Oblivious Pseudorandom Functions and Some (Magical) Applications

Hugo Krawczyk

This presentation is based on the following research papers:

https://eprint.iacr.org/2019/1275


https://eprint.iacr.org/2017/363
Oblivious PRF (OPRF)

- $f_k(x)$ is a Pseudo-Random Function (PRF) if $F_k(x)$ or $\mathbb{S}$

- $F_k(x) \text{ or } \mathbb{S}$

- $\text{Adv}$

OPRF protocol

- S(k)
- F_k
- Nothing
- C(x)
- F_k(x)

- OPRF: An interactive PRF “service” that returns PRF results without learning the input or output of the function

- A POWERFUL primitive
DH-OPRF
The Diffie-Hellman Problem

- Cyclic group $G$ of prime order $q$ with generator $g$
  - $G = \{1, g, g^2, ..., g^{q-1}\}$
  - Crucial property: for all $x, y$ in $\{0...q-1\}$: $g^{xy} = (g^x)^y = (g^y)^x = g^{yx}$

- “Diffie-Hellman problem”: Given $g^x$ and $g^y$, it’s hard to compute $g^{xy}$

- “One-More DH Assumption”:
  - Given $(g, g^k, g_1, g_2, ..., g_m)$ and $Q$ calls to a $k$-exponentiation oracle $(\cdot)^k$
  - Cannot output $g_i^k$ for more than $Q$ elements in $\{g_1, g_2, ..., g_m\}$

- We will also need: Hash function $H$ that maps arbitrary strings to random elements in $G$ (“random oracle model”)

**DH-OPRF**

- **PRF**: $F_k(x) = H(x)^k$ ; input $x$, key $k$ from $0...q-1$

- Oblivious computation via Blind DH Computation (S has $k$, C has $x$)

  - **S**: key $k$
    
    $$a = (H(x))^r$$

  - **C**: input $x$
    
    random $r$

    $$b = a^k$$

    Computes $H(x)^k \leftarrow b^{1/r}$

- $b^{1/r} = (a^k)^{1/r} = (((H(x))^r)^k)^{1/r} = (((H(x))^k)^r)^{1/r} = (H(x))^k$

- The blinding factor $r$ works as a one-time encryption key: 
  
  *hides* $H(x)$, $x$ and $F_k(x)$ perfectly from $S$ (and from any observer)
**DH-O prF**

\[ H'(x, H(x)^k) \]

- **PRF:** \( F_k(x) = H(x)^k \); input \( x \), key \( k \) from 0...\( q-1 \)

- **Oblivious computation via Blind DH Computation (S has \( k \), C has \( x \))**

  **S:** key \( k \)

  \[ a = (H(x))^r \]

  \[ b \neq a^k \]

  \[ \text{Computes } H(x)^k \leftarrow b^{1/r} \]

  **C:** input \( x \)

  random \( r \)

- **Computational cost:** one round, 2 exponentiations for C, one for S

  - **Commodity laptop:** > 10,000 exponentiations/second
  - **Variant:** fixed base exponentiation for C (even faster)
DH-OPRF

- Long history (blinded DH): [..., CP'93, SY'96, HFH'99, FK'00, AES'03, JL'10,...],
- \(H'(H(x)^k)\) treated as PRF in [NPR’99] and as OPRF in [JL’10]
- Variants \((H(x))^k, H'(H(x)^k), H'(x, H(x)^k),...\)
- Security [JL’10, JKK’14]: Secure as OPRF in the Random Oracle Mode assuming Gap-One-More-DH [BNPS’03]
- DH-OPRF: Most efficient OPRF implementation (elliptic curves)
- *Defining OPRF*: Tricky notion \(\rightarrow\) many definitions (balancing security, utility, performance)
Many applications

- Private set intersection: HFH’99, FIPR’05, JL’10, CT’10, ..., PSZ’14’15, KRRT’16, ...
- Private Keyword Search (Keyword OT/PIR) [FIPR’05]
- Pattern matching [HL08, FHV13]
- De-duplication (files, medical records, etc.) [BKR’13, BCAPR’17]
- Chameleon pseudonyms, oblivious tokenization [CL’17]
- Search on Encrypted Data [CJJKRS’13, CJKRS’13]: Uses DH-OPRF “non-interactively” by storing blinded copies of the OPRF key
New Applications

- Key management services (esp. cloud storage systems)
- Revamping the world of password protection...
What is a “Cloud KMS”?

Client (C)

Storage Server (StS)

Key Management Server (KmS)
Wrap-Unwrap Method: Wrapping

Client (C) -> (C, dek) --> Client Root Key

\[ \text{wrap} = \text{ENC}(\text{CRK}, \text{dek}) \]

\[ \text{(ObjId, wrap, Enc(dek, Obj))} \rightarrow \text{Storage Server (StS)} \]
Wrap-Unwrap Method: Unwrapping

\[(\text{ObjId}, \text{wrap}, e = \text{Enc}(dek, \text{Obj}))\]

\[
\text{Client (C)} \quad \text{CRK} \quad \text{Key Management Server (KmS)}
\]

\[
\text{Obj} = \text{Dec}(dek, e)
\]

\[dek = \text{DEC}(\text{CRK}, \text{wrap})\]
Cloud KMS — Weaknesses and Vulnerabilities

Vulnerable to Interception (e.g. TLS, CAs)

Vulnerable to KMS compromise (insiders, CDNs, middleboxes)

\[
\text{wrap} = \text{ENC}(\text{CRK}, \text{dek})
\]

\[
\text{dek} = \text{DEC}(\text{CRK}, \text{wrap})
\]
OPRF-based KMS

- OPRF replaces traditional wrap/unwrap approach

- \( \text{DEK} = \text{OPRF}(\text{key}=\text{CRK}, \text{input}=\text{DEK-id}) \), i.e., \( \text{DEK} = (H(\text{DEK-id}))^{\text{CRK}} \)
  - CRK is the client’s OPRF key, replaces the traditional wrapping key

- Keys (DEK) transmitted with perfect secrecy from network and insiders - no reliance on TLS or CA’s (even “PQ Secure”)

- KMS can’t determine which keys the user is accessing
Further Features of OPRF Approach

- **Verifiability**: If client has \( g^k \ (g \in G, \ k \text{ the client’s OPRF key}) \), it can verify that \( H(\text{DEK-ID})^k \) is correct, hence DEK is correct.

  - Note that if KMS returns wrong key/wrap data lost forever.

- **Reduced storage**: No need to store wraps in addition to key id’s; KMS can derive OPRF keys from a single key (reduces off-HSM storage).

- **Implicit authentication**: Bearer tokens, passwords, etc., input to OPRF provide authentication w/o KMS having to verify anything.

- **Threshold security**: Can distribute the OPRF into \( n \) servers (HSMs) with OPRF key secure as long as no more than \( t \) are compromised.
Threshold DH-OPRF (n-out-of-n)

- Single server solution: \( F_k(x) = (H(x))^k \) (H’ omitted for simplicity)
- Multi-server solution: Server \( S_i \) has share \( k_i \), \( k = k_1 + k_2 + \cdots + k_n \)
  \[ F_k(x) = (H(x))^{k_1} \cdot (H(x))^{k_2} \cdot \cdots \cdot (H(x))^{k_n} = (H(x))^{\sum k_i} \]
- U sends same \( a = (H(x))^r \) to each server; \( S_i \) returns \( b_i = a^{k_i} \);
  U deblinds all \( b_i \) and multiplies
- Efficiency: 2 exp’s for client (indep of n), 1 per server, 1 round
- Key \( k \) is never reconstructed: “function sharing” vs “secret sharing”
Threshold DH-OPRF (t-out-of-n)

- \( t\)-out-of-\( n \) threshold DH-OPRF: Each server \( S_i \) has share \( k_i \)

- \( F_k(x) \) computed from any set of \( t \) servers \( S_{i_1}, ..., S_{i_t} \)

  \[ F_k(x) = (H(x))^{\lambda_{i_1}k_{i_1}} \cdot (H(x))^{\lambda_{i_2}k_{i_2}} \cdot ... \cdot (H(x))^{\lambda_{i_t}k_{i_t}} \]

  - \( \lambda_{ij} \) is a Lagrange interpolation coefficient ("Shamir in the exponent")

- As before: key \( k \) is never reconstructed

  - Not even during generation/sharing: Distributed key generation
Threshold DH-OPRF (more goodies)

- Single client message $\rightarrow$ proxy-based threshold operation

- Verifiability: via ZK or interactive (latter good for proxy-based)
  - Still a single message from C, double the # of exp’s, still indep of n, t

- Distributed OPRF key generation (key never exists in one physical place)

- Share rebuilding

- Proactive security
Updatable Oblivious KMS

- KMS stores client's CRK $k$; Client stores $g$ and $y = g^k$
- To encrypt: Client sets $h = g^s$ (random $s$), sets $DEK = y^s$, stores $h$
  - $DEK = y^s = (g^k)^s = (g^s)^k = h^k$; Client can compute $h^k$ by itself w/o knowing $k$ !!
- To decrypt with $h$: Client sends $h^r$ (random $r$) to KMS, gets back $(h^r)^k$, deblinds $r$ to obtain $h^k$, sets $DEK = h^k$
  - Only decryption is interactive (at the cost of storing $h$), KMS learns nothing
- Non-interactive key update: KMS rotates $k$ to $k'$, sends $\Delta = k'/k$ to $C$, $C$ sets every DEK $h$ to $h^\Delta \rightarrow$ can decrypt with $k'$ but not with $k$
  - In regular KMS rotation, server is involved with each DEK update!
BIG MISSING PIECE:
DEFINITIONS and PROOFS
PPSS: Password Protected Secret Sharing

(password-protected distributed storage)
How to store a secret

- We want to protect *secrecy* and *availability* of information while remembering a *single* password

  - Single server = Single point of compromise for secrecy (offline dict attacks)
  - Single server = Single point of failure for availability (server gone, secret gone)

  ➔ Multi-server solution a must.

- Crypto solution: keep the secret encrypted in multiple locations; *secret share the encryption key* in multiple servers

  - Share among n servers, retrieve from t+1 servers (e.g. n=5, t=2)

- Protects availability and secrecy: *available* as long as t+1 available, *secret* as long as no more than t corrupted
Wait, but how do you authenticate to each server for share retrieval?

- Server needs to authenticate the user before delivering a share

- All we have is a user and a password
  - A strong independent password with each server? Not realistic
  - Same (or slight-variant) password for each server? Not good

⇒ Each server as a single point of compromise!

- From one point of compromise to n. We didn’t achieve much, did we?
Password Protected Secret Sharing (PPSS)

- **Init**: User secret shares a secret among n servers; *forgets secret* and keeps a *single password*.

- **Retrieval**: User contacts t + 1 servers, authenticates using the *single password* and reconstructs the secret.

- **Security**: Breaking into t servers leaks nothing about secret or password
  - Break = All server’s secret information leaks (shares, long-term keys, password file)
  - Only adversary option: Guess the password, try it in an online attack.
  - Offline attacks with ≤ t corrupted servers are useless.

+ **Soundness**: User *reconstructs the correct secret* or else rejects (CRUCIAL)

Note: No PKI except for Init, secure even if user forgets initialized servers
PPSS Solution = Threshold OPRF

- n servers share a Threshold OPRF $F_k(x)$
- U’s secret defined as $s=F_k(pwd)$
  - If U’s secret is not random (e.g., bitcoin), s can be used as an encryption key
- To retrieve $s$, U runs T-OPRF with any $t+1$ servers
- In more detail (adding crucial soundness):
  - U’s secret defined as $H(s,1)$
  - In addition to $k_i$, servers store $H(s,2)$, which they send to U together with OPRF response; if not all servers send $H(s,2)$, U aborts (soundness)
- Security bonus: Even if $t+1$ servers compromised, a full exhaustive offline attack needed to find password!
PPSS Efficiency (same as Threshold OPRF)

- Computation:
  - Single exponentiation for each server
  - Only two exponentiations in total for the client (independent of t and n)
  - t multiplications for client and for each server

- Communication: Single parallel message from user to t+1 servers, one msg back from each server. No inter-server communication.
  - No assumed PKI or secure channels (other than for initialization)
  - Any t, n (t ≤ n)

- Robustness: NIZK, interactive [2x expon], ACNP’16
Password-Authenticated Key Exchange (PAKE)
OPAQUE: Oblivious PAKE

- Asymmetric PAKE: User-Server password authentication (+ KE)
  - User has pwd, server stores pwd-related state (but not pwd!)
  - Except that in password-over-TLS, server learns password at decryption (as well as anyone that sees, legitimately or not, unencrypted traffic)

- Can we do password authentication so that server (or anyone other than the client) sees the password?

- Goal: Only feasible attacks are (unavoidable) online guesses

- Solution: OPAQUE = 1-out-of-1 PPSS!
  Use retrieved secret as private key for a key exchange protocol

You may use it one day...