we adopt has a different flavor. We start with particular efficiency requirements that we would like to achieve for a protocol for encrypted search and data sharing, and explore how strong security guarantees we can achieve while meeting these efficiency constraints.

Contributions. Computation involving large databases often needs to access only a small part of the data which is stored and the only acceptable solutions from efficiency point of view are those that use algorithms with sublinear complexity. Using generic solutions that represent the evaluated functionality as a circuit inherently incurs computational overhead linear in the size of the database. We construct a two party computation protocol that uses RAM representation for the evaluated function and achieves only polylogarithmic amortized overhead to the running time of the insecure version of the computation. We provide both a generic construction that relies on any oblivious RAM scheme and any two party computation protocol as well as an optimized construction that uses specific instantiations of both schemes.

Multivariate polynomials can be used to express the functionality of a large number of problems from linear algebra, statistics, logic and set operations. At the same time they provide a more succinct representation for these functionalities than circuits. We use this representation to construct an MPC protocol for evaluation of multivariate polynomials that improves the communication complexity of existing solutions and requires only a single round interaction among all the parties. An instantiation of our general protocol provides currently the most efficient solution for the multiparty set intersection problem in the fully malicious case.

In the setting of outsourcing we have a powerful party that provides services to weak clients. Two main types of services that can be provided are data storage and computation. When we consider computation outsourcing there are two properties that we may want to provide: privacy and verifiability. We address these questions in a construction for server-aided computation that allows offloading the works of most of the participants in an MPC protocol to a computationally powerful party (server), which does not have inputs for the protocol and just assists the execution of the computation. We also introduce a new adversarial model where parties might be misbehaving but not necessarily colluding and sharing their private information. This adversarial setting, though weaker than the case of a monolithic malicious adversary, suffices to model accurately the actual incentives for deviating behavior of the participants in many practical scenarios. At the same it allows to construct quite efficient protocols that use entirely symmetric key primitive with the only exception of a few public key operations for coin tossing.

If we restrict our requirements for delegated computation only to the verifiability property, this provides room for further efficiency improvements. At the same time such protocols are still useful for settings where the service provider is trusted to store the data in plaintext form (e.g., it is bound by legal contract not to reveal the information), but is not trusted to do honestly all the necessary work for the computation. Such techniques will be also relevant in the case where we want to be able to detect execution errors of the computation. Verifiable computation schemes aim to provide an efficient way to check the correctness of the result of a large computation. We introduce a new paradigm for constructing verifiable computation from attribute-based encryption. This construction avoids the use of expensive

primitives such as FHE and PCPs underlying other VC solutions and additionally enjoys two new useful properties: public delegation and public verification.

In the case of data outsourcing users store and access their data at a service provider. In addition to that they may wish to share data among themselves. While one way to do this is to first retrieve their own data from the provider and then send it to each other, such an approach incurs a substantial communication overhead. A much more efficient solution would allow each user to retrieve directly from the server all data (both his own as well as others'), which he is authorized to access. However, in this case access control rules that determine what data is shared among users' become private information for the users that they may not want to reveal to the provider, who at the same time needs to serve as a point of access enforcement. We propose a two level-access control solution for both reads and writes that offers tunable trade-offs between efficiency overhead and hiding properties for the access control rules and data access patterns from the storage provider.

Often when we implement a real system we are facing efficiency requirements for practical uses and the goal is to maximize the security properties of the solution while meeting these requirements. We explore such a setting in the case of encrypted search for secure data sharing. We construct a protocol that allows one party to search the database of another and retrieve matching records while providing privacy guarantees for the query and the non-matching content of the database. We design and implement a system that handles query searches on databases of size 50 GB achieving only 30% overhead compared to insecure search provided by mySQL. We achieve this efficiency performance under a relaxed security model taking advantage of two intermediary parties, which act as semi-honest intermediaries for the protocol but do not learn any of the private information of the participants (e.g., database and queries). The only allowable leakage beyond what is inherent to the results is the query pattern of the client, which is revealed to the intermediaries and the data owner.

Future Work. In this thesis we have presented several avenues to approach the problem of finding a meeting point between techniques for multiparty computation, which provide strong security guarantees, and efficiency requirements for practical application. Either of

these directions brings potential for further research. With the advent of cloud computing the question of outsourcing of computation becomes of bigger and bigger importance. We proposed a solution that improves the efficiency for most of the participants in an MPC protocol using a server-aided model of computation. Solutions based on fully homomorphic encryption improve the efficiency for all participants, but such a primitive is still quite expensive and introduces overhead for the computation party prohibitive for most practical purposes. Thus any construction that improves the efficiency for all parties with smaller overhead in the outsourced computation would be of great interest. The other desirable property for outsourced computation is verifiability. Our solution can handle the same class of functions as the class of functions that an attribute-based encryption with efficient encryption algorithm (linear in the size of the input) can admit as policies. Currently this class includes Boolean formulas. While solutions for verifiable computation for general functions exist, they employ expensive cryptographic techniques such as FHE and PCPs. Obtaining more efficient solutions for larger classes of functions as well as handling computation that depends on the inputs of multiple parties present interesting open questions. While constructions achieving privacy and verifiability separately have obvious applications in the outsourced setting, an efficient solution that provides both guarantees at the same time will be of interest for many scenarios.

The MPC construction that we propose to obtain MPC with amortized efficiency sublinear in the size of the input opens the door for applying MPC techniques to settings where the inputs for the computation are parts of large databases and only algorithms with sublinear complexity are of interest from a practical point of view. The main disadvantage of the current instantiations of our construction is that they involve multiple interactions for each memory access, which is due to the fact that all existing ORAM schemes require multiple rounds of interaction. Achieving a single round ORAM construction would greatly benefit the performance of the resulting MPC schemes.

Efforts to identify scenarios where MPC techniques will be applicable, to determine the realistic workloads for these settings as well as the acceptable efficiency overhead, provide a reference framework for the efficiency that usable implementations need to achieve. Trying to construct and implement protocols that manage to meet these requirements even at the

price of relaxed security notions is the first step to bring MPC solutions to practical uses. The setting of encrypted search that we explore in this work presents a good example of this direction. Extending this scenario with more complicated search functionalities as well as considering other setups of interest will be natural next steps.

Part IV

Appendices

Appendix A

Secure Computation with Sublinear Amortized Work

A.1 Supporting Subprotocols

In this section we describe the Yao garbled circuits that we use for the implementation of our protocol from Section 3.5.3. We use the following notation:

- v_C and v_S are shares from the virtual address $\mathsf{v}_C \oplus \mathsf{v}_S$ being sought;
- vir_C and vir_S are shares of $\mathsf{vir}_C \oplus \mathsf{vir}_S$, which is either the real or the dummy address searched in some level;
- done_C and done_S are shares of $\mathsf{done}_C \oplus \mathsf{done}_S$, which indicates whether the virtual address has already been found;
- d_C and d_S are shares of $d_C \oplus d_S$, which stores the retrieved data when the the virtual address has been found;
- $F(r)$ denotes PRF value used for encryption.
- (c_V, c_D) are the ciphertexts (encryptions of the virtual address and the data) stored in a physical position in the ORAM structure;

CheckData

Here we check whether a ciphertext (c_1, c_2) matches the input value $v = \mathsf{v}_C \oplus \mathsf{v}_S$. If it does, then we share the corresponding data between the client and the server as $d_C \oplus d_S$. If the virtual address was already found, i.e. $\mathsf{done}_C \oplus \mathsf{done}_S = 1$, we ignore the check. If the virtual address is just matched, we appropriately set the check bit $\mathsf{done}'_C \oplus \mathsf{done}'_S = 1$, re-encrypt the ciphertext as (c'_1, c'_2) to be stored at the server.

Inputs: Client: $\mathsf{v}_C, \mathsf{rw}_C, d_C$, done_C, $F_K(r_1)$, $F_K(r_2)$, $F_K(r_3)$, $F_K(r_4)$
Server: $\mathsf{v}_S, \mathsf{rw}_S, d_S$, done_S, (c_1, c_2)

Protocol:

1. Decrypt the ciphertext (c_1, c_2) to recover the values $\mathsf{v} = c_1 \oplus F_K(r_1)$ and $d = c_2 \oplus F_K(r_2)$.
2. Check whether the data value needs to be retrieved:
   - If $\mathsf{done}_C \oplus \mathsf{done}_S = 1$, compute two shares done'_S and done'_C of 1, and two new shares d'_S and d'_C of the data $d_S \oplus d_C$.
   - Else if $\mathsf{done}_C \oplus \mathsf{done}_S = 0$ and $\mathsf{v} = \mathsf{v}_C \oplus v_S$, compute two shares done'_S and done'_C of 1. If $\mathsf{rw}_C \oplus \mathsf{rw}_S = \mathit{read}$, compute two shares d'_S and d'_C of the retrieved data d.
   - Else compute shares done'_S and done'_C of 0, and another set of shares of 0: d'_S and d'_C.
3. Compute encryptions to be written back:
   - If $\mathsf{done}_C \oplus \mathsf{done}_S = 0$ and $\mathsf{v} = \mathsf{v}_C \oplus \mathsf{v}_S$, set $c' = (\text{"}\mathit{dummy}\text{"} \oplus F_K(r_3), \text{"}\mathit{dummy}\text{"} \oplus F_K(r_4))$.
   - Else $c' = (\mathsf{v} \oplus F_K(r_3), d \oplus F_K(r_4))$.

Outputs: Client: done'_C, d'_C
Server: $\mathsf{done}'_S, d'_S, c'$

Figure A.1: A functionality that enables the players to obliviously check whether a data item matches the target.

GetHashInput

We compute the virtual address that will next be looked-up in a level: the real virtual address, if there was no match so far, or a dummy address depending on the counter t, if the item was already found.

Inputs: Client: $\mathsf{done}_C, \mathsf{vir}_C, t$
Server: $\mathsf{done}_S, \mathsf{vir}_S$

Protocol:

1. If $\mathsf{done}_C \oplus \mathsf{done}_S = 0$, create a random secret sharing $\mathsf{v}_S \oplus \mathsf{v}_C = \mathsf{vir}_C \oplus \mathsf{vir}_S$.
2. If $\mathsf{done}_C \oplus \mathsf{done}_S \neq 0$, create a random secret sharing $\mathsf{v}_S \oplus \mathsf{v}_C = (\text{"}dummy\text{"} \circ t)$.

Outputs: Client: v_C
Server: v_S

Figure A.2: A functionality that determines whether a real or a dummy look-up should be performed

DataWrite

We check whether the matched item stored in $\mathsf{vir}_C \oplus \mathsf{vir}_S$ and $d_C \oplus d_S$ should be used to over-write the current position in the top level, which currently stores ciphertext (c_1, c_2). We overwrite if these are empty encryptions, or an older encryption of the same virtual address.

Inputs: Client: $\mathsf{done}_C, \mathsf{vir}_C, d_C, \mathsf{done}_C$, $F_K(r_1)$, $F_K(r_2)$, $F_K(r_3)$, $F_K(r_4)$
Server: $\mathsf{done}_S, \mathsf{vir}_S, d_S, \mathsf{done}_S$, (c_1, c_2)

Protocol:

1. Decrypt the ciphertext (c_1, c_2) to recover the values $v = c_1 \oplus F_K(r_1)$ and $d = c_2 \oplus F_K(r_2)$.
2. Compute $v' = \mathsf{vir}_C \oplus \mathsf{vir}_S$ and $d' = d_C \oplus d_S$.
3. Check whether this is the right place to write an encryption of (v', d'):
   - If $\mathsf{done}_C \oplus \mathsf{done}_S = 1$, compute two random shares done'_S and done'_C of 1, and set $(c'_1, c'_2) = (v \oplus F_K(r_3), d \oplus F_K(r_4))$.
   - Else if $v = \mathsf{vir}_C \oplus \mathsf{vir}_S$ or $v = 0$, compute two random shares done'_S and done'_C of 1, and set $(c'_1, c'_2) = (v' \oplus F_K(r_3), d' \oplus F_K(r_4))$.
   - Else, compute two random shares done'_S and done'_C of 0, and set $(c'_1, c'_2) = (v \oplus F_K(r_3), d \oplus F_K(r_4))$.

Outputs: Client: done_C
Server: $\mathsf{done}'_S, (c'_1, c'_2)$

Figure A.3: A functionality for determining whether a value should be written to a given position in the top level.

Distributed Universal Hash Function

We compute a universal hash of a value shared between the client and the server as $u_C \cdot u_S$, and send the encrypted output c to the server. Let G be a prime order group. Let a and b be parameters defining the universal hash function from [Mansour *et al.*, 1993].

Inputs: Client: $u_C \in G$, $F(r) \in \{0,1\}^m$
Server: $u_S \in G, a \in \{0,1\}^{n+m-1}, b \in \{0,1\}^m$

Protocol:

1. Set $u = u_c u_S$, and "cast" u as an integer.
2. For $1 \leq i \leq m$ compute the i-th bit of the hash as $y_i = (\oplus_{j=1}^{n}(u_j \; AND \; a_{i+j-1})) \oplus b_i$.
3. Let $y = y_1||y_2||\ldots||y_m$ and $c = y \oplus F(r)$.

Outputs: Client: no output
Server: c

Figure A.4: A functionality for the distributed computation of a universal hash function.

Oblivious Swap

We re-order two encrypted values (v_1, d_1) and (v_2, d_2) by their virtual address by decrypting, comparing (according to some criteria, described in Section **??**, swapping if necessary, and re-encrypting.

Inputs: Client: $F_K(r_1), F_K(r_2), F_K(r_3), F_K(r_4), F_K(r'_1), F_K(r'_2), F_K(r'_3), F_K(r'_4)$
Server: $(v_1 \oplus F_K(r_1), d_1 \oplus F_K(r_2)), (v_2 \oplus F_K(r_3), d_2 \oplus F_K(r_4))$

Computation:

1. Decrypt the input ciphertexts to recover the values $v_1 = (v_1 \oplus F_K(r_1)) \oplus F_K(r_1)$ and $v_2 = (v_2 \oplus F_K(r_3)) \oplus F_K(r_3)$.
2. Compare the values v_1 and v_2 :
   - If $v_1 \leq v_2$: set $b = 0$, $(c'_1, c'_2) = (v_1 \oplus F_K(r'_1), d_1 \oplus F_K(r'_2))$, and $(c'_3, c'_4) = (v_2 \oplus F_K(r'_3), d_2 \oplus F_K(r'_4))$.
   - If $v_1 > v_2$: set $b = 1$, $(c'_1, c'_2) = (v_2 \oplus F_K(r'_1), d_2 \oplus F_K(r'_2))$, and $(c'_3, c'_4) = (v_1 \oplus F_K(r'_3), d_1 \oplus F_K(r'_4))$.

Outputs: Client: no output
Server: (c'_1, c'_2) and (c'_3, c'_4)

Figure A.5: A Functionality that enables the players to obliviously compare and swap two elements. This is used repeatedly for an oblivious sort.

Remove Excess Empties

Let count be an array of n counter variables ranging from 1 to m. Let index be a bucket index from 1 to n. Let real be a boolean flag indicating whether index is associated with a real item or an empty item.

Inputs: Server: $\mathsf{index} \oplus F_K(r_1), \mathsf{count} \oplus F_K(r_2), \mathsf{real}$
Client: $F_K(r_1), F_K(r_2), F_K(r_3), F_K(r_4)$

Computation:

1. Recover the values of count and index by computing $(\mathsf{index} \oplus F_K(r_1)) \oplus F_K(r_1)$ and $(\mathsf{count} \oplus F_K(r_2)) \oplus F_K(r_2)$.
2. If $(\mathsf{count}[\mathsf{index}] < m)$
   - $\mathsf{count}[\mathsf{index}]$ ++;
   - let $(c_1, c_2) = (\mathsf{index} \oplus F_K(r_3), \mathsf{count} \oplus F_K(r_4))$
3. Else if $(\mathsf{count}[\mathsf{index}] == m$ and $\mathsf{real} == \mathsf{false})$
   - let $(c_1, c_2) = (\perp \oplus F_K(r_3), \mathsf{count} \oplus F_K(r_4))$
4. Else if $(\mathsf{count}[\mathsf{index}] == m$ and $\mathsf{real} == \mathsf{true})$
   - let $(c_1, c_2) = (\mathsf{abort!}, \mathsf{abort!})$

Outputs: The output (c_1, c_2) is sent to the server.

Figure A.6: A functionality for counting m items in each bucket and removing excess empty items.

Appendix B

Secure Multiparty Computation for Multivariate Polynomials

B.1 Proof of HEPKPV Protocol

Lemma 8 Assume that $E = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ is a CPA-secure Vector Homomorphic encryption scheme. Then protocol Π_{POK} is a zero knowledge proof of knowledge for L.

Proof. **Completeness:** Assume the Prover knows $(x_1, \ldots, x_u), (r_1, \ldots, r_u)$ such that $c_1 = \mathsf{Enc_{pk}}(\mathsf{x_1}; \mathsf{r_1})$, $\ldots, c_u = \mathsf{Enc_{pk}}(\mathsf{x_u}; \mathsf{r_u})$ and that $(c_1, \ldots, c_u) \in L$. Then the Prover can always provide correct responses to the verification challenges requested by the Verifier. Therefore, the Verifier will always accept.

Soundness: If the Prover does not know some x_i, r_i for some c_i or $(x_1, \ldots, x_u) \notin L$, the probability that the Verifier will accept is at most $1/2^k$.

Zero Knowledge: Let A_V be a non-uniform probabilistic polynomial-time real adversary that controls the Verifier. We construct a non-uniform probabilistic expected polynomial time simulator S_V. The idea behind how S_V works is that it chooses $b_1 \ldots b_k$ ahead of time. If $b_i = 0$, the simulator chooses e_{ij}, r_{ij} for $1 \leq j \leq u$ and computes $c_{ij} = \mathsf{Enc}(\mathsf{e_{ij}}; \mathsf{r_{ij}})$ such that $(c_{i1}, \ldots, c_{iu}) \in L$. The simulator then sends these values on the broadcast channel. Otherwise, if $b_i = 1$, the simulator chooses s_{ij} and rs_{ij} for $1 \leq j \leq u$ and computes $c'_{ij} = \mathsf{Enc}(\mathsf{s_{ij}}; \mathsf{rs_{ij}})$ such that $c'_{i1}, \ldots, c'_{iu}) \in L$. The simulator then sends $c_{ij} = c_j / c'_{ij}$ for

$1 \leq j \leq u$ over the broadcast channel.

S_V uses rewinding to Step 4 to ensure that the bits chosen by the coin-tossing protocol are equal to $b_1 \dots b_m$. Thus, the Simulator is able to answer all challenges correctly without knowing the witness w. The input used by A_V is distributed identically with respect to both a random $b_1 \dots b_m$ and the $b_1 \dots b_m$ chosen ahead of time. Therefore, the probability that the verifying parties $P_2, \dots, P_s$ open their commitments correctly is identical both before and after rewinding. Therefore, the expected number of times the simulator needs to rewind is 1 and so it runs in expected polynomial time.

Extraction: The idea behind how the Extractor works is that it plays the part of the verifying parties $P_2, \dots P_s$ and runs the protocol honestly until after the Prover P_1 opens the challenges corresponding to $b_1 \dots b_m$. Thus it learns either $(x_j + e_{ij}, r_j \cdot r_{ij})$ or (e_{ij}, r_{ij}) for each $1 \leq i \leq k$ and each $1 \leq j \leq u$. Now the Extractor rewinds and sends commitments for $P_2, \dots, P_s$ to different random sequences of bits, the Prover P_1 sends a different sequence of bits $v_1'' \dots v_k''$ and this results in a different set of challenge bits: $v_1 \dots v_k$. With probability at least $1 - 1/2^k$, there is some index a such that $v_a \neq b_a$. In this case, the Extractor has now seen both $(x_j + e_{aj}, r_j \cdot r_{aj})$ and (e_{aj}, r_{aj}) for all $1 \leq j \leq n$. Therefore he can now calculate (x_j, r_j) for $1 \leq j \leq u$, which is the witness for the language L.

□

B.2 Proof of Multiparty Coin Tossing

Lemma 9 If $E = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ is semantically secure homomorphic encryption scheme, Π_{coin} is a secure multiparty protocol with no honest majority among the participating parties.

Proof. Assume there is a fixed set B, $|B| \leq m$, chosen at the outset of the protocol and that a non-uniform probabilistic polynomial-time real adversary A_B controls the parties T_j such that $j \in B$. We construct a non-uniform probabilistic expected polynomial-time ideal model adversary simulator S_B.

Assume party T_i is honest, i. e. $T_i \notin B$. For each honest party T_j, S_B chooses random

R_j, r_j and sends $C_j = \mathsf{Enc}(\mathsf{R_j}, \mathsf{r_j})$ as commitment for the input of T_j. S_B uses the extractor for the multiparty HEPKPV protocol to obtain the inputs of all malicious parties and sends them to the trusted party. S_B uses the value returned from the trusted party to reconstruct R_i^*. If $i = s$ then S_B continues to the last step of the protocol and uses the simulator for the multiparty HEPKPV protocol to prove that the final value R was computed correctly. If $i \neq s$ then S_B rewinds the protocol to the step where all the commitments have already been sent but the parties have not yet opened the commitments and proved consistency. Now, S_B uses the simulator for the multiparty HEPKPV protocol to prove that the value R_i^* is consistent with the commitment sent by party T_i.

We now show that the view of A_B is indistinguishable in the Ideal Model when interacting with S_B and its view in the Real model when interacting with the honest parties. The first difference between the view of A_B in a real run of the protocol and in a simulated one is that S_B uses a simulated proof to prove that the commitment is consistent. However, the simulator for the multiparty HEPKPV guarantees indistinguishability for the two cases. The second difference is that the simulator uses a dummy commitment as the commitment of party T_i. Due to the hiding property of the commitment scheme, these two cases are also indistinguishable. □

B.3 Proofs of Input Preprocessing and Verification

Claim 1 Assume R was chosen randomly after T committed to its inputs through the Efficient Preprocessing protocol. If the parties run the Preprocessing Verification protocol and do not abort, then with all but negligible probability, the committed input shares of T are valid encryptions of $k+1$-sharing polynomials of the inputs $x_\ell, x_\ell^2, \ldots, x_\ell^{2^{\lfloor log d_\ell \rfloor}}$.

Proof. Fix x_ℓ. We prove by induction that if the verification does not abort then with probability $1 - i/2^k - .9^k \cdot i$ the sharing of $x_\ell^{2^i}$ is valid.

For the basis case, we have that the sharings of $x_\ell \in X_T$ are valid with probability 1 since the LIPEV protocol guarantees this.

Now, assume that the sharing of $x_\ell^{2^i}$ is valid with probability $1 - i/2^k - .9^k \cdot i$. We show that the sharing of $x_\ell^{2^{i+1}}$ is valid with probability $1 - (i+1)/2^k + .9^k(i+1)$.

The probability that the sharing of $x_\ell^{2^{i+1}}$ is invalid can be upper bounded by the following:

$$\begin{aligned} Pr[\text{sharing of } x_\ell^{2^{i+1}} \text{ invalid }] &\leq Pr[\text{ sharing of } x_\ell^{2^i} \text{ invalid }] + \\ &\quad Pr[\text{ sharing of } x_\ell^{2^{i+1}} \text{ invalid } | \text{ sharing of } x_\ell^{2^i} \text{ valid }] \\ &\leq i/2^k + .9^k \cdot i + \\ &\quad Pr[\text{ sharing of } x_\ell^{2^{i+1}} \text{ invalid } | \text{ sharing of } x_\ell^{2^i} \text{ valid }] \end{aligned}$$

To upper bound $Pr[$ sharing of $x_\ell^{2^{i+1}}$ invalid $|$ sharing of $x_\ell^{2^i}$ valid $]$, we note that if at least a .9-fraction of the shares of the intermediate polynomial $(x^{2^i})_\ell^2$ were computed correctly and the 0-shares of $(x^{2^i})_\ell^2, x_\ell^{2^{i+1}}$ are in $L_=$, then it must be the case that the shares of $x_\ell^{2^{i+1}}$ were all computed correctly.

Therefore, we can upper bound $Pr[$ sharing of $x_\ell^{2^{i+1}}$ invalid $|$ sharing of $x_\ell^{2^i}$ valid $]$ by the probabilities that Cut-and-choose passes on the shares of $(x^{2^i})_\ell^2$ but a .1-fraction were computed incorrectly or HEPKPV passes on 0-shares of $x_\ell^{2^{i+1}}, (x^{2^i})_\ell^2$ even though they are not in $L_=$.

By the soundness of HEPKPV and Lemma 9 above, this can be upper bounded by $1/2^k + .9^k$.

Therefore, we have that $Pr[$the sharing of $x_\ell^{2^{i+1}}$ is invalid $] \leq (i+1)/2^k + .9^k(i+1)$.

By a union bound, we get that the probability that any of the sharings are invalid is at most:

$|X| \cdot (\lfloor \log L \rfloor)^2 \cdot (1/2^k + .9^k)$, which is negligible. □

Lemma 10 For all sets X_1, X_2 where $|X_1| = |X_2| = poly(k)$, we have that the output distributions of the Verifier in consecutive executions of the Efficient Preprocessing protocol and Preprocessing Verification protocol with inputs X_1 and X_2 (respectively) are computationally indistinguishable.

Proof. Intuitively, we need to show that the verifying party does not learn anything about $P_{x_\ell}(0)$ for $x_\ell \in X$ when the Efficient Preprocessing and Preprocessing Verification protocol

are executed. Now, for the technical proof: assume there is a distinguisher $\mathcal{A}$ that distinguishes the output distribution of the verifier when using input X_1 versus X_2 for the fixed sets X_1, X_2. Thus, $\mathcal{A}$ distinguishes between the output of the (malicious) Verifier in the following two experiments: ***Expt1*** and ***Expt2*** in which the Input preprocessing verification protocol is executed with commitments from the Efficient preprocessing protocol obtained from X_1 and X_2 respectively and a verification set $R \in [|J|]$. We now show that the view of the verifiers in Expt1 and Expt2 are computationally indistinguishable.

Assume there is a polynomial-time adversarial verifier $\mathcal{A}$ that distinguishes between the verifier's output distribution in Expt1 and Expt2. Then we show that there is a polynomial-time adversary $\mathcal{A}_{CPA}$ that can break the semantic security of the encryption scheme $(\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$. Since the CPA security of $(\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ implies that $(\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ is also many-message CPA secure when $poly(k)$ ciphertexts are concatenated, it is sufficient to show that there is a polynomial-time adversary $\mathcal{A}_{CPA}$ that breaks the many-message CPA security of $(\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ when $\sum_{1 \leq j \leq s} 2 \cdot \alpha_j$ ciphertexts are concatenated.

We now describe the adversary $\mathcal{A}_{CPA}$: $\mathcal{A}_{CPA}$ forwards the vectors V_1 and V_2 to the CPA experiment where

$$V_1 = [x_\ell^i]_{x_\ell \in X_1, 1 \leq i \leq \alpha_\ell}, [(x_\ell^i)^2]_{x_\ell \in X_1, 1 \leq i \leq \alpha_\ell - 1}$$

and

$$V_2 = [x_\ell^i]_{x_\ell \in X_2, 1 \leq i \leq \alpha_\ell}, [(x_\ell^i)^2]_{x_\ell \in X_2, 1 \leq i \leq \alpha_\ell - 1}.$$

$\mathcal{A}_{CPA}$ then receives the encryptions of the values in either V_1 or V_2 encrypted under public key pk and with independent randomness.

Let R_1 and R_2 are domains for plaintext and randomness for the encryption scheme $(\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$. Let $S \in |J|$ such that $J_R \cap J_S = \emptyset$. $\mathcal{A}_{CPA}$ constructs the following encryptions:

- For each $1 \leq j \leq s, 1 \leq \ell \leq \alpha_j, i \in J_R$ $\mathcal{A}_{CPA}$ chooses uniformly at random $x_{\ell,i,j} \xleftarrow{\$} R_1$ and $r_{\ell,i,j} \xleftarrow{\$} R_2$ and computes the pair $(i, \mathsf{Enc}_{\mathsf{pk}}(\mathsf{x}_{\ell,\mathsf{i},\mathsf{j}}; \mathsf{r}_{\ell,\mathsf{i},\mathsf{j}}))$.
- For each $1 \leq j \leq s, 1 \leq \ell \leq \alpha_j - 1, i \in J_R$ choose uniformly at random $r'_{\ell,i,j} \xleftarrow{\$} R_2$ and compute the pair $(i, \mathsf{Enc}_{\mathsf{pk}}((\mathsf{x}_{\ell,\mathsf{i},\mathsf{j}})^2; \mathsf{r}'_{\ell,\mathsf{i},\mathsf{j}}))$.

- For each $1 \leq j \leq s, 1 \leq \ell \leq \alpha_j - 1, i \in J_S$ choose uniformly at random $x'_{\ell,i,j} \stackrel{\$}{\leftarrow} R_1$ and $r'_{\ell,i,j} \stackrel{\$}{\leftarrow} R_2$ and computes the pair $(i, \mathsf{Enc_{pk}}(\mathsf{x'_{\ell,i,j}}; \mathsf{r'_{\ell,i,j}}))$.

Now $\mathcal{A}_{CPA}$ uses Lagrange interpolation over encrypted values with the corresponding pairs computed above to fill in the rest of the values in the following vectors that represent encryptions of shares of the challenge ciphertexts that $\mathcal{A}_{CPA}$ receives.

- Input shares: $\mathsf{Enc_{pk}}(\mathsf{P}_{\mathsf{x_j^{2^{\ell+1}}}}(\mathsf{i}), \mathsf{r_{j,\ell,i}})^{\Delta} = \mathsf{Enc_{pk}}(\Delta \cdot \mathsf{P}_{\mathsf{x_j^{2^{\ell+1}}}}(\mathsf{i}), \mathsf{r}^{\Delta}_{\mathsf{j,\ell,i}})$ and
- Intermediate shares: $\mathsf{Enc_{pk}}(\mathsf{P}^2_{\mathsf{x_j^{2^{\ell}}}}(\mathsf{i}), \mathsf{r'_{j,\ell,i}})^{\Delta} = \mathsf{Enc_{pk}}(\Delta \cdot \mathsf{P}^2_{\mathsf{x_j^{2^{\ell}}}}(\mathsf{i}), (\mathsf{r'_{j,\ell,i}})^{\Delta})$.

$\mathcal{A}_{CPA}$ outputs these values as the commitments in the Efficient Preprocessing protocol and then uses them as inputs to the Preprocessing Verification protocol. Note that $\mathcal{A}_{CPA}$ is able to answer the challenge in the Verification protocol since $\mathcal{A}_{CPA}$ knows the plaintext and randomness in the encryptions indexed by R. $\mathcal{A}_{CPA}$ outputs whatever $\mathcal{A}$ outputs given the (malicious) verifier's output. Now we have that if the challenge ciphertext was an encryption of X_1 then the output of $\mathcal{A}$ is distributed identically to its output in **Expt1**. On the other hand, if the challenge ciphertext was an encryption of X_2 then the output of $\mathcal{A}$ is distributed identically to its output in **Expt2**. Thus $\mathcal{A}_{CPA}$ breaks the CPA security of the encryption scheme with the same probability that $\mathcal{A}$ distinguishes between the verifier's output in **Expt1** and **Expt2**. □

Appendix C

How to Delegate and Verify in Public: Verifiable Computation from Attribute-based Encryption

C.1 Note on Terminology: Attribute-based Encryption versus Predicate Encryption

We consider attribute-based encryption (ABE) schemes to be ones in which each secret key $\mathsf{ABE.SK}_F$ is associated with a function F, and can decrypt ciphertexts that encrypt a message m under an "attribute" x if and only if $F(x) = 1$. This formulation is implicit in the early definitions of ABE introduced by Goyal, Pandey, Sahai and Waters [Sahai and Waters, 2005; Goyal *et al.*, 2006]. However, their work refers to F as an access structure, and existing ABE instantiations are restricted to functions (or access structures) that can be represented as polynomial-size span programs (a generalization of Boolean formulas) [Goyal *et al.*, 2006; Ostrovsky *et al.*, 2007; Lewko *et al.*, 2010]. While such restrictions are not inherent in the definition of ABE, the fully general formulation we use above was first explicitly introduced by Katz, Sahai, and Waters, who dubbed it predicate encryption [Katz *et al.*, 2008]. Note that we do not require attribute-hiding or policy/function-hiding, properties often associated with predicate encryption schemes (there appears to be some confusion in the literature

as to whether attribute-hiding is inherent in the definition of predicate encryption [Katz *et al.*, 2008; Lewko *et al.*, 2010; Boneh *et al.*, 2011], but the original formulation [Katz *et al.*, 2008] does not seem to require it).

Thus, in a nutshell, our work can be seen as using ABE schemes for general functions, or equivalently, predicate encryption schemes that do not hide the attributes or policy, in order to construct verifiable computation protocols.

C.2 Attribute-based Encryption from Verifiable Computation

Given that we have shown how to construct a verifiable computation (VC) protocol from an attribute-based encryption (ABE) scheme, it is natural to ask whether the reverse implication holds. In other words, can we construct an ABE scheme, given a VC scheme? At first sight, the key property of a VC scheme – namely, efficient verification – does not seem to have anything to do with attribute-based encryption.

Despite this apparent mismatch of functionality, we show how to transform a (very) restricted class of (publicly) verifiable computation protocols – that we call weak "multi-function verifiable computation" – into an attribute-based encryption scheme for the same set of functions. Informally, a weak multi-function VC protocol has the following features:

- The output of $\mathsf{ProbGen}$ on an input x can be used to compute many different functions on x. Thus, in some sense, $\mathsf{ProbGen}$ is agnostic of the function that will be computed on the input.

  In particular, we now have a setup algorithm that generates a pair of public and secret parameters, a key generation algorithm KeyGen (as before) that generates a secret key SK_F for a function F given the secret parameters, and a $\mathsf{ProbGen}$ algorithm that (as before) given an input x and the public parameters generates an encoding of x together with a verification key. Thus, $\mathsf{ProbGen}$ does not know about F and KeyGen does not know about x. Indeed, this is the crucial property that gives us ABE.

- The verification key for an input x consists of a pair (VK_x^0, VK_x^1), and the verification

algorithm consists of simply applying a collision-resistant hash function H to the server's response σ_y and checking if it equals VK_x^0 or VK_x^1.

Indeed, the VC schemes we constructed from ABE can both be tweaked to have these properties.

The high level idea for the construction of an ABE scheme from a multi-function VC scheme is as follows: in order to encrypt a message under a particular attribute (in the ABE scheme), we first generate a key that can be computed only if the output of the server in the VC protocol verifies correctly. Now, decryption of the ciphertext will succeed only if the decryptor correctly performs the evaluation of the key's function on the attribute associated with the ciphertext, and the output value of the computation satisfies the decryption condition, in which case he will have the correct decryption key for the ciphertext.

Put another way, the security of a VC scheme implies it should be difficult for an adversary to produce an output that does not correspond to a legitimate computation of a function on a particular input. If we make decryption of a ciphertext dependent on having a particular output, then the computation possible given a key for the function and an attribute/input either legitimately produces the expected output, allowing decryption of the ciphertext, or produces some other output, and it is infeasible to produce the output necessary to decrypt the ciphertext.

We define the notion of a weak multi-function VC protocol below. The "weakness" in the definition comes from the fact that we only need a multi-function VC scheme that verifies that a particular output is legitimate for *some* outsourced function (i.e., a function given as input to KeyGen), rather than for a specific function.

Below, we combine these requirements in a single definition, demonstrate that both the constructions from Section 5.3 and Section 5.4 satisfy this definition, and finally show how to use such a definition to construct an ABE scheme.

Definition 39 (Weak Multi-Function Public Verifiable Computation)**.** *A VC scheme* $\mathcal{VC} = (\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{ProbGen}, \mathsf{Compute}, \mathsf{Verify})$ *is a weak multi-function public verifiable computation scheme if it has the following properties:*

- $\mathsf{Setup}(\lambda) \rightarrow (PK_{param}, SK_{param})$*: Produces the public and private parameters that do*

not depend on the functions to be evaluated.

- $\mathsf{KeyGen}_{PK_{param},SK_{param}}(F) \rightarrow PK_F$*: Produces a public key for evaluating a specific function* F*.*

- $\mathsf{ProbGen}_{PK_{param}}(x) \rightarrow (\sigma_x, VK_x = (VK_x^0, VK_x^1))$*: The algorithm requires only the public parameters, which are independent of the function that will be computed. It generates both the encoding* σ_x *for the input, and the public verification keys for each possible bit of the output, in this case, simply* VK_x^0 *and* VK_x^1*.*

- $\mathsf{Compute}_{PK_{param},PK_F}(\sigma_x) \rightarrow \sigma_y$*: The computation algorithm uses both parts of the public key to produce an encoding of the output* $y = F(x)$*.*

- $\mathsf{Verify}_{VK_x}(\sigma_y) \rightarrow y \cup \perp$*: Using the public input-specific value* VK_x*, the verification algorithm outputs* 0 *if* $VK_x^0 = H(\sigma_y)$*, outputs* 1 *if* $VK_x^1 = H(\sigma_y)$*, and outputs* $\perp$ *otherwise, to indicate that* σ_y *does not represent a valid output of some function* F*, for which* $\mathsf{KeyGen}(F)$ *has been invoked, on* x

Definition 40 (Weak Multi-Function Public Verifiable Computation Security)**.** *Let* $\mathcal{VC} = (\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{ProbGen}, \mathsf{Compute}, \mathsf{Verify})$ *be a weak multi-function public verifiable computation scheme. We define security via the following experiment.*

Experiment $\mathbf{Exp}_A^{WeakMultVerif}[\mathcal{VC}, \lambda]$

$(PK_{param}, SK_{param}) \xleftarrow{R} \mathsf{KeyGen}(\lambda)$*;*

$x \leftarrow A^{\mathcal{O}_{\mathsf{KeyGen}}(\cdot)}(PK_{param})$*;*

$(\sigma_x, VK_x) \leftarrow \mathsf{ProbGen}_{PK_{param}}(x)$*;*

$\hat{\sigma}_y \leftarrow A^{\mathcal{O}_{\mathsf{KeyGen}}(\cdot)}(PK_{param}, \sigma_x, VK_x)$*;*

$\hat{y} \leftarrow \mathsf{Verify}_{VK_x}(\hat{\sigma}_y)$

If $\hat{y} \neq \perp$ *and* $\forall F \in \mathcal{R} : \hat{y} \neq F(x)$*, output '1', else '0';*

We define the adversary's advantage and the scheme's security in the same fashion as Definition 8.

In the experiment, the adversary has oracle access to $\mathcal{O}_{\mathsf{KeyGen}}(F)$, which calls $\mathsf{KeyGen}_{PK_{param},SK_{param}}(F)$, returns PK_F, and stores F in the list $\mathcal{R}$. Eventually, the adversary returns an encoding $\hat{\sigma_y}$ which purports to be an output of some outsourced function

applied to x. The challenger runs Verify with the corresponding values of VK_x, and the adversary wins if this check passes, but the output does not correspond to the output of one of the functions in list $\mathcal{R}$.

Note that both the constructions from Section 5.3 and Section 5.4 satisfy this definition. The ABE to VC construction (Section 5.3) does not include any function verification to begin with, but it still verifies that the output, i.e., the message obtained after performing a decryption, could not have been obtained without performing a legitimate computation (decryption) with one of the keys generated by KeyGen. In contrast, Construction 1 is too strong, since it verifies the specific function used. To weaken it, we can simply add the private decryption key SK_F to the computation key, which was previously TK_F. This removes the ability to verify which function was used, and hence fits within the definition above.

Below, we describe our construction from VC to ABE in more detail.

Construction 2. *Let* $\mathcal{VC} = (\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{ProbGen}, \mathsf{Compute}, \mathsf{Verify})$ *be a weak multi-function public verifiable computation scheme, and* H *be an injective one way function. We construct the following key-policy attribute-based encryption scheme* $\mathcal{ABE}_{VC}$*.*

- $\mathsf{Setup}(\lambda, U) \rightarrow (PK, MSK)$*: Run* $\mathcal{VC}.\mathsf{Setup}(\lambda) \rightarrow (PK_{param}, SK_{param})$ *and output* $PK = PK_{param}$ *and* $MSK = SK_{param}$*.*

- $\mathsf{Enc}_{\mathsf{PK}}(\mathsf{M}, \gamma) \rightarrow \mathsf{C}$ *where* $M \in \{0,1\}$*:*

  - *Run* $(\sigma_x, VK_x = (VK_x^0, VK_x^1)) \leftarrow \mathcal{VC}.\mathsf{ProbGen}_{PK_{params}}(x)$*.*
  - *Let* σ_{ans} *be such that* $H(\sigma_{ans}) = VK_x^1$*. Choose a random value* r *and compute* $K = \langle \sigma_{ans}, r \rangle$ *where* $\langle \cdot, \cdot \rangle$ *denotes the inner product of two bit-strings (mod 2).*

  Output ciphertext $C = (\sigma_x, r, K \oplus M)$*.*

- $\mathsf{KeyGen}_{MSK}(F) \rightarrow SK_F$*: Run* $PK_F \leftarrow \mathcal{VC}.\mathsf{KeyGen}_{PK_{param}, SK_{param}}(F)$*. Output* $SK_F = PK_F$*.*

- $\mathsf{Dec}_{\mathsf{SK_F}}(\mathsf{C}) \rightarrow \mathsf{M} \cup \perp$*: Parse* C *as* (σ_x, r, D)*.*

  - *Run* $\sigma_{ans} \leftarrow \mathcal{VC}.\mathsf{Compute}_{PK_{params}, PK_F}(\sigma_x)$*.*

– *Compute* $K \leftarrow \langle \sigma_{ans}, r \rangle$. *Output* $K \oplus D$.

Correctness follows from the fact that if $F(x) = 1$, then the answer σ_{ans} produced by the server (upon running $\mathcal{VC}.\mathsf{Compute}$) is such that $H(\sigma_{ans}) = VK_x^1$. Since H is an injective one-way function, $\langle \sigma_{ans}, r \rangle \oplus D = M$.

We now proceed to showing the security of the ABE scheme. First, we state the Goldreich-Levin lemma [Goldreich and Levin, 1989].

Lemma 17 (Goldreich-Levin [Goldreich and Levin, 1989])**.** *Let* $f : \{0,1\}^n \rightarrow \{0,1\}^n$ *be a bijection computable by a circuit of size* t *and suppose there is a circuit* $\mathcal{C}$ *of size* s *such that*

$$Pr_{x,r}[C(f(x), r) = \langle x, r \rangle] = \frac{1}{2} + \epsilon.$$

Then there is a circuit $\mathcal{C}'$ *of size* $O((s+t) \cdot poly(n, 1/\epsilon))$ *such that*

$$Pr_x[\mathcal{C}'(f(x)) = x] = \frac{\epsilon}{4}.$$

Theorem 27. *If* $\mathcal{VC} = (\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{ProbGen}, \mathsf{Compute}, \mathsf{Verify})$ *is a secure weak multi-function public VC scheme (see Definitions 40), then* $\mathcal{ABE}_{VC}$*, the ABE scheme obtained with Construction 2, is IND-CPA secure (Definition 13).*

Proof Sketch. Assume for the sake of contradiction that there exists an adversary $\mathcal{A}_{ABE}$ that wins with non-negligible probability μ the security game from Definition 13. We use this to break the soundness of the VC protocol in two steps, conceptually.

First, using Lemma 17, the existence of $\mathcal{A}_{ABE}$ means that there is an adversary that, given a ciphertext of the form (σ_x, r, D), predicts an inverse of VK_x^1 under the function H. Note that this transformation creates an adversary $\mathcal{A}'_{ABE}$ that asks polynomially many more ABE secret key queries than $\mathcal{A}_{ABE}$.

Since this adversary essentially predicts the message of the server, this can then be used to construct an adversary $\mathcal{A}_{VC}$ that breaks the soundness of the VC protocol (from Definition 40) with non-negligible probability. For completeness, we describe both these transformations as one algorithm $\mathcal{A}_{VC}$ that uses $\mathcal{A}_{ABE}$:

1. $\mathcal{A}_{VC}$ receives PK_{param}, the output from $\mathcal{VC}.\mathsf{Setup}(\lambda)$ from his challenger. He forwards PK_{param} to $\mathcal{A}_{ABE}$.

2. On calls to $\mathcal{O}_{ABE.\mathsf{KeyGen}}(F)$, $\mathcal{A}_{VC}$ queries his own $\mathcal{O}_{VC.\mathsf{KeyGen}}(F)$ oracle and returns the resulting PK_F.

3. Given the challenge messages (M_0, M_1) and attributes γ, $\mathcal{A}_{VC}$ requests as his challenge $(\sigma_\gamma, VK_\gamma = (VK_\gamma^0, VK_\gamma^1)) \leftarrow \mathsf{ProbGen}_{PK_{param}}(\gamma)$.

4. $\mathcal{A}_{VC}$ runs the adversary $\mathcal{A}_{gl}$ to obtain the next value r submitted to the oracle $\mathcal{O}_x(\cdot)$. $\mathcal{A}_{VC}$ chooses a random bit $d \xleftarrow{R} \{0,1\}$ and returns the challenge ciphertext $C = (r, \sigma_\gamma, d)$ to $\mathcal{A}_{ABE}$.

5. Eventually, $\mathcal{A}_{ABE}$ will return his guess bit $\hat{b}$. $\mathcal{A}_{VC}$ computes $d \oplus M_{\hat{b}}$ and returns this value as an answer to $\mathcal{A}_{gl}$.

6. When $\mathcal{A}_{gl}$ makes a new oracle query, $\mathcal{A}_{VC}$ rewinds the execution of $\mathcal{A}_{ABE}$ to Step 4.

7. When $\mathcal{A}_{VC}$ receives an answer $\tilde{\sigma}_0$ from $\mathcal{A}_{gl}$, he returns it as his output for the computation on input $(\sigma_\gamma, VK_\gamma = (VK_\gamma^0, VK_\gamma^1))$.

The probability that $\mathcal{A}_{VC}$ succeeds in cheating is the same as the probability that $\mathcal{A}_{gl}$ succeeds in inverting $VK_\gamma^0 = H(\sigma_0)$. By assumption $\mathcal{A}_{ABE}$ wins the security game from Definition 13 with non-negligible probability μ. Therefore $\mathcal{A}_{VC}$ returns the correct value for $\sigma_0 \cdot r$ to $\mathcal{A}_{gl}$ with probability μ. Then by Lemma 17 it follows that $\mathcal{A}_{gl}$ will compute the correct output σ_0 with probability $\frac{\epsilon}{4}$, which is non-negligible. Hence $\mathcal{A}_{VC}$ wins the security game from Definition 40 with non-negligible probability. Also $\mathcal{A}_{VC}$ runs in polynomial time since $\mathcal{A}_{gl}$ is a polynomial-time algorithm. □

Appendix D

Outsourcing Multi-Party Computation with Non-Colluding Adversaries

D.1 Garbled Circuits

Informally, Garb is considered secure if $(\mathbb{G}(C), \mathbb{G}(x), \mathbb{G}(y))$ reveals no information about x and y. An added property possessed by the construction is *verifiability* which, roughly speaking, means that, given $(\mathbb{G}(C), \mathbb{G}(x), \mathbb{G}(y))$, no adversary can output some $\mathbb{G}(o)$ such that $\mathsf{Translate}(\mathbb{G}(o), \mathrm{T}) \neq f(x, y)$. Next we discuss these properties more formally.

We recall the properties of Yao's garbled circuit construction which we make use of. These include correctness, privacy and verifiability.

Definition 41 (Correctness)**.** *We say that* $\mathsf{Garb} = (\mathsf{GarbCircuit}, \mathsf{GarbIn}, \mathsf{Compute}, \mathsf{GarbOut}, \mathsf{Translate})$ *is* correct *if for all functions* f*, for all circuits* C *computing* f*, for all coins* $r \in \{0,1\}^\lambda$*, and for all* x *and* y *in the domain of* f

$$\mathsf{Translate}\Big(\mathsf{Eval}\big(\mathsf{GarbCircuit}(C; r), \mathsf{GarbIn}(C, 1, x; r), \mathsf{GarbIn}(C, 2, y; r)\big), \mathsf{GarbOut}(r)\Big) = f(x, y).$$

Informally, Garb is considered private if the garbled circuit and the garbled inputs reveal no useful information about x and y.

Definition 42 (Privacy)**.** *We say that* $\mathsf{Garb} = (\mathsf{GarbCircuit}, \mathsf{GarbIn}, \mathsf{Compute}, \mathsf{GarbOut}, \mathsf{Translate})$ *is* private *if for all functions f, for all circuits C computing f, for all inputs x, y, x' and y' in the domain of f, the following distributions are computationally indistinguishable:*

$$\left\{\mathsf{GarbCircuit}\big(C;r\big), \mathsf{GarbIn}\big(C,1,x;r\big), \mathsf{GarbIn}\big(C,2,y;r\big)\right\}$$

and

$$\left\{\mathsf{GarbCircuit}\big(C;r'\big), \mathsf{GarbIn}\big(C,1,x';r'\big), \mathsf{GarbIn}\big(C,2,y';r'\big)\right\},$$

where r and r' are chosen uniformly at random.

Finally, we consider verifiability which, roughly speaking, means that, given a garbled circuit and two garbled inputs, no adversary can find a garbled output that will result in the translation algorithm returning an incorrect output.

Definition 43 (Verifiability)**.** *We say that* $\mathsf{Garb} = (\mathsf{GarbCircuit}, \mathsf{GarbIn}, \mathsf{Compute}, \mathsf{GarbOut}, \mathsf{Translate})$ *is* verifiable *if for all functions f, for all circuits C computing f, for all inputs x and y in the domain of f, for a uniformly random seed $r \in \{0,1\}^\lambda$, and for all* PPT *adversaries $\mathcal{A}$, the following probability is negligible in k:*

$$\Pr\left[\mathsf{Translate}\big(o', \mathsf{GarbOut}(s)\big) \neq f(x,y) : o' \leftarrow \mathcal{A}\big(\mathsf{GarbCircuit}(C;r), \mathsf{GarbIn}(C,1,x;r), \mathsf{GarbIn}(C,2,y;r)\big)\right]$$

where the probability is over the coins of $\mathcal{A}$.

D.2 Secure Delegated Computation

A delegated computation scheme consists of four polynomial-time algorithms $\mathsf{Del} = (\mathsf{Gen}, \mathsf{ProbGen}, \mathsf{Compute}, \mathsf{Verify})$ that work as follows. Gen is a probabilistic algorithm that takes as input a security parameter k and a function f and outputs a public and secret key pair $(\mathrm{PK}, \mathsf{sk})$ such that the public key encodes the target function f. $\mathsf{ProbGen}$ is a probabilistic algorithm that takes as input a secret key sk and an input x in the domain of f and outputs a public encoding σ_x and a secret state τ_x. $\mathsf{Compute}$ is a deterministic algorithm that takes as input a public key PK and a public encoding σ_x and outputs a public encoding σ_y. Verify

is a deterministic algorithm that takes as input a secret key sk, a secret state τ_x and a public encoding σ_y and outputs either an element y of f's range or the failure symbol $\perp$.

We recall the formal definitions of correctness, verifiability and privacy for a delegated computation scheme.

Definition 44 (Correctness)**.** *A delegated computation scheme* $\mathsf{Del} = (\mathsf{Gen}, \mathsf{ProbGen}, \mathsf{Compute}, \mathsf{Verify})$ *is* correct *if for all functions* f*, for all* PK *and* sk *output output by* $\mathsf{Gen}(1^{\mathsf{k}}, \mathsf{f})$*, for all* x *in the domain of* f*, for all* σ_x *and* τ_x *output by* $\mathsf{ProbGen}_{\mathsf{sk}}(x)$*, for all* σ_y *output by* $\mathsf{Compute}_{\mathrm{PK}}(\sigma_x)$*,* $\mathsf{Verify}_{\mathsf{sk}}(\tau_x, \sigma_y) = f(x)$*.*

A delegated computation scheme is *verifiable* if a malicious worker cannot convince the client to accept an incorrect output. In other words, for a given function f and input x, a malicious worker should not be able to find some σ' such that the verification algorithm outputs $y' \neq f(x)$. This intuition is formalized in the following definition.

Definition 45 (Verifiability)**.** *Let* $\mathsf{Del} = (\mathsf{Gen}, \mathsf{ProbGen}, \mathsf{Compute}, \mathsf{Verify})$ *be a delegated computation scheme,* $\mathcal{A}$ *be an adversary and consider the following probabilistic experiment* $\mathsf{Ver}_{\mathsf{Del},\mathcal{A}}(k)$*:*

1. *the challenger computes* $(\mathrm{PK}, \mathsf{sk}) \leftarrow \mathsf{Gen}(1^{\mathsf{k}}, \mathsf{f})$*,*
2. *let* $\mathcal{O}(\mathsf{sk}, \cdot)$ *be a probabilistic oracle that takes as input an element* x *in the domain of* f*, computes* $(\sigma, \tau) \leftarrow \mathsf{ProbGen}_{\mathsf{sk}}(x)$ *and outputs* σ*,*
3. *given* PK *and oracle access to* $\mathcal{O}(\mathsf{sk}, \cdot)$*,* $\mathcal{A}$ *outputs an input* x*,*
4. *the challenger computes* $(\sigma_x, \tau_x) \leftarrow \mathsf{ProbGen}_{\mathsf{sk}}(x)$*,*
5. *given* σ_x*, the adversary* $\mathcal{A}$ *outputs an encoding* σ'*,*
6. *if* $\mathsf{Verify}_{\mathsf{sk}}(\tau, \sigma') \notin \{\perp, f(x)\}$ *then output* 1 *else output* 0*.*

We say that Del *is verifiable if for all* PPT *adversaries* $\mathcal{A}$*,*

$$\Pr\left[\,\mathsf{Ver}_{\mathsf{Del},\mathcal{A}}(k) = 1\,\right] \leq \mathrm{NEGL}(k)$$

where the probability is over the coins of Gen*,* $\mathcal{O}$*,* $\mathcal{A}$ *and* $\mathsf{ProbGen}$*.*

Informally, a delegated computation scheme is private if its public encodings reveal no useful information about the input x.

Definition 46 (Privacy)**.** *Let* $\mathsf{Del} = (\mathsf{Gen}, \mathsf{ProbGen}, \mathsf{Compute}, \mathsf{Verify})$ *be a delegated computation scheme,* $\mathcal{A}$ *be a stateful adversary and consider the following probabilistic experiment* $\mathsf{Priv}_{\mathsf{Del},\mathcal{A}}(k)$*:*

1. *the challenger computes* $(\mathrm{PK}, \mathsf{sk}) \leftarrow \mathsf{Gen}(1^{\mathsf{k}}, \mathsf{f})$*,*
2. *let* $\mathcal{O}(\mathsf{sk}, \cdot)$ *be a probabilistic oracle that takes as input an element* x *in the domain of* f*, computes* $(\sigma, \tau) \leftarrow \mathsf{ProbGen}_{\mathsf{sk}}(x)$ *and outputs* σ*,*
3. *given* PK *and oracle access to* $\mathcal{O}(\mathsf{sk}, \cdot)$*,* $\mathcal{A}$ *outputs two inputs* x_0 *and* x_1*,*
4. *the challenger samples a bit* b *at random and computes* $(\sigma_b, \tau_b) \leftarrow \mathsf{ProbGen}_{\mathsf{sk}}(x_b)$*,*
5. *given* σ_b*, the adversary* $\mathcal{A}$ *outputs a bit* b'*,*
6. *if* $b' = b$ *output* 1 *else output* 0*.*

We say that Del *is private if for all* PPT *adversaries* $\mathcal{A}$*,*

$$\Pr\left[\,\mathsf{Priv}_{\mathsf{Del},\mathcal{A}}(k) = 1\,\right] \leq \mathrm{NEGL}(k)$$

where the probability is over the coins of Gen*,* $\mathcal{O}$*,* $\mathcal{A}$ *and* $\mathsf{ProbGen}$*.*

D.3 Proof for Set Intersection Protocols

Theorem 14 The protocol in Figure 6.5 securely computes the 2-party set intersection functionality in the $\mathcal{F}_{\mathsf{CT}}$-hybrid model for the adversary structure Adv_5 defined as follows:

$$\mathsf{Adv}_5 = \left\{ \Big(\mathcal{A}_S[sh], \mathcal{A}_1[sh], \mathcal{A}_2[sh]\Big), \Big(\mathcal{A}_S[m], \mathcal{A}_1[h], \mathcal{A}_2[h]\Big), \Big(\mathcal{A}_S[nc_1, nc_2], \mathcal{A}_1[sh], \mathcal{A}_2[sh]\Big) \right\}.$$

Proof. We consider the different adversarial models in the following claims.

Claim 14. *The protocol* $\big(\mathcal{A}_S[sh], \mathcal{A}_1[sh], \mathcal{A}_2[sh]\big)$*-securely computes the* 2*-party set intersection functionality in the* $\mathcal{F}_{\mathsf{CT}}$*-hybrid model.*

We describe three independent transformations Sim_S, Sim_1 and Sim_2. The simulator Sim_1 simulates $\mathcal{A}_1$ as follows:

1. Sim_1 queries $\mathcal{F}_{\mathsf{CT}}$ to receive the common random string r used to derive K_1, K_2 and d, e_1, e_2 and answers $\mathcal{A}_1$'s queries to $\mathcal{F}_{\mathsf{CT}}$ correspondingly.

2. Sim_1 calls the trusted party submitting the input it has for the semi-honest $\mathcal{A}_1$ and obtains the output for $\mathcal{A}_1$

3. The simulator computes the corresponding PRP values for the output and sends those to $\mathcal{A}_1$.

We construct a simulator Sim_2 that simulates $\mathcal{A}_2$ analogously to Sim_1. The simulator Sim_S that simulates $\mathcal{A}_S$ as follows:

1. Sim_S generates two random sets X and Y of size m and n.

2. Sim_S chooses K_1, K_2 and d, e_1, e_2, computes honestly the PRP values for X and Y and submits them to $\mathcal{A}_S$.

The indistinguishability of the simulated and the real execution views follows easily from the pseudorandom properties of the PRP.

□

Claim 15. *The protocol* $\big(\mathcal{A}_S[m], \mathcal{A}_1[h], \mathcal{A}_2[h]\big)$*-securely computes the* 2*-party set intersection functionality in the* $\mathcal{F}_{\mathsf{CT}}$*-hybrid model.*

We construct a simulator Sim_S that simulates the adversary $\mathcal{A}_S$ as follows:

1. Sim_S generates two random sets X and Y of size m and n.

2. Sim_S chooses s_1, s_2 and d, e_1, e_2, computes honestly the PRP values for X and Y and submits them to $\mathcal{A}_S$.

3. Sim_S receives the output computed by $\mathcal{A}_S$. If the returned set is not the correct set of intersection PRP vales, the simulator sends an abort message to the trusted party and to $\mathcal{A}_S$.

The views of the adversary $\mathcal{A}_S$ in the real and the ideal execution are indistinguishable because of the properties of the PRP function and the fact that P_1 and P_2 are honest. In the ideal execution P_1 and P_2 receive as output the set intersection of their inputs if and only if $\mathcal{A}_S$ has computed it correctly (i.e., Sim_S has not submitted abort to the trusted party). Thus we need to show that in the real execution the probability that the parties will not abort, when the set returned by $\mathcal{A}_S$ is not the correct result, is negligible. A misbehavior of $\mathcal{A}_S$ will not be detected, if the intersection set that he returns contains all PRP values for the element d, does not contain any of the PRP values for e_1 and e_2, and for every PRP value in the returned set all the other $t-1$ PRP values that correspond to the respective element are also in the claimed intersection set. Let r be size of the set intersection. The probability that $\mathcal{A}_S$ removes $k \leq r$ values from the set intersection without being detected is (i.e., guesses the PRP values that correspond to the element d and then guesses kt PRP values that correspond to k input elements):

$$\binom{(r+1)t}{t}^{-1}\binom{r}{k}\binom{rt}{kt}^{-1}.$$

The probability that $\mathcal{A}_S$ adds $s \leq m-r$ ($s \leq n-r$) values from the set intersection returned to P_1 (P_2) without being detected is (i.e., guesses the t PRP values corresponding to e_1 (e_2) and then guesses st PRP values corresponding to s elements):

$$\binom{(n-r+1)t}{t}^{-1}\binom{(n-r)}{s}\binom{(n-r)t}{st}^{-1}.$$

The value $\binom{(r+1)t}{t}^{-1}$ is maximized when $r=1$ (if $r=0$, $\mathcal{A}_S$ cannot remove intersection values). Therefore,

$$\binom{(r+1)t}{t}^{-1} \leq \binom{2t}{t}^{-1} = \frac{t!t!}{(2t)!} = \frac{1\cdots t}{(t+1)\cdots 2t} < \frac{1}{2^t}.$$

Since $\binom{r}{k}\binom{rt}{kt}^{-1} < 1$, it follows that the probability that $\mathcal{A}_S$ removes any values from the set intersection without being detected is negligible. Similarly we get that the probability of that $\mathcal{A}_S$ adds any values from the set intersection without being detected is also negligible. Therefore, the probability that the set intersection that a party accepts as answer is not the correct result is negligible.

□

Claim 16. *The protocol* $\big(\mathcal{A}_S[nc_1, nc_2], \mathcal{A}_1[sh], \mathcal{A}_2[sh]\big)$*-securely computes the* 2*-party set intersection functionality in the* $\mathcal{F}_{\mathsf{CT}}$*-hybrid model.*

The proof of the claim follows from the above two claims and Lemma 12.

□

□

Theorem 15 The protocol in Figure 6.5 securely computes the 2-party set intersection functionality in the $\mathcal{F}_{\mathsf{CT}}$-hybrid model for the adversary structure Adv_6 defined as follows:

$$\mathsf{Adv}_6 = \mathsf{Adv}_5 \bigcup \Bigg\{ \Big(\mathcal{A}_S[h], \mathcal{A}_1[m], \mathcal{A}_2[h]\Big), \Big(\mathcal{A}_S[sh], \mathcal{A}_1[nc_S], \mathcal{A}_2[sh]\Big), \Big(\mathcal{A}_S[h], \mathcal{A}_1[h], \mathcal{A}_2[m]\Big), \Big(\mathcal{A}_S[sh], \mathcal{A}_1[sh], \mathcal{A}_2[nc_S, nc_1]\Big) \Bigg\}.$$

Proof. We consider only the cases in the adversarial structure that were not covered in the proof of Theorem 14.

Claim 17. *The protocol* $\big(\mathcal{A}_S[h], \mathcal{A}_1[m], \mathcal{A}_2[h]\big)$*-securely computes the* 2*-party set intersection functionality in the* $\mathcal{F}_{\mathsf{CT}}$*-hybrid model.*

We construct a simulator Sim_1 that simulates the adversary $\mathcal{A}_1$ as follows:

1. Sim_1 queries $\mathcal{F}_{\mathsf{CT}}$ to receive the common random string r used to derive K_1, K_2 and d, e_1, e_2 and answers $\mathcal{A}_1$'s queries to $\mathcal{F}_{\mathsf{CT}}$ correspondingly.

2. Sim_1 receives from $\mathcal{A}_1$ the PRP values that he submits, and uses K_1 and K_2 to extract the inputs. If he fails to extract these values, Sim_1 submits abort to the TP. Otherwise, Sim_1 submits the extracted values to the trusted party and receives back the set intersection.

3. The simulator verifies that $\mathcal{A}_1$ has submitted exactly t PRP values for each of his inputs. If the verification fails, he instructs the TP to send abort to the P_2.

4. Sim_1 sends to $\mathcal{A}_1$ a commitment of the PRP values from the input set sent by $\mathcal{A}_1$ corresponding to the elements in the set intersection.

5. Sim_1 and $\mathcal{A}_1$ execute the verification where $\mathcal{A}_1$ proves he has submitted exactly t PRP values for each of his inputs. If the verification fails, the simulator aborts the protocol.

6. The simulator opens his commitment and sends the corresponding PRP values to $\mathcal{A}_1$.

The view of $\mathcal{A}_1$ in the simulated and the real executions are identical. In both the real and the simulated execution P_2 receives the correct output if the set submitted by $\mathcal{A}_1$ was formed correctly and otherwise aborts.

□

Claim 18. *The protocol* $\big(\mathcal{A}_S[sh], \mathcal{A}_1[nc_S], \mathcal{A}_2[sh]\big)$*-securely computes the* 2*-party set intersection functionality in the* $\mathcal{F}_{\mathsf{CT}}$*-hybrid model.*

The proof of this claim follows from the previous claim and Lemma 12.

□

The proofs for the two remaining case when P_2 is malicious have analogous proofs.

□

Appendix E

Privacy Enhanced Access Control for Outsourced Data Sharing

E.1 Predicate Encryption and Extensions

In this section we present the construction of predicate encryption of [Katz *et al.*, 2008] and our extension that allows re-randomization of ciphertexts. The scheme $(\{,\}\,\mathsf{Enc}, \mathrm{GenKey}, \mathsf{Dec})$ is defined through the following algorithms:

- $\{(\}\,1^n)$:

  1. Choose primes p, q and r and a groups $\mathbf{G}_p$, $\mathbf{G}_q$ and $\mathbf{G}_r$ with generator g_p, g_q and g_r repectively. Let $\mathbf{G} = \mathbf{G}_p \times \mathbf{G}_q \times \mathbf{G}_r$.
  2. Choose $R_{1,i}, R_{2,i} \in \mathbf{G}_r$, $h_{1,i}, h_{2,i} \in \mathbf{G}_p$ uniformly at random for $1 \leq i \leq n$ and $R_0 \in \mathbf{G}_r$.
  3. The public parameters for the scheme are $(N = pqr, \mathbf{G}, \mathbf{G}_T, e)$. The public key is defined as:

     $$PK = (g_p, g_r, Q = g_q \cdot R_0, \{H_{1,i} = h_{1,i} \cdot R_{1,i}, H_{2,i} = h_{2,i} \cdot R_{2,i}\}_{i=1}^{n}).$$

     The master secret keys is

     $$SK = (p, q, r, g_q, \{h_{1,i}, h_{2,i}\}_{i=1}^{n}).$$

- $\mathsf{Enc}_{\mathsf{SK}}(\mathsf{x}_1, \ldots, \mathsf{x}_n)$:

  1. Choose random $s, \alpha, \beta \in \mathbf{Z}_N$, $R_{3,i}, R_{4,i} \in \mathbf{G}_r$ for $1 \leq i \leq n$.
  2. Output the following ciphertext:

$$C = \big(C_0 = g_p^s, \{C_{1,i} = H_{1,i}^s \cdot Q^{\alpha \cdot x_i} \cdot R_{3,i}, \\ C_{2,i} = H_{2,i}^s \cdot Q^{\beta \cdot x_i} \cdot R_{4,i}\}_{i=1}^n\big).$$

- $\mathrm{GenKey}_{SK}(v_1, \ldots, v_n)$:

  1. Choose random $r_{1,i}, r_{2,i} \in \mathbf{Z}_p$ for $1 \leq i \leq n$, a random $R_5 \in \mathbf{G}_r$, $f_1, f_2 \in \mathbf{Z}_q$ and $Q_6 \in \mathbf{G}_q$.
  2. Output $SK_{\mathbf{v}}$ that consists of

$$\big(K = R_5 \cdot Q_6 \cdot \prod_{i=1}^{n} h_{1,i}^{-r_{1,i}} \cdot h_{2,i}^{-r_{2,i}}, \\ \{K_{1,i} = g_p^{r_{1,i}} \cdot g_q^{f_1 \cdot v_i}, K_{2,i} = g_p^{r_{2,i}} \cdot g_q^{f_2 \cdot v_i}\}_{i=1}^n\big)$$

- $\mathsf{Dec}_{\mathsf{SK_f}}(\mathsf{c})$: The decryption algorithm outputs 1 if and only if

$$e(C_0, K) \prod_{i=1}^{n} e(C_{1,i}, K_{1,i}) \cdot e(C_{2,i}, K_{2,i}) = 1.$$

We further define an algorithm that re-randomizes any ciphertext produced by the predicate encryption as follows:

- $\overset{\$}{\leftarrow}(C)$: The ciphertext is of the form $(C_0, \{C_{1,i}, C_{2,i}\}_{i=1}^n)$. Choose a random $s' \in \mathbf{Z}_N$ and output

$$C' = C_0 \cdot g_p^{s'}, \{C_{1,i} \cdot H_{1,i}^{s'}, C_{2,i} \cdot H_{2,i}^{s'}\}_{i=1}^n.$$

  The resulting ciphertext is the same a freshly generated ciphertext for the encrypted value using random value $s + s'$, if s was the value used in C.

Now we look closely at the instantiation of the predicate encryption scheme that handles polynomial evaluation as its predicate. In this case the predicate $(v_1, \ldots, v_n)$ consists of the coefficients of the polynomial that is being evaluated and the attribute vector that is used for an evaluation point x is of the form $(1, x, x^2, \ldots, x^{n-1})$. The ciphertext for the encryption of $(1, x, x^2, \ldots, x^{n-1})$ has components $(C_0, \{C_{1,i}, C_{2,i}\}_{i=1}^n)$, where $C_{1,i}, C_{2,i}$ correspond to the vector point x^{i-1}. Thus we can view the first few components of the ciphertext $(C_0, \{C_{1,i}, C_{2,i}\}_{i=1}^2)$ as an encryption of the vector $(1, x)$ that can be used for evaluation of predicates that are linear functions.

We use the above observation in the instantiation of the tags that the cloud derives for each of the accepted write updates. He uses the token that the client has used to prove his write access to a particular block, which a predicate encryption ciphertext $(C_0, \{C_{1,i}, C_{2,i}\}_{i=1}^n)$, to derive identifier for the files with which the submitted update will be associated by taking the first part of the ciphertext $(C_0, \{C_{1,i}, C_{2,i}\}_{i=1}^2)$. This identifier cannot be used as a write access token since it is missing substantial part of the ciphertext, and no party without the master secret key can extend an identifier to a valid write token. Also any party that has read access to the file associated with the update will be given a key that would allow it to recognize the updates for that file. This key will be the predicate corresponding to the linear function that evaluates to zero at the file id.

E.2 Optimizations

We discuss several techniques to improve the performance of the two schemes that we presented in the case of multiple accesses to the same files, or as a trade-off for privacy guarantees.

Multiple File Accesses When a user is accessing a file for the first time, he needs to obtain first the access token for the file, which would allow him to obtain from the cloud the access block that contains the file. Once he gets the block, he further needs to obtain the decryption key for the file he is accessing. We can save the derivation time for the tokens and the decryption keys for subsequent accesses to the same file, if the user stores these credential. The credentials will be valid for the next request as long as no user has

been revoked access to the file in the meantime (in which case the access credentials would have been changed). If this has happened, the user will be denied access, and he would try to derive again the credential from the tree in order to obtain the updated tokens and decryption keys if he is still authorized to access the file. This optimization applies to the read and the write access tokens as well as the decryption key for read. The only exception is the encryption key for write access — the user should always derive the current public encryption key for the file, which he wants to update. Unlike the case of the decryption keys for read access, where the client would detect that he does not have a valid key by virtue of the fact that he will not be able to decrypt any of the files in the block, here the user has no way to recognize whether a cached encryption key is still valid.

During a file access request the cloud provider tests each of the access blocks that he is storing against the access token that the user provides in order to find the one that contains the requested files, if the user is authorized. The reason for this is that initially the user does not know in which access block the file is located. However, when the user receives back a block during his first request for a file, he can also get a unique identifier for the block. Thus the next time he needs to access the same file, he can directly point the cloud provider to the relevant block and he will just need to verify that the token is valid for this access block. If this verification fails, the cloud will also check all the other blocks in case the data owner has updated the mapping of the files into blocks.

Communication vs. Access Privacy The schemes that we proposed provide access pattern privacy for the users within each access block (in addition to the anonymity of the credentials). This comes at the cost of transmitting the content of the whole matched block to the user making a read request. However, there might be cases where the user's bandwidth is limited and he cannot afford to receive all the block content, or simply he is not concerned with revealing which part of the block he is accessing. In this case the client can request directly the part of the block he needs. For this we need to enable the cloud to find the requested file in the access block. However, we want to emphasize that this should happen, only if the user specifies that he is ready to reveal his exact access in the block. We should not enable the cloud provider to do this for every request since this way we will

lose the whole point of the access blocks. We can do this through an additional set of read request tokens that are constructed in the same way as predicate encryption ciphertexts with attribute the file id, however, under a different key. Now each file ciphertext would also contain a decryption key that can decrypt only ciphertexts with attribute the identifier for the file. The cloud provider can use these decryption keys to identify the exact file in the block that matches the request. With this extension of the scheme the users will have two sets of credential: access pattern hiding, just identifying a relevant block, and access pattern revealing that point directly to the requested file within the block.

Appendix F

Practical Secure Search

F.1 Key Generation for Our SADS Protocol

The key generation algorithm is not our main focus since general multiparty computation techniques [Yao, 1982; Yao, 1986; Goldreich *et al.*, 1987] can be applied to distribute the appropriate keys. However, we give here an efficient algorithm that allows the sender (S), the receiver (R) and the query router (QR) to obtain their keys. The sender and the receiver choose their keys k_S and k_R respectively. The sender chooses a random number r_S and the receiver chooses a random number r_R; the following messages are exchanged between the three parties using a public key encryption scheme $(\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ in which pk_{QR} and pk_S are public encryption keys for the third party and the sender.

$$\begin{aligned} S \rightarrow QR &: k_S \cdot r_S \\ R \rightarrow QR &: k_R \cdot r_R \\ R \rightarrow S &: r_R \\ S \rightarrow QR &: r_S \cdot r_R^{-1} \end{aligned}$$

At the end of the above message exchange the query router can compute: $k_{QR} = (k_R \cdot r_R) \cdot (k_S \cdot r_S)^{-1} \cdot r_S \cdot r_R^{-1} = (k_R) \cdot (k_S)^{-1}$.

In the above protocol a misbehaving party can cause at most invalid third party key but it cannot learn any secret. Our adversarial model assumes no colluding parties during key

generation. If the query router colludes with the sender or the receiver, they can compute from their keys the key of the non-colluding party. Collusion between sender and receiver is not possible since these parties want to protect their data from each other.

More formally we can construct a simulator $\mathcal{A}_R$ which interacts with the receiver as follows: it receives the values $k_R \cdot r_R$ and r_R that receiver sends. From those $\mathcal{A}_R$ recovers k_R and submits it to the trusted party. This execution is indistinguishable for the receiver from the real execution.

Similarly, we construct a simulator $\mathcal{A}_S$, which interacts with the sender in the following way: it receives the value $k_S \cdot r_S$, chooses and sends a random r_R to S, and then receives $r_S \cdot r_R^{-1}$. Now $\mathcal{A}_S$ can compute the value k_S and submit it to the trusted party.

Part V

Bibliography

Bibliography

[Agrawal *et al.*, 2003] Rakesh Agrawal, Alexandre Evfimievski, and Ramakrishnan Srikant. Information sharing across private databases. In *SIGMOD '03: Proceedings of the 2003 ACM SIGMOD international conference on Management of data*, pages 86–97, New York, NY, USA, 2003. ACM.

[Aiello *et al.*, 2001] William Aiello, Yuval Ishai, and Omer Reingold. Priced oblivious transfer: How to sell digital goods. In *EUROCRYPT '01: Proceedings of the International Conference on the Theory and Application of Cryptographic Techniques*, pages 119–135, London, UK, 2001. Springer-Verlag.

[Ajtai, 2010] Miklós Ajtai. Oblivious rams without cryptogrpahic assumptions. In *STOC '10: Proceedings of the 42nd ACM symposium on Theory of computing*, pages 181–190, New York, NY, USA, 2010. ACM.

[Alwen *et al.*, 2008] J. Alwen, a. Shelat, and I. Visconti. Collusion-free protocols in the mediated model. In *Advances in Cryptology - CRYPTO '08*, pages 497–514, 2008.

[Alwen *et al.*, 2009] J. Alwen, J. Katz, Y. Lindell, G. Persiano, a. Shelat, and I. Visconti. Collusion-free multiparty computation in the mediated model. In *Advances in Cryptology - CRYPTO '09*, pages 524–540, 2009.

[Applebaum *et al.*, 2004] Benny Applebaum, Yuval Ishai, and Eyal Kushilevitz. Cryptography in NC^0. In *Proceedings of the IEEE Symposium on Foundations of Computer Science (FOCS)*, 2004.

[Applebaum *et al.*, 2010] B. Applebaum, Y. Ishai, and E. Kushilevitz. From secrecy to soundness: Efficient verification via secure computation. In *Proceedings of the International Colloquium on Automata, Languages and Programming (ICALP)*, 2010.

[Asharov *et al.*, 2012] Gilad Asharov, Abhishek Jain, Adriana López-Alt, Eran Tromer, Vinod Vaikuntanathan, and Daniel Wichs. Multiparty computation with low communication, computation and interaction via threshold fhe. In *EUROCRYPT*, pages 483–501, 2012.

[Aviv *et al.*, 2007] Adam J. Aviv, Michael E. Locasto, Shaya Potter, and Angelos D. Keromytis. Ssares: Secure searchable automated remote email storage. *Computer Security Applications Conference, Annual*, 0:129–139, 2007.

[Barak and Lindell, 2002] Boaz Barak and Yehuda Lindell. Strict polynomial-time in simulation and extraction. In *STOC '02: Proceedings of the thiry-fourth annual ACM symposium on Theory of computing*, pages 484–493, 2002.

[Barbosa and Farshim, 2011] M. Barbosa and P. Farshim. Delegatable homomorphic encryption with applications to secure outsourcing of computation. Cryptology ePrint Archive, Report 2011/215, 2011.

[Beaver *et al.*, 1990] D. Beaver, S. Micali, and P. Rogaway. The round complexity of secure protocols. In *STOC '90: Proceedings of the twenty-second annual ACM symposium on Theory of computing*, pages 503–513, 1990.

[Beaver, 1992] D. Beaver. Foundations of secure interactive computing. In *Advances in Cryptology – CRYPTO '91*, pages 377–391. Springer-Verlag, 1992.

[Bellare and Rogaway, 1995] Mihir Bellare and Phillip Rogaway. Optimal asymmetric encryption – how to encrypt with rsa. In *Proceedings of EUROCRYPT'94*, 1995.

[Bellovin and Cheswick, 2007] Steven M. Bellovin and William Cheswick. Privacy-enhanced searches using encrypted bloom filters. Technical Report CUCS-034-07, Department of Computer Science, Columbia University, September 2007.

[Ben-David *et al.*, 2008] Assaf Ben-David, Noam Nisan, and Benny Pinkas. Fairplaymp: a system for secure multi-party computation. In *Proceedings of the 15th ACM conference on Computer and communications security*, CCS '08, pages 257–266, 2008.

[Ben-Or *et al.*, 1988] M. Ben-Or, S. Goldwasser, and A. Wigderson. Completeness theorems for non-cryptographic fault-tolerant distributed computation. In *Proceedings of the Twentieth Annual ACM Symposium on Theory of Computing*, 1988.

[Benabbas *et al.*, 2011] Siavosh Benabbas, Rosario Gennaro, and Yevgeniy Vahlis. Verifiable delegation of computation over large datasets. In *Proceedings of CRYPTO*, 2011.

[Bitansky *et al.*, 2011] Nir Bitansky, Ran Canetti, Alessandro Chiesa, and Eran Tromer. From extractable collision resistance to succinct non-interactive arguments of knowledge, and back again. Cryptology ePrint Archive, Report 2011/443, 2011.

[Bloom, 1970] Burton H. Bloom. Space/time trade-offs in hash coding with allowable errors. *Commun. ACM*, 13(7):422–426, 1970.

[Blum *et al.*, 1988] Manuel Blum, Paul Feldman, and Silvio Micali. Non-interactive zero-knowledge and its applications (extended abstract). In *Proceedings of the ACM Symposium on Theory of Computing (STOC)*, 1988.

[Boneh and Franklin, 2003] D. Boneh and M. Franklin. Identity-based encryption from the Weil pairing. *SIAM Journal of Computing*, 32(3):586–615, 2003.

[Boneh and Waters, 2007] Dan Boneh and Brent Waters. Conjunctive, subset, and range queries on encrypted data. In *the Theory of Cryptography Conference (TCC*, pages 535–554. Springer, 2007.

[Boneh *et al.*, 2004] Dan Boneh, Giovanni Di Crescenzo, Rafail Ostrovsky, and Giuseppe Persiano. Public key encryption with keyword search. In *Proceedings of EUROCRYPT'04*, pages 506–522, 2004.

[Boneh *et al.*, 2007] Dan Boneh, Eyal Kushilevitz, Rafail Ostrovsky, and William E. Skeith III. Public key encryption that allows PIR queries. In *Proceedings of CRYPTO'07*, 2007.

[Boneh *et al.*, 2011] Dan Boneh, Amit Sahai, and Brent Waters. Functional encryption: Definitions and challenges. In *Proceedings of the Theory of Cryptography Conference (TCC)*, 2011.

[Boneh, 2001] Dan Boneh. Simplified OAEP for the RSA and Rabin functions. *Lecture Notes in Computer Science*, 2139:275–291, 2001.

[Brakerski and Vaikuntanathan, 2011] Zvika Brakerski and Vinod Vaikuntanathan. Fully homomorphic encryption from ring-lwe and security for key dependent messages. In *CRYPTO*, pages 505–524, 2011.

[Brakerski *et al.*, 2012] Zvika Brakerski, Craig Gentry, and Vinod Vaikuntanathan. (leveled) fully homomorphic encryption without bootstrapping. In *ITCS*, pages 309–325, 2012.

[Brassard *et al.*, 1988] Gilles Brassard, David Chaum, and Claude Crépeau. Minimum disclosure proofs of knowledge. *Journal of Computer and System Sciences*, 37(2), 1988.

[Broder and Mitzenmacher, 2002] Andrei Broder and Michael Mitzenmacher. Network applications of bloom filters: A survey. In *Internet Mathematics*, pages 636–646, 2002.

[Camenisch and Zaverucha, 2009] Jan Camenisch and Gregory Zaverucha. Private intersection of certified sets. In *Proceedings of Financial Cryptography '09*, 2009.

[Canetti, 2000a] Ran Canetti. Security and composition of multiparty cryptographic protocols. *JCrypto*, 13(1):143–202, 2000.

[Canetti, 2000b] Ran Canetti. Security and composition of multiparty cryptographic protocols. *Journal of Cryptology*, 13:2000, 2000.

[Chase, 2007] Melissa Chase. Multi-authority attribute based encryption. In *Proceedings of the IACR Theory of Cryptography Conference (TCC)*, 2007.

[Chaum *et al.*, 1988a] D. Chaum, C. Crépeau, and I. Damgard. Multiparty unconditionally secure protocols. In *ACM symposium on Theory of computing (STOC '88)*, pages 11–19, 1988.

[Chaum *et al.*, 1988b] David Chaum, Claude Crépeau, and Ivan Damgard. Multiparty unconditionally secure protocols. In *STOC '88: Proceedings of the twentieth annual ACM symposium on Theory of computing*, pages 11–19, New York, NY, USA, 1988. ACM.

[cheng Chang and Mitzenmacher, 2005] Yan cheng Chang and Michael Mitzenmacher. Privacy preserving keyword searches on remote encrypted data. In *ACNS*, volume 3531, 2005.

[Cheon *et al.*, 2010] Jung Hee Cheon, Stanislaw Jarecki, and Jae Hong Seo. Multi-party privacy-preserving set intersection with quasi-linear complexity. Cryptology ePrint Archive, Report 2010/512, 2010. `http://eprint.iacr.org/`.

[Choi *et al.*, 2008] S. Choi, D. Dachman-Soled, T. Malkin, and H. Wee. Black-box construction of a non-malleable encryption scheme from any semantically secure one. In *Theory of Cryptography, Fifth Theory of Cryptography Conference, TCC*, 2008.

[Chor *et al.*, 1997] Benny Chor, Niv Gilboa, and Moni Naor. Private information retrieval by keywords. Technical Report TR-CS0917, Dept. of Computer Science, Technion, 1997.

[Chor *et al.*, 1998] Benny Chor, Oded Goldreich, Eyal Kushilevitz, and Madhu Sudan. Private information retrieval. *J. ACM*, 45(6):965–981, 1998.

[Chung *et al.*, 2010] Kai-Min Chung, Yael Kalai, and Salil Vadhan. Improved delegation of computation using fully homomorphic encryption. In *Proceedings of CRYPTO*, 2010.

[Chung *et al.*, 2011] Kai-Min Chung, Yael Kalai, Feng-Hao Liu, and Ran Raz. Memory delegation. In *Proceedings of CRYPTO*, 2011.

[Cramer *et al.*, 1999] Ronald Cramer, Ivan Damgard, Stefan Dziembowski, Martin Hirt, and Tal Rabin. Efficient multiparty computations secure against an adaptive adversary. In *Proceedings of the 17th international conference on Theory and application of cryptographic techniques*, EUROCRYPT'99, pages 311–326, 1999.

[Cramer *et al.*, 2000] Ronald Cramer, Ivan Damgard, and Ueli M. Maurer. General secure multi-party computation from any linear secret-sharing scheme. In *EUROCRYPT*, pages 316–334, 2000.

[Crescenzo *et al.*, 2000] Giovanni Di Crescenzo, Tal Malkin, and Rafail Ostrovsky. Single database private information retrieval implies oblivious transfer. In *EUROCRYPT*, pages 122–138, 2000.

[Cristofaro *et al.*, 2010] Emiliano De Cristofaro, Jihye Kim, and Gene Tsudik. Linear-complexity private set intersection protocols secure in malicious model. In *ASIACRYPT*, pages 213–231, 2010.

[Curtmola *et al.*, 2006] Reza Curtmola, Juan Garay, Seny Kamara, and Rafail Ostrovsky. Searchable symmetric encryption: improved definitions and efficient constructions. In *CCS '06: Proceedings of the 13th ACM conference on Computer and communications security*, pages 79–88, New York, NY, USA, 2006. ACM.

[Dachman-Soled *et al.*, 2009] Dana Dachman-Soled, Tal Malkin, Mariana Raykova, and Moti Yung. Efficient robust private set intersection. In *ACNS*, pages 125–142, 2009.

[Dachman-Soled *et al.*, 2011] Dana Dachman-Soled, Tal Malkin, Mariana Raykova, and Moti Yung. Secure efficient multiparty computing of multivariate polynomials and applications. In *Proceedings of the 9th international conference on Applied cryptography and network security*, ACNS'11, pages 130–146, 2011.

[Damgard and Ishai, 2005] Ivan Damgard and Yuval Ishai. Constant-round multiparty computation using a black-box pseudorandom generator. In *In CRYPTO 2005*, pages 378–394, 2005.

[Damgard and Ishai, 2006] Ivan Damgard and Yuval Ishai. Scalable secure multiparty computation. In *CRYPTO*, pages 501–520, 2006.

[Damgard and Orlandi, 2010] Ivan Damgard and Claudio Orlandi. Multiparty computation for dishonest majority: From passive to active security at low cost. In *CRYPTO*, pages 558–576, 2010.

[Damgard *et al.*, 2006] Ivan Damgard, Kasper Dupont, and Michael Pedersen. Unclonable group identification. In *Advances in Cryptology - EUROCRYPT 2006*, volume 4004, pages 555–572, 2006.

[Damgard *et al.*, 2010] Ivan Damgard, Sigurd Meldgaard, and Jesper Buus Nielsen. Perfectly secure oblivious ram without random oracles. Cryptology ePrint Archive, Report 2010/108, 2010. http://eprint.iacr.org/.

[Dawn and Song, 2005] Lea Kissner Dawn and Dawn Song. Privacy-preserving set operations. In *in Advances in Cryptology - CRYPTO 2005, LNCS*, pages 241–257. Springer, 2005.

[De Capitani di Vimercati *et al.*, 2011] S. De Capitani di Vimercati, S. Foresti, S. Paraboschi, G. Pelosi, and P. Samarati. Efficient and private access to outsourced data. In *Proc. of the 31st International Conference on Distributed Computing Systems (ICDCS 2011)*, Minneapolis, Minnesota, USA, June 2011.

[De Cristofaro *et al.*, 2009] Emiliano De Cristofaro, Stanislaw Jarecki, Jihye Kim, and Gene Tsudik. Privacy-preserving policy-based information transfer. In *Proceedings of PETS*, 2009.

[Dodis *et al.*, 2006] Yevgeniy Dodis, Aleksandr Yampolskiy, and Moti Yung. Threshold and proactive pseudo-random permutations. In *TCC*, pages 542–560, 2006.

[ElGamal, 1985] Taher ElGamal. A public key cryptosystem and a signature scheme based on discrete logarithms. In *Proceedings of CRYPTO 84 on Advances in cryptology*, pages 10–18, New York, NY, USA, 1985. Springer-Verlag New York, Inc.

[Feige *et al.*, 1994] Uri Feige, Joe Killian, and Moni Naor. A minimal model for secure computation (extended abstract). In *ACM symposium on Theory of Computing (STOC '94)*, pages 554–563, 1994.

[Feldman, 1987] Paul Feldman. A practical scheme for non-interactive verifiable secret sharing. In *FOCS*, pages 427–437, New York, NY, USA, 1987. ACM.

[Fortnow and Lund, 1991] Lance Fortnow and Carsten Lund. Interactive proof systems and alternating time-space complexity. In *STACS '91: Selected papers of the 8th annual symposium on Theoretical aspects of computer science*, pages 55–73, 1991.

[Fouque *et al.*, 2001] Pierre-Alain Fouque, Guillaume Poupard, and Jacques Stern. Sharing decryption in the context of voting or lotteries. In *FC '00: Proceedings of the 4th International Conference on Financial Cryptography*, pages 90–104, London, UK, 2001. Springer-Verlag.

[Franklin and Mohassel, 2010] Matthew Franklin and Payman Mohassel. Efficient and secure evaluation of multivariate polynomials and applications. In *Applied Cryptography and Network Security*, volume 6123, pages 236–254, 2010.

[Franklin and Yung, 1992] Matthew Franklin and Moti Yung. Communication complexity of secure computation (extended abstract). In *STOC '92: Proceedings of the twenty-fourth annual ACM symposium on Theory of computing*, pages 699–710, 1992.

[Freedman *et al.*, 2004] Michael Freedman, Kobbi Nissim, and Benny Pinkas. Efficient private matching and set intersection. In *Proceedings of EUROCRYPT'04*, 2004.

[Freedman *et al.*, 2005] Michael J. Freedman, Yuval Ishai, Benny Pinkas, and Omer Reingold. Keyword search and oblivious pseudorandom functions. In *Theory of Cryptography Conference (TCC 05*, pages 303–324, 2005.

[Fujisaki *et al.*, 2004] Eiichiro Fujisaki, Tatsuaki Okamoto, David Pointcheval, and Jacques Stern. Rsa-oaep is secure under the rsa assumption. *J. Cryptol.*, 17(2):81–104, 2004.

[Gennaro *et al.*, 2007] Rosario Gennaro, Stanislaw Jarecki, Hugo Krawczyk, and Tal Rabin. Secure distributed key generation for discrete-log based cryptosystems. *J. Cryptol.*, 20(1):51–83, 2007.

[Gennaro *et al.*, 2010] Rosario Gennaro, Craig Gentry, and Bryan Parno. Non-interactive verifiable computation: Outsourcing computation to untrusted workers. In *Proceedings of CRYPTO*, 2010.

[Gentry and Ramzan, 2005] Craig Gentry and Zulfikar Ramzan. Single-database private information retrieval with constant communication rate. In *Proceedings of the 32nd International Colloquium on Automata, Languages and Programming*, 2005.

[Gentry, 2009] C. Gentry. Fully homomorphic encryption using ideal lattices. In *ACM Symposium on Theory of Computing (STOC '09)*, pages 169–178. ACM Press, 2009.

[Gertner *et al.*, 2000] Y. Gertner, Y. Ishai, E. Kushilevitz, and T. Malkin. Protecting data privacy in private information retrieval schemes. *Journal of Computer and System Sciences*, 60(3):592–629, 2000.

[Gjøsteen, 2008] Kristian Gjøsteen. A latency-free election scheme. In *CT-RSA'08: Proceedings of the 2008 The Cryptopgraphers' Track at the RSA conference on Topics in cryptology*, pages 425–436, 2008.

[Goh, 2004] Eu-Jin Goh. Secure indexes. Cryptology ePrint Archive, Report 2003/216, 2004. http://eprint.iacr.org/2003/216/.

[Goldreich and Kahan, 1996] Oded Goldreich and Ariel Kahan. How to construct constant-round zero-knowledge proof systems for np. *Journal of Cryptology*, 9:167–189, 1996.

[Goldreich and Levin, 1989] Oded Goldreich and L.A. Levin. A hard-core predicate for all one-way functions. In *Proceedings of the ACM Symposium on Theory of Computing (STOC)*, 1989.

[Goldreich and Ostrovsky, 1996] O. Goldreich and R. Ostrovsky. Software protection and simulation on oblivious RAMs. *Journal of the ACM (JACM)*, 43(3):473, 1996.

[Goldreich *et al.*, 1987] O. Goldreich, S. Micali, and A. Wigderson. How to play any mental game. In *Proceedings of the nineteenth annual ACM symposium on Theory of computing*, pages 218–229. ACM, 1987.

[Goldreich, 2001] Oded Goldreich. *Foundations of Cryptography. Volume I: Basic Tools.* Cambridge University Press, Cambridge, England, 2001.

[Goldreich, 2004] O. Goldreich. *The Foundations of Cryptography – Volume 2.* Cambridge University Press, 2004.

[Goldreich, 2005] Oded Goldreich. Foundations of cryptography: a primer. *Found. Trends Theor. Comput. Sci.*, 1(1):1–116, 2005.

[Goldwasser and Levin, 1991] S. Goldwasser and L. Levin. Fair computation of general functions in presence of immoral majority. In *Advances in Cryptology – CRYPTO '90*, pages 77–93. Springer-Verlag, 1991.

[Goldwasser and Lindell, 2002] S. Goldwasser and Y. Lindell. Secure computation without agreement. In *International Conference on Distributed Computing (DISC '02)*, pages 17–32. Springer-Verlag, 2002.

[Goldwasser and Micali, 1982] Shafi Goldwasser and Silvio Micali. Probabilistic encryption and how to play mental poker keeping secret all partial information. In *STOC '82: Proceedings of the fourteenth annual ACM symposium on Theory of computing*, pages 365–377, New York, NY, USA, 1982. ACM.

[Goldwasser *et al.*, 2008] S. Goldwasser, Y. Kalai, and G. Rothblum. Delegating computation: interactive proofs for muggles. In *Proceedings of the 40th annual ACM symposium on Theory of computing (STOC '08)*, pages 113–122, New York, NY, USA, 2008.

[Goldwasser *et al.*, 2011] Shafi Goldwasser, Huijia Lin, and Aviad Rubinstein. Delegation of computation without rejection problem from designated verifier CS-proofs. Cryptology ePrint Archive, Report 2011/456, 2011.

[Goodrich and Mitzenmacher, 2011] Michael T. Goodrich and Michael Mitzenmacher. Privacy-preserving access of outsourced data via oblivious ram simulation. In *ICALP (2)*, pages 576–587, 2011.

[Goodrich *et al.*, 2011] Michael T. Goodrich, Michael Mitzenmacher, Olga Ohrimenko, and Roberto Tamassia. Privacy-preserving group data access via stateless oblivious ram simulation. *CoRR*, abs/1105.4125, 2011.

[Gordon *et al.*, 2011] Dov Gordon, Jonathan Katz, Vladimir Kolesnikov, Tal Malkin, Mariana Raykova, and Yevgeniy Vahlis. Secure computation with sublinear amortized work. In submission, http://eprint.iacr.org/2011/482, 2011.

[Gordon *et al.*, 2012] Dov Gordon, Jonathan Katz, Vladimir Kolesnikov, Fernando Krell, Tal Malkin, Mariana Raykova, and Yevgeniy Vahlis. Secure two-party computation in

sublinear amortized time. In *Proceedings of the 19th ACM Conference on Computer and Communications Security*, CCS '12, 2012.

[Goyal *et al.*, 2006] Vipul Goyal, Omkant Pandey, Amit Sahai, and Brent Waters. Attribute-based encryption for fine-grained access control of encrypted data. In *Proceedings of the ACM Conference on Computer and Communications Security (CCS)*, 2006.

[Green *et al.*, 2011] Matthew Green, Susan Hohenberger, and Brent Waters. Outsourcing the decryption of ABE ciphertexts. In *Proceedings of the USENIX Security Symposium*, 2011.

[Hazay and Lindell, 2008] Carmit Hazay and Yehuda Lindell. Efficient protocols for set intersection and pattern matching with security against malicious and covert adversaries. In *TCC*, pages 155–175, 2008.

[Hazay and Nissim, 2010] Carmit Hazay and Kobbi Nissim. Efficient set operations in the presence of malicious adversaries. In *Public Key Cryptography PKC 2010*, pages 312–331, 2010.

[Henecka *et al.*, 2010] Wilko Henecka, Stefan K ögl, Ahmad-Reza Sadeghi, Thomas Schneider, and Immo Wehrenberg. Tasty: tool for automating secure two-party computations. In *Proceedings of the 17th ACM conference on Computer and communications security*, CCS '10, pages 451–462, 2010.

[Huang *et al.*, 2011] Yan Huang, David Evans, Jonathan Katz, and Lior Malka. Faster secure two-party computation using garbled circuits. In *Proceedings of the 20th USENIX Security Symposium*, Usenix '11, 2011.

[Huang *et al.*, 2012] Yan Huang, David Evans, and Jonathan Katz. Quid pro quo-tocols: Strengthening semi-honest protocols with dual execution. In *Proceedings of the 33rd IEEE Symposium on Security and Privacy*, 2012.

[Hubbell *et al.*, 1988] F. Allan Hubbell, Elizabeth B. Frye, Barbara V. Akin, and Lloyd Rucker. Routine admission laboratory testing for general medical patients. *Medical Care*, 26(6), 1988.

[Ishai *et al.*, 2008] Yuval Ishai, Manoj Prabhakaran, and Amit Sahai. Founding cryptography on oblivious transfer — efficiently. In *CRYPTO 2008: Proceedings of the 28th Annual conference on Cryptology*, pages 572–591, 2008.

[Ishai *et al.*, 2009] Yuval Ishai, Manoj Prabhakaran, and Amit Sahai. Secure arithmetic computation with no honest majority. In *TCC '09: Proceedings of the 6th Theory of Cryptography Conference on Theory of Cryptography*, pages 294–314, 2009.

[Jarecki and Liu, 2009] Stanislaw Jarecki and Xiaomin Liu. Efficient oblivious pseudorandom function with applications to adaptive ot and secure computation of set intersection. In *TCC*, pages 577–594, 2009.

[Kamara *et al.*, 2011] Seny Kamara, Payman Mohassel, and Mariana Raykova. Outsourcing multi-party computation. In submission, 2011.

[Karatsuba and Ofman, 1962] A. Karatsuba and Y. Ofman. Multiplication of many-digital numbers by automatic computers. *Proceedings of the SSSR Academy of Sciences*, 145:293–294, 1962.

[Katz *et al.*, 2008] Jonathan Katz, Amit Sahai, and Brent Waters. Predicate encryption supporting disjunctions, polynomial equations, and inner products. In *Proceedings of EuroCrypt*, 2008.

[Kilian, 1992] Joe Kilian. A note on efficient zero-knowledge proofs and arguments (extended abstract). In *Proceedings of the ACM Symposium on Theory of Computing (STOC)*, 1992.

[Kilian, 1995] Joe Kilian. Improved efficient arguments (preliminary version). In *Proceedings of CRYPTO*, 1995.

[Kissner and Song, 2005] Lea Kissner and Dawn Xiaodong Song. Privacy-preserving set operations. In *CRYPTO*, pages 241–257, 2005.

[Kushilevitz *et al.*, 2011] Eyal Kushilevitz, Steve Lu, and Rafail Ostrovsky. On the (in)security of hash-based oblivious ram and a new balancing scheme. Cryptology ePrint Archive, Report 2011/327, 2011. `http://eprint.iacr.org/2011/327`.

[Kushilevitz *et al.*, 2012] Eyal Kushilevitz, Steve Lu, and Rafail Ostrovsky. On the (in)security of hash-based oblivious RAM and a new balancing scheme. In *SODA'12*, SODA '12, pages 143–156, 2012.

[Lepinksi *et al.*, 2005] Matt Lepinksi, Silvio Micali, and abhi shelat. Collusion-free protocols. In *Proceedings of the thirty-seventh annual ACM symposium on Theory of computing*, STOC '05, pages 543–552, 2005.

[Lewko *et al.*, 2010] Allison Lewko, Tatsuaki Okamoto, Amit Sahai, Katsuyuki Takashima, and Brent Waters. Fully secure functional encryption: Attribute-based encryption and (hierarchical) inner product encryption. In *Proceedings of EuroCrypt*, 2010.

[Lindell and Pinkas, 2007] Yehuda Lindell and Benny Pinkas. An efficient protocol for secure two-party computation in the presence of malicious adversaries. In *EUROCRYPT*, pages 52–78, 2007.

[Lindell, 2001] Yehuda Lindell. Parallel coin-tossing and constant-round secure two-party computation. In *Journal of Cryptology*, pages 171–189, 2001.

[Lindell, 2008] Andrew Y. Lindell. Efficient fully-simulatable oblivious transfer. In *CT-RSA*, pages 52–70, 2008.

[Lund *et al.*, 1992] Carsten Lund, Lance Fortnow, and Howard Karloff. Algebraic methods for interactive proof systems. *J. ACM*, 39(4):859–868, 1992.

[M. Bellare and O'Neill, 2007] A. Boldyareva M. Bellare and A. O'Neill. Deterministic and efficiently searchable encryption. In *Proceedings of CRYPTO'07*, 2007.

[Malkhi *et al.*, 2004] Dahlia Malkhi, Noam Nisan, Benny Pinkas, and Yaron Sella. Fairplay a secure two-party computation system. In *In Proc. 13th USENIX Security Symposium*, pages 287–302, 2004.

[Mansour *et al.*, 1993] Yishay Mansour, Noam Nisan, and Prasoon Tiwari. The computational complexity of universal hashing. *Theor. Comput. Sci.*, 107:121–133, January 1993.

[Micali and Rogaway, 1992] S. Micali and P. Rogaway. Secure computation (abstract). In *Advances in Cryptology - CRYPTO '91*, pages 392–404. Springer-Verlag, 1992.

[Micali, 1994] Silvio Micali. CS proofs (extended abstract). In *Proceedings of the IEEE Symposium on Foundations of Computer Science (FOCS)*, 1994.

[Mitzenmacher and Upfal, 2005] Michael Mitzenmacher and Eli Upfal. *Probability and computing - randomized algorithms and probabilistic analysis*. Cambridge University Press, 2005.

[Mohassel and Franklin, 2006] P. Mohassel and M. Franklin. Efficiency tradeoffs for malicious two-party computation. In *Conference on Theory and Practice of Public-Key Cryptography (PKC '06)*, volume 3958 of *Lecture Notes in Computer Science*, pages 458–473. Springer, 2006.

[Myers *et al.*, 2011] Steven Myers, Mona Sergi, and abhi shelat. Threshold fully homomorphic encryption and secure computation. Cryptology ePrint Archive, Report 2011/454, 2011. `http://eprint.iacr.org/`.

[Naor and Pinkas, 2001] Moni Naor and Benny Pinkas. Efficient oblivious transfer protocols. In *SODA '01: Proceedings of the Twelfth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 448–457, Philadelphia, PA, USA, 2001. Society for Industrial and Applied Mathematics.

[Naor and Pinkas, 2006] Moni Naor and Benny Pinkas. Oblivious polynomial evaluation. *SIAM J. Comput.*, 35(5):1254–1281, 2006.

[Naor and Reingold, 2004] Moni Naor and Omer Reingold. Number-theoretic constructions of efficient pseudo-random functions. *J. ACM*, 51:231–262, March 2004.

[Olumofin and Goldberg, 2011] Femi Olumofin and Ian Goldberg. Revisiting the computational practicality of private information retrieval. In *Financial Cryptography*, 2011.

[Ostrovsky and Shoup, 1997] Rafail Ostrovsky and Victor Shoup. Private information storage (extended abstract). In *STOC*, pages 294–303, 1997.

[Ostrovsky *et al.*, 2007] Rafail Ostrovsky, Amit Sahai, and Brent Waters. Attribute-based encryption with non-monotonic access structures. In *Proceedings of the ACM Conference on Computer and Communications Security (CCS)*, 2007.

[Paillier, 1999] Pascal Paillier. Public-key cryptosystems based on composite degree residuosity classes. In *In Proceedings of EUROCRYPT 1999*, pages 223–238. Springer-Verlag, 1999.

[Papamanthou *et al.*, 2011] Charalampos Papamanthou, Roberto Tamassia, and Nikos Triandopoulos. Optimal verification of operations on dynamic sets. In *Proceedings of CRYPTO*, 2011.

[Pappas *et al.*, 2011] Vasilis Pappas, Mariana Raykova, Binh Vo, Steven Bellovin, and Tal Malkin. Private search in the real world. In *Annual Computer Security Applications Conference (ACSAC)*, 2011.

[Parno *et al.*, 2012] Bryan Parno, Mariana Raykova, and Vinod Vaikuntanathan. How to delegate and verify in public: Verifiable computation from attribute-based encryption. In *Theory of Cryptography Conference (TCC), to appear*, 2012.

[Patra *et al.*, 2009a] Arpita Patra, Ashish Choudhary, and C. Rangan. Information theoretically secure multi party set intersection re-visited. In Michael Jacobson, Vincent Rijmen, and Reihaneh Safavi-Naini, editors, *Selected Areas in Cryptography*, volume 5867 of *Lecture Notes in Computer Science*, pages 71–91. Springer Berlin / Heidelberg, 2009.

[Patra *et al.*, 2009b] Arpita Patra, Ashish Choudhary, and C. Pandu Rangan. Round efficient unconditionally secure mpc and multiparty set intersection with optimal resilience. In *Proceedings of the 10th International Conference on Cryptology in India: Progress in Cryptology*, INDOCRYPT '09, pages 398–417, 2009.

[Pinkas and Reinman, 2010] B. Pinkas and T. Reinman. Oblivious RAM Revisited. *Advances in Cryptology–CRYPTO 2010*, pages 502–519, 2010.

[Pinkas *et al.*, 2009] Benny Pinkas, Thomas Schneider, Nigel P. Smart, and Stephen C. Williams. Secure two-party computation is practical. In *Proceedings of the 15th International Conference on the Theory and Application of Cryptology and Information Security: Advances in Cryptology*, ASIACRYPT '09, pages 250–267, 2009.

[Pohlig and Hellman, 1978] Stephen Pohlig and Martin Hellman. An improved algorithm for computing logarithms over GF(p)and its cryptographic significance. *IEEE Transactions on Information Theory*, 24(1):106–110, 1978.

[Raykova *et al.*, 2009] Mariana Raykova, Binh Vo, Steven Bellovin, and Tal Malkin. Secure anonymous database search. In *CCSW 2009.*, 2009.

[Raykova *et al.*, 2012] Mariana Raykova, Hang Zhao, and Steven Bellovin. Privacy enhanced access control for outsourced data sharing. In *Financial Crypto, to appear*, 2012.

[Reynolds and Vahdat, 2003] Patrick Reynolds and Amin Vahdat. Efficient peer-to-peer keyword searching. In *Middleware 2003*, pages 21–40, 2003.

[Sahai and Seyalioglu, 2010] Amit Sahai and Hakan Seyalioglu. Worry-free encryption: functional encryption with public keys. In *Proceedings of the ACM Conference on Computer and Communications Security (CCS)*, 2010.

[Sahai and Waters, 2005] Amit Sahai and Brent Waters. Fuzzy identity-based encryption. In *Proceedings of EuroCrypt*, 2005.

[Sang and Shen, 2009] Yingpeng Sang and Hong Shen. Efficient and secure protocols for privacy-preserving set operations. *ACM Trans. Inf. Syst. Secur.*, 13:9:1–9:35, November 2009.

[Shamir, 1979] Adi Shamir. How to share a secret. *Commun. ACM*, 22(11):612–613, 1979.

[Shamir, 1992] Adi Shamir. Ip = pspace. *J. ACM*, 39(4):869–877, 1992.

[Shetty and Adibi, 2004] Jitesh Shetty and Jafar Adibi. The enron email dataset database schema and brief statistical report. Technical report, USC, 2004.

[Shi *et al.*, 2007] Elaine Shi, John Bethencourt, T-H. Hubert Chan, Dawn Song, and Adrian Perrig. Multi-dimensional range query over encrypted data. In *SP '07: Proceedings of the 2007 IEEE Symposium on Security and Privacy*, pages 350–364, Washington, DC, USA, 2007. IEEE Computer Society.

[Shi *et al.*, 2011] Elaine Shi, T.-H. Hubert Chan, Emil Stefanov, and Mingfei Li. Oblivious RAM with $o((\log n)^3)$ worst-case cost. In *ASIACRYPT*, pages 197–214, 2011.

[Song *et al.*, 2000] Dawn Xiaodong Song, David Wagner, and Adrian Perrig. Practical techniques for searches on encrypted data. In *SP '00: Proceedings of the 2000 IEEE Symposium on Security and Privacy*, page 44, Washington, DC, USA, 2000. IEEE Computer Society.

[Talbot and Osborne, 2007] David Talbot and Miles Osborne. Randomised language modelling for statistical machine translation. In *In Proceeding of 45th Annual Meeting of the Association for Computational Linguistics*, pages 512–519, 2007.

[Vimercati *et al.*, 2010] S. De Capitani di Vimercati, S. Foresti, S. Jajodia, S. Paraboschi, G. Pelosi, and P. Samarati. Encryption-based policy enforcement for cloud storage. In *Proceedings of the 2010 IEEE 30th International Conference on Distributed Computing Systems Workshops*, ICDCSW '10, pages 42–51, Washington, DC, USA, 2010. IEEE Computer Society.

[Waters *et al.*, 2004] B. Waters, D. Balfanz, G. Durfee, and D. Smetters. Building an encrypted and searchable audit log. In *NDSS 2004.*, 2004.

[Williams and Sion, 2008] Peter Williams and Radu Sion. Usable PIR. In *NDSS*, 2008.

[Williams *et al.*, 2008] Peter Williams, Radu Sion, and Bogdan Carbunar. Building castles out of mud: practical access pattern privacy and correctness on untrusted storage. In *CCS '08: Proceedings of the 15th ACM conference on Computer and communications security*, pages 139–148, New York, NY, USA, 2008. ACM.

[Yao, 1982] Andrew Chi-Chih Yao. Protocols for secure computations. In *FOCS*, pages 160–164, 1982.

[Yao, 1986] Andrew Yao. How to generate and exchange secrets. In *Proceedings of the IEEE Symposium on Foundations of Computer Science*, 1986.

[Yu *et al.*, 2010] Shucheng Yu, Cong Wang, Kui Ren, and Wenjing Lou. Achieving secure, scalable, and fine-grained data access control in cloud computing. In *Proceedings of the 29th conference on Information communications*, INFOCOM'10, pages 534–542, Piscataway, NJ, USA, 2010. IEEE Press.

[Zobel and Moffat, 1998] Justin Zobel and Alistair Moffat. Inverted files versus signature files for text indexing. *ACM Transactions on Database Systems*, 23:453–490, 1998.