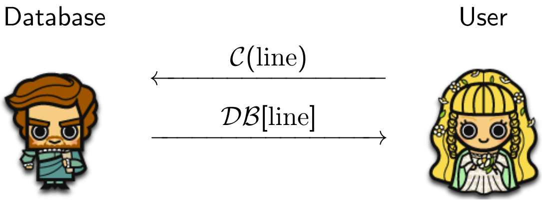
Structure Preserving Smooth Projective Hashing

Olivier Blazy, Céline Chevalier

Conditional Actions

Oblivious Transfer



$\rightsquigarrow$ The User learns the value of line but nothing else

$\rightsquigarrow$ The Database learns nothing

Conditional Actions

Password Authenticated Key Exchange

Bob

$$\xleftarrow{\quad f(\mathsf{pw}_A) \quad}$$

$$\xrightarrow{\quad f(\mathsf{pw}_B, f_A) \quad}$$

Alice

$\rightsquigarrow$ The Users obtain the same key iff their passwords match

$\rightsquigarrow$ An Adversary learns nothing

UC Requirements for Adaptive Corruptions

- First flow should be extractable
- First flow should be equivocable
- Memory should be adapted accordingly

Memory as a scalar

No real trapdoor possible $\rightsquigarrow$ Partial Erasure is the only way

UC Requirements for Adaptive Corruptions

- First flow should be extractable
- First flow should be equivocable
- Memory should be adapted accordingly

Memory as a scalar

No real trapdoor possible $\rightsquigarrow$ Partial Erasure is the only way

Memory as a group element

Allows extra trapdoor

UC Requirements for Adaptive Corruptions

- First flow should be extractable
- First flow should be equivocable
- Memory should be adapted accordingly

Memory as a scalar

No real trapdoor possible $\rightsquigarrow$ Partial Erasure is the only way

Memory as a group element

Allows extra trapdoor

UC Requirements for Adaptive Corruptions

- First flow should be extractable
- First flow should be equivocable
- Memory should be adapted accordingly

Memory as a scalar

No real trapdoor possible $\rightsquigarrow$ Partial Erasure is the only way

Memory as a group element

Allows extra trapdoor

UC Requirements for Adaptive Corruptions

- First flow should be extractable
- First flow should be equivocable
- Memory should be adapted accordingly

Memory as a scalar

No real trapdoor possible $\rightsquigarrow$ Partial Erasure is the only way

Memory as a group element

Allows extra trapdoor

Definition (Encryption Scheme)

$\mathcal{E} = (\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Encrypt}, \mathsf{Decrypt})$:

- $\mathsf{Setup}(\mathfrak{K})$: param;
- KeyGen(param): public *encryption* key pk, private *decryption* key dk;
- Encrypt(pk, m; r): encrypts $m \in \mathcal{M}$ in c using pk;
- Decrypt(dk, c): decrypts c under dk.

Indistinguishability under Chosen Ciphertext Attack

Definition (Smooth Projective Hash Functions) [CS02]

Let $\{H\}$ be a family of functions:

- X, domain of these functions
- L, subset (a language) of this domain

such that, for any point x in L, $H(x)$ can be computed by using

- either a *secret* hashing key hk: $H(x) = \mathsf{Hash}_L(\mathsf{hk}; x)$;
- or a *public* projected key hp: $H'(x) = \mathsf{ProjHash}_L(\mathsf{hp}; x, w)$

Public mapping $\mathsf{hk} \mapsto \mathsf{hp} = \mathsf{ProjKG}_L(\mathsf{hk}, x)$

Properties

For any $x \in X$, $H(x) = \mathsf{Hash}_L(\mathsf{hk}; x)$
For any $x \in L$, $H(x) = \mathsf{ProjHash}_L(\mathsf{hp}; x, w)$ $\quad w$ witness that $x \in L$

Smoothness

For any $x \notin L$, $H(x)$ and hp are independent

Pseudo-Randomness

For any $x \in L$, $H(x)$ is pseudo-random, without a witness w

Properties

For any $x \in X$, $H(x) = \mathsf{Hash}_L(\mathsf{hk}; x)$
For any $x \in L$, $H(x) = \mathsf{ProjHash}_L(\mathsf{hp}; x, w)$ w witness that $x \in L$

Smoothness

For any $x \notin L$, $H(x)$ and hp are independent

Pseudo-Randomness

For any $x \in L$, $H(x)$ is pseudo-random, without a witness w

Properties

For any $x \in X$, $H(x) = \mathsf{Hash}_L(\mathsf{hk}; x)$
For any $x \in L$, $H(x) = \mathsf{ProjHash}_L(\mathsf{hp}; x, w)$ $\quad$ w witness that $x \in L$

Smoothness

For any $x \notin L$, $H(x)$ and hp are independent

Pseudo-Randomness

For any $x \in L$, $H(x)$ is pseudo-random, without a witness w

Definition (Structure Preserving Smooth Projective Hash Functions)

- $X = \mathbb{G}_*^k$, $L \subsetneq \mathbb{G}_*^k$

such that, for any point x in L, $H(x)$ can be computed as:

- $H(x) = \mathsf{Hash}_L(\mathsf{hk}; x) \in \mathbb{G}_T$;
- $H'(x) = \mathsf{ProjHash}_L(\mathsf{hp}; x, w)$

hp, x, w are group elements

Definition (Structure Preserving Smooth Projective Hash Functions)

- $X = \mathbb{G}_*^k$, $L \subsetneq \mathbb{G}_*^k$

such that, for any point x in L, $H(x)$ can be computed as:

- $H(x) = \mathsf{Hash}_L(\mathsf{hk}; x) \in \mathbb{G}_T$;
- $H'(x) = \mathsf{ProjHash}_L(\mathsf{hp}; x, w)$

hp, x, w are group elements

Why?

Witnesses can now be Group Elements

This means, compatible with Groth Sahai Proofs (QA-NIZK, . . .)

Witnesses can now have trapdoors

Why?

Witnesses can now be Group Elements

This means, compatible with Groth Sahai Proofs (QA-NIZK, ...)

Witnesses can now have trapdoors

Retro-Compatibilty

	SPHF	SP-SPHF
Word $\mathbf{u}$	$[\omega \odot \Gamma(\mathbf{u})]_1$	$[\omega \odot \Gamma(\mathbf{u})]_1$
Witness w	ω	$\boldsymbol{\Lambda} = [f \odot \omega]_2$
hk	$\boldsymbol{\lambda}$	$\boldsymbol{\lambda}$
hp $= [\gamma(\mathbf{u})]_1$	$[\Gamma(\mathbf{u}) \odot \boldsymbol{\lambda}]_1$	$[\Gamma(\mathbf{u}) \odot \boldsymbol{\lambda}]_1$
Hash(hk, $\mathbf{u}$)	$[\Theta(\mathbf{u}) \odot \boldsymbol{\lambda}]_1$	$[f \odot \Theta(\mathbf{u}) \odot \boldsymbol{\lambda}]_T$
ProjHash(hp, $\mathbf{u}$, w)	$[\omega \odot \gamma(\mathbf{u})]_1$	$[\boldsymbol{\Lambda} \odot \gamma(\mathbf{u})]_T$

	SPHF	SP-SPHF
DH	h^r, g^r	h^r, g^r
Witness w	r	g_2^r
hk	λ, μ	λ, μ
hp	$h^\lambda g^\mu$	$h^\lambda g^\mu$
Hash(hk, $\mathbf{u}$)	$(h^r)^\lambda (g^r)^\mu$	$e((h^r)^\lambda (g^r)^\mu, g_2)$
ProjHash(hp, $\mathbf{u}$, w)	hp^r	$e(\mathsf{hp}, g_2^r)$

Figure: Example of conversion of classical SPHF into SP-SPHF

Oblivious Transfer [Rab81]

A user U wants to access a line ℓ in a database D composed of t of them:

- U learns nothing more than the value of the line ℓ
- D does not learn which line was accessed by U

Correctness: if U request a single line, he learns it

Security Notions

- Oblivious: D does not learn which line was accessed ;
- Semantic Security: U does not learn any information about the other lines.

A user U wants to access a line ℓ in a database D composed of t of them:

- U learns nothing more than the value of the line ℓ
- D does not learn which line was accessed by U

Correctness: if U request a single line, he learns it

Security Notions

- Oblivious: D does not learn which line was accessed ;
- Semantic Security: U does not learn any information about the other lines.

A user U wants to access a line ℓ in a database D composed of t of them:

- U learns nothing more than the value of the line ℓ
- D does not learn which line was accessed by U

Correctness: if U request a single line, he learns it

Security Notions

- Oblivious: D does not learn which line was accessed ;
- Semantic Security: U does not learn any information about the other lines.

Generic 1-out-of-t Oblivious Transfer (Simplified)

- User U picks ℓ:
  Computes $\mathcal{C} = \text{Encrypt}(\ell; \mathbf{s})$ with a UC commit SPHF friendly ($\mathbf{d}$ being the decommit information). He sends $\mathcal{C}$ and keeps $\mathbf{d}$ while erasing the rest.
- For each line L_j, server S computes hk_j, hp_j, and $H_j = \text{Hash}_{\mathcal{L}_j}(\text{hk}_j, \mathcal{C})$, $M_j = H_j \oplus L_j$ and sends M_j, hp_j.
- For the line ℓ, user computes $H'_\ell = \text{ProjHash}_{\mathcal{L}_\ell}(\text{hp}_\ell, \mathcal{C}, \mathbf{d})$, and then $L_\ell = M_\ell \oplus H'_\ell$

Generic 1-out-of-t Oblivious Transfer (Simplified)

- User U picks ℓ:
  Computes $\mathcal{C} = \text{Encrypt}(\ell; \mathbf{s})$ with a UC commit SPHF friendly ($\mathbf{d}$ being the decommit information). He sends $\mathcal{C}$ and keeps $\mathbf{d}$ while erasing the rest.
- For each line L_j, server S computes hk_j, hp_j, and $H_j = \text{Hash}_{\mathcal{L}_j}(\text{hk}_j, \mathcal{C})$, $M_j = H_j \oplus L_j$ and sends M_j, hp_j.
- For the line ℓ, user computes $H'_\ell = \text{ProjHash}_{\mathcal{L}_\ell}(\text{hp}_\ell, \mathcal{C}, \mathbf{d})$, and then $L_\ell = M_\ell \oplus H'_\ell$

Generic 1-out-of-t Oblivious Transfer (Simplified)

- User U picks ℓ:
  Computes $\mathcal{C} = \text{Encrypt}(\ell; \mathbf{s})$ with a UC commit SPHF friendly ($\mathbf{d}$ being the decommit information). He sends $\mathcal{C}$ and keeps $\mathbf{d}$ while erasing the rest.
- For each line L_j, server S computes hk_j, hp_j, and $H_j = \text{Hash}_{\mathcal{L}_j}(\text{hk}_j, \mathcal{C})$, $M_j = H_j \oplus L_j$ and sends M_j, hp_j.
- For the line ℓ, user computes $H'_\ell = \text{ProjHash}_{\mathcal{L}_\ell}(\text{hp}_\ell, \mathcal{C}, \mathbf{d})$, and then $L_\ell = M_\ell \oplus H'_\ell$

Generic Password Authenticated Key Exchange

- Each user U_i computes $\mathcal{C}_i = \mathsf{Encrypt}(\mathsf{pw}_i; \mathbf{s_i})$ with a UC commitment SPHF friendly, and $\mathbf{d_i}$ the decommit information.
  He computes $\mathsf{hp}_i, \mathsf{hk}_i$ for the language of valid passwords.
  He sends $\mathcal{C}_i, \mathsf{hp}_i$ and keeps $\mathbf{d_i}, \mathsf{hk}_i$ while erasing the rest.
- Receiving $\mathcal{C}_j, \mathsf{hp}_j$, compute $H'_i \cdot H_j = \mathsf{ProjHash}(\mathsf{hp}_j, \mathbf{d_i}) \cdot \mathsf{Hash}(\mathsf{hk}_i, \mathcal{C}_j)$

Generic Password Authenticated Key Exchange

- Each user U_i computes $\mathcal{C}_i = \mathsf{Encrypt}(\mathsf{pw}_i; \mathbf{s_i})$ with a UC commitment SPHF friendly, and $\mathbf{d_i}$ the decommit information.
  He computes $\mathsf{hp}_i, \mathsf{hk}_i$ for the language of valid passwords.
  He sends $\mathcal{C}_i, \mathsf{hp}_i$ and keeps $\mathbf{d_i}, \mathsf{hk}_i$ while erasing the rest.
- Receiving $\mathcal{C}_j, \mathsf{hp}_j$, compute $H'_i \cdot H_j = \mathsf{ProjHash}(\mathsf{hp}_j, \mathbf{d_i}) \cdot \mathsf{Hash}(\mathsf{hk}_i, \mathcal{C}_j)$

Generic Anonymous Credential-Based Message Transmission

Credential Use by User i:

1. UC commits to his credential in $\mathbf{C}$, and keeps his decommit info $\mathbf{d}$
2. Stores $\mathbf{d}$, sends $\mathbf{C}$ and erases the rest

Database input M with policy P:

1. Computes $\mathsf{hk}_P \stackrel{R}{\leftarrow} \mathsf{HashKG}(\mathcal{L}_P)$, $\mathsf{hp}_P \leftarrow \mathsf{ProjKG}(\mathsf{hk}_P, \mathcal{L}_P)$, $K_P \leftarrow \mathsf{Hash}(\mathsf{hk}_P, (\mathcal{L}_P, \mathbf{C}))$, and $N_P \leftarrow K_P \oplus M$
2. Server erases everything except (hp_P, N_P) and sends them

Data recovery:

Upon receiving (hp_P, N_P), User computes
$K \leftarrow \mathsf{ProjHash}(\mathsf{hp}_P, (\mathcal{L}_P, \mathbf{C}), \mathbf{d})$ and gets $M \leftarrow K \oplus N_P$.

High Level

- Do a Linear Cramer-Shoup Encryption of M with randomness $r, s \rightsquigarrow \mathbf{C}$
- Do a Groth Sahai proof of knowledge of $r, s \rightsquigarrow \mathbf{d}$

One Round UC Commitment [FLM11]

High Level

- Do a Linear Cramer-Shoup Encryption of M with randomness $r, s \rightsquigarrow \mathbf{C}$
- Do a Groth Sahai proof of knowledge of $r, s \rightsquigarrow \mathbf{d}$

Comparison with existing SXDH UC-secure OT schemes

	Flow	Communication Complexity	1-out-of
[CKWZ13]	4	$26\ \mathbb{G} + 7\ \mathbb{Z}_p$	2
[ABBCP13]	3	$(m + 8 \log m)\ \mathbb{G}_1 + \log m\ \mathbb{G}_2 + 1\ \mathbb{Z}_p$	m
Us	3	$4\ \mathbb{G}_1 + (4 + 4m)\ \mathbb{G}_2 + m\ \mathbb{Z}_p$	m
Us	3	$4\ \mathbb{G}_1 + 12\ \mathbb{G}_2 + 2\ \mathbb{Z}_p$	2

Comparison with UC-secure PAKE where $|\text{password}| = m$

	Adaptive	One-round	Communication complexity	Assumption
[ACP09]	✓	✗	$2 \times (2m + 22m\mathfrak{K}) \times \mathbb{G}$ + OTS	DDH
[KV11]	✗	✓	$\approx 2 \times 70\ \mathbb{G}$	DLIN
[BBCPV13]	✗	✓	$2 \times (6\ \mathbb{G}_1 + 5\ \mathbb{G}_2)$	SXDH
[ABBCP13]	✓	✓	$2 \times (10m\ \mathbb{G}_1 + m\ \mathbb{G}_2)$	SXDH
[JR15]	✓	✓	$4\ \mathbb{G}_1 + 4\ \mathbb{G}_2$	SXDH
Us	✓	✓	$2 \times (4\ \mathbb{G}_1 + 5\ \mathbb{G}_2)$	SXDH

k-MDDH abstraction [EHKRV13]

- Allows to abstract every Diffie Hellman assumptions
- Given $\mathbf{A}$, $\mathbf{z}$ decides whether there exists $\mathbf{s}$ such that $\mathbf{As} = \mathbf{z}$

A framework for everything

Compatible with linear constructions (CCA2, FLM-like, SPHF, and so SPSPHF)

k-MDDH abstraction [EHKRV13]

- Allows to abstract every Diffie Hellman assumptions
- Given $\mathbf{A}$, $\mathbf{z}$ decides whether there exists $\mathbf{s}$ such that $\mathbf{A}\mathbf{s} = \mathbf{z}$

A framework for everything

Compatible with linear constructions (CCA2, FLM-like, SPHF, and so SPSPHF)

To sum up

- ✓ Generic Transformation (keeps security, extra property)
- ✓ Allows to use NIZK as witnesses
- ✓ Leads to efficient protocols by using existing results
- ✓ All constructions can be transposed to MDDH

To sum up

- ✓ Generic Transformation (keeps security, extra property)
- ✓ Allows to use NIZK as witnesses
- ✓ Leads to efficient protocols by using existing results
- ✓ All constructions can be transposed to MDDH

To sum up

- ✓ Generic Transformation (keeps security, extra property)
- ✓ Allows to use NIZK as witnesses
- ✓ Leads to efficient protocols by using existing results
- ✓ All constructions can be transposed to MDDH

To sum up

- ✓ Generic Transformation (keeps security, extra property)
- ✓ Allows to use NIZK as witnesses
- ✓ Leads to efficient protocols by using existing results
- ✓ All constructions can be transposed to MDDH