

Random Robust Secret Sharing with Perfect Privacy and its Applications

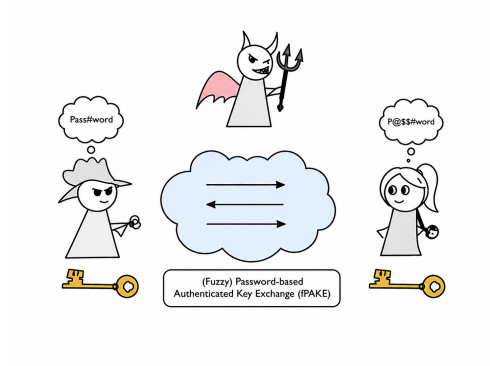
Mohammad Hassan Ameri Jeremiah Blocki

Purdue University

SPRAI Weekly Seminar, UIUC
Fall 2026

⟨To appear in ACM CCS 2026 || The Hague, Netherlands⟩

Fuzzy PAKE: Key Exchange from Close Passwords



Fuzzy PAKE: Key Exchange from Close Passwords

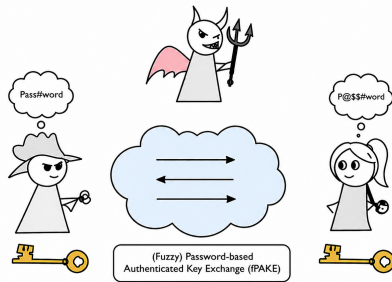
Password Authenticated Key Exchange: No PKI and no trusted third party only the users' passwords.

- **PAKE:** agree on a key when the passwords match exactly.
- **fPAKE:** agree on a key when the passwords are *sufficiently* close.

Alice: $pwd_A \in \Sigma^n$ Bob: $pwd_B \in \Sigma^n$

$\text{Ham}(pwd_A, pwd_B) \leq \ell \implies K_A = K_B$

$\text{Ham}(pwd_A, pwd_B) > \ell \implies K_A \perp K_B$



Fuzzy PAKE: Key Exchange from Close Passwords

Password Authenticated Key Exchange: No PKI and no trusted third party only the users' passwords.

- **PAKE:** agree on a key when the passwords match exactly.
- **fPAKE:** agree on a key when the passwords are *sufficiently* close.

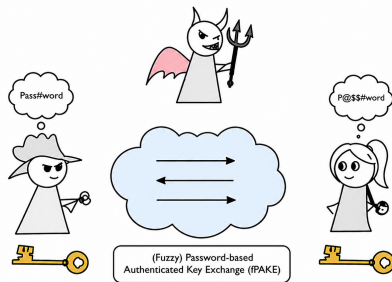
Security goal: Reveal nothing else to an attacker with wrong pwd .

- No session key/password
- No offline attack on password
- Not even which characters match

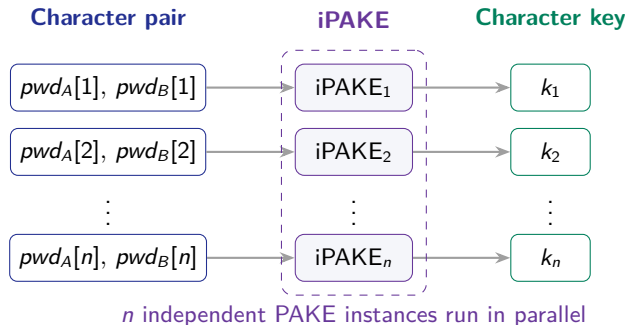
Alice: $pwd_A \in \Sigma^n$ Bob: $pwd_B \in \Sigma^n$

$\text{Ham}(pwd_A, pwd_B) \leq \ell \implies K_A = K_B$

$\text{Ham}(pwd_A, pwd_B) > \ell \implies K_A \perp K_B$



Fuzzy PAKE: Efficient Template?!



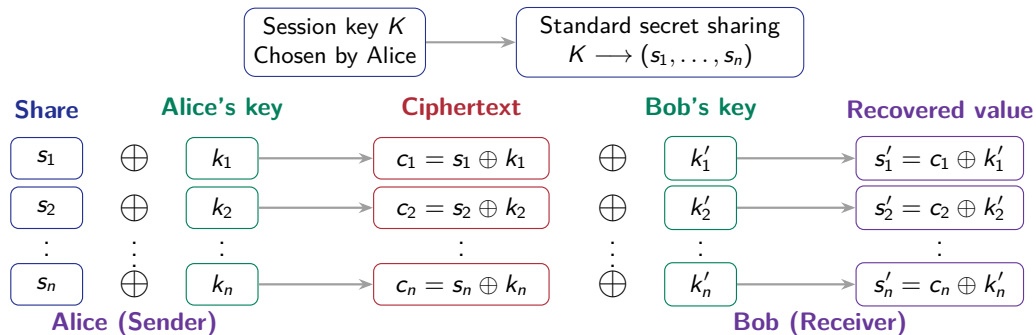
► **Matching character:**

If $pwd_A[i] = pwd_B[i]$, both parties receive the **same key** k_i from iPAKE.

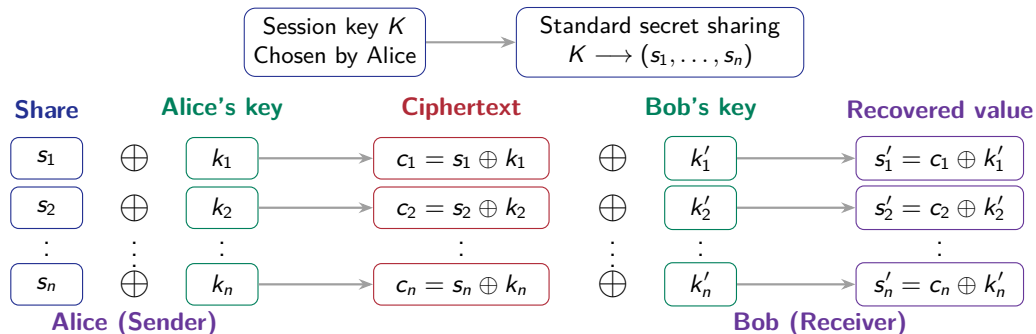
○ **Mismatching character:**

If $pwd_A[i] \neq pwd_B[i]$, they receive **independent keys** from iPAKE.

Fuzzy PAKE: Efficient Template?! (Continued)

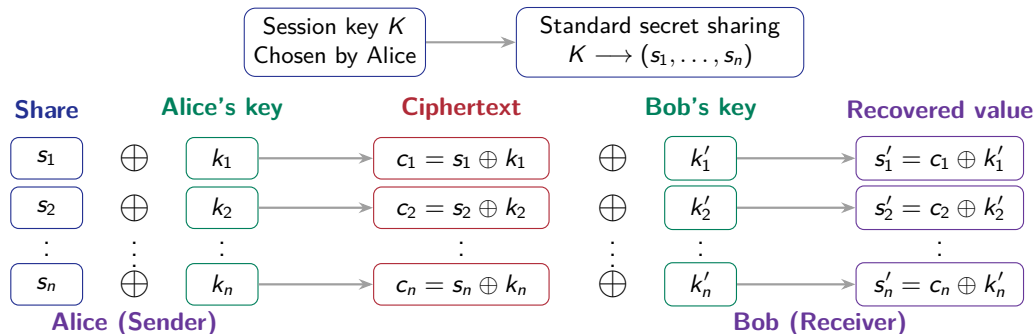


Fuzzy PAKE: Efficient Template?! (Continued)



- Matching character: Bob derives the same key $k'_i = k_i \Rightarrow s'_i = s_i$
 $\Rightarrow$ He obtains a **valid share**.

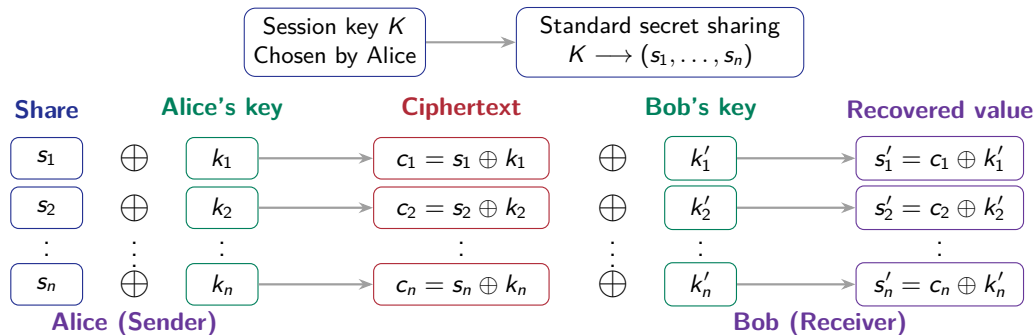
Fuzzy PAKE: Efficient Template?! (Continued)



- Matching character: Bob derives the same key $k'_i = k_i \Rightarrow s'_i = s_i$
 $\Rightarrow$ He obtains a **valid share**.

- Mismatching character: $k'_i \neq k_i$
 $\Rightarrow s'_i = s_i \oplus k_i \oplus k'_i$
 $\Rightarrow$ He obtains a **uniform random string**.

Fuzzy PAKE: Efficient Template?! (Continued)



- Matching character: Bob derives the same key $k'_i = k_i \Rightarrow s'_i = s_i$
 $\Rightarrow$ He obtains a **valid share**.

- Mismatching character: $k'_i \neq k_i$
 $\Rightarrow s'_i = s_i \oplus k_i \oplus k'_i$
 $\Rightarrow$ He obtains a **uniform random string**.

At least $t = n - \ell$ matching characters $\Rightarrow \exists$ enough valid shares to reconstruct K .

Problem 1: Standard (e.g., Shamir) Secret Sharing Is Not Efficient Here!

Bob receives a mixture of candidate shares:



- Standard reconstruction assumes the correct shares are known.
- Problem: Bob does **not** know which candidate shares are valid.
- Brute force?** $\binom{n}{\ell}$ combinations $\Rightarrow$ **Not Efficient**

We need robustness.

A robust reconstruction should recover K without being told the valid locations.

Problem 2: Standard Secrecy Guarantee is Insufficient!

- ✓ **Standard secrecy:** Any $t - 1$ shares reveal no information about the secret K .
- ✓ $K = K_0$ and $K = K_1$ induce the same view.

The secret is hidden.

Problem 2: Standard Secrecy Guarantee is Insufficient!

✓ **Standard secrecy:** Any $t - 1$ shares reveal no information about the secret K .

✓ $K = K_0$ and $K = K_1$ induce the same view.

The secret is hidden.

✗ **Problem:** Share may not Look Random
⇒ Valid robust shares need not look like uniformly random strings.

⇒ valid encoding vs. random mismatch

Their joint distributions may be distinguishable from uniform.

Problem 2: Standard Secrecy Guarantee is Insufficient!

✓ **Standard secrecy:** Any $t - 1$ shares reveal no information about the secret K .

✓ $K = K_0$ and $K = K_1$ induce the same view.

The secret is hidden.

✗ **Problem:** Share may not Look Random
 $\Rightarrow$ Valid robust shares need not look like uniformly random strings.

$\Rightarrow$ valid encoding vs. random mismatch

Their joint distributions may be distinguishable from uniform.

Distinguish **valid** from **invalid** (potentially Random) candidate shares $\Rightarrow$ learn which password characters match $\Rightarrow$ **Breaking fPAKE security**

Robust Secret Sharing Schemes: shares typically distinguishable from random string

The Missing Guarantee Is $t - 1$ -Perfect Privacy

- **$(t - 1)$ -privacy (Standard):**

For every $|S| \leq t - 1$, the shares $(s_i)_{i \in S}$ reveal no information about the secret K .

- **$(t - 1)$ -perfect privacy (Stronger)**

For every $S \subseteq [n]$ with $|S| \leq t - 1$:

*The joint distribution of $(s_i)_{i \in S}$ is **uniform***

$\Rightarrow$ Valid shares look exactly like **random mismatch** values.

Prior fPAKE constructions [EC18:DHPRY], [AC23:BFHHO]:

- Secret sharing with t_1 -perfect privacy and t_2 -robustness
- **Gap:** $t_1 < t_2 - 1$
- **Gap $\rightarrow$ Significantly weaker/relaxed security guarantee**

The Missing Guarantee Is $t - 1$ -Perfect Privacy

- **$(t - 1)$ -privacy (Standard):**

For every $|S| \leq t - 1$, the shares $(s_i)_{i \in S}$ reveal no information about the secret K .

- **$(t - 1)$ -perfect privacy (Stronger)**

For every $S \subseteq [n]$ with $|S| \leq t - 1$:

*The joint distribution of $(s_i)_{i \in S}$ is **uniform***

$\Rightarrow$ Valid shares look exactly like **random mismatch** values.

Prior fPAKE constructions [EC18:DHPRY], [AC23:BFHHO]:

- Secret sharing with t_1 -perfect privacy and t_2 -robustness
- **Gap:** $t_1 < t_2 - 1$
- **Gap $\rightarrow$ Significantly weaker/relaxed security guarantee**

Why the Gap Matters for Password Security

Prior efficient fPAKE [EC18:DHPY], [AC23:BFHHO] realizes a relaxed ideal functionality.

For a password pw and an adversarial guess $\widehat{\text{pw}}$, let

$$M = \{i \in [n] : \text{pw}_i = \widehat{\text{pw}}_i\}, \quad m = |M|.$$

Leakage inside the gap: $t_1 < m < t_2$

The guess is outside the guaranteed recovery region, but the relaxed functionality permits revealing M : **the positions of matching characters**.

Why the Gap Matters for Password Security

Prior efficient fPAKE [EC18:DHPRY], [AC23:BFHHO] realizes a relaxed ideal functionality.

For a password pw and an adversarial guess $\widehat{\text{pw}}$, let

$$M = \{i \in [n] : \text{pw}_i = \widehat{\text{pw}}_i\}, \quad m = |M|.$$

Leakage inside the gap: $t_1 < m < t_2$

The guess is outside the guaranteed recovery region, but the relaxed functionality permits revealing M : **the positions of matching characters**.

- Passwords typically have **low entropy**. Even one revealed character can substantially reduce offline guessing cost of $\mathcal{A}$.
- [AC23:BFHHO] reduces the gap; our goal is to eliminate it.

Why the Gap Matters for Password Security

Prior efficient fPAKE [EC18:DHPRY], [AC23:BFHHO] realizes a relaxed ideal functionality.

For a password pw and an adversarial guess $\widehat{\text{pw}}$, let

$$M = \{i \in [n] : \text{pw}_i = \widehat{\text{pw}}_i\}, \quad m = |M|.$$

Leakage inside the gap: $t_1 < m < t_2$

The guess is outside the guaranteed recovery region, but the relaxed functionality permits revealing M : **the positions of matching characters**.

- Passwords typically have **low entropy**. Even one revealed character can substantially reduce offline guessing cost of $\mathcal{A}$.
- [AC23:BFHHO] reduces the gap; our goal is to eliminate it.

Full realization of the fPAKE ideal functionality requires: $t_1 = t_2 - 1$
Impossible if we require robustness against malicious share corruption

Why the Gap Matters for Password Security

Prior efficient fPAKE [EC18:DHPY], [AC23:BFHHO] realizes a relaxed ideal functionality.

For a password pw and an adversarial guess $\widehat{\text{pw}}$, let

$$M = \{i \in [n] : \text{pw}_i = \widehat{\text{pw}}_i\}, \quad m = |M|.$$

Leakage inside the gap: $t_1 < m < t_2$

The guess is outside the guaranteed recovery region, but the relaxed functionality permits revealing M : **the positions of matching characters**.

- Passwords typically have **low entropy**. Even one revealed character can substantially reduce offline guessing cost of $\mathcal{A}$.
- [AC23:BFHHO] reduces the gap; our goal is to eliminate it.

Full realization of the fPAKE ideal functionality requires: $t_1 = t_2 - 1$

Impossible if we require robustness against malicious share corruption



Why Arbitrary Corruptions Prevent Closing the Gap

Impossibility of Strong Robustness: For $1 \leq t < n$, a (t, n) secret sharing scheme with $(t-1)$ -perfect privacy cannot always recover the secret after up to $n - t$ arbitrary share corruptions.

Why Arbitrary Corruptions Prevent Closing the Gap

Impossibility of Strong Robustness: For $1 \leq t < n$, a (t, n) secret sharing scheme with $(t-1)$ -perfect privacy cannot always recover the secret after up to $n - t$ arbitrary share corruptions.

Proof idea: one received vector, two possible secrets.

1. Perfect privacy allows that two distinct secrets $K \neq K'$ have valid share vectors that agree on the first $t - 1$ positions (Intuition: $t - 1$ shares looks **uniform** random):

$$\mathbf{x} = (a_1, \dots, a_{t-1}, x_t, x_{t+1}, \dots, x_n).$$

$$\mathbf{y} = (a_1, \dots, a_{t-1}, y_t, y_{t+1}, \dots, y_n).$$

Why Arbitrary Corruptions Prevent Closing the Gap

Impossibility of Strong Robustness: For $1 \leq t < n$, a (t, n) secret sharing scheme with $(t-1)$ -perfect privacy cannot always recover the secret after up to $n - t$ arbitrary share corruptions.

Proof idea: one received vector, two possible secrets.

1. Perfect privacy allows that two distinct secrets $K \neq K'$ have valid share vectors that agree on the first $t - 1$ positions (Intuition: $t - 1$ shares look **uniform** random):

$$\begin{aligned}\mathbf{x} &= (a_1, \dots, a_{t-1}, x_t, x_{t+1}, \dots, x_n). \\ \mathbf{y} &= (a_1, \dots, a_{t-1}, y_t, y_{t+1}, \dots, y_n).\end{aligned}$$

2. Consider the mixed vector $\mathbf{z} = (\underbrace{a_1, \dots, a_{t-1}, x_t}_{t \text{ shares matching } \mathbf{x}}, y_{t+1}, \dots, y_n)$.

It also **matches $\mathbf{y}$ at all $n - 1 \geq t$ positions other than position t .**

Why Arbitrary Corruptions Prevent Closing the Gap

Impossibility of Strong Robustness: For $1 \leq t < n$, a (t, n) secret sharing scheme with $(t-1)$ -perfect privacy cannot always recover the secret after up to $n - t$ arbitrary share corruptions.

Proof idea: one received vector, two possible secrets.

1. Perfect privacy allows that two distinct secrets $K \neq K'$ have valid share vectors that agree on the first $t - 1$ positions (Intuition: $t - 1$ shares looks **uniform** random):

$$\begin{aligned}\mathbf{x} &= (a_1, \dots, a_{t-1}, x_t, x_{t+1}, \dots, x_n). \\ \mathbf{y} &= (a_1, \dots, a_{t-1}, y_t, y_{t+1}, \dots, y_n).\end{aligned}$$

2. Consider the mixed vector $\mathbf{z} = (\underbrace{a_1, \dots, a_{t-1}, x_t}_{t \text{ shares matching } \mathbf{x}}, y_{t+1}, \dots, y_n)$.

It also **matches $\mathbf{y}$ at all $n - 1 \geq t$ positions** other than position t .

3. Strong robustness therefore requires

$$\text{SecretRecover}(\mathbf{z}) = K \quad \text{and} \quad \text{SecretRecover}(\mathbf{z}) = K',$$

Contradicting $K \neq K'!$

Our Main Contribution: Random Robust Secret Sharing

Key Insight: Share corruptions in fPAKE are **Random** not **Adversarial**

- ▶ If $\text{pwd}[i] \neq \text{pwd}'[i] \Rightarrow s'_i = s_i \oplus k_i \oplus k'_i$
- ▶ By iPAKE security $k_i \neq k'_i \Rightarrow s'_i$ is uniform

Our Main Contribution: Random Robust Secret Sharing

Key Insight: Share corruptions in fPAKE are **Random** not **Adversarial**

- ▶ If $\text{pwd}[i] \neq \text{pwd}'[i] \Rightarrow s'_i = s_i \oplus k_i \oplus k'_i$
- ▶ By iPAKE security $k_i \neq k'_i \Rightarrow s'_i$ is uniform

We introduce and construct RRSS.

- ✓ $t_1 = (t - 1)$ -perfect privacy
- ✓ $t_2 = t$ -Robustness against **Random** Corruption
- ✓ negligible failure probability

Our Main Contribution: Random Robust Secret Sharing

Key Insight: Share corruptions in fPAKE are **Random** not **Adversarial**

- ▶ If $\text{pwd}[i] \neq \text{pwd}'[i] \Rightarrow s'_i = s_i \oplus k_i \oplus k'_i$
- ▶ By iPAKE security $k_i \neq k'_i \Rightarrow s'_i$ is uniform

We introduce and construct RRSS.

- ✓ $t_1 = (t - 1)$ -perfect privacy
- ✓ $t_2 = t$ -Robustness against **Random** Corruption
- ✓ negligible failure probability

The privacy-robustness gap is closed: $t_1 = t_2 - 1$ under the **Uniformly Random** corruption model.

RRSS Syntax and Share Generation

$\Pi_{\text{RRSS}} = (\text{Setup}, \text{ShareGen}, \text{Identify}, \text{SecretRecover})$

- $\text{pp} \leftarrow \text{Setup}(1^\lambda, n, t).$
- $\vec{c} \leftarrow \text{ShareGen}(s, \text{pp}).$
- $S \leftarrow \text{Identify}(\vec{c}', \text{pp}).$
- $s \leftarrow \text{SecretRecover}(S, \vec{c}'_S, \text{pp}).$

RRSS Syntax and Share Generation

$\Pi_{\text{RRSS}} = (\text{Setup}, \text{ShareGen}, \text{Identify}, \text{SecretRecover})$

- $\text{pp} \leftarrow \text{Setup}(1^\lambda, n, t)$.
- $\vec{c} \leftarrow \text{ShareGen}(s, \text{pp})$.
- $S \leftarrow \text{Identify}(\vec{c}', \text{pp})$.
- $s \leftarrow \text{SecretRecover}(S, \vec{c}'_S, \text{pp})$.

Share generation:

Setup publishes $3n$ nonzero field elements
 $\vec{a} = (a_1, \dots, a_{3n})$.

Share the main secret with (n, t) -Shamir:

$$([s]_1, \dots, [s]_n) \leftarrow \text{Shamir}_{n,t}(s).$$

For every $j \in [3n]$, share the public value a_j
with $(2n, 2t)$ -Shamir:

$$([a_j]_1, \dots, [a_j]_{2n}) \leftarrow \text{Shamir}_{2n,2t}(a_j).$$

RRSS Syntax and Share Generation

$\Pi_{\text{RRSS}} = (\text{Setup}, \text{ShareGen}, \text{Identify}, \text{SecretRecover})$

- $\text{pp} \leftarrow \text{Setup}(1^\lambda, n, t).$
- $\vec{c} \leftarrow \text{ShareGen}(s, \text{pp}).$
- $S \leftarrow \text{Identify}(\vec{c}', \text{pp}).$
- $s \leftarrow \text{SecretRecover}(S, \vec{c}'_S, \text{pp}).$

Share generation:

Setup publishes $3n$ nonzero field elements $\vec{a} = (a_1, \dots, a_{3n}).$

Share the main secret with (n, t) -Shamir:

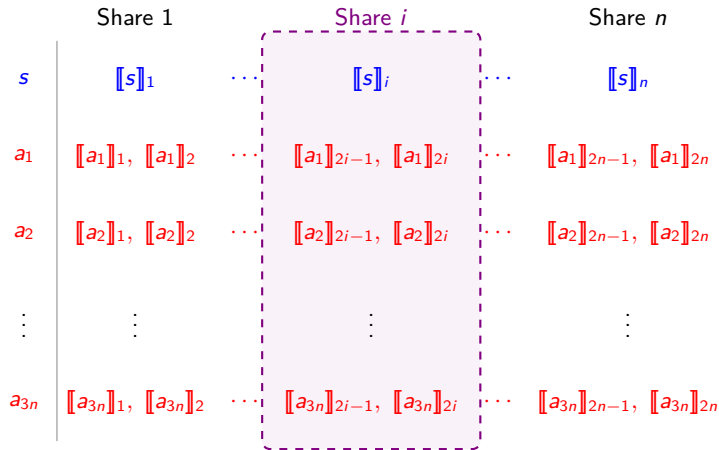
$$([s]_1, \dots, [s]_n) \leftarrow \text{Shamir}_{n,t}(s).$$

For every $j \in [3n]$, share the public value a_j with $(2n, 2t)$ -Shamir:

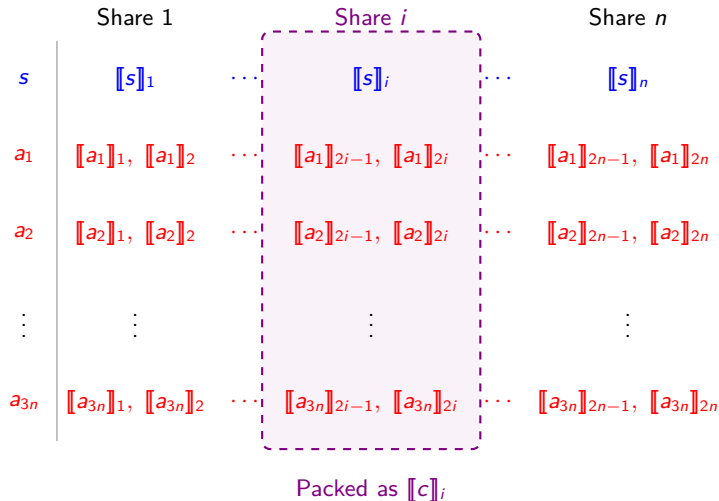
$$([a_j]_1, \dots, [a_j]_{2n}) \leftarrow \text{Shamir}_{2n,2t}(a_j).$$

$$i\text{-th share format: } [c]_i = \left(\underbrace{[s]_i}_{\text{main secret share}}, \underbrace{([a_j]_{2i-1}, [a_j]_{2i})_{j \in [3n]}}_{\text{two auxiliary shares for each public check } a_j} \right).$$

Packing the Shamir Shares into RSS Shares



Packing the Shamir Shares into RRSS Shares



Each RRSS share contains **one main share** and **$6n$ auxiliary shares**.

RRSS Identification: Solving for Valid Shares

Input: possibly corrupted shares.

- $\vec{c}' = ([c']_1, \dots, [c']_n)$
- Auxiliary part: $([a'_j]_{2i-1}, [a'_j]_{2i})_{j \in [3n]}$
- Up to $n - t$ shares may have been replaced by uniformly random values.
- A valid share $\Rightarrow$ 2 valid shares of a_j

RRSS Identification: Solving for Valid Shares

Input: possibly corrupted shares.

- $\vec{c}' = ([c']_1, \dots, [c']_n)$
- Auxiliary part: $([a'_j]_{2i-1}, [a'_j]_{2i})_{j \in [3n]}$
- Up to $n - t$ shares may have been replaced by uniformly random values.
- A valid share $\Rightarrow$ 2 valid shares of a_j

Building System of Linear Equations:

Form a $3n \times 2n$ matrix A from the auxiliary shares:

$$A_{j,2i-1} = [a'_j]_{2i-1}, A_{j,2i} = [a'_j]_{2i}.$$

Solve the over-constrained system

$$A\vec{x} = \vec{a}, \quad \vec{x} = (x_1, \dots, x_{2n}).$$

RRSS Identification: Solving for Valid Shares

Input: possibly corrupted shares.

- $\vec{c}' = ([c']_1, \dots, [c']_n)$
- Auxiliary part: $([a'_j]_{2i-1}, [a'_j]_{2i})_{j \in [3n]}$
- Up to $n - t$ shares may have been replaced by uniformly random values.
- A valid share $\Rightarrow$ 2 valid shares of a_j

Building System of Linear Equations:

Form a $3n \times 2n$ matrix A from the auxiliary shares:

$$A_{j,2i-1} = [a'_j]_{2i-1}, A_{j,2i} = [a'_j]_{2i}.$$

Solve the over-constrained system

$$A\vec{x} = \vec{a}, \quad \vec{x} = (x_1, \dots, x_{2n}).$$

Identify valid indices:

* **Intuition:** Random corrupted share:

- Unlikely being consistent with all $3n$ constraints.
- $\Rightarrow$ its two coefficients are zero with high probability.

* Define $S = \{i \in [n] : x_{2i-1} \neq 0 \vee x_{2i} \neq 0\}$.

* Output $\perp$, if $|S| < t$.

* If $|S| \geq t$: x_i s are **Lagrange** Coefficients.

✓ As One Possible Solution

RRSS Identification: Solving for Valid Shares

Input: possibly corrupted shares.

- $\vec{c}' = ([c']_1, \dots, [c']_n)$
- Auxiliary part: $([a'_j]_{2i-1}, [a'_j]_{2i})_{j \in [3n]}$
- Up to $n - t$ shares may have been replaced by uniformly random values.
- A valid share $\Rightarrow$ 2 valid shares of a_j

Building System of Linear Equations:

Form a $3n \times 2n$ matrix A from the auxiliary shares:

$$A_{j,2i-1} = [a'_j]_{2i-1}, A_{j,2i} = [a'_j]_{2i}.$$

Solve the over-constrained system

$$A\vec{x} = \vec{a}, \quad \vec{x} = (x_1, \dots, x_{2n}).$$

Identify valid indices:

* **Intuition:** Random corrupted share:

- Unlikely being consistent with all $3n$ constraints.
- $\Rightarrow$ its two coefficients are zero with high probability.

* Define $S = \{i \in [n] : x_{2i-1} \neq 0 \vee x_{2i} \neq 0\}$.

* Output $\perp$, if $|S| < t$.

* If $|S| \geq t$: x_i s are **Lagrange** Coefficients.

✓ As One Possible Solution

Recover the secret:

1. If $|S| \geq t$, obtain $\{[s']_i\}_{i \in S}$
2. run standard Shamir reconstruction:

$$s \leftarrow \text{ShamirRecover}_{n,t}(S, \{[s']_i\}_{i \in S}).$$

Result: recover correctly with probability at least $1 - 2^{-\lambda}$.

RRSS Identification: Solving for Valid Shares

Input: possibly corrupted shares.

- $\vec{c}' = ([c']_1, \dots, [c']_n)$
- Auxiliary part: $([a'_j]_{2i-1}, [a'_j]_{2i})_{j \in [3n]}$
- Up to $n - t$ shares may have been replaced by uniformly random values.
- A valid share $\Rightarrow$ 2 valid shares of a_j

Building System of Linear Equations:

Form a $3n \times 2n$ matrix A from the auxiliary shares:

$$A_{j,2i-1} = [a'_j]_{2i-1}, A_{j,2i} = [a'_j]_{2i}.$$

Solve the over-constrained system

$$A\vec{x} = \vec{a}, \quad \vec{x} = (x_1, \dots, x_{2n}).$$



Identify valid indices:

* **Intuition:** Random corrupted share:

- Unlikely being consistent with all $3n$ constraints.
- $\Rightarrow$ its two coefficients are zero with high probability.

* Define $S = \{i \in [n] : x_{2i-1} \neq 0 \vee x_{2i} \neq 0\}$.

* Output $\perp$, if $|S| < t$.

* If $|S| \geq t$: x_i s are **Lagrange** Coefficients.

✓ As One Possible Solution

Recover the secret:

1. If $|S| \geq t$, obtain $\{[s']_i\}_{i \in S}$
2. run standard Shamir reconstruction:

$$s \leftarrow \text{ShamirRecover}_{n,t}(S, \{[s']_i\}_{i \in S}).$$

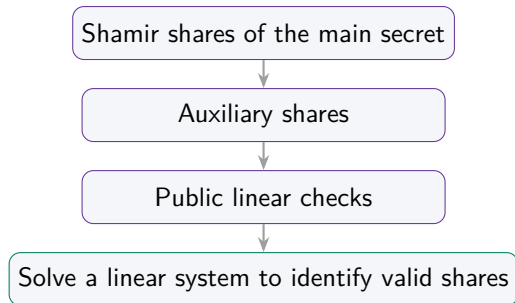
Result: recover correctly with probability at least $1 - 2^{-\lambda}$.

Proof Intuition: Random Robustness

- **Bad Event:** Suppose a corrupted share contributes a nonzero variable.
- Since the share is uniformly random, each linear constraint is satisfied only with small probability: $\equiv \frac{1}{|\mathbb{F}|} = 2^{-\lambda}$
 - Let $x_{2i} \neq 0$ for a random corruption at i -th share, all $\llbracket a_j \rrbracket_{2i x_{2i}}$ are random.
- $\Rightarrow \forall 1 \leq j \leq 3n : \Pr \left[a_j = \llbracket a_j \rrbracket_{2i x_{2i}} + \sum_{\forall k \in [2n] - \{i\}} A_{j,k} x_k \right] = \frac{1}{|\mathbb{F}|} = 2^{-\lambda}$
- There are $3n$ (independent) constraints $\Rightarrow$ at most $(2^{-\lambda})^{3n} = 2^{-3n\lambda}$
- Union bound over all $2^{2n\lambda}$ possible assignments ($2n$ variables and each has 2^λ possibilities)
 $\Rightarrow (2^\lambda)^{2n} \times 2^{-3n\lambda} = 2^{-n\lambda}$ (negligible failure prob.)

Optimization (Two field sizes): The main secret is shared over the large field $\mathbb{F}_{2^\lambda}$, e.g., $\lambda = 128$. Auxiliary shares use the smaller field $\mathbb{F}_{2^{\lambda_1}}$. $\lambda_1 \geq \max \left\{ \lceil \log_2(2n) \rceil, \left\lceil \frac{\lambda}{n} \right\rceil \right\} \implies 2^{-n\lambda_1} \leq 2^{-\lambda}$.

RRSS Efficient Construction with C++ Implementation



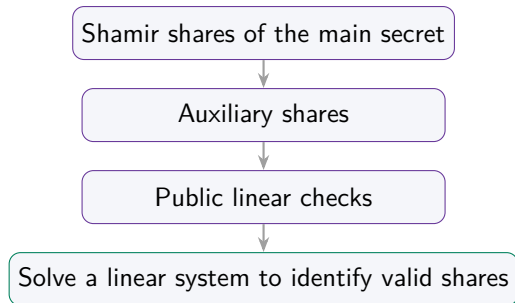
Guarantees:

- ✓ Any $t - 1$ shares are jointly uniform.
- ✓ Recover with up to $n - t$ random corruptions.
- ✓ Failure probability at most $2^{-\lambda}$.

Identification time: $O(n^\omega)$

(ω : matrix-multiplication exponent)

RRSS Efficient Construction with C++ Implementation



Guarantees:

- ✓ Any $t - 1$ shares are jointly uniform.
- ✓ Recover with up to $n - t$ random corruptions.
- ✓ Failure probability at most $2^{-\lambda}$.

Identification time: $O(n^\omega)$
(ω : matrix-multiplication exponent)

Github link to the C++ code



Application: The First Practical fPAKE with Full UC Security

Prior Practical Work

[EC18:DHPRY], [AC23:BFHHO]

Efficient constructions

Robustness-privacy gap

Does not fully realize the ideal functionality

Prior Fully Secure Work

[EC18:DHPRY], [EC26:RRX]

Strong theoretical security

No practical implementation

Our RRSS-Based fPAKE

Full ideal-functionality security

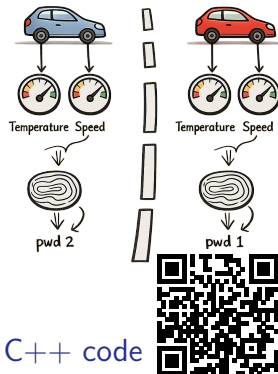
Efficient

Implemented in C++

Full UC security and practical efficiency

A prior relaxed fPAKE construction was used in **FastZip** for vehicle-to-vehicle authentication [MobiSys21:FHAL+].

See the paper for the complete comparison with our fPAKE using RRSS.



Github link to the C++ code

Conditional Encryption: Motivation

Personalized typo correction.

- TypTop [CCS:Chatterjee et. al,17] encrypts a user's password typos using PKE.
- Processing mistyped PWDs by decrypting the pwd ctxts (using sk extracted from Original PWD.)

Problem: Using other accounts password (far from the original one)



Conditional Encryption: Motivation

Personalized typo correction.

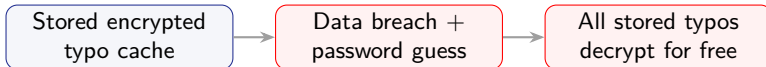
- TypTop [CCS:Chatterjee et. al,17] encrypts a user's password typos using PKE.
- Processing mistyped PWDs by decrypting the pwd ctxts (using sk extracted from Original PWD.)

Problem: Using other accounts password (far from the original one)



The breach problem.

- Traditional PKE: All the ctxt are decryptable to a typo.
- Successful Breach: Recovering the sk, decrypting typos.
 - ⇒ Obtaining PWDs of more sensitive accounts (e.g., Bank account PWDs).



Conditional Encryption: Reveal a Typo Only When It Is Safe

Core idea. Encrypt the sensitive payload (e.g., typo) *conditionally*: it should decrypt only when the guessed password is *sufficiently* close to the original password.

Conditional Encryption: Reveal a Typo Only When It Is Safe

Core idea. Encrypt the sensitive payload (e.g., typo) *conditionally*: it should decrypt only when the guessed password is *sufficiently* close to the original password.

Close password Typo.

If $\text{Ham}(\text{pwd}, \text{pwd}') \leq \ell$, the conditional ciphertext decrypts to the intended typo.

Conditional Encryption: Reveal a Typo Only When It Is Safe

Core idea. Encrypt the sensitive payload (e.g., typo) *conditionally*: it should decrypt only when the guessed password is *sufficiently* close to the original password.

Close password Typo.

If $\text{Ham}(\text{pwd}, \text{pwd}') \leq \ell$, the conditional ciphertext decrypts to the intended typo.

Far password Typo.

If $\text{Ham}(\text{pwd}, \text{pwd}') > \ell$, the conditional Ctxt decrypts to *random garbage*.

Conditional Encryption: Reveal a Typo Only When It Is Safe

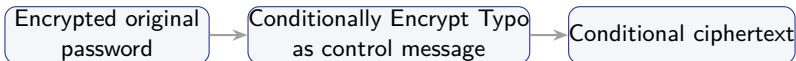
Core idea. Encrypt the sensitive payload (e.g., typo) *conditionally*: it should decrypt only when the guessed password is *sufficiently* close to the original password.

Close password Typo.

If $\text{Ham}(\text{pwd}, \text{pwd}') \leq \ell$, the conditional ciphertext decrypts to the intended typo.

Far password Typo.

If $\text{Ham}(\text{pwd}, \text{pwd}') > \ell$, the conditional Ctxt decrypts to *random garbage*.



Conditional Encryption: Syntax

$$\Pi_{\text{CE}} = (\text{Setup}, \text{KG}, \text{Enc}, \text{CEnc}, \text{Dec}).$$

Conditional Encryption: Syntax

$$\Pi_{\text{CE}} = (\text{Setup}, \text{KG}, \text{Enc}, \text{CEnc}, \text{Dec}).$$

Inputs.

$$c_1 \leftarrow \text{Enc}_{pk}(m_1)$$

$$c' \leftarrow \text{CEnc}_{pk}(c_1, m_2, m_3).$$

- c_1 encrypting the unknown message m_1 (e.g., PWD)
- m_2 is the control message (e.g., typo).
- m_3 is the payload (e.g., typo or AE secret Key).

Conditional Encryption: Syntax

$$\Pi_{\text{CE}} = (\text{Setup}, \text{KG}, \text{Enc}, \text{CEnc}, \text{Dec}).$$

Inputs.

$$\begin{aligned} c_1 &\leftarrow \text{Enc}_{pk}(m_1) \\ c' &\leftarrow \text{CEnc}_{pk}(c_1, m_2, m_3). \end{aligned}$$

- c_1 encrypting the unknown message m_1 (e.g., PWD)
- m_2 is the control message (e.g., typo).
- m_3 is the payload (e.g., typo or AE secret Key).

Correctness and security.

$$P(m_1, m_2) = 1 \implies \text{Dec}_{sk}(c') = m_3.$$

$$P(m_1, m_2) = 0 \implies c' \approx \text{Sim}(1^\lambda),$$

Even for an adversary who **knows** sk .

Conditional Encryption: Syntax

$$\Pi_{\text{CE}} = (\text{Setup}, \text{KG}, \text{Enc}, \text{CEnc}, \text{Dec}).$$

Inputs.

$$\begin{aligned} c_1 &\leftarrow \text{Enc}_{pk}(m_1) \\ c' &\leftarrow \text{CEnc}_{pk}(c_1, m_2, m_3). \end{aligned}$$

- c_1 encrypting the unknown message m_1 (e.g., PWD)
- m_2 is the control message (e.g., typo).
- m_3 is the payload (e.g., typo or AE secret Key).

Correctness and security.

$$P(m_1, m_2) = 1 \implies \text{Dec}_{sk}(c') = m_3.$$

$$P(m_1, m_2) = 0 \implies c' \approx \text{Sim}(1^\lambda),$$

Even for an adversary who **knows** sk .

Our Target Predicate.

$$P_{\text{Ham} \leq \ell}(m_1, m_2) = 1 \iff \text{Ham}(m_1, m_2) \leq \ell.$$

CCS'24 [AmeBlo24] Construction: Inefficient Conditional Decryption

Equality test from Paillier.

Paillier: $c_1[i] = (1 + N)^{m_1[i]} r_i^N \bmod N^2$

Idea: For each character i , conditional encryption of i -th share of a key $K \in_R \{0, 1\}^\lambda$:

$$m_1[i] = m_2[i] \implies \text{correct share } \llbracket K \rrbracket_i.$$

$$m_1[i] \neq m_2[i] \implies \text{uniform random share.}$$

$$C_i = c_1[i]^{R_i} (1 + N)^{-R_i(m_2[i] + \llbracket K \rrbracket_i)} r_i'^N \bmod N^2$$

$$C_i = (1 + N)^{R_i(m_1[i] - m_2[i] + \llbracket K \rrbracket_i)} (r_i^R r_i')^N \bmod N^2.$$

CCS'24 [AmeBlo24] Construction: Inefficient Conditional Decryption

Equality test from Paillier.

Paillier: $c_1[i] = (1 + N)^{m_1[i]} r_i^N \bmod N^2$

Idea: For each character i , conditional encryption of i -th share of a key $K \in_R \{0, 1\}^\lambda$:

$$m_1[i] = m_2[i] \implies \text{correct share } \llbracket K \rrbracket_i.$$

$$m_1[i] \neq m_2[i] \implies \text{uniform random share.}$$

$$C_i = c_1[i]^{R_i} (1 + N)^{-R_i(m_2[i] + \llbracket K \rrbracket_i)} r_i'^N \bmod N^2$$

$$C_i = (1 + N)^{R_i(m_1[i] - m_2[i] + \llbracket K \rrbracket_i)} (r_i^R r_i')^N \bmod N^2.$$

Extend equality to Hamming distance.

Share a fresh symmetric key K with (n, t) -Shamir, where $t = n - \ell$.

Run one equality-based conditional encryption for each character/share.

(Over Simplified Construction!)

CCS'24 [AmeBlo24] Construction: Inefficient Conditional Decryption

Equality test from Paillier.

Paillier: $c_1[i] = (1 + N)^{m_1[i]} r_i^N \bmod N^2$

Idea: For each character i , conditional encryption of i -th share of a key $K \in_R \{0, 1\}^\lambda$:

$$m_1[i] = m_2[i] \implies \text{correct share } \llbracket K \rrbracket_i.$$

$$m_1[i] \neq m_2[i] \implies \text{uniform random share.}$$

$$C_i = c_1[i]^{R_i} (1 + N)^{-R_i(m_2[i]) + \llbracket K \rrbracket_i} r_i'^N \bmod N^2$$

$$C_i = (1 + N)^{R_i(m_1[i] - m_2[i]) + \llbracket K \rrbracket_i} (r_i^R r_i')^N \bmod N^2.$$

Extend equality to Hamming distance.

Share a fresh symmetric key K with (n, t) -Shamir, where $t = n - \ell$.

Run one equality-based conditional encryption for each character/share.

(Over Simplified Construction!)

Why this hides matching positions.

Shamir $(t - 1)$ -perfect privacy is essential: it hides which characters match.

CCS'24 [AmeBlo24] Construction: Inefficient Conditional Decryption

Equality test from Paillier.

Paillier: $c_1[i] = (1 + N)^{m_1[i]} r_i^N \bmod N^2$

Idea: For each character i , conditional encryption of i -th share of a key $K \in_R \{0, 1\}^\lambda$:

$$m_1[i] = m_2[i] \implies \text{correct share } \llbracket K \rrbracket_i.$$

$$m_1[i] \neq m_2[i] \implies \text{uniform random share.}$$

$$C_i = c_1[i]^{R_i} (1 + N)^{-R_i(m_2[i]) + \llbracket K \rrbracket_i} r_i'^N \bmod N^2$$

$$C_i = (1 + N)^{R_i(m_1[i] - m_2[i]) + \llbracket K \rrbracket_i} (r_i^R r_i')^N \bmod N^2.$$

Why this hides matching positions.

Shamir $(t - 1)$ -perfect privacy is essential: it hides which characters match.

Extend equality to Hamming distance.

Share a fresh symmetric key K with (n, t) -Shamir, where $t = n - \ell$.

Run one equality-based conditional encryption for each character/share.

(Over Simplified Construction!)

The bottleneck.

Shamir sharing is not robust:

- $\Rightarrow$ decryption cannot identify which shares are correct.
- $\Rightarrow$ It must try all candidate subsets: $O\left(\binom{n}{\ell}\right)$
- $\Rightarrow$ Efficient only for very small ℓ .

Our Construction: Replace Shamir with RSS

What happens at each character.

$m[i] = m_1[i] \implies$ correct RSS share.

$m[i] \neq m_1[i] \implies$ independent uniform RSS share.

Thus, mismatches create exactly the **random corruptions** RSS is designed to handle.

Our Construction: Replace Shamir with RRSS

What happens at each character.

$m[i] = m_1[i] \implies$ correct RRSS share.

$m[i] \neq m_1[i] \implies$ independent uniform RRSS share.

Thus, mismatches create exactly the **random corruptions** RRSS is designed to handle.

Efficient decryption.

Obtain valid shares using:

$$\Pi_{\text{RRSS}}.\text{Identify} \implies O(n^\omega).$$

RRSS Recovers K to get payload.

Our Construction: Replace Shamir with RRSS

What happens at each character.

$m[i] = m_1[i] \implies$ correct RRSS share.

$m[i] \neq m_1[i] \implies$ independent uniform RRSS share.

Thus, mismatches create exactly the **random corruptions** RRSS is designed to handle.

Efficient decryption.

Obtain valid shares using:

$$\Pi_{\text{RRSS}}.\text{Identify} \implies O(n^\omega).$$

RRSS Recovers K to get payload.

Result.

$$\binom{n}{\ell}$$

CCS'24: exhaustive search



$$O(n^\omega)$$

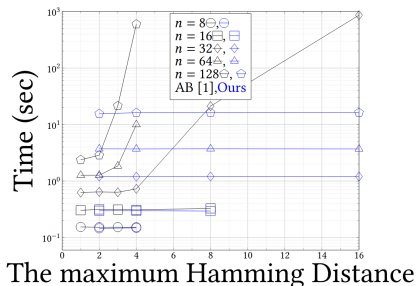
this work: linear algebra

Conditional Encryption Performance

- ✗ Previous construction [CCS24:AmeriBlocki]:
 - efficient only for **small ℓ** ;
 - about **14.4 minutes** for $n = 32, \ell = 16$.
- ✓ Our construction using RRSS:
 - decryption time is essentially **independent of ℓ** ;
 - about **1.2 seconds** for $n = 32, \ell = 16$.
- Ciphertexts are larger but remain manageable.

Conditional Encryption Performance

- ✗ Previous construction [CCS24:AmeriBlocki]:
 - efficient only for **small** ℓ ;
 - about **14.4 minutes** for $n = 32, \ell = 16$.
- ✓ Our construction using RSS:
 - decryption time is essentially **independent of** ℓ ;
 - about **1.2 seconds** for $n = 32, \ell = 16$.
- Ciphertexts are larger but remain manageable.



Implementation

- ✓ RRSS implemented as a modular C++ library.
- ✓ Valid-share identification uses Gauss–Jordan elimination over $GF(256)$.
- ✓ Conditional encryption uses Paillier, GMP, CryptoPP, and AES-GCM.
- ✓ fPAKE implementation modifies the FastZIP codebase by replacing RSS with RRSS.
 - Designing a translator to use/compile our C++ implementation as python library as well
- ✓ All implementations are available as an **open source C++** library
 - **Github Link:** <https://github.com/mhassanameri/RRSS>

Try It, Use It, and Come Talk to Me :-)

- **Paper and open-source C++ implementation**
- $\langle \text{RRSS} \parallel \text{fPAKE} \parallel \text{Conditional Encryption} \rangle$
 - o ePrint 2026/653
 - o github.com/mhassanameri/RRSS
- ✓ **fPAKE**: full UC security and practical efficiency.
- ✓ **Second Application**: Efficient **CondEnc** for arbitrary-Hamming-distance.

Thank You!
Questions ?

📌 **I am on the 26-27 job market**

Seeking postdoctoral in:

Applied/Theoretical foundation of
Cryptography

mameriek@purdue.edu



Scan for paper

Github link to the C++ code

