The Efficient Dual Receiver Cryptosystem and Its Applications

Ted Diament, Homin K. Lee, Angelos D. Keromytis, and Moti Yung
(Corresponding author: Angelos D. Keromytis)

Department of Computer Science, Columbia University
1214 Amsterdam Avenue, M.C. 0401, New York, NY 10027, USA (Email: angelos@cs.columbia.edu)

(Received Mar. 10, 2010; revised and accepted June 15, 2010)

Abstract

We put forth the notion of efficient dual receiver cryptosystems and implement it based on bilinear pairings over certain elliptic curve groups. The cryptosystem is simple and efficient yet powerful, as it helps to solve two problems of practical importance whose solutions had proven to be elusive until now: (1) A provably secure "combined" public-key cryptosystem (with a single secret key per user) where the key is used for both decryption and signing and where encryption can be escrowed and recovered, while the signature capability never leaves its owner. This is an open problem proposed by the work of Haber and Pinkas. (2) A puzzle is a method for rate-limiting remote users by forcing them to solve a computational task (the puzzle). Puzzles have been based on cryptographic challenges in the past, but the successful design of embedding a useful cryptographic task inside a puzzle, originally posed by Dwork and Naor, has remained problematic. We model and present "useful security puzzles" applicable as an online transaction server (such as a Web server).

Keywords: Denial of service, pairing-based cryptography, puzzles, useful secure computation

1 Introduction

We introduce the notion of an efficient dual receiver cryptosystem, which enables a ciphertext to be decrypted by two independent receivers. To implement a dual receiver cryptosystem, one may use the methodology suggested by Naor and Yung [44], by using the first receiver's key and the second receiver's key and encrypting the same plaintext with both. This, however, makes the ciphertext (which should also include a proof of consistent encryption) inefficient. Even in a practice-oriented adaptation of the dual ciphertext methodology [18], such a multi-cryptosystem scheme is not an efficient solution.

To achieve a practical system, we build on any bilinear map between two groups, and in particular, we use a pairing defined on certain elliptic curves. Several papers have used pairings to construct cryptosystems, and these designs inspired our construction's components. The first and most basic design is the three-party one-round Diffie-Hellman key exchange proposed by Joux [33]. A particularly elegant and surprising cryptosystem is the identity-based encryption scheme proposed by Boneh and Franklin [7], in which the public key is a user's identity and a key generation authority assigns the user a private key. Hierarchical identity-based systems were given by Gentry and Silverberg [24], and Boneh, Lynn, and Shacham [8] used pairings to generate short signatures. Various other constructions have been proposed.

Our basic design is a cryptosystem that transforms the three-party one-round Diffie-Hellman key exchange proposed by Joux [33] into a dual receiver public key cryptosystem. This is analogous to the construction of the El Gamal encryption from the Diffie-Hellman key exchange protocol, and is therefore simple and efficient. We show that this simple construction is quite powerful by demonstrating how it solves two open issues in the literature: how to construct public key cryptosystems that support encryption and signature generation with a single private key per user (called "combined cryptosystems"), and how to construct "useful" computational puzzles secure under certain assumptions.

We begin with an overview of these two problems before we explain the details of the dual receiver cryptosystem. We present the dual receiver cryptosystem in Section 4. Section 5 discusses a combined public key cryptosystem, and we present our client-generated puzzles in Section 6. We conclude in Section 7.

2 Combined Cryptosystems

Public key systems support both encryption and signature generation with a single private key portion per user. Haber and Pinkas [29] model a public key in a constrained environment that can afford only a single key per user. The user is required to both sign and decrypt with this single secret key. (This design may arise in constrained environments, or as a flexible design tool in cases where a system is specified to support only signatures but with the outlook that an encryption requirement may be added later after the design is finished). Various systems, *e.g.,* PGP, used combined schemes before, but without any formal proof of security. Coron *et al.* [13] showed how RSA with padding, originally designed for secure signatures by Bellare and Rogaway [10], actually remains secure when using the same padding in a combined system. Similar work was done by Komano and Ohta [38].

The use of a single key for both encryption and signature generation is a problem in certain applications where the encrypted information must be recoverable by a third party (*e.g.,* a security officer). This is true for medical and financial records where privacy and secrecy are becoming mandatory (*e.g.,* the Health Insurance Portability and Accountability Act (HIPAA)), yet the necessity for emergency access remains. However, no authority, and in fact no one but the user, should have the capability to sign the user's name in order to preserve non-repudiation. This issue has been discussed in the literature [19], which recommended that different tasks use separate keys due to escrow.

Ideally, the same encryption system should be able to create recoverable ciphertexts for official use and non-recoverable ciphertexts for private use. Even better would be a securely combined encryption and signature system that uses the same private key for both functions, allowing each installation of the system to adopt a "key recovery policy" in a flexible way. The problem of having a *single private key per user* together with simultaneous escrow of the decryption capability and non-escrow of the signing capability has been considered self-contradictory and perhaps somewhat paradoxical, and was one of the earliest criticism raised against combined cryptosystems when they were first suggested [47].

Our work answers this open problem. We propose a combined public key system composed of an encryption scheme secure against chosen-ciphertext attacks in which the encrypted message is either non-recoverable or recoverable by parties that can be chosen differently for each message, and a signature scheme secure against adaptive chosen-message attacks (including attacks by the escrow agent). Both signatures and ciphertexts that use the same secret key consist of a single cryptosystem. Furthermore, a sender can escrow her message to a party of the sender's choosing without this party being involved in any pre-processing. Voluntary escrow can also be useful for encrypted storage. (Many file and disk encryption products, *e.g.,* Windows EFS, offer key escrow capability.) This achieves full flexibility in key management and recovery policy (and the recovering party is only involved in the recovery). Obviously, we do not escrow keys directly in our design but employ the dual receiver cryptosystem method.

As noted by Boneh and Franklin, key escrow is inherent in identity-based schemes, since the key generation authority knows all the users' private keys. Verheul [53] had suggested that the user have two keys, one that is escrowed to a designated recovery agent who is involved in key generation and a second one that is not escrowed and can also be used for signing. However, this scheme does not achieve our goals of allowing a single key per user or per server with careful modeling and security proofs. In fact, we are not aware of any prior work that has solved the separation of key management of combined cryptosystems.

2.1 Random Oracles

The security model used throughout the paper is that of the random oracle model, which has been employed in constructions based on pairings such as those proposed by Boneh and Franklin [7], and in many efficient constructions for chosen ciphertext secure encryption, *e.g.,* [45]. A random oracle is a function $H : X \to Y$ chosen uniformly at random from the set of all functions from X to Y, Y finite. An algorithm can query the random oracle for any $x \in X$ and receive $H(x) \in Y$ in response. Random oracles are an idealized model for cryptographic hash functions, and thus security proofs in this model only prove security against attackers that are confined to this model as well. Nevertheless, in many recent designs that employ hash functions as a black box, this design approach followed by a proof in the random oracle model gives a certain validation to the strength of the system design methodology. (Koblitz and Menezes have provided a defense of the model [36].)

3 Useful Puzzles

Computational puzzles have been proposed as a means to protect servers from resource-depletion attacks [17, 31, 54], such as TCP SYN floods [51]. The basic idea is to require every client to perform some computationally expensive but easily verifiable (by the server) computation. Attackers issuing large numbers of concurrent requests will need considerable amounts of computational resources, making such attacks difficult to mount, while low-rate legitimate clients will not be severely affected.

Puzzle schemes typically use a one-way function (OWF) to construct a hard computational challenge with a shortcut: knowledge of an input to the OWF allows the server to efficiently compute the result, while a client presented with a result must exhaustively search (brute force) through the space of potential inputs (applying the OWF to each) to determine the correct value. Thus, a server can ask the client a question similar to *"which 32-bit number, when supplied as input to the SHA1 OWF, results in the value 0xdeadbeef"?* The server can pick the input value at random, and may vary its size to reflect the computational resources of the clients and attackers.

However, solving a puzzle represents "useless" computation in the sense that, other than rate-limiting requests, it serves no other purpose. One can imagine a server that uses the clients that request some service to solve a useful problem. Such a "useful puzzle" must have specific properties:

Computational intensity. The puzzle should be a moderate but serious computational task, assuring a certain slow-down of the accessing party (client).

Reliability. It should be computationally efficient for the challenger (the server) to verify the result of the puzzle (much easier to check than to compute).

Usefulness. The result of the computation should be useful to the server.

Non-dependability. If the puzzle is not actually solved by the client, the server should still be able to solve it (or give it to another client to solve).

Security. The client must not learn the result of the computation (if it is considered sensitive), any long-term cryptographic keys, or any other secrets of the server.

The question is, how can a puzzle be useful, reliable and secure? If it is to be reliable, it means the challenger has to repeat the work of the accessing party in order to verify, thus hurting the notion of "usefulness." If it is to be useful, the accessing party needs some secret information that is required to do the useful work, which may result in giving away the *security* of the challenger (who needs to provide trapdoor information or other secrets that are easier to check). The only computational puzzle to meet these requirements thus far has been one that creates e-coins for the MicroMint scheme of Rivest and Shamir [32]. In this paper, we give two partial answers to the question on the existence of useful puzzles, first posed by Dwork and Naor [16].

At the heart of the solution is the dual receiver cryptosystem. We exploit a cryptosystem that effectively has two receivers: the challenger, who needs to have a puzzle solved as part of a protocol, and a second receiver that is someone who has a different (perhaps temporary) trapdoor that the challenger can provide him with. In our scheme, the second receiver is another client that is contacting the same challenger (server) and is requesting some service in the same protocol. Getting the trapdoor and doing the useful work is only part of the decryption, since the accessing party should not learn the plaintext, and the result should be easily checkable. What we exploit are padding schemes for chosen-ciphertext secure schemes that separate the trapdoor action from the part that involves checking the integrity of computation and the extraction of plaintexts. The combination of the dual receiver encryption and the separability of the padding scheme enables the useful security puzzles under certain threat models. We show an example whereby the useful task is part of a cryptosystem operation.

3.1 Related Work on Puzzles

Early work defending against resource depletion attacks focused around the concept of the "cookie," an opaque bit-string that the initiator of a connection request needs to return verbatim to the server before the request is allowed to proceed. Thus, cookies were used only to establish the validity of the peer in terms of network address reachability; in other words, cookies protect against attackers spoofing their IP address. Cookie-based solutions [41] were used against TCP connection-depletion (also known as TCP SYN) attacks [26, 51], and in security protocols such as Photuris [40], IKE [27], JFK [1], and others [28, 46]. More generally, The advantages of being stateless, at least in the beginning of a protocol run, were recognized in the context of security protocols by Janson *et al.* [35] and Aura and Nikander [5].

Computational client puzzles as a means to defend against denial of service attacks were first introduced by Juels and Brainard [31]. In that work, client puzzles were used to counter TCP SYN attacks from attackers that were willing to expose their IP address, as opposed to attackers with spoofed IP addresses who would require some form of packet marking to identify [11, 25, 42]. Although TCP cookies are ineffective in that scenario, client puzzles can mitigate the effects of such an attack by an adversary that is CPU-limited. However, in recent years, attackers have demonstrated their ability to effectively utilize large numbers of subverted hosts in their attacks [30]. Gligor [21] argues that solutions requiring client proofs of work (*e.g.,* computational client puzzles such as those using hash functions) are both ineffective and unnecessary in open networks, such as the Internet, when strong access guarantees (*e.g.,* maximum waiting time) are desired.

Jakobsson and Juels [32] first proposed the concept of a useful puzzle, which they call a "bread pudding protocol." The particular scheme they use, applied to minting e-coins for the MicroMint micropayment system [49] does not appear to be easily generalizable to other types of useful work.

Client puzzles have also been used in the context of security protocols [4, 32, 43], most notably for protecting SSL against computational denial of service attacks [17]. Other uses of client puzzles involve junk email mitigation [16], fair exchange [9, 20], protection of sensor networks against DoS attacks [55], and time-lock puzzles [50]. The latter aims to encrypt a message such that it cannot be decrypted, even by the sender, until some predetermined future time. A summary of other uses of client puzzles (also known as "hash cash") is presented by Back [6]. Abadi *et al.* [3] introduced the concept of a memory-bound puzzle, which aims to impose the same solving delay as traditional client puzzles by increasing the number of memory accesses a client needs to perform to solve the challenge.

Wang and Reiter [54] introduce the idea of a puzzle auction as a way to ease some of the practical deployment difficulties, *e.g.,* selecting the appropriate hardness for the puzzles. Their approach lets clients bid for the resources by adjusting the difficulty of the puzzles they solve. When the server is attacked, legitimate clients gradually increase their bids (puzzle difficulty), eventually bringing the cost outside the adversary's capabilities.

Based on our original paper on the dual receiver cryptogram [15], Zhang *et al.* [57] develop a general framework for constructing useful client puzzles. Their framework is based on identity-based cryptography and authenticated encryption techniques that are not based on the random oracle model. They show how to construct various puzzles that meet different system requirements. Shi and Yan [52] show that with the proper choice of parameters, based on the underlying system architecture, it is possible to create even faster versions of elliptic curve algorithms purely in software.

4 The Dual Receiver Cryptosystem

The dual receiver cryptosystem is the key component to our useful puzzle construction. After reviewing some definitions and our complexity assumptions, we present a semantically secure dual receiver scheme. We then present the chosen-ciphertext secure dual receiver cryptosystem based on the semantically secure scheme that will be used to construct our useful puzzle.

4.1 Definitions

The key management of our scheme enables a second receiver to decrypt the ciphertext. We formally define dual receiver public key encryption (PKE) schemes below.

Definition 4.1 (Dual Receiver Public Key Encryption Scheme) *A* dual receiver public key encryption scheme *consists of four randomized polynomial-time algorithms* $\mathsf{Enc} = (\mathcal{K}, \mathcal{E}, \mathcal{D}, \mathcal{R})$ *as follows:*

- *The* key generation algorithm $\mathcal{K}$ *is a randomized algorithm that takes a security parameter k as an input, and produces a pair (e, d) of corresponding public encryption and private decryption keys. We write $\mathcal{K}(k) = (e, d)$. (Let $\mathcal{K}(k) = (f, g)$ be another key pair in the following.)*

- *The* encryption algorithm $\mathcal{E}$ *is a randomized algorithm that takes public encryption keys e and f, and a message $m \in \mathcal{M}$ (where $\mathcal{M}$ is the message space) as inputs, and produces a ciphertext $c \in \mathcal{C}$ (where $\mathcal{C}$ is the ciphertext space). We write $\mathcal{E}_{e,f}(m) = c$.*

- *The* decryption algorithm $\mathcal{D}$ *is a deterministic algorithm that takes a private decryption key d, a public encryption key f, and a ciphertext $c \in \mathcal{C}$ as inputs, and produces a message $m \in \mathcal{M}$ or a special* reject *symbol. We write $\mathcal{D}_{d,f}(c) = m$.*

- *The* recovery algorithm $\mathcal{R}$ *is a deterministic algorithm that takes a public encryption key e, a private decryption key g, and a ciphertext $c \in \mathcal{C}$ as inputs, and produces a message $m \in \mathcal{M}$ or a special* reject *symbol. We write $\mathcal{R}_{e,g}(c) = m$.*

We require that if $\mathcal{K}(k)$ outputs (e, d) and (f, g), and $\mathcal{E}_{e,f}(m)$ outputs c, all with positive probability, then $\mathcal{D}_{d,f}(c)$ and $\mathcal{R}_{e,g}(c)$ both output m for all $m \in \mathcal{M}$.

We now formally define the security notions we use. Informally, if no probabilistic polynomial-time (PPT) attacker can recover the whole plaintext from a given ciphertext, then the public key encryption scheme is said to be *one-way.*

Definition 4.2 (One-Wayness of a Dual Receiver PKE Scheme) *Given a dual receiver public key encryption scheme* Enc *and a sufficiently large security parameter k, generate keys $\mathcal{K}(k) = (e, d)$ and $\mathcal{K}(k) = (f, g)$. The success probability of an adversary $\mathcal{A}$,* $\mathrm{Succ}(\mathcal{A})$*, is defined to be* $\Pr[\mathcal{A}(\mathcal{E}_{e,f}(m)) = m]$ *where m is a random message in $\mathcal{M}$.* Enc *is (t, ϵ)-*OW*, if for any such adversary $\mathcal{A}$ with running time bounded by $t(k)$,* $\mathrm{Succ}(\mathcal{A}) < \epsilon(k)$.

A *plaintext-checking oracle* takes as input a plaintext m and a ciphertext c and outputs whether or not c encrypts m. If the adversary above has access to a plaintext-checking oracle, it is playing out a *one-way plaintext-checking attack*, or OW-PCA.

Informally, if no PPT attacker can learn any bit of information about the plaintext from the ciphertext, except the length, then a public key encryption scheme is said to be *semantically secure*, or equivalently *polynomial-time indistinguishable* (notated as IND) [22]. The following definition is the logical extension of semantic security to a dual receiver public key encryption scheme.

Definition 4.3 (Semantic Security of a Dual Receiver PKE Scheme) *Given a dual receiver public key encryption scheme* Enc *and a sufficiently large security parameter k, generate keys $\mathcal{K}(k) = (e, d)$ and $\mathcal{K}(k) = (f, g)$. Given an adversary $\mathcal{A}$ consisting of two PPT algorithms A_1 and A_2, have A_1 choose two equal-length messages (m_0, m_1) from $\mathcal{M}$. For a random bit $b \leftarrow \{0, 1\}$, encrypt the corresponding message $\mathcal{E}_{e,f}(m_b) = c$. The advantage of adversary $\mathcal{A}$,* $\mathrm{Adv}(\mathcal{A})$*, is defined to be* $|\Pr[A_2(m_0, m_1, c) = b] - 1/2|$. Enc *is (t, ϵ)-*IND*, or* semantically secure*, if for any such adversary $\mathcal{A}$ with running time bounded by $t(k)$,* $\mathrm{Adv}(\mathcal{A}) < \epsilon(k)$.

The adversary considered above is playing out a *chosen-plaintext attack*, or CPA, since she is able to encrypt any plaintext of her choice. If the adversary has access to both a decryption oracle, and in our case a recovery oracle, then she is playing out a *chosen-ciphertext attack.* Naturally, we do not allow the adversary to ask that m_0 and m_1 be decrypted. If the adversary's access to the oracle is limited in time, the attack is called *non-adaptive* [44]. If access is unlimited, the attack is called *adaptive* or CCA [48]. Chosen-ciphertext security is the strongest security notion that one can expect in the standard model of communication.

4.2 Complexity Assumptions

Three related complexity assumptions form the basis of security for cryptography done using discrete logarithms in a group. The security of our encryption scheme is based on the difficulty of the Bilinear Diffie-Hellman Problem, which is an extension of the three problems [14] described below for a multiplicative group $\mathbb{G}$ of prime order q.

Definition 4.4 (Discrete Logarithm (DL) Problem) *Given two group elements g and h, find an integer n such that $h = g^n$ whenever such an integer exists.*

Definition 4.5 (Computational Diffie-Hellman (CDH) Problem) *Given three group elements g, g^a, and g^b, $a, b \in \mathbb{Z}$, find an element h such that $h = g^{ab}$.*

Definition 4.6 (Decision Diffie-Hellman (DDH) Problem) *Given four group elements g, g^a, g^b, g^c with $a, b, c \in \mathbb{Z}$, decide whether or not $c = ab$ (modulo the order of g).*

Note that the DDH problem is no harder than the CDH problem, and that the CDH problem is no harder than the DL problem.

A *CDH parameter generator* $\mathcal{G}$ is a randomized algorithm that takes a security parameter k, and outputs the description of a group $\mathbb{G}$. $\mathcal{G}$ should run in time polynomial in k, and the order of $\mathbb{G}$ is determined by k. We write $\mathbb{G} \leftarrow \mathcal{G}(1^k)$. If the CDH problem is hard for $\mathbb{G}$, then we say that $\mathcal{G}$ satisfies the *CDH assumption.* The CDH problem is considered to be hard if $\Pr[\mathcal{A}(\mathbb{G}, g, g^a, g^b) = g^{ab}]$ is negligible in k for all PPT algorithms $\mathcal{A}$, where the probability is taken over $\mathcal{G}$, the random choices of $g \in \mathbb{G}$, and $a, b \in \mathbb{Z}_q^*$.

We will make extensive use of a bilinear map between two abelian groups $\mathbb{G}_1$ and $\mathbb{G}_2$. Let q_1 and q_2 be the orders of the groups. The map $e : \mathbb{G}_1 \times \mathbb{G}_1 \rightarrow \mathbb{G}_2$ is also referred to as a pairing, and the pairing of two elements $g_1, g_2 \in \mathbb{G}_1$ is denoted $e(g_1, g_2) \in \mathbb{G}_2$. Due to the bilinearity condition, for all $g_1, g_2 \in \mathbb{G}_1$ and $a, b \in \mathbb{Z}_{q_1}^*$, the pair $e(g_1^a, g_2^b) = e(g_1, g_2)^{ab}$. We will require that the pairing be non-degenerate (*i.e.,* $e(g, g) \neq 1$). Note that the DL problem should be hard in $\mathbb{G}_2$ so that the pairing is not easily invertible and the DL problem in $\mathbb{G}_1$ is not easily solved [33]. Good candidates for such bilinear maps are the Weil and the Tate pairings defined over points on an elliptic curve defined over a finite field. Currently, the best algorithms for computing the pairing require $O(\log |\mathbb{G}_1|)$ exponentiations in $\mathbb{G}_1$.

The following are natural extensions of the Diffie-Hellman problems:

Definition 4.7 (Bilinear Diffie-Hellman (BDH) Problem) *Given the elements g, g^a, g^b, $g^c \in \mathbb{G}_1$ with $a, b, c \in \mathbb{Z}_{q_1}^*$, find an element $h \in \mathbb{G}_2$ such that $h = e(g, g)^{abc}$.*

Definition 4.8 (Decision Bilinear Diffie-Hellman (DBDH) Problem) *Given the elements $g, g^a, g^b, g^c \in \mathbb{G}_1$ with $a, b, c \in \mathbb{Z}_{q_1}^*$ and $h \in \mathbb{G}_2$, decide whether or not $h = e(g, g)^{abc}$. If $h = e(g, g)^{abc}$, then (g, g^a, g^b, g^c, h) is called a* valid DBDH tuple.

Definition 4.9 (Gap Bilinear Diffie-Hellman (GBDH) Problem) *Solve a given instance, (g, g^a, g^b, g^c), of the BDH problem with the help of a DBDH oracle that is able to decide whether or not a tuple $(g, g^{a'}, g^{b'}, g^{c'}, h)$ is valid.*

A *BDH parameter generator* $\mathcal{G}$ is a randomized algorithm that takes a security parameter k, and outputs the description of two groups $\mathbb{G}_1$ and $\mathbb{G}_2$, and the description of a non-degenerate bilinear map. $\mathcal{G}$ should run in time polynomial in k, and the orders of $\mathbb{G}_1$ and $\mathbb{G}_2$ are determined by k. We write $(\mathbb{G}_1, \mathbb{G}_2, e) \leftarrow \mathcal{G}(1^k)$. If the BDH problem is hard for $(\mathbb{G}_1, \mathbb{G}_2, e)$, then we say that $\mathcal{G}$ satisfies the *BDH assumption.* The BDH problem is considered to be hard if $\Pr[\mathcal{A}(\mathbb{G}_1, \mathbb{G}_2, e, g, g^a, g^b, g^c) = e(g, g)^{abc}]$ is negligible in k for all PPT algorithms $\mathcal{A}$, where the probability is taken over $\mathcal{G}$, the random choices of $g \in \mathbb{G}_1$, and $a, b, c \in \mathbb{Z}_{q_1}^*$.

Joux [34] gives a detailed analysis of the BDH problem in his survey of the Weil and Tate pairings as building blocks for cryptosystems.

4.3 The Semantically Secure Dual Receiver Scheme

The dual receiver public key encryption scheme, SEnc, provides semantic security. The message space is $\mathcal{M} = \{0, 1\}^n$. The user may choose any public key g^y as the second receiver, and may even choose herself if she does not wish a third-party to have access to the message. Note that the decryption algorithm and the recovery algorithm are the same operations using different keys.

We require that there be a hash function H_x associated with each public key g^x; it is easy to base such a family on a given random oracle hash (by first attaching a proper prefix derived from g^x to any string to be hashed).

Key Generation. Groups $\mathbb{G}_1$ and $\mathbb{G}_2$ are chosen using a BDH parameter generator $\mathcal{G}$, along with a random element $g \in \mathbb{G}_1$, and $x \in \mathbb{Z}_{q_1}^*$. The public key is (g, g^x) together with a cryptographic hash function $H_x : \mathbb{G}_2 \rightarrow \{0, 1\}^n$. The private key is x.

Encryption. The input is a plaintext $m \in \{0, 1\}^n$. The encryption algorithm chooses a random element $r \in \mathbb{Z}_{q_1}^*$ and computes $u_1 = g^r$, $u_2 = g^y$, and $u_3 = m \oplus H_x(e(g^x, g^y)^r)$, where H_x is the primary receiver's hash function and g^y is the secondary receiver's public key. The ciphertext is (u_1, u_2, u_3).

Decryption. The decryption algorithm for private key x computes: $u_3 \oplus H_x(e(u_1, u_2)^x) = m$. Note that $e(u_1, u_2)^x = e(g^r, g^y)^x = e(g^x, g^y)^r = e(g, g)^{rxy}$.

Recovery. The recovery algorithm for private key y computes: $u_3 \oplus H_x(e(u_1, g^x)^y) = m$. Note that $e(u_1, g^x)^y = e(g^r, g^x)^y = e(g^x, g^y)^r = e(g, g)^{rxy}$.

Security. SEnc is a semantically secure (IND-CPA) dual receiver public key encryption scheme if the BDH problem is assumed to be hard.

Theorem 4.1 *Let H_x be a random oracle from $\mathbb{G}_2$ to $\{0,1\}^n$. Let $\mathcal{A}$ be an adversary with running time bounded by t that has advantage ϵ against* SEnc*. Suppose $\mathcal{A}$ makes a total of $q_{H_x} > 0$ queries to H_x. Then there is an algorithm $\mathcal{B}$ that solves the BDH problem for $\mathcal{G}$ with advantage at least $2\epsilon/q_{H_x}$ and running time $O(t)$.*

Proof. The proof of this theorem closely follows Lemma 4.3 in Boneh and Franklin's work [7]. Algorithm $\mathcal{B}$ is given the BDH parameters produced by $\mathcal{G}$ and an instance of the BDH problem for these parameters, (g, g^a, g^b, g^c). $\mathcal{B}$ uses the adversary $\mathcal{A}$ to find $h = e(g,g)^{abc}$, the solution to the BDH problem, as follows. First, $\mathcal{B}$ creates a public key for SEnc by setting $g^x = g^a$ and $g^y = g^b$, and sends (g^x, g^y, H_x) to $\mathcal{A}$.

- *H_x-queries:* Here H_x is a random oracle controlled by $\mathcal{B}$ where $\mathcal{B}$ keeps a list of pairs, the H_x-list. When $\mathcal{A}$ issues a query, q_i, to H_x, $\mathcal{B}$ checks to see if q_i is on the H_x-list. If q_i appears in a pair (q_i, h_i), then $\mathcal{B}$ responds with $H_x(q_i) = h_i$. Otherwise, $\mathcal{B}$ picks a random string $h_i \leftarrow \{0,1\}^n$, adds the pair (q_i, h_i) to the H_x-list, and responds with $H_x(q_i) = h_i$.

- *Challenge:* The adversary $\mathcal{A}$ produces two messages m_0 and m_1 on which it wishes to be challenged. $\mathcal{B}$ picks a random string $u \in \{0,1\}^n$, defines the ciphertext to be (g^c, u), and gives the ciphertext as the challenge to $\mathcal{A}$. Note that the decryption of the ciphertext is: $u \oplus H_x(e(g^a, g^b)^c) = u \oplus H_x(h)$, due to the way we defined g^x and g^y.

- *Guess:* The adversary $\mathcal{A}$ outputs its guess γ from $\{\text{YES}, \text{NO}\}$. $\mathcal{B}$ responds by outputting a random q_i that appears on the H_x-list as the solution to the given instance of the BDH problem.

Let Q be the event that $\mathcal{A}$ issues a query for h. Then $\Pr[Q]$ in the simulation is the same as $\Pr[Q]$ in the real attack. Before any queries are made $\Pr[Q] = 0$ in both cases. Let Q_i be the event that a query for h was made in the first i queries. $\Pr[Q_i] = \Pr[Q_i|Q_{i-1}]\Pr[Q_{i-1}] + \Pr[Q_i|\neg Q_{i-1}]\Pr[\neg Q_{i-1}]$, and using induction we only have to show that $\Pr[Q_i|\neg Q_{i-1}]$ in the simulation is the same as $\Pr[Q_i|\neg Q_{i-1}]$ in the real attack. Note that the public key and the challenge are distributed as in the real attack, and all responses to the H_x-queries are uniform and independent in $\{0,1\}^n$. Thus, $\Pr[Q_i|\neg Q_{i-1}]$ is the same in both the simulation and the real attack, and $\Pr[Q]$ is the same in both the simulation and the real attack.

If $\mathcal{A}$ never issues a query for h, then the decryption of the ciphertext is independent of $\mathcal{A}$'s view. Let the true answer to the BDH problem be γ. Therefore in the real attack $\Pr[\gamma = \gamma'|\neg Q] = 1/2$. Since $\mathcal{A}$ has advantage ϵ, $|\Pr[\gamma = \gamma'] - 1/2| \geq \epsilon$.

$$\begin{aligned}\Pr[\gamma = \gamma'] &= \Pr[\gamma = \gamma'|\neg Q]\Pr[\neg Q] + \Pr[\gamma = \gamma'|Q]\Pr[Q] \\ &\leq \frac{1}{2}\Pr[\neg Q] + \Pr[Q] = \frac{1}{2} + \frac{1}{2}\Pr[Q].\end{aligned}$$

Therefore $\epsilon \leq |\Pr[\gamma = \gamma'] - 1/2| \leq 1/2\Pr[Q]$, and $\Pr[Q] \geq 2\epsilon$. The probability that h appears in some pair on the H_x-list is at least 2ϵ, and thus $\mathcal{B}$ produces the correct answer with probability at least $2\epsilon/q_{H_x}$. □

SEnc is also a (OW-PCA) dual receiver public key encryption scheme if the GBDH problem is assumed to be hard.

Lemma 4.1 *Let PCO be a plaintext-checking oracle, and let H_x be a random oracle from $\mathbb{G}_2$ to $\{0,1\}^n$. Let $\mathcal{A}$ be an adversary with running time bounded by t that has success probability ϵ against* SEnc*. Suppose $\mathcal{A}$ makes a total of $q_{H_x} > 0$ queries to H_x and PCO. Then there is an algorithm $\mathcal{B}$ that solves the GBDH problem for $\mathcal{G}$ with advantage at least $(\epsilon - \frac{1}{2^n})/q_{H_x}$, and a running time $O(t)$.*

Proof. Algorithm $\mathcal{B}$ is given the BDH parameters produced by $\mathcal{G}$ and an instance of the GBDH problem for these parameters, (P, g^a, g^b, g^c). $\mathcal{B}$ uses the adversary $\mathcal{A}$ and a DBDH oracle to find $g = e(g,g)^{abc}$, the solution to the GBDH problem, as follows. First, $\mathcal{B}$ creates a public key for SEnc by setting $g^x = g^a$ and $g^y = g^b$, and sends (g^x, g^y, H_x) to $\mathcal{A}$.

- *H_x-queries:* H_x-queries are handled as they are in Theorem 4.1.

- *PCO-queries:* Here *PCO* is a plaintext-checking oracle controlled by $\mathcal{B}$. *PCO*-queries are equivalent to reverse H_x-queries. When $\mathcal{A}$ issues a query, $(m_i, c_i = (u_{1_i}, u_{2_i}, u_{3_i}))$, to *PCO*, $\mathcal{B}$ checks to see if $h_i = m_i \oplus u_{3_i}$ is on the H_x-list. If h_i does not appear in a pair (q_i, h_i), then $\mathcal{B}$ picks a random element $r \leftarrow \mathbb{Z}_{q_2}$, sets $q_i = e(g,g)^r$, and adds the pair (q_i, h_i) to the H_x-list. $\mathcal{B}$ uses a DBDH oracle to determine if $(g^x, u_{1_i}, u_{2_i}, q_i)$ is a valid DBDH tuple. If it is, $\mathcal{B}$ responds YES to $\mathcal{A}$, and otherwise responds NO.

- *Challenge:* $\mathcal{B}$ picks a random string $u \in \{0,1\}^n$, defines the ciphertext to be (g^c, g^y, u), and gives the ciphertext as the challenge to $\mathcal{A}$. Note that the decryption of the ciphertext is: $u \oplus H_x(e(g^a, g^b)^c) = u \oplus H_x(g)$, due to the way we defined g^x and g^y.

- *Guess:* The adversary $\mathcal{A}$ outputs its guess $m \in \{0,1\}^n$. $\mathcal{B}$ responds by outputting a random q_i that appears on the H_x-list as the solution to the given instance of the GBDH problem.

Let Q be the event that $\mathcal{A}$ issues a query for g in a H_x-query or a query for $H_x(g)$ (actually a query for some $(m', (g^x, g^y, m' \oplus H_x(g))))$ in a *PCO*-query. Then $\Pr[Q]$ in the simulation is the same as $\Pr[Q]$ in the real attack. Before any queries are made $\Pr[Q] = 0$ in both cases. Let Q_i

be the event that a query for g or $H_x(g)$ was made in the first i queries. All the responses by the PCO-queries are valid and only create entries in the H_x-list that are uniform and independent in $\{0,1\}^n$. Using similar reasoning to that used in the proof for Theorem 4.1, $\Pr[Q_i|\neg Q_{i-1}]$ is the same in both the simulation and the real attack, and thus $\Pr[Q]$ is the same in both the simulation and the real attack.

Let $S \overset{\text{def}}{=} \Pr[\mathcal{A}((g^c, g^y, u))]$. If $\mathcal{A}$ never issues a query for g or $H_x(g)$, then the decryption of the ciphertext is independent of $\mathcal{A}$'s view. Therefore in the real attack $\Pr[S|\neg Q] = 1/2^n$. Since $\mathcal{A}$ has success probability ϵ, $\Pr[S] \geq \epsilon$.

$$\begin{aligned}\Pr[S] &= \Pr[S|\neg Q]\Pr[\neg Q] + \Pr[S|Q]\Pr[Q] \\ &\leq \frac{1}{2^n}\Pr[\neg Q] + \Pr[Q] = \frac{1}{2^n} + \left(1 - \frac{1}{2^n}\right)\Pr[Q].\end{aligned}$$

Therefore $\epsilon \leq \Pr[S] \leq \frac{1}{2^n} + \left(1 - \frac{1}{2^n}\right)\Pr[Q]$, and

$$\Pr[Q] \geq \frac{\epsilon - \frac{1}{2^n}}{1 - \frac{1}{2^n}} \geq \epsilon - \frac{1}{2^n}.$$

The probability that g or H_x appears in some pair on the H_x-list is at least $\epsilon - \frac{1}{2^n}$, and thus $\mathcal{B}$ produces the correct answer with probability at least $(\epsilon - \frac{1}{2^n})/q_{H_x}$. □

4.4 The Chosen-Ciphertext Secure Dual Receiver Scheme

The encryption scheme CEnc provides chosen-ciphertext security and allows a specified third party to decrypt the ciphertext. We use the REACT conversion introduced by Okamoto and Pointcheval [45] to convert the SEnc scheme into a chosen-ciphertext secure scheme. (We could use the GEM conversion [12] instead to obtain a shorter ciphertext, but at the expense of allowing for the session key to precomputed.) The message space is $\mathcal{M} = \{0,1\}^n$, b_2 is the length of the bit-string representation of a point in $\mathbb{G}_2$, and n' is a security parameter.

Key Generation. Groups $\mathbb{G}_1$ and $\mathbb{G}_2$ are chosen using a BDH parameter generator $\mathcal{G}$, as are a random element $g \in \mathbb{G}_1$, and $x \in \mathbb{Z}_{q_1}^*$. The public key is (g, g^x) together with cryptographic hash functions $H_x : \mathbb{G}_2 \rightarrow \{0,1\}^n$, $G : \{0,1\}^n \rightarrow \{0,1\}^n$, and $F : \{0,1\}^{4n+b_2} \rightarrow \{0,1\}^{n'}$. The private key is x.

Encryption. The input is a plaintext $m \in \{0,1\}^n$. The encryption algorithm chooses a random element $r \in \mathbb{Z}_{q_1}^*$, a random element $\rho \in \{0,1\}^n$, and computes $u_1 = g^r$, $u_2 = g^y$, $u_3 = \rho \oplus H_x(e(g^x, g^y)^r)$, $u_4 = m \oplus G(\rho)$, and $u_5 = F(\rho, m, u_3, u_4, e(g^x, g^y)^r)$. The ciphertext is $(u_1, u_2, u_3, u_4, u_5)$.

Decryption. The decryption algorithm computes $u_3 \oplus H_x(e(u_1, u_2)^x) = \rho$ and $G(\rho) \oplus u_4 = m$ given a ciphertext $(u_1, u_2, u_3, u_4, u_5)$. Then it checks that $u_5 = F(\rho, m, u_3, u_4, e(u_1, u_2)^x)$, and if u_5 is correct, the algorithm outputs m. Otherwise, it outputs Reject.

Recovery. The recovery algorithm computes $u_3 \oplus H_x(e(u_1, g^x)^y) = \rho$ and $G(\rho) \oplus u_4 = m$, given a ciphertext $(u_1, u_2, u_3, u_4, u_5)$. Then it checks that $u_5 = F(\rho, m, u_3, u_4, e(u_1, g^x)^y)$, and if u_5 is correct, the algorithm outputs m. Otherwise, it outputs Reject.

Security. CEnc is a chosen-ciphertext secure dual receiver public key encryption scheme if the GBDH problem is assumed to be hard. Since SEnc is OW-PCA and one-time pads (the XORs) are semantically secure, the conversion is chosen-ciphertext secure in the random oracle model. (See Theorem 1 in Okamoto and Pointcheval [45].)

Non-mandatory Escrow Encryption. Our dual receiver encryption scheme can easily allow escrow encryption at the sender's discretion, without any sacrifices to security. The sender can choose a specific escrow public key to be the second key to recover the information. The sender can also choose her own public key to be the second key if she does not wish a third-party to have access to the message. This gives a very flexible key management component that can be used in designing secure file and storage systems, and general recovery and backup policies. The designation of tasks in the system can be managed via a key certification process.

5 A Combined Public Key Cryptosystem

In this section we present a signature scheme secure against adaptive chosen-message attacks, that can be used in conjunction with the chosen-ciphertext attack secure dual receiver encryption scheme while using the same private key.

5.1 The Public Key Signature Scheme

5.1.1 Definitions

Definition 5.1 (Public Key Signature Scheme) *A public key signature scheme* Sig $= (\mathcal{K}, \mathcal{S}, \mathcal{V})$ *consists of three randomized polynomial-time algorithms as follows:*

- *The* key generation algorithm $\mathcal{K}$ *is a randomized algorithm that takes a security parameter k as an input, and produces a pair (s, v) of corresponding private signature and public verification keys. We write $\mathcal{K}(k) = (s, v)$.*
- *The* signature algorithm $\mathcal{S}$ *is a randomized algorithm that takes a private signature key s and a message $m \in \mathcal{M}$ as inputs, and produces a signature $\sigma \in \{0,1\}^*$. We write $\mathcal{S}_s(m) = \sigma$.*

- *The* verification algorithm $\mathcal{V}$ *is a randomized algorithm that takes a public verification key v and a message-signature pair (m, σ) as inputs, and produces as output either* accept *or* reject. *We write* $\mathcal{V}_v(m, \sigma) = \mathsf{accept}$.

We require that if $\mathcal{K}(k)$ outputs (s, v) and $\mathcal{S}_s(m)$ outputs σ, both with positive probability, then $\mathcal{V}_v(m, \sigma) = \mathsf{accept}$, and for any other pair (m, σ'), $\mathcal{V}_v(m, \sigma') = \mathsf{reject}$.

When considering signature schemes, if no PPT attacker can forge a signature on one message, then the signature scheme is secure against *existential forgery*.

Definition 5.2 (Security of a Public Key Signature Scheme) *Given* $\mathsf{Sig} = (\mathcal{K}, \mathcal{S}, \mathcal{V})$*, a public key signature scheme, and a sufficiently large security parameter k, generate keys* $\mathcal{K}(k) = (s, v)$*. Given an adversary* $\mathcal{A}$ *consisting of a PPT algorithm A, A outputs a message/signature pair,* (m, σ)*.* $\mathrm{Adv}(\mathcal{A})$ *is defined as* $\Pr[\mathcal{V}_v(m, \sigma) = \mathsf{accept}]$*.* Sig *is* (t, ϵ)-secure *against* existential forgery *if for any such adversary* $\mathcal{A}$ *with running time bounded by* $t(k)$*,* $\mathrm{Adv}(\mathcal{A}) < \epsilon(k)$*.*

The above adversary is playing out a *key-only attack*, but if the adversary observes valid message/signature pairs chosen and produced by the signer, then she is playing out a *known signature attack*. If the adversary is allowed to ask the signer to sign a number of messages of her choice, then she is playing out a *chosen message attack*. Naturally, we do not allow the adversary to ask that the challenge message be signed. If the adversary's access to the signer is limited in time, the attack is called *non-adaptive*, and if access is unlimited, the attack is called *adaptive* or CMA [23].

5.1.2 The Actual Combined Scheme

The signature scheme Sig provides security against existential forgery under a chosen message attack if the CDH problem is assumed to be hard in $\mathbb{G}_1$. The message space is $\mathcal{M} = \{0, 1\}^n$. This scheme is similar to Boneh, Lynn, and Shacham's signature scheme [8].

Key Generation. Groups $\mathbb{G}_1$ and $\mathbb{G}_2$ are chosen using a BDH parameter generator $\mathcal{G}$, as are a random element $g \in \mathbb{G}_1$, and $x \in \mathbb{Z}^*_{q_1}$. The public verification key is (g, g^x) together with a cryptographic hash function $I : \{0, 1\}^n \to \mathbb{G}_1$. The private signature key is x.

Signature. The input is a private signature key $x \in \mathbb{Z}^*_{q_1}$ and a plaintext $m \in \{0, 1\}^n$. The signature algorithm calculates $\sigma = I(m)^x$. The signature is σ.

Verification. Given a public key (g, g^x), and a message-signature pair (m, σ), the verification algorithm verifies that $e(g, \sigma) = e(g^x, I(m))$.

Security. Sig is secure against existential forgery under adaptive chosen-message attacks.

Theorem 5.1 *Let I be a random oracle from* $\{0, 1\}^n$ *to* $\mathbb{G}_1$*. Let* $\mathcal{A}$ *be an adversary with running time bounded by t that has advantage* ϵ *against* Sig*. Suppose* $\mathcal{A}$ *makes a total of* $q_I > 0$ *queries to I and* $q_S > 0$ *signature queries. Then there is an algorithm* $\mathcal{B}$ *that solves the CDH problem for* $\mathbb{G}_1$ *with advantage at least* $\epsilon/e(q_S + 1)$ *(where e is the base of the natural logarithm) and a running time at most* $t + j(q_I + 2q_S)$*, where j is the time taken to multiply two points in* $\mathcal{G}_1$*.*

Proof. Algorithm $\mathcal{B}$ is given the CDH parameters produced by $\mathcal{G}$ and an instance of the CDH problem for these parameters, (g, g^a, g^b). $\mathcal{B}$ uses the adversary $\mathcal{A}$ to find $h = g^{ab}$, the solution to the CDH problem, as follows. First, $\mathcal{B}$ creates a verification key for Sig by setting $g^x = g^{a+r}$ for a random $r \in \mathbb{Z}$, and sends (g, g^x, I) to $\mathcal{A}$.

- *I-queries:* Here I is a random oracle controlled by $\mathcal{B}$ where $\mathcal{B}$ keeps a list of tuples, the I-list. When $\mathcal{A}$ issues a query, q_i, to I, $\mathcal{B}$ checks to see if q_i is on the I-list. If q_i appears in a tuple (q_i, i_i, r_i, c_i), then $\mathcal{B}$ responds with $I(q_i) = i_i$. Otherwise, $\mathcal{B}$ picks a random $r_i \in \mathbb{Z}$ and generates a random coin $c_i \in \{0, 1\}$ where $\Pr[c_i = 0] = 1/(q_S + 1)$. If $c_i = 0$, $\mathcal{B}$ sets $i_i = g^{b+r_i}$. If $c_i = 1$, $\mathcal{B}$ sets $i_i = g^{r_i}$. $\mathcal{B}$ adds the tuple (q_i, i_i, r_i, c_i) to the I-list, and responds with $I(q_i) = i_i$.

- *Signature Queries:* When $\mathcal{A}$ issues a signature query, q_i, $\mathcal{B}$ obtains the corresponding tuple by making a I-query as outlined above. If $c_i = 0$, $\mathcal{B}$ reports failure and terminates. If $c_i = 1$, $\mathcal{B}$ sets $\sigma_i = g^{ar_i} + i_i^r = g^{r_i(a+r)}$. Note that σ_i is a valid signature for q_i under the public key g^x which was set to g^{ar}. $\mathcal{B}$ gives σ_i to $\mathcal{A}$.

- *Challenge:* The adversary $\mathcal{A}$ produces m, σ, a message-signature pair on which it wishes to be challenged such that m was never a signature query. If σ is not a valid signature on m, $\mathcal{B}$ reports failure and terminates. Otherwise, $\mathcal{B}$ obtains the corresponding tuple by making a I-query as outlined above. If $c_i = 1$, then $\mathcal{B}$ reports failure and terminates. Otherwise, $c_i = 0$ and $i_i = g^{b+r_i}$. Thus, $\sigma = g^{(a+r)(r_i+b)}$. Then $\mathcal{B}$ outputs the required solution to the CDH problem as $g^{ab} = \sigma g^{-rb} g^{-ar_i} + g^{-rr_i}$).

Let Q_1 be the event that $\mathcal{B}$ does not abort during $\mathcal{A}$'s signature queries. The probability that $\mathcal{B}$ does not abort during one query is $1 - 1/(q_S + 1)$, and since $\mathcal{A}$ makes at most q_S signature queries, $\Pr[Q_1] \geq (1 - 1/(q_S + 1))^{q_S} \geq 1/e$.

Let Q_2 be the event that $\mathcal{A}$ produces a valid message-signature pair given that the Challenge stage was successfully reached. The public key given to $\mathcal{A}$ is from the same distribution as a public key produced by the key generation algorithm, and the responses to the I-queries are uniformly and independently distributed in $\mathbb{G}_1$. Thus $\Pr[Q_2] \geq \epsilon$.

Let Q_3 be the event that the final I-query made by $\mathcal{B}$ does not fail given that $\mathcal{B}$ did not report a failure up to this point. The probability that $c = 0$ is $1/(q_s + 1)$ so $\Pr[Q_3] \geq 1/(q_s + 1)$.

If Q_1, Q_2, and Q_3 are true then $\mathcal{B}$ produces the correct answer. Thus $\mathcal{B}$ solves the CDH problem with probability at least $\epsilon/e(q_s + 1)$. $\mathcal{B}$'s running time is the time it takes for $\mathcal{A}$ to run plus the time it takes to respond to $(q_I + q_S)$ hash queries and q_S signature queries. If a multiplication in $\mathbb{G}_1$ takes time j, then the total running time is at most $t + j(q_I + 2q_S)$. □

5.2 The Combined Scheme

Recall that the combined cryptosystem is a public key cryptosystem supporting both encryption and signature generation with a single private key per user. We first present our signature scheme then our combined scheme.

5.2.1 Definitions

Definition 5.3 (Combined Scheme) *Given a dual receiver PKE scheme* $\mathsf{Enc} = (\mathcal{K}_E, \mathcal{E}, \mathcal{D}, \mathcal{R})$ *and a public key signature scheme* $\mathsf{Sig} = (\mathcal{K}_S, \mathcal{S}, \mathcal{V})$, *the* combined scheme *consists of six randomized polynomial-time algorithms* $\mathsf{Comb} = (\mathcal{K}, \mathcal{E}, \mathcal{D}, \mathcal{R}, \mathcal{S}, \mathcal{V})$ *as follows:*

- *The* key generation algorithm $\mathcal{K}$ *is a randomized algorithm that takes a security parameter k as an input, and produces two pairs of keys $[(e, d), (s, v)]$, the first for* Enc *and the second for* Sig.
- *Encryption, decryption, and message recovery are performed with $\mathcal{E}$, $\mathcal{D}$, and $\mathcal{R}$, and signature generation and verification are performed with $\mathcal{S}$ and $\mathcal{V}$, exactly as in the original schemes.*

Note that the only differences between the combined and the original schemes are in the key generation algorithms.

When combining a dual receiver public key encryption scheme Enc with a public key signature scheme Sig we must verify that sharing keys between the two does not degrade the security of either scheme. Thus, an adversary for Enc with access to a signature-generation oracle should not have a greater probability of success in attacking Enc than it would if it did not have access to the oracle. Similarly, an adversary for Sig with access to a decryption oracle and a recovery oracle should not have a greater probability of success in attacking Sig than it would if it did not have access to either oracle. We prove that the security of a scheme is not degraded in the presence of an oracle by constructing a simulator that does not have the private keys of the scheme, yet can answer the adversary's queries in a manner that is indistinguishable from that of an oracle. If a signature-generation oracle, a decryption oracle, and a recovery oracle can be simulated, then both Enc and Sig can be used in combination without compromising the security of either. Our analysis of the security of the combined scheme uses the technique used by Haber and Pinkas [29] to combine other encryption and signature schemes.

Definition 5.4 (Security of an Encryption Scheme in a Combined Scheme) *The combined scheme* $\Sigma = (\mathsf{Enc}, \mathsf{Sig})$ *does not compromise the security of* Enc *if for any PPT adversary $\mathcal{A}$ with unlimited access to an oracle for $\mathcal{S}_s$, there exists an adversary $\mathcal{A}'$ for* Enc *alone with success probability at most negligibly worse than the success probability of $\mathcal{A}$.*

Definition 5.5 (Security of a Signature in a Combined Scheme) *The combined scheme* $\Sigma = (\mathsf{Enc}, \mathsf{Sig})$ *does not compromise the security of* Sig *if for any PPT adversary $\mathcal{A}$ with unlimited access to an oracle for $\mathcal{D}_{d,f}$ and $\mathcal{R}_{e,g}$, there exists an adversary $\mathcal{A}'$ for* Sig *alone with success probability at most negligibly worse than the success probability of $\mathcal{A}$.*

Definition 5.6 (Security of a Combined Scheme) *The combined scheme* $\Sigma = (\mathsf{Enc}, \mathsf{Sig})$ *is* CCA-CMA *secure if no PPT adversary $\mathcal{A}$ has a non-negligible advantage against a challenger $\mathcal{A}'$ in a joint* CCA-CMA *game. The adversary is allowed q_1 adaptive queries to signature and decryption oracles, and then picks between a* CCA *or a* CMA *challenge. Once the challenger lays out the challenge, the adversary is allowed q_2 adaptive queries before producing her guess.*

5.2.2 The Actual Signature Scheme

Recall that a combined public key scheme leaves the encryption, decryption, recovery, signature generation, and verification algorithms unchanged, but needs a new key generation algorithm.

Key Generation. Groups $\mathbb{G}_1$ and $\mathbb{G}_2$ are chosen using a BDH parameter generator $\mathcal{G}$, along with a random element $g \in \mathbb{G}_1$, and $x \in \mathbb{Z}^*_{q_1}$. The public key is (g, g^x) together with a cryptographic hash function $H_x : \mathbb{G}_2 \to \{0,1\}^n$. The private decryption key is x. The public verification key is (g, g^x) together with a cryptographic hash function $I : \{0,1\}^n \to \mathbb{G}_1$. The private signature key is x.

Security of *CEnc* in the Presence of *Sig*. The combined scheme does not compromise the security of *CEnc*.

Lemma 5.1 *Let I be a random oracle from $\{0,1\}^n$ to $\mathbb{G}_1$. Let $\mathcal{A}$ be an adversary that has advantage ϵ against* CEnc *in a* CCA *attack with unlimited access to I and a signature oracle for* Sig. *Then there is an algorithm $\mathcal{B}$ that has advantage ϵ against* CEnc.

Proof. Given an adversary $\mathcal{A}$ that attacks CEnc when used together with Sig, we construct an adversary $\mathcal{B}$ attacking CEnc alone.

Algorithm $\mathcal{B}$ is given (g, g^x, g^y), the encryption key for CEnc, and $\mathcal{B}$ sends the seven-tuple $(g, g^x, g^y, H_x, G, F, I)$ to $\mathcal{A}$ where I is a random oracle controlled by $\mathcal{B}$.

- *I-queries:* Here I is a random oracle controlled by $\mathcal{B}$ where $\mathcal{B}$ keeps a list of tuples, the I-list. When $\mathcal{A}$ issues a query, q_i, to I, $\mathcal{B}$ checks to see if q_i is on the I-list. If q_i appears in a tuple (q_i, i_i, r_i), then $\mathcal{B}$ responds with $I(q_i) = i_i$. Otherwise, $\mathcal{B}$ picks a random $r_i \in \mathbb{Z}^*_{q_1}$ and sets $i_i = g^{r_i}$. $\mathcal{B}$ adds the tuple (q_i, i_i, r_i) to the I-list, and responds with $I(q_i) = i_i$.

- *Signature Queries:* When $\mathcal{A}$ issues a signature query, q_i, $\mathcal{B}$ obtains the corresponding tuple, (q_i, i_i, r_i), by making a I-query as outlined above. $\mathcal{B}$ sets $\sigma_i = (g^{xr_i})$. Note that σ_i is a valid signature for q_i under the public key g^x. $\mathcal{B}$ gives σ_i to $\mathcal{A}$.

$\mathcal{A}$'s view of the signature is identical to that of a real signature, and thus its probability of success in breaking the encryption scheme is unchanged. $\square$

Security of *Sig* in the Presence of *CEnc*. The combined scheme does not compromise the security of *Sig*.

Lemma 5.2 *Let H_x be a random oracle from $\mathbb{G}_2$ to $\{0,1\}^n$, and let F be a random oracle from $\{0,1\}^{4n+b_2}$ to $\{0,1\}^{n'}$. Let $\mathcal{A}$ be an adversary that has advantage ϵ against* Sig *with unlimited access to H_x, F, and a decryption oracle for* CEnc*. Then there is an algorithm $\mathcal{B}$ that has advantage ϵ against* Sig.

Proof. Given an adversary $\mathcal{A}$ that attacks Sig when used together with CEnc, we construct an adversary $\mathcal{B}$ attacking Sig alone.

Algorithm $\mathcal{B}$ is given the verification key for Sig, (g, g^x), and $\mathcal{B}$ sends (g, g^x, H_x, G, F) to $\mathcal{A}$ where H_x and F are random oracles controlled by $\mathcal{B}$.

- *H_x-queries:* Here H_x is a random oracle controlled by $\mathcal{B}$ where $\mathcal{B}$ keeps a list of tuples, the H_x-list. When $\mathcal{A}$ issues a query, q_i, to H_x, $\mathcal{B}$ checks to see if q_i is on the H_x-list. If q_i appears in a tuple (q_i, h_i), then $\mathcal{B}$ responds with $H_x(q_i) = h_i$. Otherwise, $\mathcal{B}$ picks a random $h_i \in \{0,1\}^n$, adds the tuple (q_i, h_i) to the H_x-list, and responds with $H_x(q_i) = h_i$.

- *F-queries:* Here F is a random oracle controlled by $\mathcal{B}$ where $\mathcal{B}$ keeps a list of tuples, the F-list. When $\mathcal{A}$ issues a query, q_i, to F, $\mathcal{B}$ checks to see if q_i is on the F-list (note that q_i is a bit-string of length $\{0,1\}^{4n+b_2}$). If q_i appears in a tuple (q_i, r_i), then $\mathcal{B}$ responds with $F(q_i) = r_i$. Otherwise, $\mathcal{B}$ picks a random $r_i \in \{0,1\}^{n'}$, adds the tuple (q_i, r_i) to the F-list, and responds with $F(q_i) = r_i$.

- *Decryption Queries:* $\mathcal{B}$ responds as follows when $\mathcal{A}$ issues the five-tuple decryption query, $q_i = (u_1, u_2, u_3, u_4, u_5)$. $\mathcal{B}$ obtains the corresponding tuple, (q_i, u_5), from the F-list and sets g to the last b_2 bits of q_i. If u_5 is not in any tuple, then $\mathcal{B}$ picks a random $g \in \{0,1\}^{b_2}$. $\mathcal{B}$ then obtains the corresponding tuple (g, h_i) from the H_x list by making a H_x-query as outlined above. $\mathcal{B}$ sets $\rho = u_3 \oplus h_i$, and outputs $m = u_4 \oplus G(\rho)$.

$\mathcal{A}$'s view of the decryption is identical to that of a real decryption (as is its view of the recovery for they are simulated the same way), and thus its probability of success in forging a signature is unchanged. $\square$

Security of Σ=(*CEnc*,*Sig*). The combined scheme Σ is CCA-CMA secure, and thus does not compromise its own security with respect to an adversary that is trying to compromise either the encryption or the signature.

Theorem 5.2 *Let H_x, F, and I be random oracles, then Σ is* CCA-CMA *secure, assuming that the GBDH and the CDH problems are hard.*

Proof. Queries to H_x, F, I, the decryption and signature oracles are exactly as in Lemmas 5.1 and 5.2. Due to the chosen-ciphertext security of *CEnc*, Theorem 5.1, and Lemmas 5.1 and 5.2, if $\mathcal{A}$ has advantage ϵ over Σ, then there is an algorithm $\mathcal{B}$ that can solve either the GBDH or the CDH problem with non-negligible probability. $\square$

6 Useful Security Puzzles

Our approach for fulfilling the requirements for a useful security puzzle (computational intensity, reliability, usefulness, non-dependability, and security) is to use our dual receiver cryptosystem to construct a *client-generated useful puzzle.* We first describe the features of a client-generated useful puzzle, then we show how they fulfill the original requirements we laid out in Section 3. Then we show two implementations (CUP1 and CUP2) of the client-generated useful puzzle using our dual receiver scheme. The first is only secure in the secure channel model. The second is secure without that assumption, but it requires more work of the server. Finally, we describe how to use our scheme in an online transaction server.

6.1 Client-generated Useful Puzzles

We define *client-generated useful puzzles* so that they will easily fulfill the requirements we laid out in Section 3. Recall that the requirements were: computational intensity, reliability, usefulness, non-dependability, and security. An important feature of our definition is that a client does most of the work for generating the puzzle.

Definition 6.1 (Client-generated useful puzzle) *A* client-generated useful puzzle *consists of six randomized polynomial-time algorithms* CUP $= (\mathcal{S}, \mathcal{G}, \mathcal{R}, \mathcal{C}, \mathcal{H})$ *as follows:*

- *The* set-up algorithm $\mathcal{S}$ *is a randomized algorithm that takes a security parameter k as input and produces a pair (e, d) of corresponding public encryption and private decryption keys, and a pair (f, g) of corresponding auxiliary public encryption and private decryption keys. We write $\mathcal{S}(k) = \{e, d, f, g\}$.*

- *The* client puzzle-generation algorithm $\mathcal{G}$ *is a randomized algorithm that takes encryption keys e and f as input and produces a puzzle in two parts p_0 and p_1. We write $\mathcal{G}(e, f) = \{p_0, p_1\}$.*

- *The* recovery algorithm $\mathcal{R}$ *is a deterministic algorithm that takes a public encryption key e, a private decryption key d, and two puzzle pieces (p_0, p_1), as inputs, and produces the solution $m \in \mathcal{M}$ to the puzzle or a special* reject *symbol. We write $\mathcal{R}_{e,d}(p_0, p_1) = m$.*

- *The* client algorithm $\mathcal{C}$ *is a deterministic algorithm that takes p_1 and outputs a value p_2. We write $\mathcal{C}(p_1) = p_2$.*

- *The* client checking algorithm $\mathcal{H}$ *is a deterministic algorithm that takes p_0, p_2, and r as inputs, and produces the solution $m \in \mathcal{M}$ to the puzzle or a special* reject *symbol. We write $\mathcal{H}(p_0, p_2, r) = m$. $\mathcal{H}$ should require fewer time steps than $\mathcal{R}$.*

We now show how the client-generated useful puzzle satisfies computational intensity, reliability, usefulness, non-dependability, and security. As long as the client-generated m is a useful piece of information, the usefulness requirement is fulfilled. The client checking algorithm $\mathcal{C}$ insures reliability and produces m. The recovery algorithm $\mathcal{R}$ insures non-dependability as m can be recovered without using $\mathcal{C}$. As long as no information about m, d, and g is leaked, security is maintained. The computational intensity comes from the computational complexity of $\mathcal{C}$, and as long as the complexity of $\mathcal{H}$ is less than the complexity of $\mathcal{C}$, then work is being saved.

After describing our implementations in Sections 6.2 and 6.3, we will prove that security and computational intensity are maintained. Then in Section 6.4, we will show an example of when a client-generated m can be useful.

6.2 An Honest-but-Curious Useful Puzzle Implementation

The overall idea is that one client, C_0, will encrypt a message with the dual receiver encryption scheme using the server's public key and an auxiliary public key, and send the ciphertext to the server. The server will split the ciphertext into two parts, p_0 and p_1. The server will then send p_1 and the auxiliary private key to a different client, C_1, to do a computationally intense partial decryption. Given this partial decryption and p_0, the server will be able to decrypt the message. Note that the server can do the full decryption itself as it has the private key corresponding to its public key.

This implementation, which we call CUP1, is insecure if an adversary can listen in on all the channels. Given the auxiliary private key, anyone can decode the ciphertext. If the client C_1 is listening in, she can decode the ciphertext as well. We present CUP1 as a version of a client-generated useful puzzle that is secure when everyone is honest-but-curious. It has the advantage that the server does very little work. Let Enc$= (\mathcal{K}, \mathcal{E}, \mathcal{D}, \mathcal{R})$ be as in Section 4.1 in the following description.

Setup. Groups $\mathbb{G}_1$ and $\mathbb{G}_2$ are chosen using a BDH parameter generator $\mathcal{G}$, along with a random element $g \in \mathbb{G}_1$, and $x, y \in \mathbb{Z}^*_{q_1}$. The server's public key is (g, g^x) together with a cryptographic hash function $H_x : \mathbb{G}_2 \to \{0,1\}^n$. The private key is x. An auxiliary public key g^y is generated as well, with corresponding private key y.

Puzzle-Generation. To generate the puzzle, the server sends the auxiliary public key to a client, C_0. The client uses the encryption algorithm $\mathcal{E}$ from the dual receiver cryptosystem to encrypt a message m, and sends the cipher text $(u_1 = g^c, u_2 = \rho \oplus H_x(e(g^x, g^y)^c), u_3 = m \oplus G(\rho), u_4 = F(\rho, m, u_1, u_2, u_3))$ to the server. We set $p_0 = u_4$ and $p_1 = (u_1, u_2, u_3)$.

Recovery. Given the ciphertext (u_1, u_2, u_3, u_4), the server can use the decryption algorithm $\mathcal{D}$ and its private key x to recover the message m.

Client Algorithm. Given g^x, H_x, y, and p_1, the client C_1 computes $p_2 = H_x(e(g^x, u_1)^y) \oplus u_2$ and sends p_2 to the server.

Checking. The server computes $m = G(p_2) \oplus u_3$. The server accepts C_1's computation if $u_4 = F(\rho, m, u_1, u_2, u_3)$.

We now argue that CUP1 satisfies the useful puzzle requirements given the caveat that all parties can only listen in on their own channel.

Computational Intensity. It takes the server one bitwise XOR and a check of a simple has function to recover the message. The client has to do the same in the client algorithm, but also has to do an expensive pairing computation and exponentiation.

Reliability. The hash check in the Recovery algorithm assures the faithful operation by the client.

Non-dependability. If the Checking algorithm fails to pass verification, the server can do the puzzle itself from scratch using the Recovery algorithm, or give it to another client.

Usefulness. The usefulness will be discussed in Section 6.4.

Security. The scheme is a chosen-ciphertext secure public key encryption. The client, seeing only part of the ciphertext that recovers to a random value ρ, has no idea what the message is (she only sees values that could have been computed without seeing the message).

6.3 A Secure Implementation of the Useful Puzzle

In this implementation, the server outsources the work of computing the pairing function to a client. The overall idea is that one client, C_0, will encrypt a message with the dual receiver encryption scheme using the server's public key and an auxiliary public key, and send the ciphertext to the server. The server will split the ciphertext into two parts, p_0 and p_1, and send p_1 to a different client, C_1, to do the computationally intense pairing computation. Given the client's output, p_2, and p_0, the server will be able to decrypt the message. Note that the server can do the full decryption itself as it has the private key corresponding to its public key.

This implementation, which we call CUP2, has the disadvantage that the server has to do an exponentiation. Let Enc$=(\mathcal{K}, \mathcal{E}, \mathcal{D}, \mathcal{R})$ be as in Section 4.1 in the following description.

Setup. Groups $\mathbb{G}_1$ and $\mathbb{G}_2$ are chosen using a BDH parameter generator $\mathcal{G}$, along with a random element $g \in \mathbb{G}_1$, and $x, y \in \mathbb{Z}_{q_1}^*$. The value $e(g, g)$ is precomputed. The server's public key is (g, g^x) together with a cryptographic hash function $H_x : \mathbb{G}_2 \to \{0,1\}^n$. The private key is x. An auxiliary public key g^y is generated as well, with corresponding private key y.

Puzzle-Generation. To generate the puzzle, the server sends the auxiliary public key to a client, C_0. The client uses the encryption algorithm $\mathcal{E}$ from the dual receiver cryptosystem to encrypt a message m, and sends the cipher text $(u_1 = g^c, u_2 = \rho \oplus H_x(e(g^x, g^y)^c), u_3 = m \oplus G(\rho), u_4 = F(\rho, m, u_1, u_2, u_3))$ to the server. We set $p_0 = (u_2, u_3, u_4)$ and $p_1 = u_1$.

Recovery. Given the ciphertext (u_1, u_2, u_3, u_4), the server can use the decryption algorithm $\mathcal{D}$ and its private key x to recover the message m.

Client Algorithm. Given p_1, the client C_1 computes $p_2 = e(g, p_1)$ and sends p_2 to the server.

Checking. The server computes: $(p_2 \cdot e(g, g))^{xy} := v$, and then computes $\rho = u_2 \oplus H_x(v)$ and $m = G(\rho) \oplus u_3$. The server accepts C_1's computation if $u_4 = F(\rho, m, u_1, u_2, u_3)$.

Computational Intensity. It takes the server one exponentiation and two multiplications to check the client's work. This constant number of exponentiations requires much less work on the part of the server than the pairing computation done by the clients.

Reliability. The hash check in the Recovery algorithm assures the faithful operation by the client.

Non-dependability. If the Recovery algorithm fails to pass verification, the server can do the puzzle itself from scratch, or give it to another client.

Usefulness. The usefulness will be discussed in Section 6.4.

Security. The scheme is a chosen-ciphertext secure public key encryption. The security of the puzzle is maintained as the only public values are the public keys, uniform random values, and the cipher text of the chosen-ciphertext secure encryption scheme.

6.4 Employing Useful Security Puzzles in TLS

We now briefly discuss an application of CUP2, which trades off security against certain kinds of eavesdropping adversaries with resistance to computational denial of service attacks. Consider a cryptographic protocol such as TLS, a simplified version of which is shown in Figure 1 (inspired by a similar figure in [39]). In Message 3 of this protocol, the client encrypts a randomly chosen secret value, m, with the server's public key (obtained from the certificate sent by the server in Message 2). The server must decrypt this secret value, which both parties use to derive a session key. In almost all cases, the RSA algorithm is used to encrypt m. Note that the server also uses its private key to authenticate (via a signature) to the client.

We envision using useful security puzzles as a substitute for the RSA encryption shown in Figure 2. The server will have a long-term public/private key pair (e and d), and will periodically select a new auxiliary key pair (f and g). A client C_0 that contacts the server will receive both public keys (f and e). The client will then select a secret m, which will be encrypted along with a server-provided stateless nonce N_s, using both public keys creating ciphertext p_0, p_1, as described in Section 6.3. If the server is lightly loaded, it may simply decrypt this value using g and d itself. Otherwise, the server selects another client, C_1, at random and forwards p_1 to it. On a busy server, such as a popular e-commerce web site, there will be a constant stream of new clients connecting, to which p_1 can be forwarded to. Similarly, the original client may receive another p_1', produced by another client connecting to the server.

Client C_1 will now use e and f to produce the intermediate value p_2 and send it back to the server as proof of work done. The server will verify the solution (as described in Section 6.3) and the nonce, and will allow the connection from client C_1 to proceed. At the same time, the server has retrieved the secret value m produced by client C_0 for use in deriving a session key. The purpose of the nonce is to force colluding clients (*e.g.,*

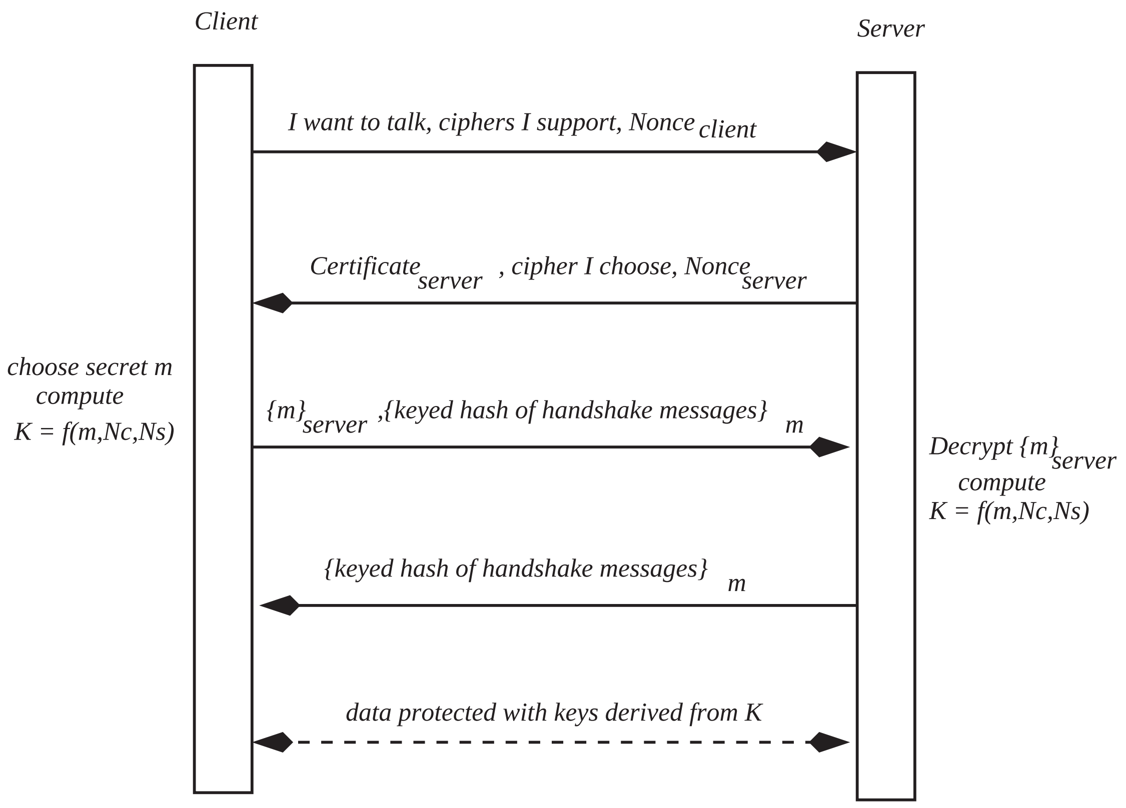
Client
Server
I want to talk, ciphers I support, Nonce client
Certificate server , cipher I choose, Nonce server
choose secret m
compute
K = f(m,Nc,Ns)
{m} server ,{keyed hash of handshake messages} m
Decrypt {m} server
compute
K = f(m,Nc,Ns)
{keyed hash of handshake messages} m
data protected with keys derived from K

Figure 1: Simplified SSLv3/TLS.

machines controlled by the same attacker) to communicate with each other, mitigating the impact of an influx of such clients on the throttling properties of our scheme. If client C_0 also provides a correct response to the challenge p'_1 (which may have been generated by client C_1 or some other client contacting the server in the same window of time as C_0), it will be allowed to proceed with its connection. Neither C_0 nor C_1 have learned anything about the secret values they helped decrypt, nor have they learned anything that would allow them to impersonate the server to other clients (*e.g.,* the server's private key). The server has throttled down the clients by forcing them to perform some useful computation; under previous such schemes [31], the client would have to perform the same work in addition to solving a "useless" puzzle, while the server itself would have to do more of the protocol's cryptographic work.

Our two schemes, CUP1 and CUP2, offer different tradeoffs in terms of security and performance improvement to the server. Although CUP2 is not susceptible to a passive eavesdropping adversary (compared to CUP1), this security comes at the price of increased processing cost. In particular, CUP2 requires exponentiation operations of the same complexity as the original (standard) TLS protocol with RSA encryption. Thus, other than throttling down clients, the CUP2 scheme does not improve the server performance, as CUP1 does. On the other hand, if a protocol (*e.g.,* a future version of TLS) already uses pairing-based cryptography, CUP2 can both improve performance and increase resistance to DoS attacks.

If we use the client-generated useful puzzle CUP1 in this environment the server's work would be limited to hash functions and XOR operations. Unfortunately, a powerful adversary that is capable of monitoring all the server's communication links can obtain enough information (specifically, $\{p_0, p_1, g\}$ or $\{p_0, p_1, p_2\}$) to decrypt the original message from the client to the server, thus violating the security of the TLS-like protocol. From a denial-of-service perspective, such an adversary can potentially perform much more powerful attacks (*e.g.,* shut down these links); however, this scheme has the potential to make things worse from a security perspective.

We offer two potential approaches to mitigating the threat when using CUP1. First, we can treat the first iteration of the TLS-like protocol as a pre-authentication phase, establishing a key which can be used to quickly validate the client's traffic to the server's router using a scheme such as the one proposed by Yaar, Perrig and Song [56]; a second authentication phase (without using puzzles) is subsequently used to secure the end-to-end path. Second, we can use a distributed set of servers through which the main server routes (and receives) Messages 3, 4 and 5 (per Figure 2). These messages are transmitted under pre-established security associations, preventing an attacker eavesdropping on the server's direct links from obtaining enough information. Such an

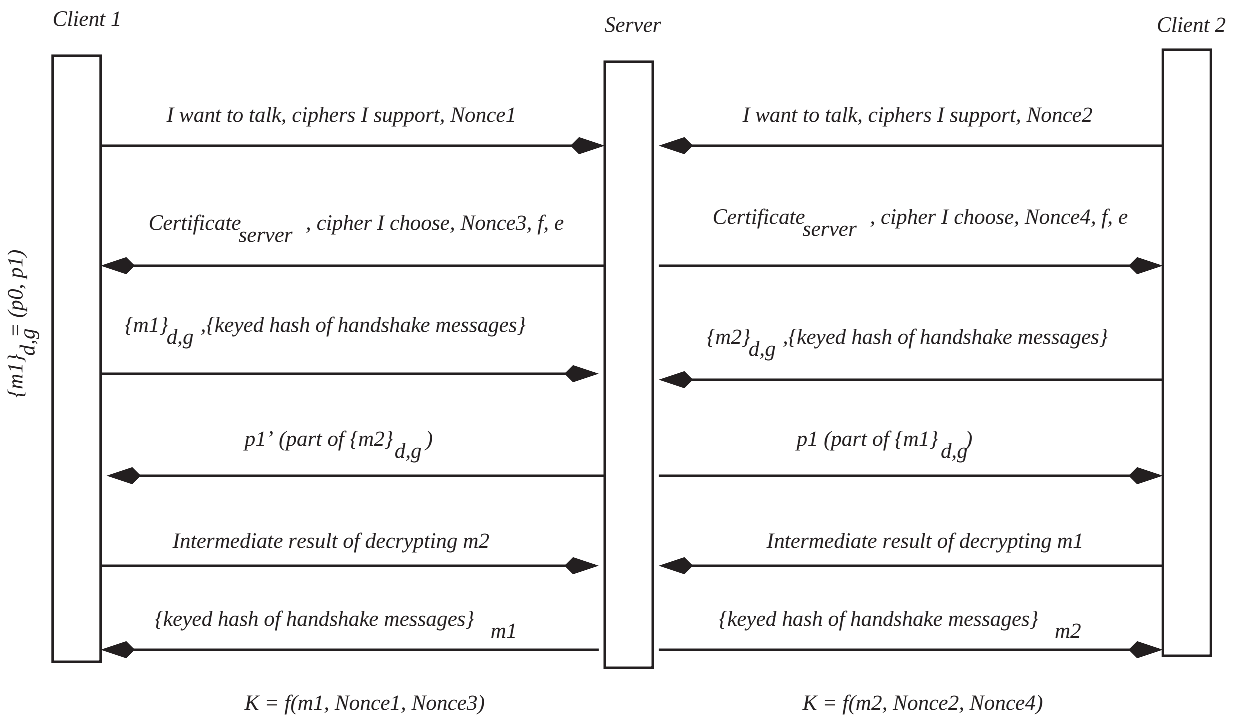
Client 1
Server
Client 2
I want to talk, ciphers I support, Nonce1
I want to talk, ciphers I support, Nonce2
Certificate server , cipher I choose, Nonce3, f, e
Certificate server , cipher I choose, Nonce4, f, e
{m1} d,g = (p0, p1)
{m1} d,g ,{keyed hash of handshake messages}
{m2} d,g ,{keyed hash of handshake messages}
p1' (part of {m2} d,g)
p1 (part of {m1} d,g)
Intermediate result of decrypting m2
Intermediate result of decrypting m1
{keyed hash of handshake messages} m1
{keyed hash of handshake messages} m2
K = f(m1, Nonce1, Nonce3)
K = f(m2, Nonce2, Nonce4)

Figure 2: A TLS-like protocol using useful puzzles. For simplicity, only two clients are shown, each partially decrypting the other's secret value m on behalf of the server.

attacker would instead need to eavesdrop the links for all these servers. Recent work has shown that such overlay-based mechanisms offer reasonable security guarantees [37] and performance characteristics [2].

CUP1 has the further problem that the auxiliary public key needs to be periodically refreshed to prevent an attacker from posing as a client and using the auxiliary private key to decrypt. If the key is refreshed for each message then the server is doing an exponentiation at each step which is exactly what we wanted to avoid. The server can avoid this problem by having several auxiliary keys prepared and then giving different keys to different clients. Depending on the amount of traffic the server is experiencing the keys should be periodically refreshed.

6.5 Protecting Honest Users

A potential problem with the CUP2-TLS scheme is that an adversarial client could generate a bad ciphertext (puzzle), so that an honest client will not be able to verify itself. Our approach to this problem is to provide the same puzzle to multiple clients and, conversely, to require clients to solve several puzzles. Furthermore, the server injects some puzzles who solutions are already known, *e.g.,* puzzles that were solved in the past by other clients, or puzzles that were solved by the server during times of light load. All puzzles (including those whose solution is already known) are "anonymized" by the server, such that colluding clients do not know that they are solving the same puzzle.

First, we add a new algorithm to CUP2 and change the client and checking algorithms accordingly:

Privacy-preservation. Given p_1, the server generates $r \in_R \mathbb{Z}^*_{q_1}$ and computes $\tilde{p}_1 = g^r \cdot p_1$.

Client Algorithm. Given $\tilde{p}_1$, the client C_1 computes $p_2 = e(g, \tilde{p}_1)$ and sends p_2 to the server.

Checking. The server computes $e(g,g)^{-r}$ and then computes: $(p_2 \cdot e(g,g)^{-r})^{xy} := v$. The server computes $\rho = u_2 \oplus H_x(v)$ and $m = G(\rho) \oplus u_3$. The server accepts C_1's computation if $u_4 = F(\rho, m, u_1, u_2, u_3)$.

Let the modified version of CUP2 be called CPUP (for Client-generated Privacy-preserving Useful Puzzle). Note that the privacy-preserving algorithm ensures that the outsourced computation is independent of the original puzzle, and the client doing the computation has no idea which puzzle it is associated with. This allows the server to send the same puzzle out to multiple clients; colluding clients will not be able to share their answers.

If the server sends out ζ different puzzles to each client, let α be the number of puzzles that the server already knows the answer to, thus leaving $\zeta - \alpha$ puzzles that the server actually needs solved. Again, note that the same client puzzle will be sent to a number of clients. The server will dismiss all clients that do not solve the α known puzzles correctly. The probability that a client that computes β puzzles correctly will have computed the α known puzzles among them is at most: $\binom{\zeta-\alpha}{\beta-\alpha}/\binom{\zeta}{\beta} + |\mathbb{G}_2|^{-1}$. For example, an honest client will compute the puzzles with probability 1 ($\beta = \zeta$). A client trying to guess which are the α and only computing those will be correct with probability $\binom{\zeta}{\alpha}^{-1}$ with some negligible probability of randomly guessing the correct answers. Any client computing fewer than α puzzles will have no better than a $|\mathbb{G}_2|^{-1}$ chance of guessing the correct values.

After dismissing the "obviously" malicious clients by checking the known solutions, the server will run the checking algorithm on the computations of the remaining clients. If there is any disagreement among the clients' answers, the server will take the majority's vote and dismiss the minority clients irrespective of whether or not the checking algorithm accepts the answer. Thus, if there are even only two honest clients, two malicious clients will have to give the same answer for all $\zeta - \beta$ puzzles for the server to be fooled. Since each copy of each puzzle is independently generated uniformly at random, the probability of this happening is at most $|\mathbb{G}_2|^{-(\zeta-\beta)}$. If a majority of the clients give the same answer to a puzzle, but the client algorithm does not accept the answer, then the client that generated the puzzle will be dismissed. Thus, honest clients can only fall victim to malicious adversaries with a very small probability that is adjustable by the server.

In terms of the server's computation, adding the privacy-preservation property adds two more exponentiations. Thus it takes the server three exponentiations and two multiplications to check the client's work in CPUP. This constant number of exponentiations requires much less work on the part of the server than the pairing computation done by the clients.

7 Conclusions

We introduced the notion of a "dual receiver cryptosystem" which enables a ciphertext to be decrypted by two independent receivers. We presented a construction and illustrated its use in two important applications that address heretofore open problems in the literature.

The first application, first suggested by Dwork and Naor, is a client-generated useful puzzle scheme. Here, a server can effectively delegate the decryption of an encrypted message to a client in the form of a puzzle. The puzzle-solving client facilitates the decryption without learning anything about the encrypted message or the server's private key. The remaining cryptographic workload for the server (including verification that the puzzle was correctly solved) is reduced to a bitwise XOR and the computation of a simple hash in our first construction. We believe that this scheme will have important applications in preventing denial-of-service attacks, and we explore its use in a TLS-like protocol. The happy irony is that a DoS attacker that seeks to shut down a server by inducing it to perform computationally intensive cryptographic computations, is forced to facilitate the server's pending cryptographic tasks on behalf of legitimate clients.

We are still left with the open question of whether or not one can construct a useful puzzle in a TLS-like protocol that: (1) does not require the server to do any exponentiations; and (2) is secure against adversaries that can monitor all communications between the server and the clients. Our first protocol fulfills the first requirement, and our second protocol fulfills the second. Ideally, we would like the best of both worlds.

The second application, inspired by the work of Haber and Pinkas, is a "combined cryptosystem" wherein multiple participants, each maintaining only a single public/private-key pair, can both encrypt and sign messages, and can also delegate decryption (escrow) capabilities to a specified user (on a per-message basis, if desired). The escrow is achieved without compromising the security of the signature scheme or the security of any other message encryption.

References

[1] B. Aiello, S. Bellovin, M. Blaze, R. Canetti, J. Ioannidis, A. Keromytis, and O. Reingold, "Efficient, dos-resistant secure key exchange for Internet protocols," *ACM Computers and Communications Security conference (CCS)*, pp. 48-58, 2002.

[2] D. G. Andersen, H. Balakrishnan, M. F. Kaashoek, and R. Morris, "Resilient overlay networks,"*Proceedings of the 18th Symposium on Operating Systems Principles (SOSP)*, Oct. 2001.

[3] M. Abadi, M. Burrow, M. Manasse, and T. Wobber, "Moderately hard, memory-bound functions," *Proceedings of the ISOC Symposium on Network and Distributed Systems Security (SNDSS)*, Feb. 2003.

[4] T. Aura, J. Leiwo, and P. Nikander, "Dos-resistant authentication with client puzzles," *Proceedings Security Protocols Workshop 2000*, LNCS 2133, pp. 170-181, 2000.

[5] T. Aura, and P. Nikander, "Stateless con-nections," *Proceedings of International Conferenec on Information and Communi-cations Security (ICICS)*, LNCS 1334, pp. 87-97, *Springer-Verlag*, Nov. 1997.

[6] A. Back, "Hashcash- A denial of service counter-measure", (http: //www.cypherspace.org/hashcash/hashcash.pdf)

[7] D. Boneh, and M. Franklin, "Identity-based encryption from the Weil pairing," Joe Kilian, editor, *Advances in Cryptology CRYPTO 2001, LNCS 2139*, pp. 213-229. *Springer-Verlag*, 2001.

[8] D. Boneh, B. Lynn, and H. Shacham, "Short signatures from the Weil pairing," Colin Boyd, editor, *Advances in Cryptology - ASIACRYPT 2001, LNCS 2248*, pp. 514V532. *Springer-Verlag*, 2001.

[9] D. Boneh, and M. Naor, "Timed commitments (extended abstract)," *Proceedings of CRYPTO*, pp. 236V254, Aug. 2000.

[10] M. Bellare, and P. Rogaway, "The exact security of digital signature - how to sign with RSA and Rabin,"

Ueli Maurer, editor, *Advances in Cryptology -EUROCRYPT' 96, LNCS 1070*, pp. 399V416m *Springer-Verlag*, 1996.

[11] Y. Chen, S. Das, P. Dhar, A. El Saddik, and A. Nayak, "Detecting and preventing IP-spoofed distributed DoS attacks," *International Journal of Network Security (IJNS)*, vol. 7, no. 1, pp. 69-80, July 2008.

[12] J. S. Coron, H. Handschuh, M. Joye, P. Paillier, D. Pointcheval, and C. Tymen, "GEM: A generic chosen-ciphertext secure encryption method," Bart Preneel, editor, *Topics in Cryptology - CTRSA* 2002, *LNCS 2271*, *Springer-Verlag*, pp. 263V276, 2002.

[13] J. S., M. Joye, D. Naccache, and P. Paillier, "Universalpadding schemes for RSA," *Advances in Cryptology (CRYPTO'2002)*, LNCS 2442, pp. 226-241, 2002.

[14] W. E. Difie, and M. E. Hellman, "New directions in cryptography," *IEEE Transaction on Information Theory*, IT vol. 22, no. 6, pp. 644-654, Nov. 1976.

[15] T. Diament, H. K. Lee, A. D. Keromytis, and M. Yung, "The dual receiver cryptosystem and its applications," *Proceedings of the 11th ACM conference on Computer and Communications Security(CCS)*, pp. 330V343, Oct. 2004.

[16] C. Dwork, and M. Naor, "Pricing via processing, or combating junk mail," *Proceedings of CRYPTO*, pp. 139V147, Aug. 1992.

[17] D. Dean, and A. Stubbleeld, "Using client puzzles to protect TLS," *Proceedings of the 10th USENIX UNIX Security Symposium*, Aug. 2001.

[18] P. Fouque, and D. Pointcheval, "Thrshold cryptosystems secure against chosen-ciphertext attacks," *Advances in Cryptology - ASIACRYPT 2001*, LNCS 2248, pp. 351-368, 2001.

[19] Y. Frankel, and M. Yung, "Escrowencryption systems visited: Attacks, analysis and designs," *Advances in Cryptology (CRYPTO'1995)*, LNCS 963, pp. 222-235, 1995.

[20] J. A. Garay, and M. Jakobsson, "Timed release of standard digital signatures," *Proceedings of the 6th Conference on Financial Cryptography*, pp. 168-182, Feb. 2002.

[21] V. D. Gligor, "Guaranteeing access in spite of distributed service-flooding attacks," *Proceedings of the Security Protocols Workshop*, Apr. 2003.

[22] S. Goldwasser, and S. Micali, "Probabilistic encryption," *Journal of Computer and System Sciences*, vol. 28, no. 2, pp. 270-299, Apr. 1984.

[23] S. Goldwasser, S. Micali, and R. Rivest, "A digital signature scheme secure against adaptive chosen-message attacks," *SIAM Journal on Computing*, vol. 17, no. 2, pp. 281-308, Apr. 1988.

[24] C. Gentry, and A. Silverberg, "Hierarchical ID-based cryptography," *Advances in Cryptology (ASIACRYPT'2002)*, LNCS 2501, pp. 548-566, 2002.

[25] C. Gong, and K. Sarac, "Toward a practical packet marking approach for IP traceback," *International Journal of Network Security (IJNS)*, vol. 8, no. 3, pp. 271-281, May 2009.

[26] L. T. Heberlein, and M. Bishop, "Attack class: Address spoofing," *Proceedings of the 19th National Information Systems Security Conference*, pp. 371-377, Oct. 1996.

[27] D. Harkins, and D. Carrel, *The Internet Key Exchange (IKE) Request for Comments (Proposed Standard) 2409,* Internet Engineering Task Force, Nov. 1998.

[28] S. Hirose, and K. Matsuura, "Enhancing the resistance of a provably secure key agreement protocol to a denial-of-service attack," *Proceedings of the 2nd International Conference on Information and Communication Security (ICICS)*, pp. 169-182, Nov. 1999.

[29] S. Haber, and B. Pinkas, "Securely combining public-key cryptosystems," Pierangela Samarti, editor, *Proceeding 8th ACM Conference on Computer and Communications Security*, *ACM Press*, pp. 215-224, 2001.

[30] K. Houle, G. Weaver, N. Long, and R. Thomas, "Trends in denial of service attack technology," CERT and CERT Coordination Center, 2001. (http://www.cert.org/archive/pdf/DoS_trends.pdf)

[31] A. Juels, and J. Brainard, "Client puzzles: A cryptographic countermeasure against connection depletion attacks," *Proceedings of the ISOC Symposium on Network and Distributed Systems Security (SNDSS)*, pp. 151-165, Feb. 1999.

[32] M. Jakobsson, and A. Juels, "Proofs of work and bread pudding protocols," *Proceedings of the IFIP TC6 and TC11 Joint Working Conference on Communications and Multi-media Security*, Sep. 1999.

[33] A. Joux, and K. Nguyen, "Separating decision difie-hellman from diffie-hellman in cryptographic groups," 2001. (http://eprint.iacr.org)

[34] A. Joux, "A one-round protocol for tripartite Diffie-Hellman," Wieb Bosma, editor, *Proceeding Algorithmic Number Theory, 4th International Symposium (ANTS-IV)*, LNCS 1838, pp. 385-394, 2000.

[35] P. Janson, G. Tsudik, and M. Yung, "Scalability and flexibility in authentication services: The KryptoKnight approach," *Proceedings of IEEE INFOCOM*, pp. 725-736, Apr. 1997.

[36] N. Koblitz, and A. J. Menezes, "Another Look at 'Provable Security', Manuscript, 2004. (http://eprint.iacr.org)

[37] A. D. Keromytis, V. Misra, and D. Rubenstein, "SOS: Secure overlay services," *Proceedings of ACM SIGCOMM*, pp. 61-72, Aug. 2002.

[38] Y. Komano, and K. Ohta, "Efficient universal padding techniques for multiplicative trap-door one-way functions," Dan Boneh, editor, *Advances in Cryptology - CRYPTO 2003*, LNCS 2729, pp. 366V38, 2003.

[39] C. Kaufman, R. Perlman, and M. Speciner, *Network Security*, 2nd Edition, Prentice Hall, 2002.

[40] P. Karn, and W. Simpson, "Photuris: Session key management protocol," Request for Comments (Experimental) 2522, *Internet Engineering Task Force*, Mar. 1999.

[41] J. Lemmon, "Resisting SYN-flood DoS attacks with a SYN cache," *Proceedings of the USENIX BSD Conference (BSDCon)*, Feb. 2001.

[42] M. Lee, Y. He, and Z. Chen, "Towards improving an algebraic marking scheme for tracing DDoS attacks," *International Journal of Network Security (IJNS)*, vol. 9, no. 3, pp. 204-213, Nov. 2009.

[43] J. Leiwo, P. Nikander, and T. Aura, "Towards network denial of service resistant protocols," *Proceedings of the 15th International Information Security Conference (IFIP/SEC)*, Aug. 2000.

[44] M. Naor, and M. Yung, "Public-key cryptosystems provably secure against chosen ciphertext attacks," *Proceedings 22nd Annual ACM Symposium on Theory of Computing(STOC)*, LNCS 547, pp. 427-437, 1990.

[45] T. Okamoto, and D. Pointcheval, "REACT: Rapid enhanced-security asymmetric cryptosystem transform," *Topics in Cryptology (CT-RSA'2002),* LNCS 2271, pp. 159-175, 2002.

[46] R. Oppliger,"Protecting key exchange and management protocols against resource clogging attacks," *Proceedings of the IFIP TC6 and TC11 Joint Working Conference on Communications and Multimedia Security (CMS)*, pp. 163-175, Sep. 1999.

[47] B. Pinkas, "Personal communication,"

[48] C. Racko, and D. R. Simon, "Non-interactive zero-knowledge proof of knowledge and chosen ciphertext attack," *Advances in Cryptology (CRYPTO'1991)*, LNCS 576, pp. 433-444, 1991.

[49] R. Rivest, and A. Shamir, "PayWord and micromint," *CryptoBytes*, vol. 2, no. 1, pp. 7-11, 1996.

[50] R. L. Rivest, A. Shamir, and D. A. Wagner, *Time-lock Puzzles and Timed-release Crypto,* Technical Report MIT/LCS/TR-684, MIT, 1996.

[51] C. Schuba and I. Krsul and M. Kuhn and E. Spafford and A. Sundaram and D. Zamboni, "Analysis of a denial of service attack on TCP," *Proceedings of IEEE Security and Privacy Conference*, pp. 208-223, May 1997.

[52] Z. J. Shi and H. Yan, "Software implementations of elliptic curve cryptography," *International Journal of Network Security (IJNS)*, vol. 7, no. 1, pp. 141-150, July 2008.

[53] E. R. Verheul, "Evidence that XTR is more secure than supersingluar elliptic curve cryptosystems," *Proceedings of Advances in Cryptology (EUROCRYPT'2001)*, LNCS 2045, pp. 195-210, 2001.

[54] X. Wang and M. K. Reiter, "Defending against denial-of-service attacks with puzzle auctions (extended abstract)," *Proceedings of the IEEE Symposium on Security and Privacy*, May 2003.

[55] A. D. Wood and J. A. Stankovic, "Denial of service in sensor networks," *IEEE Computer*, vol. 35, no. 10, pp. 54-62, 2002.

[56] A. Yaar and A. Perrig and D. Song, "Pi: A path identification mechanism to defend against DDoS attacks," *Proceedings of the IEEE Symposium on Security and Privacy*, May 2003.

[57] R. Zhang and G. Hanaoka and H. Imai, "A generic construction of useful client puzzles," *Proceedings of the 4^{th} International Symposium on Information, Computer, and Communications Security (ASIACCS)*, pp. 70-79, 2009.

Ted Diament has worked at Google, Inc. and was a Ph.D. student at Columbia University during the time the work described in this paper was conducted.

Homin K. Lee is a postdoctoral student with the University of Texas at Austin. He received his Ph.D. in Computer Science from Columbia University in New York.

Angelos D. Keromytis is an Associate Professor of Computer Science and the Director of the Network Security Laboratory at Columbia University in New York. He received his Ph.D. and M.Sc. from the University of Pennsylvania, and his B.Sc. from the University of Crete, in Greece.

Moti Yung works at Google, Inc. and is a visiting research scientist in the Computer Science Department at Columbia University in New York. Previously, Moti has also been an industry consultant, the Director of Advanced Authentication Research at RSA Laboratories, the Chief Scientist of CertCo and a researcher at IBM's T.J. Watson Research Center. He received his Ph.D. in computer science from Columbia University in New York.