

Implicit proofs of membership

Olivier Blazy



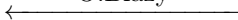
March 2nd, 2021

A Brave New World

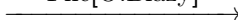
Patient Record



O.Blazy



File[O.Blazy]



Physician



Main Goal

Access a patient file in a database

A Brave New World

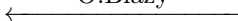
Patient Record



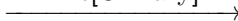
Physician



O.Blazy



File[O.Blazy]



Danger

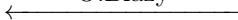
An outsider may want to access the patient file

A Brave New World

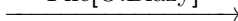
Patient Record



O.Blazy



File[O.Blazy]



Physician



Danger

An outsider may want to access the patient file

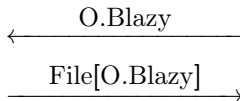
The Database can try to learn the value of the record

A Brave New World

Patient Record



Oncologist



Danger

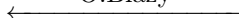
An outsider may want to access the patient file
The Database can try to learn the value of the record
or link the identity of the patient to a specialist

A Brave New World

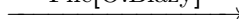
Patient Record



O.Blazy



File[O.Blazy]



Physician



Danger

An outsider may want to access the patient file

The Database can try to learn the value of the record

or link the identity of the patient to a specialist

A Physician can try to access more than one record at a time

The World is dark and full of terrors



$\text{pw}_B = 123$



$\text{pw}_A = 123$

Main Goal

Obtain a shared secret if the password match

The World is dark and full of terrors



$$pw_B = 123$$

$$pw_A = 123$$

Dangers

An outside adversary may try to learn the password

An outside adversary may try to learn whether the passwords match

The World is dark and full of terrors



???



$\text{pw}_A = 123$

Dangers

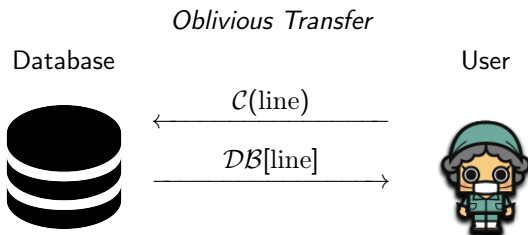
An outside adversary may try to learn the password

An outside adversary may try to learn whether the passwords match

A corrupted participant may try to obtain the shared key without knowing the password

A corrupted participant may try to learn the password

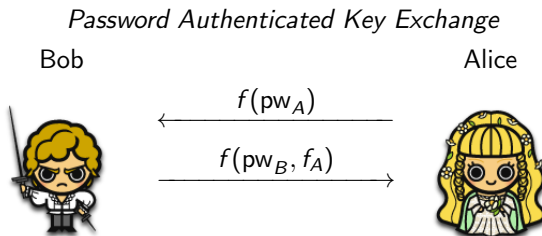
Oblivious Transfer



Security

The User learns the value of line but nothing else.
The Database learns nothing.

Password Authenticated Key Exchange



Security

The Users obtain a shared key iff their passwords match
An Adversary learns nothing

- 1 Motivation
- 2 Smooth Projective Hash Function
- 3 Building Hash Proofs
- 4 Applications

- 1 Motivation
- 2 Smooth Projective Hash Function
- 3 Building Hash Proofs
- 4 Applications

- 1 Motivation
- 2 Smooth Projective Hash Function
- 3 Building Hash Proofs
- 4 Applications

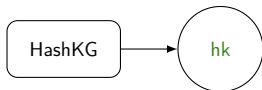
- 1 Motivation
- 2 Smooth Projective Hash Function
- 3 Building Hash Proofs
- 4 Applications

- 1 Motivation
- 2 Smooth Projective Hash Function
 - Definition
 - Direct Applications
 - Languages
- 3 Building Hash Proofs
- 4 Applications

Smooth Projective Hash Functions (SPHFs)

Definition [CS02]

NP language $\mathcal{L}$: $W \in \mathcal{L} \subseteq \mathcal{X} \iff \exists w, \mathcal{R}(W, w) = 1$

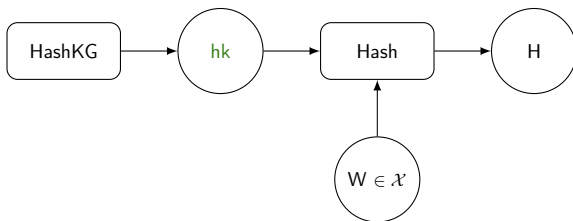


$hk \stackrel{R}{\leftarrow} \text{HashKG}()$

Smooth Projective Hash Functions (SPHFs)

Definition [CS02]

NP language $\mathcal{L}$: $W \in \mathcal{L} \subseteq \mathcal{X} \iff \exists w, \mathcal{R}(W, w) = 1$



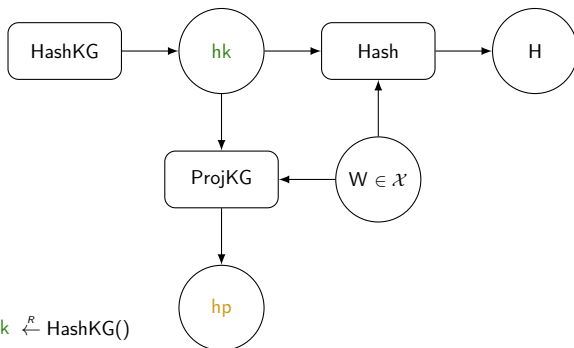
$hk \xleftarrow{R} \text{HashKG}()$

$H \leftarrow \text{Hash}(hk, W)$

Smooth Projective Hash Functions (SPHFs)

Definition [CS02]

NP language $\mathcal{L}$: $W \in \mathcal{L} \subseteq \mathcal{X} \iff \exists w, \mathcal{R}(W, w) = 1$



$hk \xleftarrow{R} \text{HashKG}()$

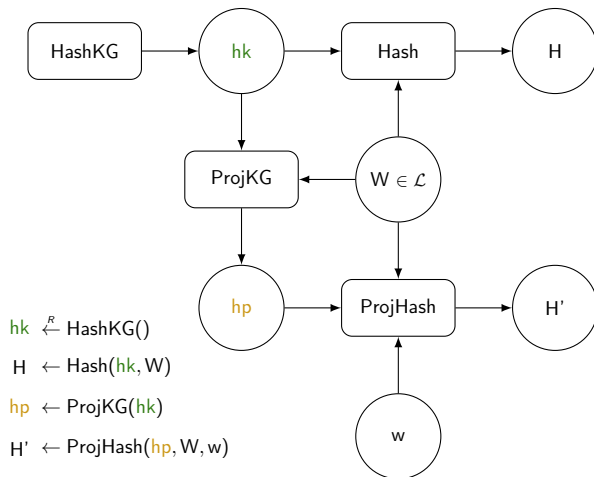
$H \leftarrow \text{Hash}(hk, W)$

$hp \leftarrow \text{ProjKG}(hk)$

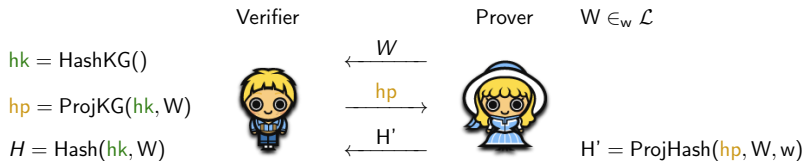
Smooth Projective Hash Functions (SPHFs)

Definition [CS02]

NP language $\mathcal{L}$: $W \in \mathcal{L} \subseteq \mathcal{X} \iff \exists w, \mathcal{R}(W, w) = 1$

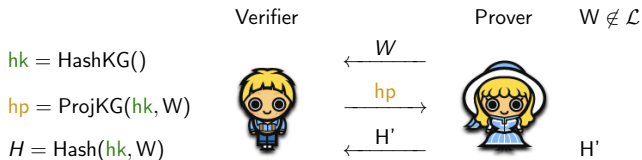


Properties: Correctness



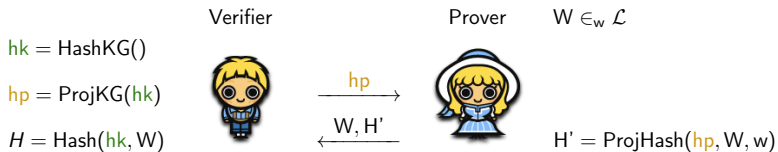
Correctness: $H = H'$

Properties: Smoothness



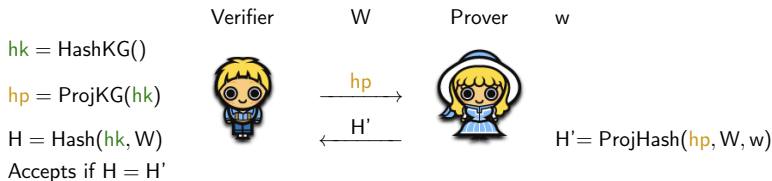
Smoothness: H random from the view of hp

Properties: Word-Independence



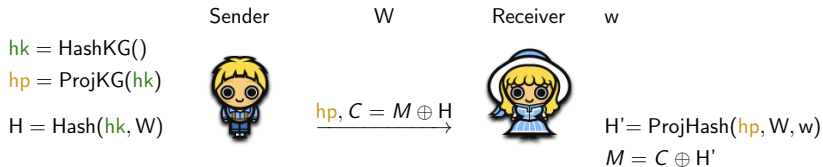
Direct Applications of SPHF's

Honest-Verifier Zero-Knowledge Proofs



Direct Applications of SPHF

Implicit Arguments / Witness Encryption



Languages Handled by SPHF

- Any NP Language?
 - Polynomial Hierarchy Collapses [GGSW13 + WE from SPHF], so let's say no?
 - But *any* encryption of an NP-Language
- Advanced Languages

Languages Handled by SPHF

- Any NP Language?
 - Polynomial Hierarchy Collapses [GGSW13 + WE from SPHF], so let's say no?
 - But *any* encryption of an NP-Language
- Advanced Languages

Languages Handled by SPHF

- Any NP Language?
 - Polynomial Hierarchy Collapses [GGSW13 + WE from SPHF], so let's say no?
 - But *any* encryption of an NP-Language

- Advanced Languages

BBCPV13 Pairing-Product Equation / Algebraic Statement

BCV15 Proof of Proof ($\leftrightarrow$ Negative Statement)

BCV16 Structure-Preserving (Using a CS proof as a witness)

BCV17 Proof of Proof (Using a CS proof as a witness)

BCV18 Proof of Proof (Using a CS proof as a witness)

Languages Handled by SPHF

- Any NP Language?
 - Polynomial Hierarchy Collapses [GGSW13 + WE from SPHF], so let's say no?
 - But *any* encryption of an NP-Language
- Advanced Languages

BBCPV13 Pairing-Product Equation / Algebraic Statement

BCV15 Proof of Proof ($\rightsquigarrow$ Negative Statement)

BC16 Structure-Preserving (Using a GS proof as a witness)

Languages Handled by SPHF

- Any NP Language?
 - Polynomial Hierarchy Collapses [GGSW13 + WE from SPHF], so let's say no?
 - But *any* encryption of an NP-Language

- Advanced Languages

BBCPV13 Pairing-Product Equation / Algebraic Statement

BCV15 Proof of Proof ($\rightsquigarrow$ Negative Statement)

BC16 Structure-Preserving (Using a GS proof as a witness)

Languages Handled by SPHF

- Any NP Language?
 - Polynomial Hierarchy Collapses [GGSW13 + WE from SPHF], so let's say no?
 - But *any* encryption of an NP-Language
- Advanced Languages
 - BBCPV13 Pairing-Product Equation / Algebraic Statement
 - BCV15 Proof of Proof ($\leadsto$ Negative Statement)
 - BC16 Structure-Preserving (Using a GS proof as a witness)

Languages Handled by SPHF

- Any NP Language?
 - Polynomial Hierarchy Collapses [GGSW13 + WE from SPHF], so let's say no?
 - But *any* encryption of an NP-Language
- Advanced Languages
 - BBCPV13 Pairing-Product Equation / Algebraic Statement
 - BCV15 Proof of Proof ($\leadsto$ Negative Statement)
 - BC16 Structure-Preserving (Using a GS proof as a witness)

Languages Handled by SPHF's

- Languages of ciphertexts of a given message M
 - Cyclic-group based encryption schemes: ElGamal, Cramer-Shoup, ...
 - QR-based, Paillier, ...
 - Lattice-based:
 - Code-based:

Languages Handled by SPHF's

- Languages of ciphertexts of a given message M
 - Cyclic-group based encryption schemes: ElGamal, Cramer-Shoup, ...
 - QR-based, Paillier, ...
 - Lattice-based:
 - NTRU "homomorphic" encryption schemes with associated applications
 - FHE: Fully Homomorphic Encryption
 - HECC: Homomorphic Encryption over Commutative Groups
 - Code-based:

Languages Handled by SPHF's

- Languages of ciphertexts of a given message M
 - Cyclic-group based encryption schemes: ElGamal, Cramer-Shoup, ...
 - QR-based, Paillier, ...
 - Lattice-based:
 - KV09 "unnatural" decryption procedure and non-word-independent
 - ZY17 Random Oracle Model
 - BDQ18 Probabilistic Encoding
 - Code-based:

Languages Handled by SPHF's

- Languages of ciphertexts of a given message M
 - Cyclic-group based encryption schemes: ElGamal, Cramer-Shoup, ...
 - QR-based, Paillier, ...
 - Lattice-based:
 - KV09 "unnatural" decryption procedure and non-word-independent
 - ZY17 Random Oracle Model
 - BBDQ18 Probabilistic Rounding
 - Code-based:

Languages Handled by SPHF's

- Languages of ciphertexts of a given message M
 - Cyclic-group based encryption schemes: ElGamal, Cramer-Shoup, ...
 - QR-based, Paillier, ...
 - Lattice-based:
 - KV09 "unnatural" decryption procedure and non-word-independent
 - ZY17 Random Oracle Model
 - BBDQ18 Probabilistic Rounding
 - Code-based:

Languages Handled by SPHF's

- Languages of ciphertexts of a given message M
 - Cyclic-group based encryption schemes: ElGamal, Cramer-Shoup, ...
 - QR-based, Paillier, ...
 - Lattice-based:
 - KV09 "unnatural" decryption procedure and non-word-independent
 - ZY17 Random Oracle Model
 - BBDQ18 Probabilistic Rounding
 - Code-based:

Languages Handled by SPHF's

- Languages of ciphertexts of a given message M
 - Cyclic-group based encryption schemes: ElGamal, Cramer-Shoup, ...
 - QR-based, Paillier, ...
 - Lattice-based:
 - KV09 "unnatural" decryption procedure and non-word-independent
 - ZY17 Random Oracle Model
 - BBDQ18 Probabilistic Rounding
 - Code-based:
 - Per13 A try?...
 - BBBGG217 Gapless, works with NIST candidates

Languages Handled by SPHF's

- Languages of ciphertexts of a given message M
 - Cyclic-group based encryption schemes: ElGamal, Cramer-Shoup, ...
 - QR-based, Paillier, ...
 - Lattice-based:
 - KV09 "unnatural" decryption procedure and non-word-independent
 - ZY17 Random Oracle Model
 - BDQ18 Probabilistic Rounding
 - Code-based:
 - Per13 *A try?....*
 - BBBCG21? Gapless, works with NIST candidates

Languages Handled by SPHF's

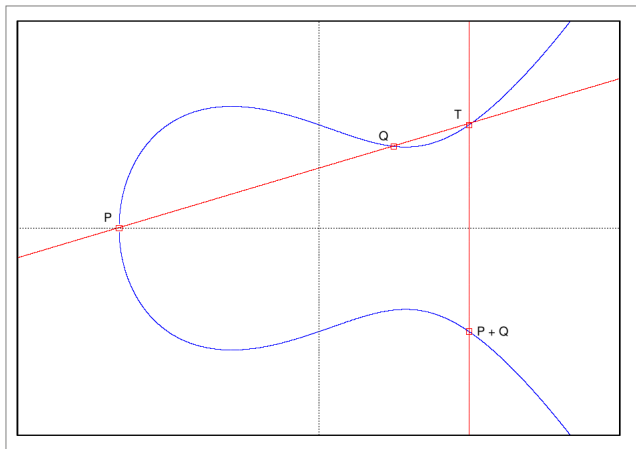
- Languages of ciphertexts of a given message M
 - Cyclic-group based encryption schemes: ElGamal, Cramer-Shoup, ...
 - QR-based, Paillier, ...
 - Lattice-based:
 - KV09 "unnatural" decryption procedure and non-word-independent
 - ZY17 Random Oracle Model
 - BDQ18 Probabilistic Rounding
 - Code-based:
 - Per13 *A try?....*
 - BBBCG21? Gapless, works with NIST candidates

Languages Handled by SPHF's

- Languages of ciphertexts of a given message M
 - Cyclic-group based encryption schemes: ElGamal, Cramer-Shoup, ...
 - QR-based, Paillier, ...
 - Lattice-based:
 - KV09 "unnatural" decryption procedure and non-word-independent
 - ZY17 Random Oracle Model
 - BDQ18 Probabilistic Rounding
 - Code-based:
 - Per13 *A try?...*
 - BBG21? Gapless, works with NIST candidates

- 1 Motivation
- 2 Smooth Projective Hash Function
- 3 Building Hash Proofs
 - ElGamal
 - Lattices
 - Code-based
 - Supersingular Isogeny
- 4 Applications

Starting on Elliptic Curve



$(\mathbb{G}, +)$ cyclic group of prime order p , generator g :

- Secret key: $s \in \mathbb{Z}_p$
- Public key: $h = g \cdot s \in \mathbb{G}$
- Ciphertext of M : $r \xleftarrow{R} \mathbb{Z}_p$

$$\mathbf{c} = (u, v) \quad \text{with} \quad \begin{cases} u &= g \cdot r \\ v &= h \cdot r + M \end{cases}$$

- Decryption:

$$M = v - u \cdot s$$

SPHF for ElGamal

Language of ciphertexts of $M = 0$:

$$\mathcal{L} = \{\mathbf{c} = (u, v) \mid \exists r \in \mathbb{Z}_p, u = g \cdot r \text{ and } v = h \cdot r\}$$

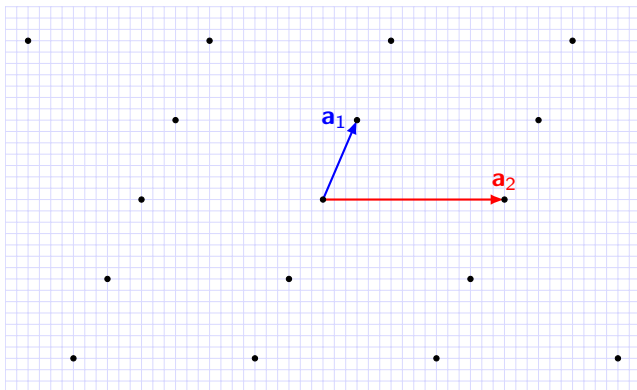
$$\text{Hashing key } \mathbf{hk} = (\alpha, \beta) \xleftarrow{R} \mathbb{Z}_p^2$$

$$\text{Projection Key } \mathbf{hp} = \alpha \cdot g + \beta \cdot h$$

$$\text{Hash value } H = \alpha \cdot u + \beta \cdot v$$

$$\text{Projected hash value } H' = \mathbf{hp} \cdot r$$

Lattices: Taming the Noise



LWE-Based IND-CPA Encryption

Encryption Scheme

- Public key: $\mathbf{A} \in \mathbb{Z}_q^{m \times n}$, define lattice $\Lambda = \{\mathbf{A} \cdot \mathbf{s} \bmod q \mid \mathbf{s} \in \mathbb{Z}_q^n\}$
- Secret key: trapdoor for $\mathbf{A}$
- Ciphertext of $M \in \{0, 1\}$: $\mathbf{s} \xleftarrow{R} \mathbb{Z}_q^n$, noise $\mathbf{e}$:

$$\mathbf{c} = \mathbf{A} \cdot \mathbf{s} + \mathbf{e} + M \cdot (0, \dots, 0, \lfloor q/2 \rfloor)^\top$$

- Decrypt to:

$$\begin{cases} 0 & \text{if } d(\mathbf{c}, \Lambda) \leq B' \\ 1 & \text{if } d(\mathbf{c} - M \cdot (0, \dots, 0, \lfloor q/2 \rfloor)^\top, \Lambda) \leq B' \\ \perp & \text{otherwise} \end{cases}$$

LWE-Based IND-CPA Encryption

SPHF — First try

Language of ciphertexts of $M = 0$:

$$\mathcal{L} = \{\mathbf{c} \mid \exists \mathbf{s} \in \mathbb{Z}_q^n, \exists \mathbf{e} \in \mathbb{Z}_q^m, \|\mathbf{e}\| \leq B, \quad \mathbf{c} = \mathbf{A} \cdot \mathbf{s} + \mathbf{e}\}$$

Hashing key	$\mathbf{hk} = \mathbf{h}$	$\in \mathbb{Z}_q^{1 \times m}$ “small”
Projection Key	$\mathbf{hp} = \mathbf{p}$	$= \mathbf{h} \cdot \mathbf{A}$
Hash value	$\mathbf{H}$	$= \mathbf{h} \cdot \mathbf{c}$
Projected hash value	$\mathbf{H}'$	$= \mathbf{p} \cdot \mathbf{s}$

LWE-Based IND-CPA Encryption

SPHF — First try

Language of ciphertexts of $M = 0$:

$$\mathcal{L} = \{\mathbf{c} | \exists \mathbf{s} \in \mathbb{Z}_q^n, \exists \mathbf{e} \in \mathbb{Z}_q^m, \|\mathbf{e}\| \leq B, \quad \mathbf{c} = \mathbf{A} \cdot \mathbf{s} + \mathbf{e}\}$$

Hashing key	$\mathbf{hk} = \mathbf{h}$	$\in \mathbb{Z}_q^{1 \times m}$ “small”
Projection Key	$\mathbf{hp} = \mathbf{p}$	$= \mathbf{h} \cdot \mathbf{A}$
Hash value	$\mathbf{H}$	$= \mathbf{h} \cdot \mathbf{c}$
Projected hash value	$\mathbf{H}'$	$= \mathbf{p} \cdot \mathbf{s}$

Correctness?

Not exactly $\rightarrow$ rounding function R

$$\mathbf{H} = R(\mathbf{h} \cdot \mathbf{c}) \approx R(\mathbf{p} \cdot \mathbf{s}) = \mathbf{H}'$$

LWE-Based IND-CPA Encryption

SPHF — First try

Language of ciphertexts of $M = 0$:

$$\mathcal{L} = \{\mathbf{c} \mid \exists \mathbf{s} \in \mathbb{Z}_q^n, \exists \mathbf{e} \in \mathbb{Z}_q^m, \|\mathbf{e}\| \leq B, \quad \mathbf{c} = \mathbf{A} \cdot \mathbf{s} + \mathbf{e}\}$$

Hashing key $\mathbf{hk} = \mathbf{h} \in \mathbb{Z}_q^{1 \times m}$ “small”

Projection Key $\mathbf{hp} = \mathbf{p} = \mathbf{h} \cdot \mathbf{A}$

Hash value $H = R(\mathbf{h} \cdot \mathbf{c})$

Projected hash value $H' = R(\mathbf{p} \cdot \mathbf{s})$

Correctness?

Not exactly $\rightarrow$ rounding function R

$$H = R(\mathbf{h} \cdot \mathbf{c}) = R(\mathbf{p} \cdot \mathbf{s}) = H'$$

with reasonable proba if R well chosen

LWE-Based IND-CPA Encryption

SPHF — First try

Language of ciphertexts of $M = 0$:

$$\mathcal{L} = \{\mathbf{c} \mid \exists \mathbf{s} \in \mathbb{Z}_q^n, \exists \mathbf{e} \in \mathbb{Z}_q^m, \|\mathbf{e}\| \leq B, \quad \mathbf{c} = \mathbf{A} \cdot \mathbf{s} + \mathbf{e}\}$$

Hashing key	$\mathbf{hk} = \mathbf{h}$	$\in \mathbb{Z}_q^{1 \times m}$ "small"
Projection Key	$\mathbf{hp} = \mathbf{p}$	$= \mathbf{h} \cdot \mathbf{A}$
Hash value	H	$= R(\mathbf{h} \cdot \mathbf{c})$
Projected hash value	H'	$= R(\mathbf{p} \cdot \mathbf{s})$

Smoothness?

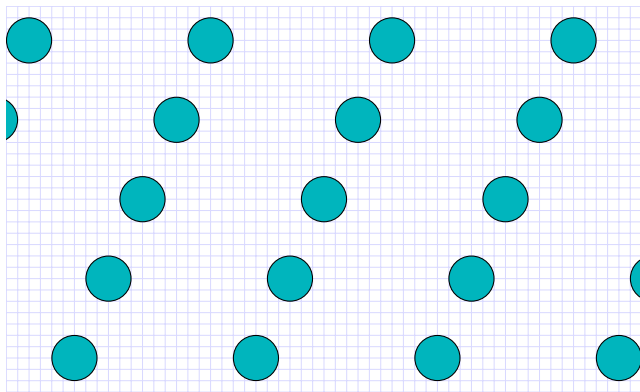
Encrypting with LWE: Languages

Correctness for valid encryption of 0:

$$\mathcal{L} = \{\mathbf{c} | \exists \mathbf{s} \in \mathbb{Z}_q^n, \exists \mathbf{e} \in \mathbb{Z}_q^m, \|\mathbf{e}\| \leq B, \mathbf{c} = \mathbf{A} \cdot \mathbf{s} + \mathbf{e}\}$$

Smoothness for ciphertexts that do not decrypt to 0:

$$\mathcal{L} = \{\mathbf{c} | \exists \mathbf{s} \in \mathbb{Z}_q^n, \exists \mathbf{e} \in \mathbb{Z}_q^m, \|\mathbf{e}\| \leq B', \mathbf{c} = \mathbf{A} \cdot \mathbf{s} + \mathbf{e}\}$$



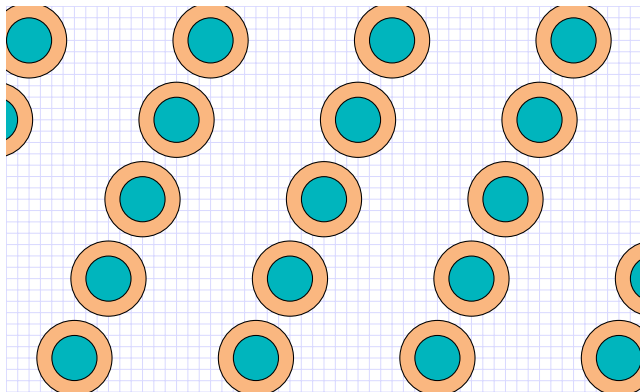
Encrypting with LWE: Languages

Correctness for valid encryption of 0:

$$\mathcal{L} = \{\mathbf{c} \mid \exists \mathbf{s} \in \mathbb{Z}_q^n, \exists \mathbf{e} \in \mathbb{Z}_q^m, \|\mathbf{e}\| \leq B, \mathbf{c} = \mathbf{A} \cdot \mathbf{s} + \mathbf{e}\}$$

Smoothness for ciphertexts that do not decrypt to 0:

$$\mathcal{L} = \{\mathbf{c} \mid \exists \mathbf{s} \in \mathbb{Z}_q^n, \exists \mathbf{e} \in \mathbb{Z}_q^m, \|\mathbf{e}\| \leq B', \mathbf{c} = \mathbf{A} \cdot \mathbf{s} + \mathbf{e}\}$$



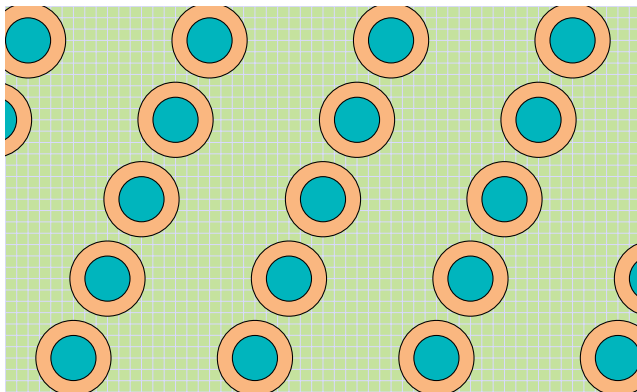
Encrypting with LWE: Languages

Correctness for valid encryption of 0:

$$\mathcal{L} = \{\mathbf{c} \mid \exists \mathbf{s} \in \mathbb{Z}_q^n, \exists \mathbf{e} \in \mathbb{Z}_q^m, \|\mathbf{e}\| \leq B, \mathbf{c} = \mathbf{A} \cdot \mathbf{s} + \mathbf{e}\}$$

Smoothness for ciphertexts that do not decrypt to 0:

$$\mathcal{L} = \{\mathbf{c} \mid \exists \mathbf{s} \in \mathbb{Z}_q^n, \exists \mathbf{e} \in \mathbb{Z}_q^m, \|\mathbf{e}\| \leq B', \mathbf{c} = \mathbf{A} \cdot \mathbf{s} + \mathbf{e}\}$$

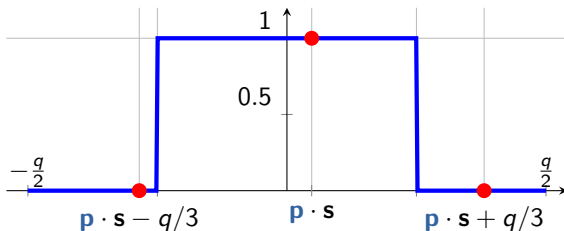


Smoothness?

$$R(x) = 1 + \left\lfloor \frac{2x}{q} \right\rfloor \bmod 2 = \begin{cases} 1 & \text{if } -q/4 \leq x < q/4 \\ 0 & \text{otherwise} \end{cases}$$

$$\mathbf{c} = \mathbf{A} \cdot \mathbf{s} + (0, \dots, 0, q/3)^\top$$

$$\Rightarrow H = R(\mathbf{h} \cdot \mathbf{c}) = R(\mathbf{p} \cdot \mathbf{s} + \frac{h_m \cdot q}{3} \bmod q) \quad \text{heavily biased}$$



LWE-Based IND-CPA Encryption

Smoothness — Solution of [KV09]

Basically smoothness when

$$\text{for all } j \in \mathbb{Z}_q^*, d(j \cdot \mathbf{c}, \Lambda) > B'$$

Drawbacks:

- Unnatural decryption algorithm
- Requires q lattice decoding
- No way to work with q superpolynomial
→ in particular unlikely to get word-independent SPHF

LWE-Based IND-CPA Encryption

Smoothness — Solution of [KV09]

Basically smoothness when

$$\text{for all } j \in \mathbb{Z}_q^*, d(j \cdot \mathbf{c}, \Lambda) > B'$$

Drawbacks:

- Unnatural decryption algorithm
- Requires q lattice decoding
- No way to work with q superpolynomial
→ in particular unlikely to get word-independent SPHF

Some help from Harmonic Analysis

Generalization: R is randomized and $r(x) = \Pr[R(x) = 1]$

Fourier decomposition of r : $r(x) = \sum_{j=0}^{q-1} \hat{r}_j \cdot e^{2i\pi jx/q}$

Fix $\mathbf{p} = \mathbf{h} \cdot \mathbf{A}$ and $\mathbf{c}$, and apply Poisson formula, goal:

$$P = \Pr[R(\mathbf{h} \cdot \mathbf{c}) = 1] = 1/2 + \text{negl}$$

$$P \approx \hat{r}_0 \cdot \rho'_{q/t}(\Lambda) + \sum_{j \neq 0} \hat{r}_j \cdot \rho'_{q/t}(\Lambda - j \cdot \mathbf{c})$$

Some help from Harmonic Analysis

Generalization: R is randomized and $r(x) = \Pr[R(x) = 1]$

Fourier decomposition of r : $r(x) = \sum_{j=0}^{q-1} \hat{r}_j \cdot e^{2i\pi jx/q}$

Fix $\mathbf{p} = \mathbf{h} \cdot \mathbf{A}$ and $\mathbf{c}$, and apply Poisson formula, goal:

$$P = \Pr[R(\mathbf{h} \cdot \mathbf{c}) = 1] = 1/2 + \text{negl}$$

$$P \approx \hat{r}_0 \cdot \rho'_{q/t}(\Lambda) + \sum_{j \neq 0} \hat{r}_j \cdot \rho'_{q/t}(\Lambda - j \cdot \mathbf{c})$$

Solution [KV09]

- 1 use $R(x) = 1 + \left\lfloor \frac{2x}{q} \right\rfloor \bmod q \Rightarrow \hat{r}_j = \Theta(1/j)$ for even j
- 2 ensure $j \cdot \mathbf{c}$ far from Λ for all j

Some help from Harmonic Analysis

Generalization: R is randomized and $r(x) = \Pr[R(x) = 1]$

Fourier decomposition of r : $r(x) = \sum_{j=0}^{q-1} \hat{r}_j \cdot e^{2i\pi jx/q}$

Fix $\mathbf{p} = \mathbf{h} \cdot \mathbf{A}$ and $\mathbf{c}$, and apply Poisson formula, goal:

$$P = \Pr[R(\mathbf{h} \cdot \mathbf{c}) = 1] = 1/2 + \text{negl}$$

$$P \approx \hat{r}_0 \cdot \rho'_{q/t}(\Lambda) + \sum_{j \neq 0} \hat{r}_j \cdot \rho'_{q/t}(\Lambda - j \cdot \mathbf{c})$$

Our solution [BBDQ18]

Ensure that $\hat{r}_j = 0$ for $j \geq 2$, i.e.:

$$r(x) = \Pr[R(x) = 1] = \frac{1}{2} + \frac{1}{2} \cos\left(\frac{2\pi x}{q}\right)$$

Construction

Hashing key	$hk = \mathbf{h}$	$\stackrel{R}{\leftarrow} D_{\mathbb{Z}, t}^{1 \times m}$
Projection Key	$hp = \mathbf{p}$	$= \mathbf{h} \cdot \mathbf{A}$
Hash value	H	$= R(\mathbf{h} \cdot \mathbf{c})$
Projected hash value	H'	$= R(\mathbf{p} \cdot \mathbf{s})$

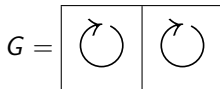
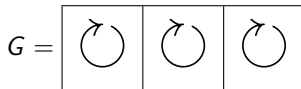
$$r(x) = \Pr[R(x) = 1] = \frac{1}{2} + \frac{1}{2} \cos\left(\frac{2\pi x}{q}\right)$$

- **Smoothness:** proven previously
- **Correctness:** approximate

$$\Pr[H = H'] \geq 3/4 + o(1)$$

→ can be extended to classical correctness
(ECC + parallel repetitions)

To Code and Beyond



RQC Encryption

Assume parameters: $\text{param} = (n, k, \delta, w, w_r, w_e) \in \mathbb{N}^6, \mathcal{R} = \mathcal{F}_{q^m}[X]/(X^n - 1)$ and $\mathbf{G} \in \mathcal{M}_{k,n}(\mathcal{F}_{q^m})$

- Secret key: $(\mathbf{x}, \mathbf{y}) \xleftarrow{\$} \mathcal{R}^2$ ($\omega(\mathbf{x}) = \omega(\mathbf{y}) = w$)
- Public key: $\mathbf{h}, \mathbf{s} = \mathbf{x} + \mathbf{h} \cdot \mathbf{y}$ (with $\mathbf{h} \xleftarrow{\$} \mathcal{R}$)
- Ciphertext of m : $\mathbf{r} = (\mathbf{r}_1, \mathbf{r}_2, \mathbf{r}_3) \xleftarrow{\$} \mathcal{R}^3$,

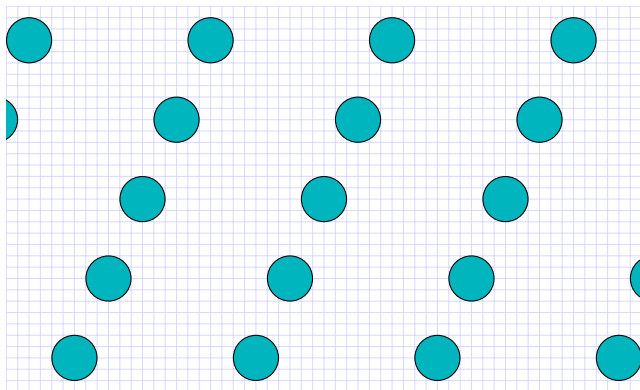
$$\mathbf{c} = (c_1, c_2) \quad \text{with} \quad \begin{cases} c_1 &= \mathbf{h} \cdot \mathbf{r}_2 + \mathbf{r}_1 \\ c_2 &= \mathbf{s} \cdot \mathbf{r}_2 + \mathbf{r}_3 + m\mathbf{G} \end{cases}$$

- Decryption : $\mathcal{C}.\text{Decode}(\mathbf{c}_2 - \mathbf{y} \cdot \mathbf{c}_1)$.
- Naive $\text{hp} = \mathbf{h} \cdot \alpha_1 + \mathbf{s} \cdot \alpha_2 + \alpha_3$

Encrypting with RQC: Languages

Correctness for valid encryption of 0: $\mathcal{L} = \{\mathbf{c} | \exists \mathbf{r}_* \in \mathcal{R}, \|\mathbf{r}\| \leq w_r, \mathbf{c} = \dots\}$

Smoothness for ciphertexts that do not decrypt to 0

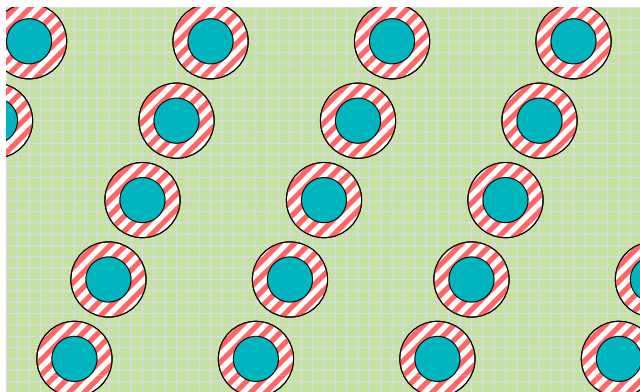


Verifiable Encryption $\rightsquigarrow$ Gapless SPHF

Encrypting with RQC: Languages

Correctness for valid encryption of 0: $\mathcal{L} = \{\mathbf{c} | \exists \mathbf{r}_* \in \mathcal{R}, \|\mathbf{r}\| \leq w_r, \mathbf{c} = \dots\}$

Smoothness for ciphertexts that do not decrypt to 0



Verifiable Encryption $\rightsquigarrow$ Gapless SPHF

Supersingular Isogeny



Supersingular Isogeny



We only have collection of orbits,
We only know to build SPHF around modules (or eventually ring action)



Alamati, De Feo, Montgomery, Patranabis at AC'20: *Cryptographic Group Action...* but...

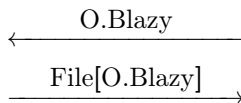
- 1 Motivation
- 2 Smooth Projective Hash Function
- 3 Building Hash Proofs
- 4 Applications
 - Oblivious Transfer
 - Password Authenticated Key Exchange

Oblivious Transfer

Patient Record



Physician



Security

The User learns the value requested but nothing else.

Oblivious Transfer

Patient Record



Physician



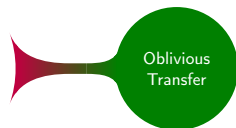
$$\begin{array}{c} \xleftarrow{\mathcal{C}(\text{O.Blazy})} \\ \xrightarrow{\text{SPHF}_{\text{patient}}(\mathcal{C}) \oplus \text{File}[\text{patient}]} \end{array}$$

Security

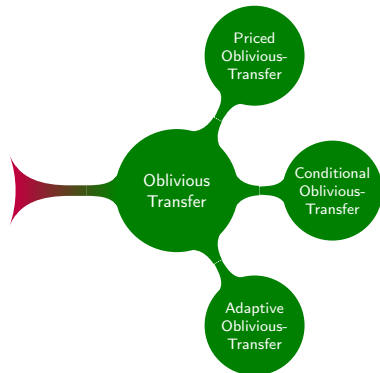
The User learns the value requested but nothing else.

The Database learns nothing.

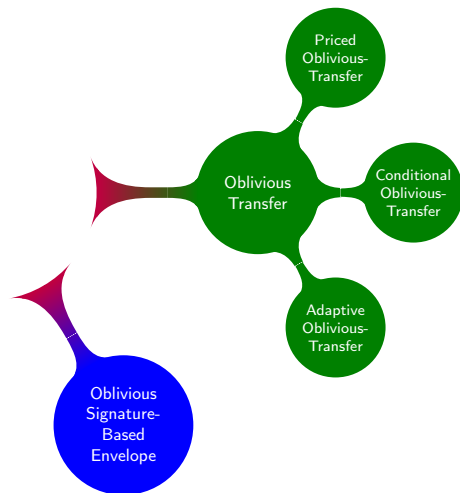
A family of similar problems



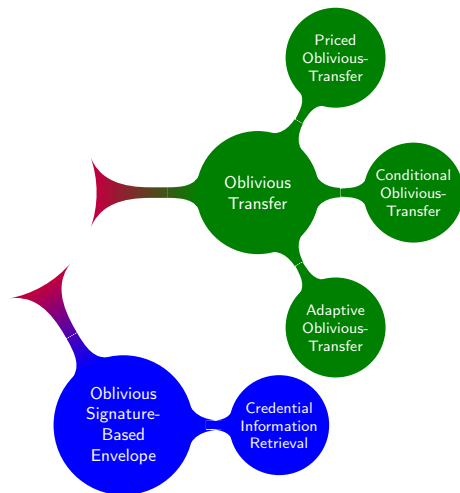
A family of similar problems



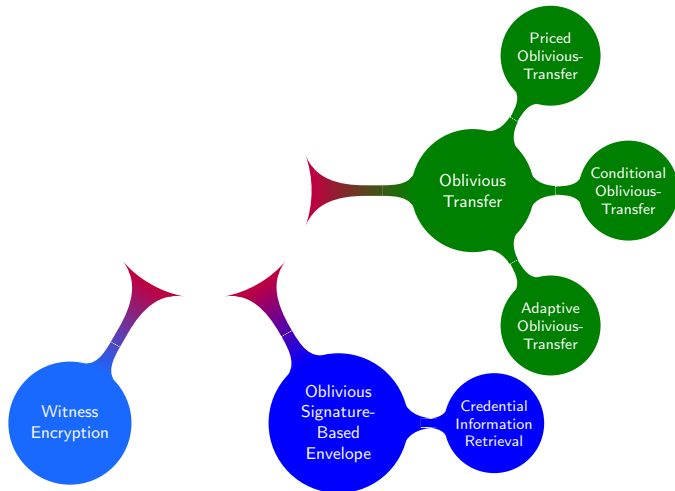
A family of similar problems



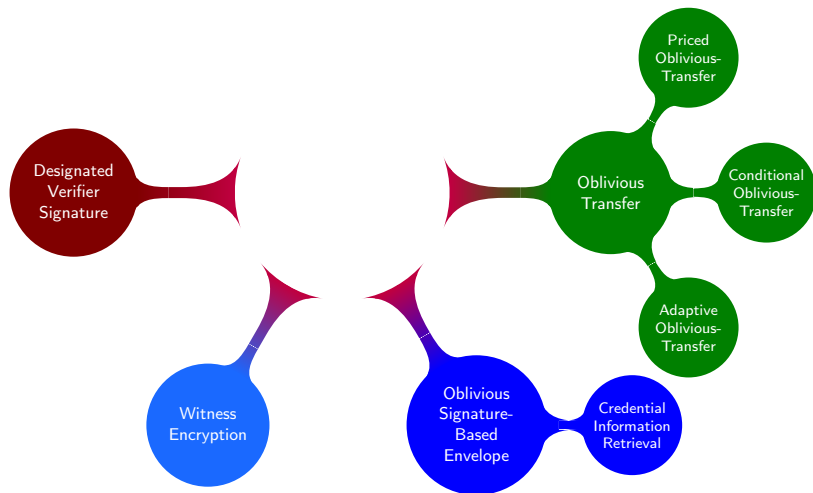
A family of similar problems



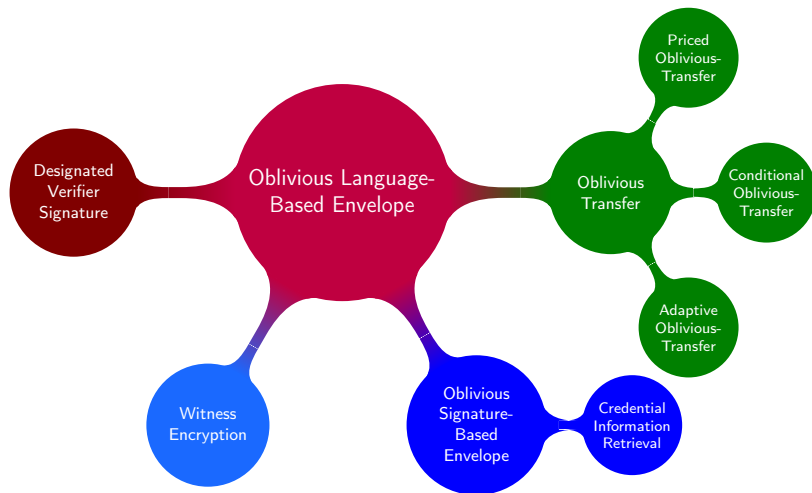
A family of similar problems

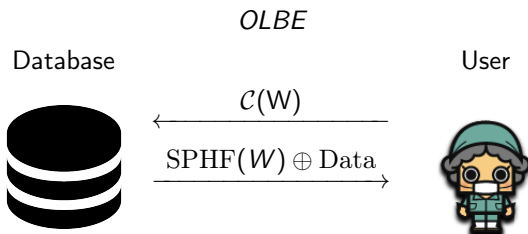


A family of similar problems



A family of similar problems





Security

The User learns the values linked to his word but nothing else.
The Database learns nothing.

Via a simple argument...

simulation of the server. When the server has sent his flow, he has already created all his random coins.

However, there would have been an issue when the user is corrupted after the server has sent its flow, but before the user receives it, since he has kept π in this world ready to the adversary to recover $\text{sk}(W)$ (from pk_i, W) and his $\text{pk}_{i,T}$. This is the goal of the ephemeral random R that provides a secure decommit.

E Proof of the Generic Construction of Oblivious Language-Based Decommit

We prove the adaptive¹² security of this protocol via a sequence of games, starting from the real game, where the adversary of *adversaries* with the real players, and ending with the ideal game, where we have built a simulator $\mathcal{S}$ that makes the interface between the ideal functionality $\mathcal{F}$ and the adversary $\mathcal{A}$.

We denote as S the sender (*i.e.* the server) and R the receiver (*i.e.* the user). The main idea is that, by assumption, the simulator can always obtain the common output sk of the collection of languages $\{L_1, \dots, L_n\}$ and use it to commit to a word stochastically belonging to all the languages. In case of adaptive corruption, we face two cases either the word was not correct, in which case, following the real protocol, the user should have used his randomness, so that the simulator does not have anything to reveal. Or the word was correct and should belong to a certain language L_i , and the simulator can then adapt the word and randomness so that it seems to belong to L_i (only). This enables us to avoid the use of commitments both extractable and equivocal (which is the usual tool for adaptive corruption).

We say that a flow is *semi-generated* if it was sent by a honest player (or the simulator) and received without any alteration by the adversary. It is said *non-epoch-generated* otherwise.

Game G_0 : This is the real game.

Game G_1 : In this game, the simulator generates correctly every flow on behalf of the honest players, as they would do themselves, knowing the inputs $(P_1, \dots, P_n)$ and W sent by the environment to the sender and the receiver. In all the subsequent games, the players use the label $\ell = (\text{id}, \text{sid}, S, R)$. In case of corruption, the simulator can give the internal data generated on behalf of the honest players.

Game G_2 : In this game, we replace the setup algorithm Setup by Setup_T , allowing the estimator of a trapdoor to find words in the intersection of the collection of languages. We also allow the simulator to program $\text{Setup}_{\text{adv}}$ in the CRS, enabling it to learn the extraction trapdoor of the CCA-secure scheme. The indistinguishability of the setup makes this game indistinguishable from the former one for the environment. Corruptions are handled the same way.

¹² One can obtain the proof of the static version by removing the parts related to the ephemeral random R and to the adaptive corruption in the proof below.

Game G_3 : We first deal with epoch-generated flows from the sender S when receiving a commitment C . The simulator extracts the committed value sk . By testing with the help of the algorithm Verify , it recovers i such that $W \in L_i$. If R recovers i such that $W \in L_i$, then the adversary has broken the indistinguishability of finding a word in an intersection of languages without knowing the trapdoor and we abort the game. Otherwise, instead of computing the v_i , for $i = 1, \dots, n$ with the hash function, the simulator then chooses $v_i \in G$ for $i \neq i$.

With an hybrid proof, applying the smoothness for every honest sender, on every index $i \neq i$, since G is extracted to $W \in L_i$, for any $i \neq i$, the hash value is indistinguishable from a random value.

In case of corruption, everything has been found (except after the pre-flow, where the simulator can reveal the keys (pk, sk) generated honestly). This game is thus indistinguishable from the previous one under the smoothness.

Game G_4 : Still in this case, when receiving a commitment C , the simulator extracts the committed value sk , giving it the number i . Instead of proceeding as the sender would do on $(P_1, \dots, P_n)$, the simulator proceeds on $(P'_1, \dots, P'_n)$, with $P'_i = P_i$ for $i \neq i$ for all $i \neq i$. Since the number v_i , for $i \neq i$, are random, this game is perfectly indistinguishable from the previous one.

Game G_5 : We now deal with epoch-generated flows from the receiver R : the simulator now uses the trapdoor to find a word W in the intersection of all languages.

With an hybrid proof, applying a security game in each random in which the simulator does not know the trapdoor sk , one can show the indistinguishability of this game from the previous one, one learns the already known value i .

Game G_6 : We deal with the generation of R for honest senders S on epoch-generated queries (adaptive case only). If S and R are honest at least until t , the simulator uses $R = P_i$ for both S and R , with P_i a random value, instead of $R = F(P_i)$.

With an hybrid proof, applying the IND-CPA property for each random, one can show the indistinguishability of this game with the previous one.

Game G_7 : Still in the same case, the simulator sets R as a random value, instead of $R = P_i$.

With an hybrid proof, applying the PHE property for each random, one can show the indistinguishability of this game with the previous one.

Game G_8 : We now deal with the generation of v_i for honest senders S on epoch-generated queries.

in the static case (the pre-flow is only needed to compute (pk, sk) , and thus we assume $R = \emptyset$), the simulator chooses $v_i \in G$ (for $i \neq i$), the simulator already chooses $v_i \in G$, where i is the index given by the ideal functionality to the honest receiver R .

With an hybrid proof, applying the pseudo-randomness for every honest sender, the hash value is indistinguishable from a random value, because the adversary does not know any decommitment information for C .

In the adaptive case, and thus with the additional random mask R , one can send a random P_i and v_i can be computed later (when R actually receives its flow).

As above, but only if R has not been corrupted before receiving its flow, the simulator chooses $v_i \in G$. With an hybrid proof, applying the pseudo-randomness, for every honest sender, the hash value is indistinguishable from a random value, because the adversary does not know any decommitment information for C . If the player R involved in the pseudo-randomness game gets corrupted (but the decommitment information is unknown), we are not in this case, and we can thus abort it.

In case of corruption of S , everything has been found (except after the pre-flow, where the simulator can reveal the keys (pk, sk) generated honestly). In case of corruption of the receiver R , and thus receiving the value P_i , the simulator chooses R (because it was a random value unknown to the adversary and all the other v_i are independent random values too) such that

$$R \in \text{PreHash}(\text{pk}_i, (L, \text{param}), (C, C), \tau) \oplus P_i = 0.$$

This game is thus indistinguishable from the previous one under the pseudo-randomness.

Game G_9 : Still in this case, the simulator proceeds on $(P'_1, \dots, P'_n)$ with $P'_i = R$ for all i . Since the number $v_i \in R$, for any $i = 1, \dots, n$, are independent random values (the v_i , for $i \neq i$ are independent random values, and v_i is also independently random in the static case, while R is independently random in the adaptive case), this game is perfectly indistinguishable from the previous one. We remark that it is therefore no more necessary to know the index i given by the ideal functionality to the honest receiver R , to simulator S (it is still necessary to simulate R).

Game G_{10} : We can stop approximating the knowledge of i when simulating on honest senders S . The simulator generates a word W in the intersection of the languages and $C \in \mathcal{C} \cdot \text{Encrypt}_{\text{adv}}(W, r)$ with $r = (\text{id}, \text{sid}, S, R)$, so send C during the first phase of honest receivers. This does not change anything from the previous game since the randomness needed to open to a word is another language (it never reveals).

When S thereafter receives $(\text{Send}, \text{id}, \text{sid}, S, R, (Q_1, \dots, Q_n, \text{sk}_R))$ from the adversary, the simulator computes, for $i = 1, \dots, k$, v_i

$$v_i = \text{PreHash}(\text{pk}_i, (L, \text{param}), (C, C), \tau)$$

and $P_i = v_i \oplus R \oplus Q_i$. This provides the decommit, obtained by the sender.

Game G_{11} : We can now make use of the functionality, which leads to the following situation:

— when receiving a *Send*-message from the ideal functionality, which means that an honest sender has sent a pre-flow, the simulator generates a key $\text{pk}_i = \text{KeyGen}(1^k)$ and sends pk_i as pre-flow;

— after receiving a pre-flow pk_i (from an honest or a corrupted sender) and a *Receive*-message from the ideal functionality, which means that an honest receiver has sent a first flow, the simulator generates a word W in the intersection of languages $\mathcal{C} \in \text{Encrypt}_{\text{adv}}(W, r)$ with $r = (\text{id}, \text{sid}, R, S)$ and $C \in \mathcal{C} \cdot \text{Encrypt}(\text{pk}_i, r)$ where r is a random value;

— when receiving a commitment C and a ciphertext c , generated by the adversary (from a corrupted receiver), the simulator extracts the committed value W and recovers i (obtaining a list of candidate values), and uses it to send a *Receive*-message to the ideal functionality (and also decommit the ciphertext c as J , and outputs $R = F(J)$);

— when receiving $(Q_1, \text{sk}_1, \dots, Q_k, \text{sk}_k)$ from the adversary (a corrupted sender) the simulator computes, for $i = 1, \dots, k$, v_i

$$v_i = \text{PreHash}(\text{pk}_i, (L, \text{param}), (C, \text{param}), (C, C), \tau)$$

and $P_i = v_i \oplus R \oplus Q_i$. It uses them to send a *Send*-message to the ideal functionality.

— when receiving a *Receive*-message from the ideal functionality, together with P_i on behalf of a corrupted receiver, from the extracted W the simulator c (instead of proceeding on the sender would do on $(P_1, \dots, P_n)$), leading to c proceeds on $(P'_1, \dots, P'_n)$ with $P'_i = P_i$ for $i \neq i$ and $P'_i = R$ for all $i \neq i$.

— when receiving a commitment C and a ciphertext c , generated by an honest sender (i.e., by the simulator itself), the simulator proceeds as above on $(P'_1, \dots, P'_n)$, with $P'_i = R$ for all i , but this choice R uniformly at random (instead of choosing i as $R = F(J)$), in case of corruption afterwards, the simulator will adapt. If such that $R \in \text{PreHash}(\text{pk}_i, (L, \text{param}), (C, C), \tau) \oplus Q_i = P_i$ where P_i is the message actually received by the receiver.

Any corruption either reveals i earlier, which allows a correct simulation of the receiver, or reveals $(P_1, \dots, P_n)$ earlier, which allows a correct simulation of the sender. When the sender has sent his flow, he has already earned all his randomness.

However, there would have been an issue when the receiver is corrupted after the sender has sent its flow, but before the receiver receives it, since he has kept τ . This would enable the adversary to recover P_i from Q_i and sk_i . This is the goal of the ephemeral random R that provides a secure decommit.

F Oblivious Primitives Obtained by the Framework

The framework presented in Section 4 page 30 provides a generic way to achieve cryptographic protocols under automated trust negotiation. In this section, we show how one can use this framework to obtain various cryptographic primitives and can be seen as examples of Oblivious Language-Based Decommit. We illustrate the power and how by revealing the languages (and choosing appropriate masks).

A complete family

- Singleton: Oblivious Transfer [ABBCP13, BC15]
- Solution to an equation (signature): OSBE [BPV12]
- Non-Interactive solution to an equation: Witness Encryption [BBDQ18]
- And others (Conditional Oblivious Transfer, DVS, ...) [BCGJ17, BBCF20]

A complete family

- Singleton: Oblivious Transfer [ABBCP13, BC15]
- Solution to an equation (signature): OSBE [BPV12]
- Non-Interactive solution to an equation: Witness Encryption [BBDQ18]
- And others (Conditional Oblivious Transfer, DVS, ...) [BCGJ17, BBCF20]

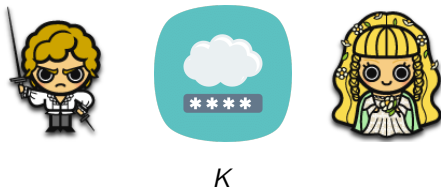
A complete family

- Singleton: Oblivious Transfer [ABCP13, BC15]
- Solution to an equation (signature): OSBE [BPV12]
- Non-Interactive solution to an equation: Witness Encryption [BDQ18]
- And others (Conditional Oblivious Transfer, DVS, ...) [BCGJ17, BBCF20]

A complete family

- Singleton: Oblivious Transfer [ABCP13, BC15]
- Solution to an equation (signature): OSBE [BPV12]
- Non-Interactive solution to an equation: Witness Encryption [BDQ18]
- And others (Conditional Oblivious Transfer, DVS, ...) [BCGJ17, BCF20]

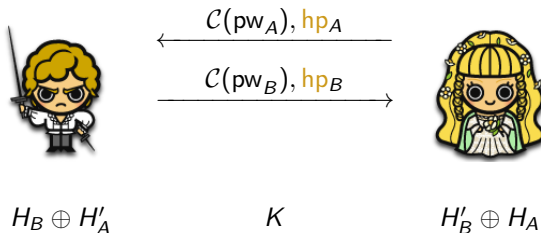
Password Authenticated Key Exchange



Security

Protocol succeeds iff the passwords are the same
Nobody learns the password

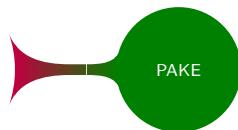
Password Authenticated Key Exchange



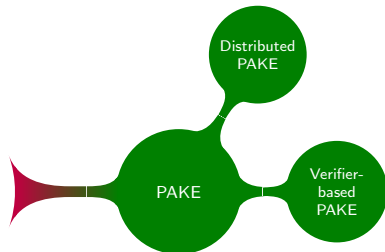
Security

Protocol succeeds iff the passwords are the same
Nobody learns the password

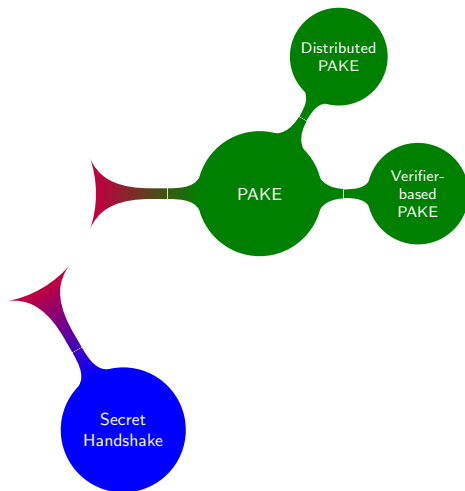
A family of similar problems



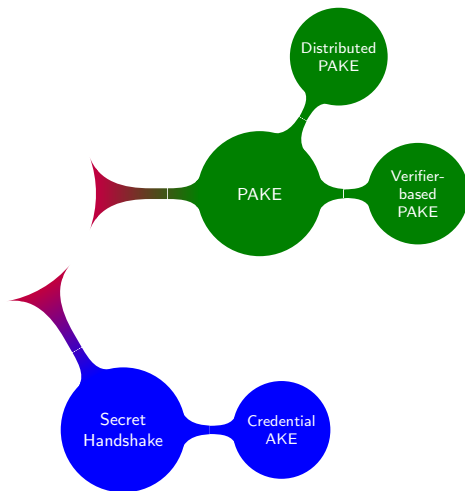
A family of similar problems



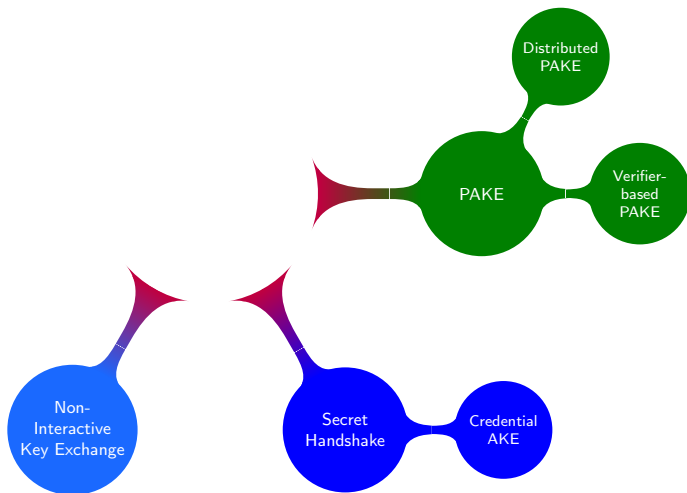
A family of similar problems



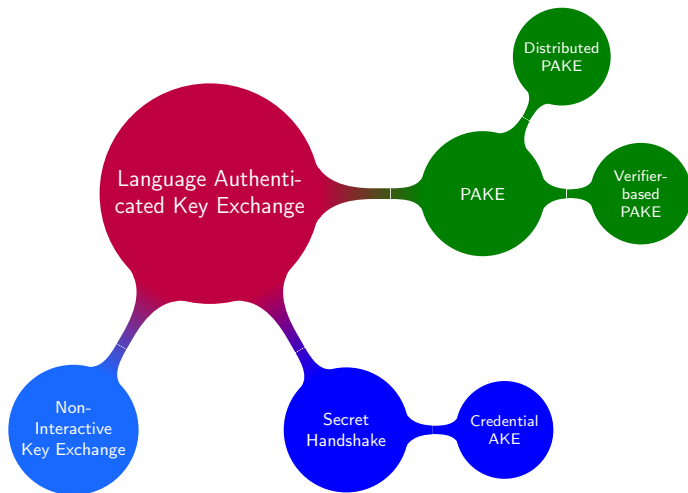
A family of similar problems

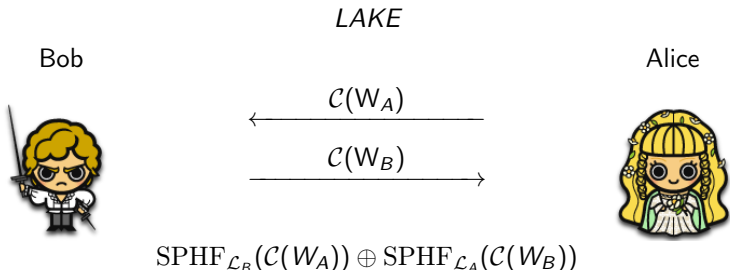


A family of similar problems



A family of similar problems





Security

The Users obtain a shared key iff their words fit
An Adversary learns nothing

A whole new world

- Singleton: PAKE [BBCPV14, ABBCP14, BBDQ18]
- Solution to an equation (signature): Secret Handshake [BCFK19?]
- And others [BCV16, BBL16, BC18]

A whole new world

- Singleton: PAKE [BB_{CPV14}, AB_{BCP14}, BB_{DQ18}]
- Solution to an equation (signature): Secret Handshake [BC_{FK19?}]
- And others [BC_{V16}, BB_{L16}, BC₁₈]

A whole new world

- Singleton: PAKE [BBCPV14, ABBCP14, BBDQ18]
- Solution to an equation (signature): Secret Handshake [BCFK19?]
- And others [BCV16, BBL16, BC18]

What we achieved

- ✓ Smooth Hash Proof Systems now exist for most Cryptographic Worlds
- ✓ They fit well in already interactive protocols, and allow to reduce rounds
- ✓ Increase privacy by removing the explicit validation
- ✓ Can do common languages classically manageable with a ZKPK
- ✓ Most efficient solutions in the standard model for ECC
- ✓ A library in Rust implements our constructions, and allow fast, constant time instantiation

What we achieved

- ✓ Smooth Hash Proof Systems now exist for most Cryptographic Worlds
- ✓ They fit well in already interactive protocols, and allow to reduce rounds
- ✓ Increase privacy by removing the explicit validation
- ✓ Can do common languages classically manageable with a ZKPK
- ✓ Most efficient solutions in the standard model for ECC
- ✓ A library in Rust implements our constructions, and allow fast, constant time instantiation

What we achieved

- ✓ Smooth Hash Proof Systems now exist for most Cryptographic Worlds
- ✓ They fit well in already interactive protocols, and allow to reduce rounds
- ✓ Increase privacy by removing the explicit validation
- ✓ Can do common languages classically manageable with a ZKPK
- ✓ Most efficient solutions in the standard model for ECC
- ✓ A library in Rust implements our constructions, and allow fast, constant time instantiation

What we achieved

- ✓ Smooth Hash Proof Systems now exist for most Cryptographic Worlds
- ✓ They fit well in already interactive protocols, and allow to reduce rounds
- ✓ Increase privacy by removing the explicit validation
- ✓ Can do common languages classically manageable with a ZKPK
- ✓ Most efficient solutions in the standard model for ECC
- ✓ A library in Rust implements our constructions, and allow fast, constant time instantiation

What we achieved

- ✓ Smooth Hash Proof Systems now exist for most Cryptographic Worlds
- ✓ They fit well in already interactive protocols, and allow to reduce rounds
- ✓ Increase privacy by removing the explicit validation
- ✓ Can do common languages classically manageable with a ZKPK
- ✓ Most efficient solutions in the standard model for ECC
- ✓ A library in Rust implements our constructions, and allow fast, constant time instantiation

What we achieved

- ✓ Smooth Hash Proof Systems now exist for most Cryptographic Worlds
- ✓ They fit well in already interactive protocols, and allow to reduce rounds
- ✓ Increase privacy by removing the explicit validation
- ✓ Can do common languages classically manageable with a ZKPK
- ✓ Most efficient solutions in the standard model for ECC
- ✓ A library in Rust implements our constructions, and allow fast, constant time instantiation

Open problems

- ? Other applications (Generic construction of IBE from Trapdoor SPHF)
- ? Efficient PostQuantum SPHF
- ? GapLess Lattice-based SPHF
- ? Isogeny-based SPHF (Collection of orbits vs Modules/Ring Actions)
- ? Counter Example Language for SPHF / Word-Independent SPHF

Open problems

- ? Other applications (Generic construction of IBE from Trapdoor SPHF)
- ? Efficient PostQuantum SPHF
- ? GapLess Lattice-based SPHF
- ? Isogeny-based SPHF (Collection of orbits vs Modules/Ring Actions)
- ? Counter Example Language for SPHF / Word-Independent SPHF

Open problems

- ? Other applications (Generic construction of IBE from Trapdoor SPHF)
- ? Efficient PostQuantum SPHF
- ? GapLess Lattice-based SPHF
- ? Isogeny-based SPHF (Collection of orbits vs Modules/Ring Actions)
- ? Counter Example Language for SPHF / Word-Independent SPHF

Open problems

- ? Other applications (Generic construction of IBE from Trapdoor SPHF)
- ? Efficient PostQuantum SPHF
- ? GapLess Lattice-based SPHF
- ? Isogeny-based SPHF (Collection of orbits vs Modules/Ring Actions)
- ? Counter Example Language for SPHF / Word-Independent SPHF

Open problems

- ? Other applications (Generic construction of IBE from Trapdoor SPHF)
- ? Efficient PostQuantum SPHF
- ? GapLess Lattice-based SPHF
- ? Isogeny-based SPHF (Collection of orbits vs Modules/Ring Actions)
- ? Counter Example Language for SPHF / Word-Independent SPHF

Questions?

Thank you!



And thanks to my co-authors:

OPPPPPQS