Implicit Cryptography

Olivier Blazy

Xlim – Institut de Recherche

Université de Limoges

Implicit Cryptography

Michel Abdalla, Fabrice BenHamouda, Slim Bettaieb, Loïc Bidoux, *Olivier Blazy* ,Emmanuel Conchon, Yann Connan, Céline Chevalier, Leo Ducas, Philippe Gaborit, Paul Germouty, Amandine Jambert, David Pointcheval, Willy Quatch, Damien Vergnaud

Conditional Actions

Protocols

Database

Info!
←

Valid?A : B
→

User

Conditional Actions

Oblivious Transfer

Database | User

$\xleftarrow{\mathcal{C}(\text{line})}$

$\xrightarrow{\mathcal{DB}[\text{line}]}$

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

$\rightsquigarrow$ The Database learns nothing

Conditional Actions

Password Authenticated Key Exchange

Bob

$$f(\mathrm{pw}_A) \quad \longleftarrow$$

$$f(\mathrm{pw}_B, f_A) \quad \longrightarrow$$

Alice

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

$\rightsquigarrow$ An Adversary learns nothing

Conditional Actions

Protocols

Database | User

Info! ←

Valid?A : B →

Conditional Actions

Protocols

Database

Info! ←

Valid?A : B →

User

⇝ Anyone can learn the secret user information

⇝ The Database learns the info, and whether the execution succeeds

UC Requirements for Adaptive Corruptions

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

UC Requirements for Adaptive Corruptions

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

UC Requirements for Adaptive Corruptions

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

Definition (Encryption Scheme)

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

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

Indistinguishability under Chosen Ciphertext Attack

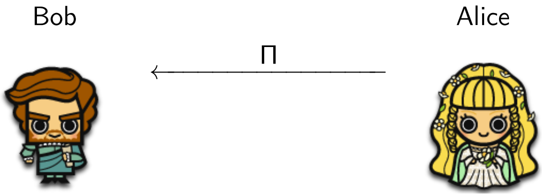
Zero-Knowledge [GMR85]

Bob

$\xleftarrow{\Pi}$

Alice

Zero-Knowledge [GMR85]



Soundness: no cheating prover can convince the verifier that a false statement is true

Zero-Knowledge: no cheating verifier learns anything other than the veracity of the statement.

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

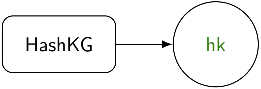
Smoothness

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

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

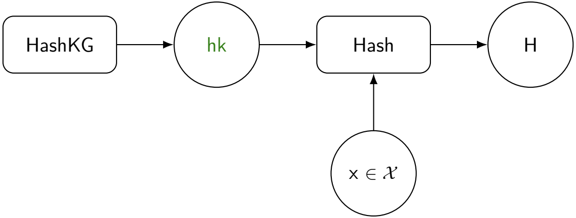


$hk \xleftarrow{R} \mathsf{HashKG}()$

Smooth Projective Hash Functions (SPHFs)

Definition

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



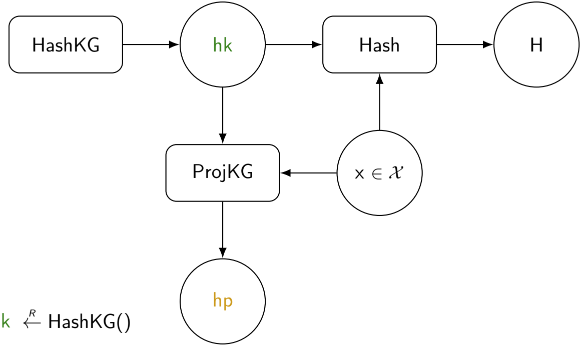
$\mathsf{hk} \xleftarrow{R} \mathsf{HashKG}()$

$\mathsf{H} \leftarrow \mathsf{Hash}(\mathsf{hk}, x)$

Smooth Projective Hash Functions (SPHFs)

Definition

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



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

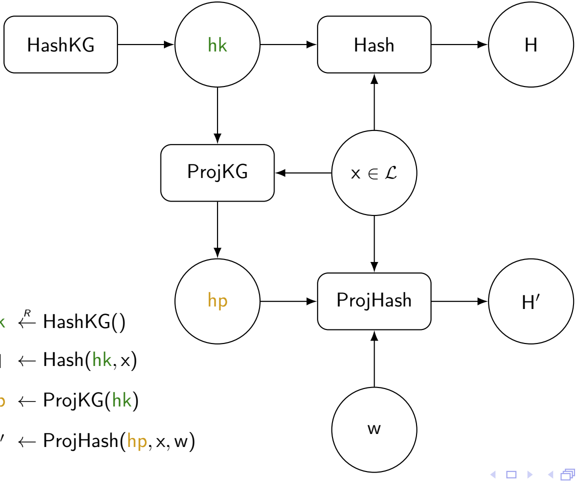
$\mathsf{H} \leftarrow \mathsf{Hash}(\mathsf{hk}, x)$

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

Smooth Projective Hash Functions (SPHFs)

Definition

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



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

$\mathsf{H} \leftarrow \mathsf{Hash}(\mathsf{hk}, x)$

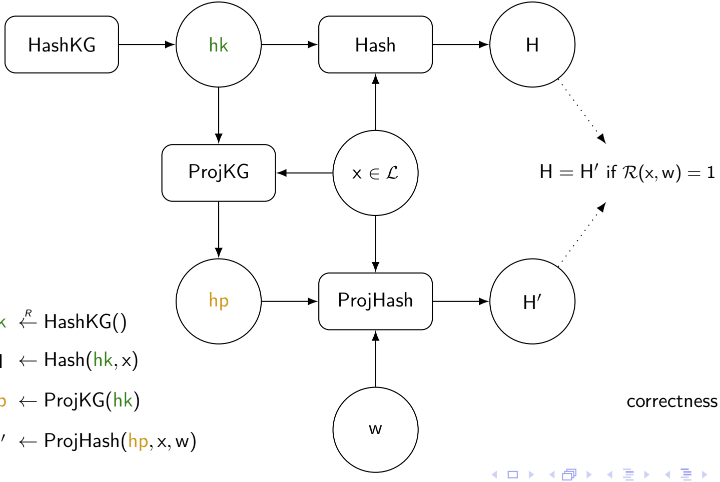
$\mathsf{hp} \leftarrow \mathsf{ProjKG}(\mathsf{hk})$

$\mathsf{H}' \leftarrow \mathsf{ProjHash}(\mathsf{hp}, x, w)$

Smooth Projective Hash Functions (SPHFs)

Definition

NP language $\mathcal{L}$: $\quad \mathsf{x} \in \mathcal{L} \subseteq \mathcal{X} \quad \Longleftrightarrow \quad \exists \mathsf{w},\ \mathcal{R}(\mathsf{x}, \mathsf{w}) = 1$



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

$\mathsf{H} \leftarrow \mathsf{Hash}(\mathsf{hk}, \mathsf{x})$

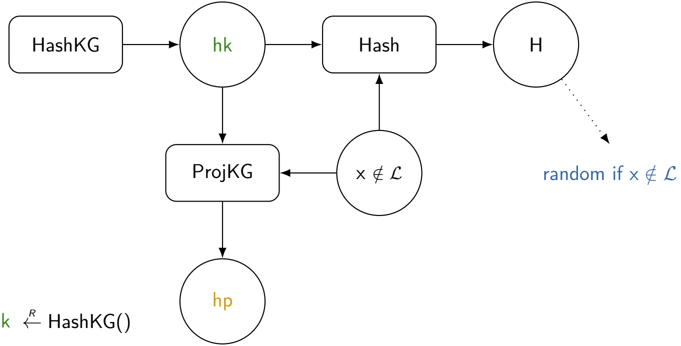
$\mathsf{hp} \leftarrow \mathsf{ProjKG}(\mathsf{hk})$

$\mathsf{H}' \leftarrow \mathsf{ProjHash}(\mathsf{hp}, \mathsf{x}, \mathsf{w})$

Smooth Projective Hash Functions (SPHFs)

Definition

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



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

$\mathsf{H} \leftarrow \mathsf{Hash}(\mathsf{hk}, x)$

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

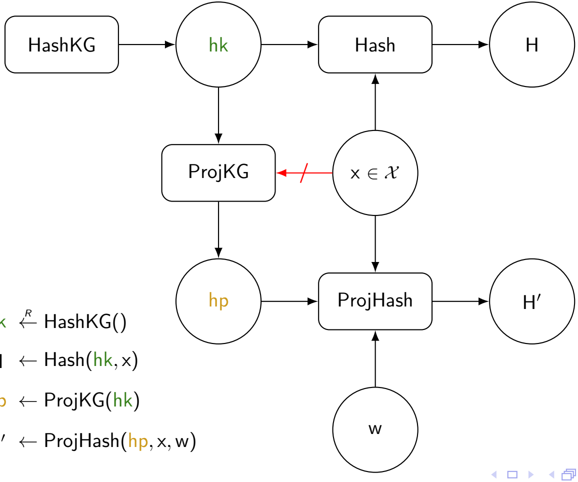
$\mathsf{H}' \leftarrow \mathsf{ProjHash}(\mathsf{hp}, x, w)$

smoothness

Smooth Projective Hash Functions (SPHFs)

Word-Independent SPHF

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



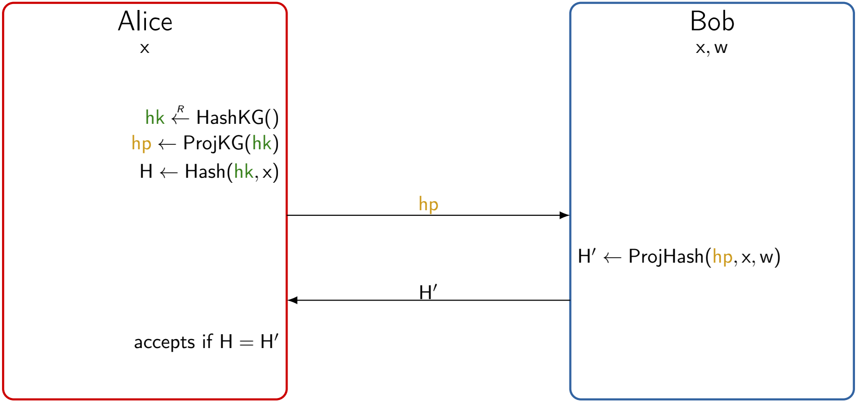
$\mathsf{hk} \stackrel{R}{\leftarrow} \mathsf{HashKG}()$

$\mathsf{H} \leftarrow \mathsf{Hash}(\mathsf{hk}, x)$

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

$\mathsf{H}' \leftarrow \mathsf{ProjHash}(\mathsf{hp}, x, w)$

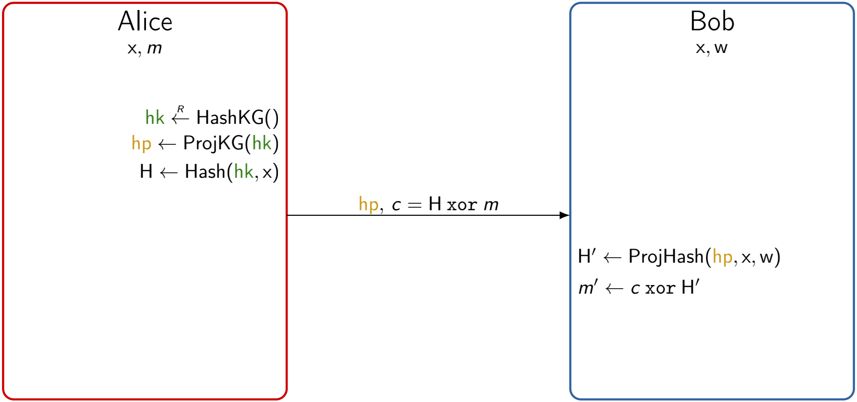
Bob wants to prove to Alice that $x \in \mathcal{L}$.



Direct Applications of SPHFs

Implicit Arguments / Witness Encryption

Alice wants to send m to Bob if $\mathsf{x} \in \mathcal{L}$.



Languages Handled by SPHFs?

- Any NP language?
  - $\Rightarrow$ polynomial hierarchy collapses [GGSW13]
  - But any encryption of an NP-Language ...
- Languages of ciphertexts of a given message M
  - Cyclic-group-based encryption schemes: ElGamal, Cramer-Shoup, ...
  - QR-based, Paillier, ...
- Complement of an algebraic (ish) language:

BCV15 Proof of Proof

Languages Handled by SPHFs?

- Any NP language?
  - ⇒ polynomial hierarchy collapses [GGSW13]
  - But any encryption of an NP-Language ...
- 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:

    Double

    Gapless
- Complement of an algebraic (ish) language:

BCV15 Proof of Proof

Languages Handled by SPHFs?

- Any NP language?
  - ⇒ polynomial hierarchy collapses [GGSW13]
  - But any encryption of an NP-Language . . .
- 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 *Doable*

    BBBCG19? Gapless
- Complement of an algebraic (ish) language:

BCV15 Proof of Proof

Languages Handled by SPHFs?

- Any NP language?
  - $\Rightarrow$ polynomial hierarchy collapses [GGSW13]
  - But any encryption of an NP-Language ...
- 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 *Doable*

    BBBCG19? Gapless
- Complement of an algebraic (ish) language:

  BCV15 Proof of Proof

Languages Handled by SPHFs?

- Any NP language?
  - $\Rightarrow$ polynomial hierarchy collapses [GGSW13]
  - But any encryption of an NP-Language . . .
- 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 *Doable*
BBBCG19? Gapless

- Complement of an algebraic (ish) language:

BCV15 Proof of Proof

Languages Handled by SPHFs?

- Any NP language?
  - ⇒ polynomial hierarchy collapses [GGSW13]
  - But any encryption of an NP-Language ...
- 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 *Doable*

    BBBCG19? Gapless
- Complement of an algebraic (ish) language:

BCV15 Proof of Proof

Languages Handled by SPHFs?

- Any NP language?
  - $\Rightarrow$ polynomial hierarchy collapses [GGSW13]
  - But any encryption of an NP-Language ...
- 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 *Doable*

    BBBCG19? Gapless
- Complement of an algebraic (ish) language:

BCV15 Proof of Proof

ElGamal Encryption Scheme

$(\mathbb{G}, +)$ 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: $r \stackrel{R}{\leftarrow} \mathbb{Z}_p$

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

- Decryption:

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

ElGamal Encryption Scheme

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

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

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

- Decryption:

$$\boxed{M} = \boxed{v} - \boxed{u} \cdot s$$

ElGamal Encryption Scheme

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

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

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

- Decryption:

$$\boxed{M} = \boxed{v} - \boxed{u} \cdot s$$

ElGamal Encryption Scheme

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

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

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

- Decryption:

$$\boxed{M} = \boxed{v} - \boxed{u} \cdot s$$

SPHF for ElGamal

Language of ciphertexts of $\boxed{M} = \boxed{0}$:

$$\mathcal{L} = \left\{ \boxed{\mathbf{c}} = (\boxed{u}, \boxed{v}) \;\middle|\; \exists r \in \mathbb{Z}_p,\ \boxed{u} = \boxed{g} \cdot r \quad \text{and} \quad \boxed{v} = \boxed{h} \cdot r \right\}$$

Hashing key	hk	$= (\alpha, \beta) \stackrel{R}{\leftarrow} \mathbb{Z}_p^2$
Projection Key	$\boxed{\text{hp}}$	$= \alpha \cdot \boxed{g} + \beta \cdot \boxed{h} = (\alpha \quad \beta) \cdot \begin{pmatrix} \boxed{g} \\ \boxed{h} \end{pmatrix}$
Hash value	$\boxed{\text{H}}$	$= \alpha \cdot \boxed{u} + \beta \cdot \boxed{v} = (\alpha \quad \beta) \cdot \boxed{\mathbf{c}}$
Projected hash value	$\boxed{\text{H}'}$	$= \boxed{\text{hp}} \cdot r$

SPHF for ElGamal

Language of ciphertexts of $\boxed{M} = \boxed{0}$:

$$\mathcal{L} = \left\{ \boxed{\mathbf{c}} = (\boxed{u}, \boxed{v}) \;\middle|\; \exists r \in \mathbb{Z}_p,\ \boxed{u} = \boxed{g} \cdot r \quad \text{and} \quad \boxed{v} = \boxed{h} \cdot r \right\}$$

Hashing key	hk	$= (\alpha, \beta) \stackrel{R}{\leftarrow} \mathbb{Z}_p^2$
Projection Key	$\boxed{\text{hp}}$	$= \alpha \cdot \boxed{g} + \beta \cdot \boxed{h} = \begin{pmatrix} \alpha & \beta \end{pmatrix} \cdot \begin{pmatrix} \boxed{g} \\ \boxed{h} \end{pmatrix}$
Hash value	$\boxed{\text{H}}$	$= \alpha \cdot \boxed{u} + \beta \cdot \boxed{v} = \begin{pmatrix} \alpha & \beta \end{pmatrix} \cdot \boxed{\mathbf{c}}$
Projected hash value	$\boxed{\text{H}'}$	$= \boxed{\text{hp}} \cdot r$

Correctness

If $\boxed{u} = r \cdot \boxed{g}$ and $\boxed{v} = r \cdot \boxed{h}$}:

$$\boxed{\text{H}} = \alpha \cdot \boxed{u} + \beta \cdot \boxed{v} = (\alpha \cdot \boxed{g} + \beta \cdot \boxed{h}) \cdot r = \boxed{\text{hp}} \cdot r = \boxed{\text{H}'}$$

SPHF for ElGamal

Language of ciphertexts of $\boxed{M} = \boxed{0}$:

$$\mathcal{L} = \left\{ \boxed{\mathbf{c}} = \begin{pmatrix} \boxed{u} \\ \boxed{v} \end{pmatrix} \;\middle|\; \exists r \in \mathbb{Z}_p,\ \mathbf{c} = \begin{pmatrix} \boxed{u} \\ \boxed{v} \end{pmatrix} = \begin{pmatrix} \boxed{g} \\ \boxed{h} \end{pmatrix} \cdot r \right\}$$

Hashing key	hk	$= (\alpha, \beta) \stackrel{R}{\leftarrow} \mathbb{Z}_p^2$
Projection Key	$\boxed{\text{hp}}$	$= \alpha \cdot \boxed{g} + \beta \cdot \boxed{h} \quad = (\alpha \quad \beta) \cdot \begin{pmatrix} \boxed{g} \\ \boxed{h} \end{pmatrix}$
Hash value	$\boxed{\text{H}}$	$= \alpha \cdot \boxed{u} + \beta \cdot \boxed{v} \quad = (\alpha \quad \beta) \cdot \boxed{\mathbf{c}}$
Projected hash value	$\boxed{\text{H}'}$	$= \boxed{\text{hp}} \cdot r$

Correctness

If $\boxed{u} = \boxed{g} \cdot r$ and $\boxed{v} = \boxed{h} \cdot r$:

$$\boxed{\text{H}} = (\alpha \quad \beta) \cdot \boxed{\mathbf{c}} = (\alpha \quad \beta) \cdot \begin{pmatrix} \boxed{g} \\ \boxed{h} \end{pmatrix} \cdot r = \boxed{\text{hp}} \cdot r = \boxed{\text{H}'}$$

SPHF for ElGamal

Language of ciphertexts of $\boxed{M} = \boxed{0}$:

$$\mathcal{L} = \left\{ \boxed{\mathbf{c}} = \begin{pmatrix} \boxed{u} \\ \boxed{v} \end{pmatrix} \middle| \exists r \in \mathbb{Z}_p,\ \mathbf{c} = \begin{pmatrix} \boxed{u} \\ \boxed{v} \end{pmatrix} = \begin{pmatrix} \boxed{g} \\ \boxed{h} \end{pmatrix} \cdot r \right\}$$

Hashing key	hk	$= (\alpha, \beta) \stackrel{R}{\leftarrow} \mathbb{Z}_p^2$
Projection Key	$\boxed{\text{hp}}$	$= \alpha \cdot \boxed{g} + \beta \cdot \boxed{h} \quad = (\alpha \quad \beta) \cdot \begin{pmatrix} \boxed{g} \\ \boxed{h} \end{pmatrix}$
Hash value	$\boxed{\text{H}}$	$= \alpha \cdot \boxed{u} + \beta \cdot \boxed{v} \quad = (\alpha \quad \beta) \cdot \boxed{\mathbf{c}}$
Projected hash value	$\boxed{\text{H}'}$	$= \boxed{\text{hp}} \cdot r$

Smoothness

$$\begin{pmatrix} \boxed{\text{hp}} & \boxed{\text{H}} \end{pmatrix} = (\alpha \quad \beta) \cdot \begin{pmatrix} \boxed{g} & \boxed{u} \\ \boxed{h} & \boxed{v} \end{pmatrix}$$

and $\begin{pmatrix} \boxed{g} & \boxed{u} \\ \boxed{h} & \boxed{v} \end{pmatrix}$ is invertible iff $\boxed{\mathbf{c}} \notin \mathcal{L}$

LWE-Based IND-CPA Encryption à la Micciancio-Peikert

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 $M \in \{0,1\}$: $s \xleftarrow{R} \mathbb{Z}_q^n$, noise $\mathbf{e}$:

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

- Decrypt to:

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

- 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 $M \in \{0,1\}$: $s \xleftarrow{R} \mathbb{Z}_q^n$, noise $\mathbf{e}$:

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

- Decrypt to:

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

LWE-Based IND-CPA Encryption à la Micciancio-Peikert

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 $M \in \{0,1\}$: $s \stackrel{R}{\leftarrow} \mathbb{Z}_q^n$, noise $\mathbf{e}$:

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

- Decrypt to:

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

LWE-Based IND-CPA Encryption à la Micciancio-Peikert

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 $M \in \{0, 1\}$: $s \stackrel{R}{\leftarrow} \mathbb{Z}_q^n$, noise $\mathbf{e}$:

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

- Decrypt to:

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

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

LWE-Based IND-CPA Encryption à la Micciancio-Peikert

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

Correctness?

No!

$$\mathsf{H} = \mathbf{h} \cdot \mathbf{c} \approx \mathbf{p} \cdot \mathbf{s} = \mathsf{H}'$$

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

Correctness?

No! $\rightarrow$ rounding function R

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

with reasonable proba if R well chosen

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

Smoothness?

Choose:

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

Smoothness:

- Unlikely to hold for: $\mathbf{c} \notin L \iff d(\mathbf{c}, \Lambda) > B$
  $\longrightarrow$ only for: $\mathbf{c}$ not decrypt to 0 $\iff d(\mathbf{c}, \Lambda) > B'$ with $B' \gg B$

- Still insufficient: if

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

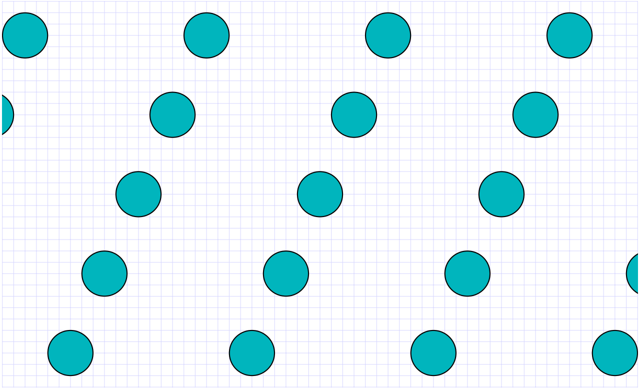
then

$$\mathrm{H} = R(\mathbf{h} \cdot \mathbf{c}) = R(\mathbf{p} \cdot \mathbf{s} + \frac{h_m \cdot q}{3} \bmod q)$$

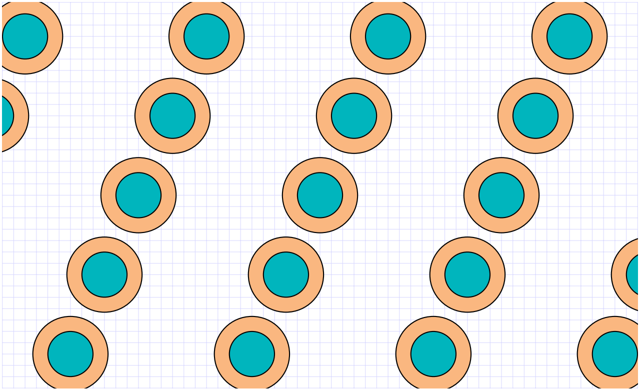
heavily biased

Encrypting with LWE: Languages

Correctness for valid encryption of 0:
$\mathcal{L} = \{\mathbf{c} \mid \exists \mathbf{s} \in \mathbb{Z}_q^n,\ \exists \mathbf{e} \in \mathbb{Z}_q^m,\ \|\mathbf{e}\| \leq B, \quad \mathbf{c} = \mathbf{A} \cdot \mathbf{s} + \mathbf{e}\}$
Smoothness for ciphertexts that do not decrypt to 0:
$\mathcal{L} = \{\mathbf{c} \mid \exists \mathbf{s} \in \mathbb{Z}_q^n,\ \exists \mathbf{e} \in \mathbb{Z}_q^m,\ \|\mathbf{e}\| \leq B', \quad \mathbf{c} = \mathbf{A} \cdot \mathbf{s} + \mathbf{e}\}$

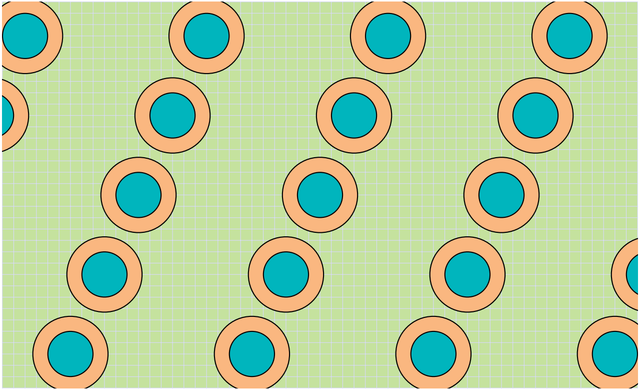


Correctness for valid encryption of 0:
$\mathcal{L} = \{\mathbf{c} \mid \exists \mathbf{s} \in \mathbb{Z}_q^n,\ \exists \mathbf{e} \in \mathbb{Z}_q^m,\ \|\mathbf{e}\| \leq B, \quad \mathbf{c} = \mathbf{A} \cdot \mathbf{s} + \mathbf{e}\}$
Smoothness for ciphertexts that do not decrypt to 0:
$\mathcal{L} = \{\mathbf{c} \mid \exists \mathbf{s} \in \mathbb{Z}_q^n,\ \exists \mathbf{e} \in \mathbb{Z}_q^m,\ \|\mathbf{e}\| \leq B', \quad \mathbf{c} = \mathbf{A} \cdot \mathbf{s} + \mathbf{e}\}$



Encrypting with LWE: Languages

Correctness for valid encryption of 0:
$\mathcal{L} = \{\mathbf{c} \mid \exists \mathbf{s} \in \mathbb{Z}_q^n,\ \exists \mathbf{e} \in \mathbb{Z}_q^m,\ \|\mathbf{e}\| \leq B, \quad \mathbf{c} = \mathbf{A} \cdot \mathbf{s} + \mathbf{e}\}$
Smoothness for ciphertexts that do not decrypt to 0:
$\mathcal{L} = \{\mathbf{c} \mid \exists \mathbf{s} \in \mathbb{Z}_q^n,\ \exists \mathbf{e} \in \mathbb{Z}_q^m,\ \|\mathbf{e}\| \leq B', \quad \mathbf{c} = \mathbf{A} \cdot \mathbf{s} + \mathbf{e}\}$



Choose:

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

Smoothness:

- Unlikely to hold for: $\mathbf{c} \notin L \iff d(\mathbf{c}, \Lambda) > B$
  $\longrightarrow$ only for: $\mathbf{c}$ not decrypt to 0 $\iff d(\mathbf{c}, \Lambda) > B'$ with $B' \gg B$
- Still insufficient: if

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

then

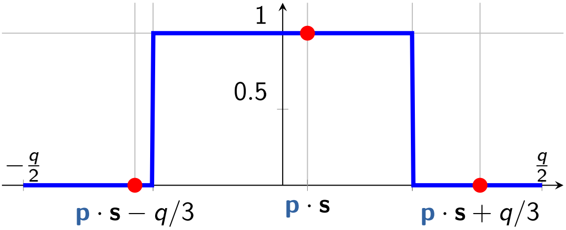
$$\mathrm{H} = R(\mathbf{h} \cdot \mathbf{c}) = R(\mathbf{p} \cdot \mathbf{s} + \frac{h_m \cdot q}{3} \bmod q)$$

heavily biased

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

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

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



LWE-Based IND-CPA Encryption à la Micciancio-Peikert

Smoothness — Solution of Katz and Vaikuntanathan

Basically smoothness when

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
  $\rightarrow$ in particular unlikely to get word-independent SPHF

Basically smoothness when

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
  $\rightarrow$ in particular unlikely to get word-independent SPHF

Let's Introduce Harmonic Analysis

Lattice $\Lambda = \{\mathbf{A} \cdot \mathbf{s} \mid \mathbf{s} \in \mathbb{Z}_q^n\}$
SPHF correct for $\mathbf{c}$ s.t. $d(\mathbf{c}, \Lambda) \leq B$; smooth for $\mathbf{c}$ s.t. $d(\mathbf{c}, \Lambda) > B'$

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

Let's Introduce Harmonic Analysis

Lattice $\Lambda = \{\mathbf{A} \cdot \mathbf{s} \mid \mathbf{s} \in \mathbb{Z}_q^n\}$
SPHF correct for $\mathbf{c}$ s.t. $d(\mathbf{c}, \Lambda) \leq B$; smooth for $\mathbf{c}$ s.t. $d(\mathbf{c}, \Lambda) > B'$

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

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

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

Recall for $R(x) = 1 + \left\lfloor \frac{2x}{q} \right\rceil \bmod 2$: $\hat{r}_j = \Theta(1/j)$ for even j.

Let's Introduce Harmonic Analysis

Lattice $\Lambda = \{\mathbf{A} \cdot \mathbf{s} \mid \mathbf{s} \in \mathbb{Z}_q^n\}$
SPHF correct for $\mathbf{c}$ s.t. $d(\mathbf{c}, \Lambda) \leq B$; smooth for $\mathbf{c}$ s.t. $d(\mathbf{c}, \Lambda) > B'$

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

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

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

Recall for $R(x) = 1 + \left\lfloor \frac{2x}{q} \right\rceil \bmod 2$: $\quad \hat{r}_j = \Theta(1/j)$ for even j.

Let's Introduce Harmonic Analysis

Lattice $\Lambda = \{\mathbf{A} \cdot \mathbf{s} \mid \mathbf{s} \in \mathbb{Z}_q^n\}$
SPHF correct for $\mathbf{c}$ s.t. $d(\mathbf{c}, \Lambda) \leq B$; smooth for $\mathbf{c}$ s.t. $d(\mathbf{c}, \Lambda) > B'$

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

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

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

Recall for $R(x) = 1 + \left\lfloor \frac{2x}{q} \right\rceil \bmod 2$: $\quad \hat{r}_j = \Theta(1/j)$ for even j.

Let's Introduce Harmonic Analysis

Lattice $\Lambda = \{\mathbf{A} \cdot \mathbf{s} \mid \mathbf{s} \in \mathbb{Z}_q^n\}$
SPHF correct for $\mathbf{c}$ s.t. $d(\mathbf{c}, \Lambda) \leq B$; smooth for $\mathbf{c}$ s.t. $d(\mathbf{c}, \Lambda) > B'$

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

Gaussian weight function on $\mathbb{Z}^m$ for parameter $t > 0$:

$$\rho_t(\mathbf{x}) = exp(-\frac{\pi \|\mathbf{x}\|^2}{t^2})$$

Discrete Gaussian distribution $D_{\Lambda,t}$ over a lattice Λ:

$$\forall \mathbf{x} \in \Lambda, \quad D_{\Lambda,t}(\mathbf{x}) = \rho_t(\mathbf{x}) / \rho_t(\Lambda)$$

Let's Introduce Harmonic Analysis

Lattice $\Lambda = \{\mathbf{A} \cdot \mathbf{s} \mid \mathbf{s} \in \mathbb{Z}_q^n\}$
SPHF correct for $\mathbf{c}$ s.t. $d(\mathbf{c}, \Lambda) \leq B$; smooth for $\mathbf{c}$ s.t. $d(\mathbf{c}, \Lambda) > B'$

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

Fix $\mathbf{p} = \mathbf{h}_0 \cdot \mathbf{A}$ and $\mathbf{c}$ far from Λ, goal: $P = \Pr[R(\mathbf{h} \cdot \mathbf{c}) = 1] \approx 1/2$
Using harmonic analysis (Poisson formula):

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

where $\rho'_{q/t}(\Lambda - j \cdot \mathbf{c}) = \sum_{\mathbf{y} \in \Lambda - j \cdot \mathbf{c}} \rho_{q/t}(\mathbf{y}) \cdot e^{2i\pi \mathbf{h}_0 \cdot \mathbf{y}/q}$

Let's Introduce Harmonic Analysis

Lattice $\Lambda = \{\mathbf{A} \cdot \mathbf{s} \mid \mathbf{s} \in \mathbb{Z}_q^n\}$
SPHF correct for $\mathbf{c}$ s.t. $d(\mathbf{c}, \Lambda) \leq B$; smooth for $\mathbf{c}$ s.t. $d(\mathbf{c}, \Lambda) > B'$

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

Fix $\mathbf{p} = \mathbf{h}_0 \cdot \mathbf{A}$ and $\mathbf{c}$ far from Λ, goal: $P = \Pr[R(\mathbf{h} \cdot \mathbf{c}) = 1] \approx 1/2$
Using harmonic analysis (Poisson formula):

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

where $\rho'_{q/t}(\Lambda - j \cdot \mathbf{c}) = \sum_{\mathbf{y} \in \Lambda - j \cdot \mathbf{c}} \rho_{q/t}(\mathbf{y}) \cdot e^{2i\pi \mathbf{h}_0 \cdot \mathbf{y}/q}$

Let's Introduce Harmonic Analysis (2)

Fix $\mathbf{p}$ and $\mathbf{c}$, goal: $P = \Pr[R(\mathbf{h} \cdot \mathbf{c}) = 1] = 1/2 + \text{negl}$

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

1. if balanced r, for t large enough: $\hat{r}_0 \cdot \rho'_{q/t}(\Lambda) \approx 1/2$
2. for t large enough:

$$\rho'_{q/t}(\Lambda - j \cdot \mathbf{c}) \approx \begin{cases} 0 & \text{if } j \cdot \mathbf{c} \text{ far from } \Lambda \\ 1 & \text{if } j \cdot \mathbf{c} \text{ close to } \Lambda \end{cases}$$

$\longrightarrow$ for $j = 1$: $\rho'_{q/t}(\Lambda - j \cdot \mathbf{c}) \approx 0$

Let's Introduce Harmonic Analysis (2)

Fix $\mathbf{p}$ and $\mathbf{c}$, goal: $P = \Pr[R(\mathbf{h} \cdot \mathbf{c}) = 1] = 1/2 + \mathrm{negl}$

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

1. if balanced r, for t large enough: $\hat{r}_0 \cdot \rho'_{q/t}(\Lambda) \approx 1/2$
2. for t large enough:

$$\rho'_{q/t}(\Lambda - j \cdot \mathbf{c}) \approx \begin{cases} 0 & \text{if } j \cdot \mathbf{c} \text{ far from } \Lambda \\ 1 & \text{if } j \cdot \mathbf{c} \text{ close to } \Lambda \end{cases}$$

$\longrightarrow$ for $j = 1$: $\rho'_{q/t}(\Lambda - j \cdot \mathbf{c}) \approx 0$

Solution KV

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

Let's Introduce Harmonic Analysis (2)

Fix $\mathbf{p}$ and $\mathbf{c}$, goal: $P = \Pr[R(\mathbf{h} \cdot \mathbf{c}) = 1] = 1/2 + \text{negl}$

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

1. if balanced r, for t large enough: $\hat{r}_0 \cdot \rho'_{q/t}(\Lambda) \approx 1/2$
2. for t large enough:

$$\rho'_{q/t}(\Lambda - j \cdot \mathbf{c}) \approx \begin{cases} 0 & \text{if } j \cdot \mathbf{c} \text{ far from } \Lambda \\ 1 & \text{if } j \cdot \mathbf{c} \text{ close to } \Lambda \end{cases}$$

$\longrightarrow$ for $j = 1$: $\rho'_{q/t}(\Lambda - j \cdot \mathbf{c}) \approx 0$

Our solution

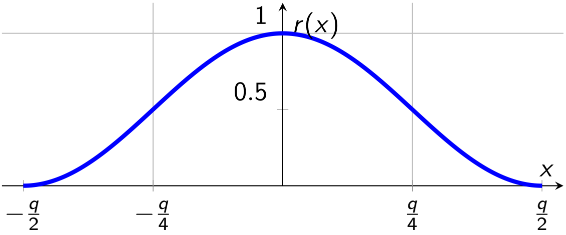
Ensure that $\hat{r}_j = 0$ for $j \geq 2$, i.e.:

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

Construction

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

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



Construction

Hashing key	$\mathsf{hk} = \mathbf{h}$	$\stackrel{R}{\leftarrow} D_{\mathbb{Z},t}^{1\times m}$
Projection Key	$\mathsf{hp} = \mathbf{p}$	$= \mathbf{h} \cdot \mathbf{A}$
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

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

$\longrightarrow$ can be extended to classical correctness
(ECC + parallel repetitions)

Drawback: word-dependent

Construction

Hashing key	$\mathsf{hk} = \mathbf{h}$	$\xleftarrow{R} D_{\mathbb{Z},t}^{1\times m}$
Projection Key	$\mathsf{hp} = \mathbf{p}$	$= \mathbf{h} \cdot \mathbf{A}$
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

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

$\longrightarrow$ can be extended to classical correctness
(ECC + parallel repetitions)
Drawback: word-dependent

NIST Candidate HQC/RQC

- Setup($1^{\mathfrak{K}}$): Generates param $= (n, k, \delta, w, w_r, w_{\mathbf{e}}) \in \mathbb{N}^6$.
- KeyGen(param): Let denotes $\mathcal{R} = \mathcal{F}_{q^m}[X]/(X^n - 1)$. Samples $\mathbf{h} \stackrel{\$}{\leftarrow} \mathcal{R}$, $\mathbf{G} \in \mathcal{M}_{k,n}(\mathcal{F}_{q^m})$ of $\mathcal{C}$,
  sk $= (\mathbf{x}, \mathbf{y}) \stackrel{\$}{\leftarrow} \mathcal{R}^2$ $(\omega(\mathbf{x}) = \omega(\mathbf{y}) = w)$, pk $= (\mathbf{h}, \mathbf{s} = \mathbf{x} + \mathbf{h} \cdot \mathbf{y})$,
  returns (pk, sk).
- Encrypt(pk, $\mathbf{m}$): $\mathbf{r} = (\mathbf{r}_1, \mathbf{r}_2, \mathbf{r}_3, \stackrel{\$}{\leftarrow} \mathcal{R}^3$ $(\ \omega(\mathbf{r}_*) = w_{\mathbf{r}})$,
  $\mathbf{c}_1 = \mathbf{r}_1 + \mathbf{h} \cdot \mathbf{r}_2$, $\mathbf{c}_2 = \mathbf{mG} + \mathbf{s} \cdot \mathbf{r}_2 + \mathbf{r}_3$,
  returns $\mathbf{c} = (\mathbf{c}_1, \mathbf{c}_2)$.
- Decrypt(sk, $\mathbf{c}$): Returns $\mathcal{C}$.Decode$(\mathbf{c}_2 - \mathbf{y} \cdot \mathbf{c}_1)$.

- Naive ProjHash: hp $= \mathbf{h} \cdot \alpha_1 + \mathbf{s} \cdot \alpha_2 + \alpha_3$

NIST Candidate HQC/RQC

- Setup($1^{\mathfrak{K}}$): Generates param $= (n, k, \delta, w, w_r, w_{\mathbf{e}}) \in \mathbb{N}^6$.
- KeyGen(param): Let denotes $\mathcal{R} = \mathcal{F}_{q^m}[X]/(X^n - 1)$. Samples $\mathbf{h} \stackrel{\$}{\leftarrow} \mathcal{R}$, $\mathbf{G} \in \mathcal{M}_{k,n}(\mathcal{F}_{q^m})$ of $\mathcal{C}$,
  $\mathsf{sk} = (\mathbf{x}, \mathbf{y}) \stackrel{\$}{\leftarrow} \mathcal{R}^2$ $(\omega(\mathbf{x}) = \omega(\mathbf{y}) = w)$, $\mathsf{pk} = (\mathbf{h}, \mathbf{s} = \mathbf{x} + \mathbf{h} \cdot \mathbf{y})$,
  returns (pk, sk).
- Encrypt(pk, $\mathbf{m}$): $\mathbf{r} = (\mathbf{r}_1, \mathbf{r}_2, \mathbf{r}_3, \stackrel{\$}{\leftarrow} \mathcal{R}^3$ $(\omega(\mathbf{r}_*) = w_{\mathbf{r}})$,
  $\mathbf{c}_1 = \mathbf{r}_1 + \mathbf{h} \cdot \mathbf{r}_2$, $\mathbf{c}_2 = \mathbf{mG} + \mathbf{s} \cdot \mathbf{r}_2 + \mathbf{r}_3$,
  returns $\mathbf{c} = (\mathbf{c}_1, \mathbf{c}_2)$.
- Decrypt(sk, $\mathbf{c}$): Returns $\mathcal{C}$.Decode($\mathbf{c}_2 - \mathbf{y} \cdot \mathbf{c}_1$).

- Naive ProjHash: $\mathsf{hp} = \mathbf{h} \cdot \alpha_1 + \mathbf{s} \cdot \alpha_2 + \alpha_3$

NIST Candidate HQC/RQC

- Setup($1^{\mathfrak{K}}$): Generates param $= (n, k, \delta, w, w_r, w_{\mathbf{e}}) \in \mathbb{N}^6$.
- KeyGen(param): Let denotes $\mathcal{R} = \mathcal{F}_{q^m}[X]/(X^n - 1)$. Samples $\mathbf{h} \stackrel{\$}{\leftarrow} \mathcal{R}$, $\mathbf{G} \in \mathcal{M}_{k,n}(\mathcal{F}_{q^m})$ of $\mathcal{C}$,
  $\mathsf{sk} = (\mathbf{x}, \mathbf{y}) \stackrel{\$}{\leftarrow} \mathcal{R}^2$ $(\omega(\mathbf{x}) = \omega(\mathbf{y}) = w)$, $\mathsf{pk} = (\mathbf{h}, \mathbf{s} = \mathbf{x} + \mathbf{h} \cdot \mathbf{y})$,
  returns $(\mathsf{pk}, \mathsf{sk})$.
- Encrypt($\mathsf{pk}, \mathbf{m}$): $\mathbf{r} = (\mathbf{r}_1, \mathbf{r}_2, \mathbf{r}_3, \stackrel{\$}{\leftarrow} \mathcal{R}^3$ $(\ \omega(\mathbf{r}_*) = w_{\mathbf{r}})$,
  $\mathbf{c}_1 = \mathbf{r}_1 + \mathbf{h} \cdot \mathbf{r}_2$, $\mathbf{c}_2 = \mathbf{mG} + \mathbf{s} \cdot \mathbf{r}_2 + \mathbf{r}_3$,
  returns $\mathbf{c} = (\mathbf{c}_1, \mathbf{c}_2)$.
- Decrypt($\mathsf{sk}, \mathbf{c}$): Returns $\mathcal{C}$.Decode($\mathbf{c}_2 - \mathbf{y} \cdot \mathbf{c}_1$).
- Naive ProjHash: $\mathsf{hp} = \mathbf{h} \cdot \alpha_1 + \mathbf{s} \cdot \alpha_2 + \alpha_3$

NIST Candidate HQC/RQC

- Setup($1^{\mathfrak{K}}$): Generates param $= (n, k, \delta, w, w_r, w_{\mathbf{e}}) \in \mathbb{N}^6$.
- KeyGen(param): Let denotes $\mathcal{R} = \mathcal{F}_{q^m}[X]/(X^n - 1)$. Samples $\mathbf{h} \stackrel{\$}{\leftarrow} \mathcal{R}$, $\mathbf{G} \in \mathcal{M}_{k,n}(\mathcal{F}_{q^m})$ of $\mathcal{C}$,
  sk $= (\mathbf{x}, \mathbf{y}) \stackrel{\$}{\leftarrow} \mathcal{R}^2$ $(\omega(\mathbf{x}) = \omega(\mathbf{y}) = w)$, pk $= (\mathbf{h}, \mathbf{s} = \mathbf{x} + \mathbf{h} \cdot \mathbf{y})$,
  returns (pk, sk).
- Encrypt(pk, $\mathbf{m}$): $\mathbf{r} = (\mathbf{r}_1, \mathbf{r}_2, \mathbf{r}_3, \stackrel{\$}{\leftarrow} \mathcal{R}^3$ ($\omega(\mathbf{r}_*) = w_{\mathbf{r}}$),
  $\mathbf{c}_1 = \mathbf{r}_1 + \mathbf{h} \cdot \mathbf{r}_2$, $\mathbf{c}_2 = \mathbf{mG} + \mathbf{s} \cdot \mathbf{r}_2 + \mathbf{r}_3$,
  returns $\mathbf{c} = (\mathbf{c}_1, \mathbf{c}_2)$.
- Decrypt(sk, $\mathbf{c}$): Returns $\mathcal{C}$.Decode($\mathbf{c}_2 - \mathbf{y} \cdot \mathbf{c}_1$).

- Naive ProjHash: hp $= \mathbf{h} \cdot \alpha_1 + \mathbf{s} \cdot \alpha_2 + \alpha_3$

NIST Candidate HQC/RQC

- Setup($1^{\mathfrak{K}}$): Generates param $= (n, k, \delta, w, w_r, w_{\mathbf{e}}) \in \mathbb{N}^6$.
- KeyGen(param): Let denotes $\mathcal{R} = \mathcal{F}_{q^m}[X]/(X^n - 1)$. Samples $\mathbf{h} \stackrel{\$}{\leftarrow} \mathcal{R}$, $\mathbf{G} \in \mathcal{M}_{k,n}(\mathcal{F}_{q^m})$ of $\mathcal{C}$,
  sk $= (\mathbf{x}, \mathbf{y}) \stackrel{\$}{\leftarrow} \mathcal{R}^2$ $(\omega(\mathbf{x}) = \omega(\mathbf{y}) = w)$, pk $= (\mathbf{h}, \mathbf{s} = \mathbf{x} + \mathbf{h} \cdot \mathbf{y})$,
  returns (pk, sk).
- Encrypt(pk, $\mathbf{m}$): $\mathbf{r} = (\mathbf{r}_1, \mathbf{r}_2, \mathbf{r}_3, \stackrel{\$}{\leftarrow} \mathcal{R}^3$ $(\ \omega(\mathbf{r}_*) = w_{\mathbf{r}})$,
  $\mathbf{c}_1 = \mathbf{r}_1 + \mathbf{h} \cdot \mathbf{r}_2$, $\mathbf{c}_2 = \mathbf{mG} + \mathbf{s} \cdot \mathbf{r}_2 + \mathbf{r_3}$,
  returns $\mathbf{c} = (\mathbf{c}_1, \mathbf{c}_2)$.
- Decrypt(sk, $\mathbf{c}$): Returns $\mathcal{C}$.Decode$(\mathbf{c}_2 - \mathbf{y} \cdot \mathbf{c}_1)$.

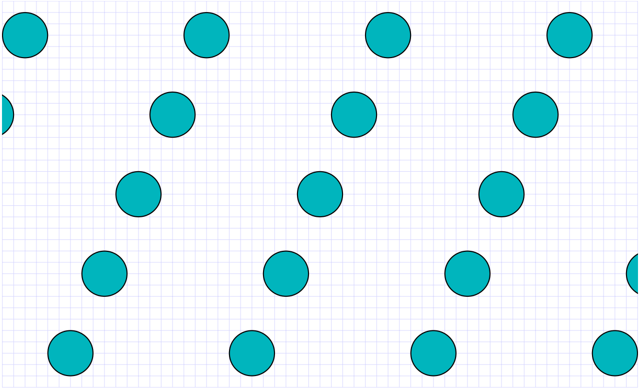
- Naive ProjHash: hp $= \mathbf{h} \cdot \boldsymbol{\alpha}_1 + \mathbf{s} \cdot \boldsymbol{\alpha}_2 + \boldsymbol{\alpha}_3$

Encrypting with HQC/RQC: Languages

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

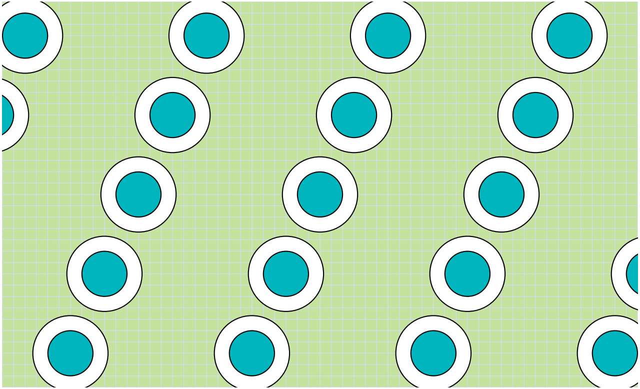
Smoothness for ciphertexts that do not decrypt to 0:

$\mathcal{L} = \{\mathbf{c} \mid \exists \mathbf{r}_* \in \mathcal{R}; \|\mathbf{r}\| \leq w_r, \quad \mathbf{c} = (\mathbf{r}_1 + \mathbf{h} \cdot \mathbf{r}_2, \mathbf{s} \cdot \mathbf{r}_2 + \mathbf{r}_3)\}$



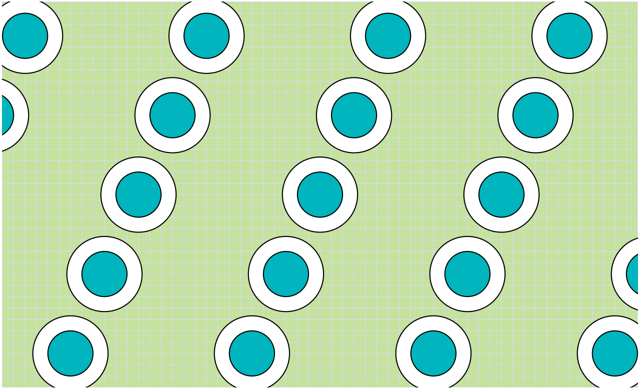
Encrypting with HQC/RQC: Languages

Correctness for valid encryption of 0: $\mathcal{L} = \{\mathbf{c} \mid \exists \mathbf{r}_* \in \mathcal{R}; \|\mathbf{r}\| \leq w_r, \quad \mathbf{c} = \ldots\}$
Smoothness for ciphertexts that do not decrypt to 0:
$\mathcal{L} = \{\mathbf{c} \mid \exists \mathbf{r}_* \in \mathcal{R}; \|\mathbf{r}\| \leq w_r, \quad \mathbf{c} = (\mathbf{r}_1 + \mathbf{h} \cdot \mathbf{r}_2, \mathbf{s} \cdot \mathbf{r}_2 + \mathbf{r}_3)\}$



Encrypting with HQC/RQC: Languages

Correctness for valid encryption of 0: $\mathcal{L} = \{\mathbf{c} \mid \exists \mathbf{r}_* \in \mathcal{R}; \|\mathbf{r}\| \leq w_r, \quad \mathbf{c} = \ldots\}$
Smoothness for ciphertexts that do not decrypt to 0:
$\mathcal{L} = \{\mathbf{c} \mid \exists \mathbf{r}_* \in \mathcal{R}; \|\mathbf{r}\| \leq w_r, \quad \mathbf{c} = (\mathbf{r}_1 + \mathbf{h} \cdot \mathbf{r}_2, \mathbf{s} \cdot \mathbf{r}_2 + \mathbf{r}_3)\}$



Verifiable Encryption $\rightsquigarrow$ Gapless SPHF

Supersingular Isogeny

Supersingular Isogeny

We used to rely on modules (Ring action could be enough)

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.

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.

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

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

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

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

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

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

Oblivious Transfer

Oblivious Transfer

Database | User

$\xleftarrow{\mathcal{C}(\text{line})}$

$\xrightarrow{\mathsf{hp}_i,\, \mathcal{DB}[i] \texttt{ xor } \mathsf{Hash}_i(\mathcal{C}, \mathsf{hk}_i)}$

Smoothness $\rightsquigarrow$ The User learns the value of $\mathcal{DB}[i]$ but nothing else

Indistinguishability $\rightsquigarrow$ The Database learns nothing

Efficiency of 1-out-of-n OT

UC World

Theoretical lower bound: $\log n + n * s$

***Plain* Model**

Theoretical lower bound: $\log n + s$
Just do FHE / Functional Encryption.

Efficiency of 1-out-of-n OT

UC World

Theoretical lower bound: $\log n + n * s$

Plain Model

Theoretical lower bound: $\log n + s$
Just do FHE / Functional Encryption.

Generic Password Authenticated Key Exchange

- Each user U_i computes $\mathcal{C}_i = \mathsf{Encrypt}(\mathsf{pw}_i; \mathbf{s_i})$ with a commitment, 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 commitment, 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 commitment, 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 $H'_i \cdot H_j = \mathsf{ProjHash}(\mathsf{hp}_j, \mathbf{d_i}) \cdot \mathsf{Hash}(\mathsf{hk}_i, \mathcal{C}_j)$

Password Authenticated Key Exchange

PAKE

Bob		Alice
	$\xleftarrow{\mathcal{C}(\mathsf{pw}_A),\ \mathsf{hp}_A}$ $\xrightarrow{\mathcal{C}(\mathsf{pw}_B),\ \mathsf{hp}_B}$	

$$H_B = \mathsf{Hash}(\mathcal{C}_A, \mathsf{hk}_B)$$
$$H'_A = \mathsf{ProjHash}(\mathcal{C}_B, \mathsf{hp}_A, w_B)$$

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

PseudoRandomness $\rightsquigarrow$ An Adversary learns nothing

And many more

LAKE

2 Users expect a word in a language

- Singleton {pwd}: PAKE
- Solution to an equation (Signature): Secret handshAKE
- Solution to an equation *Certified*: CAKE
- And others

And many more

LAKE

2 Users expect a word in a language

- Singleton {pwd}: PAKE
- Solution to an equation (Signature): Secret handshAKE
- Solution to an equation *Certified*: CAKE
- And others

And many more

LAKE

2 Users expect a word in a language

- Singleton {pwd}: PAKE
- Solution to an equation (Signature): Secret handshAKE
- Solution to an equation *Certified*: CAKE
- And others

And many more

LAKE

2 Users expect a word in a language

- Singleton $\{pwd\}$: PAKE
- Solution to an equation (Signature): Secret handshAKE
- Solution to an equation *Certified*: CAKE
- And others

And many more

OLBE

An information is going to be transmitted if a word is in a language

- Singleton {line}: Oblivious Transfer
- Solution to an equation: (Signature): OSBE
- Solution to an equation *Certified*: Credential Information Retrieval
- Non-Interactive solution to an equation: Witness Encryption
- And others (Conditional Oblivious Transfer, Priced OT, ...)

To sum up

- ✓ Smooth Hash Proof Systems exist on most Cryptographic Worlds
- ✓ They fit well in already interactive protocol, and allow to reduce communications
- ✓ Increase privacy by removing the validation step
- ✓ Can do classical languages proven with a ZKPK

- ? Efficient PostQuantum HPS
- ? GapLess Lattice-based HPS
- ? Isogeny-based HPS
- ? Counter-Example language for SPHF / Word Independent SPHF
- ? Other Applications

To sum up

- ✓ Smooth Hash Proof Systems exist on most Cryptographic Worlds
- ✓ They fit well in already interactive protocol, and allow to reduce communications
- ✓ Increase privacy by removing the validation step
- ✓ Can do classical languages proven with a ZKPK

- ? Efficient PostQuantum HPS
- ? GapLess Lattice-based HPS
- ? Isogeny-based HPS
- ? Counter-Example language for SPHF / Word Independent SPHF
- ? Other Applications

To sum up

- ✓ Smooth Hash Proof Systems exist on most Cryptographic Worlds
- ✓ They fit well in already interactive protocol, and allow to reduce communications
- ✓ Increase privacy by removing the validation step
- ✓ Can do classical languages proven with a ZKPK

- ? Efficient PostQuantum HPS
- ? GapLess Lattice-based HPS
- ? Isogeny-based HPS
- ? Counter-Example language for SPHF / Word Independent SPHF
- ? Other Applications

To sum up

- ✓ Smooth Hash Proof Systems exist on most Cryptographic Worlds
- ✓ They fit well in already interactive protocol, and allow to reduce communications
- ✓ Increase privacy by removing the validation step
- ✓ Can do classical languages proven with a ZKPK

- ? Efficient PostQuantum HPS
- ? GapLess Lattice-based HPS
- ? Isogeny-based HPS
- ? Counter-Example language for SPHF / Word Independent SPHF
- ? Other Applications

To sum up

- ✓ Smooth Hash Proof Systems exist on most Cryptographic Worlds
- ✓ They fit well in already interactive protocol, and allow to reduce communications
- ✓ Increase privacy by removing the validation step
- ✓ Can do classical languages proven with a ZKPK

- ? Efficient PostQuantum HPS
- ? GapLess Lattice-based HPS
- ? Isogeny-based HPS
- ? Counter-Example language for SPHF / Word Independent SPHF
- ? Other Applications

To sum up

- ✓ Smooth Hash Proof Systems exist on most Cryptographic Worlds
- ✓ They fit well in already interactive protocol, and allow to reduce communications
- ✓ Increase privacy by removing the validation step
- ✓ Can do classical languages proven with a ZKPK

- ? Efficient PostQuantum HPS
- ? GapLess Lattice-based HPS
- ? Isogeny-based HPS
- ? Counter-Example language for SPHF / Word Independent SPHF
- ? Other Applications

To sum up

- ✓ Smooth Hash Proof Systems exist on most Cryptographic Worlds
- ✓ They fit well in already interactive protocol, and allow to reduce communications
- ✓ Increase privacy by removing the validation step
- ✓ Can do classical languages proven with a ZKPK

- ? Efficient PostQuantum HPS
- ? GapLess Lattice-based HPS
- ? Isogeny-based HPS
- ? Counter-Example language for SPHF / Word Independent SPHF
- ? Other Applications

To sum up

- ✓ Smooth Hash Proof Systems exist on most Cryptographic Worlds
- ✓ They fit well in already interactive protocol, and allow to reduce communications
- ✓ Increase privacy by removing the validation step
- ✓ Can do classical languages proven with a ZKPK

- ? Efficient PostQuantum HPS
- ? GapLess Lattice-based HPS
- ? Isogeny-based HPS
- ? Counter-Example language for SPHF / Word Independent SPHF
- ? Other Applications

To sum up

- ✓ Smooth Hash Proof Systems exist on most Cryptographic Worlds
- ✓ They fit well in already interactive protocol, and allow to reduce communications
- ✓ Increase privacy by removing the validation step
- ✓ Can do classical languages proven with a ZKPK

- ? Efficient PostQuantum HPS
- ? GapLess Lattice-based HPS
- ? Isogeny-based HPS
- ? Counter-Example language for SPHF / Word Independent SPHF
- ? Other Applications