Cryptographic proofs for remote storage: models and construction

Julien Lavauzelle[1], Françoise Levy-dit-Vehel[1,2]

[1] LIX & INRIA Saclay, Université Paris-Saclay
[2] ENSTA ParisTech

Journées codage & cryptographie 2018, Aussois, France
12/10/2018

1. Proofs-of-* for secure remote storage

2. A generic construction of proof-of-retrievability
   - Model and definition
   - A generic construction of PoR
   - Some instances

1. Proofs-of-* for secure remote storage

2. A generic construction of proof-of-retrievability
   - Model and definition
   - A generic construction of PoR
   - Some instances

Informal issue

Checking storage properties on remote servers, *e.g.* verifying that:

- the server actually stores the file,
- the server has fully deleted some data,
- a file is retrievable from the server,
- some space is used/available on a server.

Informal issue

Checking storage properties on remote servers, *e.g.* verifying that:

- the server actually stores the file,
- the server has fully deleted some data,
- a file is retrievable from the server,
- some space is used/available on a server.

Practical application:

- cryptocurrency based on a decentralized cloud storage network
- Storj, FileCoin, SpaceMint.

PoR and PDP

Proof of Retrievability (PoR):[1] a verifier checks extractability of a file m.

[1] introduced in *PoRs: Proofs of Retrievability for Large Files*, Juels, Kaliski CCS'07

PoR and PDP

Proof of Retrievability (PoR):[1] a verifier checks extractability of a file m.

	Verifier		Prover
Initialisation	$\kappa \leftarrow \mathsf{KeyGen}(1^\lambda)$		
	$w \leftarrow \mathsf{Init}(m, \kappa)$	$\xrightarrow{\quad w \quad}$	
	delete m, (most of) w		

[1] introduced in *PoRs: Proofs of Retrievability for Large Files*, Juels, Kaliski CCS'07

PoR and PDP

Proof of Retrievability (PoR):[1] a verifier checks extractability of a file m.

	Verifier		Prover
Initialisation	$\kappa \leftarrow \mathsf{KeyGen}(1^\lambda)$		
	$w \leftarrow \mathsf{Init}(m, \kappa)$	$\xrightarrow{w}$	
	delete m, (most of) w		
Verification	$u \leftarrow_R \mathcal{Q}$	$\xrightarrow{u}$	
		$\xleftarrow{r_u}$	$r_u \leftarrow \mathsf{Resp}(u, w)$
	$0/1 \leftarrow \mathsf{Check}(u, r_u, \kappa)$		

[1] introduced in *PoRs: Proofs of Retrievability for Large Files*, Juels, Kaliski CCS'07

PoR and PDP

Proof of Retrievability (PoR):[1] a verifier checks extractability of a file m.

	Verifier		Prover
Initialisation	$\kappa \leftarrow \mathsf{KeyGen}(1^\lambda)$		
	$w \leftarrow \mathsf{Init}(m, \kappa)$	$\xrightarrow{w}$	
	delete m, (most of) w		
Verification	$u \leftarrow_{\mathrm{R}} \mathcal{Q}$	$\xrightarrow{u}$	
		$\xleftarrow{r_u}$	$r_u \leftarrow \mathsf{Resp}(u, w)$
	$0/1 \leftarrow \mathsf{Check}(u, r_u, \kappa)$		
Extraction		$\{(u, r_u) : u \in \mathcal{Q}\}$ $\xleftarrow{\quad}$	
	$m/\perp \leftarrow \mathsf{Extract}(\{(u, r_u) : u \in \mathcal{Q}\}, \kappa)$		

[1] introduced in *PoRs: Proofs of Retrievability for Large Files*, Juels, Kaliski CCS'07

PoR and PDP

Proof of Retrievability (PoR):[1] a verifier checks extractability of a file m.

	Verifier		Prover
Initialisation	$\kappa \leftarrow \mathsf{KeyGen}(1^\lambda)$		
	$w \leftarrow \mathsf{Init}(m, \kappa)$	$\xrightarrow{\quad w \quad}$	
	delete m, (most of) w		
Verification	$u \leftarrow_R \mathcal{Q}$	$\xrightarrow{\quad u \quad}$	
		$\xleftarrow{\quad r_u \quad}$	$r_u \leftarrow \mathsf{Resp}(u, w)$
	$0/1 \leftarrow \mathsf{Check}(u, r_u, \kappa)$		
Extraction		$\{(u, r_u) : u \in \mathcal{Q}\}$ $\xleftarrow{\quad\quad}$	
	$m/\perp \leftarrow \mathsf{Extract}(\{(u, r_u) : u \in \mathcal{Q}\}, \kappa)$		

A **Proof of Data Possession (PDP)** is essentially a PoR without explicit extractor.

[1] introduced in *PoRs: Proofs of Retrievability for Large Files*, Juels, Kaliski CCS'07

Other proofs for remote storage

Proving **deletion of data**:

- **Proof of Secure Erasure (PoSE)**[2] [*One-time computable self-erasing functions*, Dziembowski, Kazan, Wichs TCC'11]

[2] originally introduced by Perito and Tsudik, *Secure code update for embedded devices via proofs of secure erasure* (ESORICS 2010)

Other proofs for remote storage

Proving **deletion of data**:

- **Proof of Secure Erasure (PoSE)**[2] [*One-time computable self-erasing functions*, Dziembowski, Kazan, Wichs TCC'11]

Proving that some **space/time** is invested:

- **Proof of Space (PoS)** [*Proofs of Space*, Dziembowski, Faust, Kolmogorov, Pietrzak, CRYPTO'15]

[2] originally introduced by Perito and Tsudik, *Secure code update for embedded devices via proofs of secure erasure* (ESORICS 2010)

Other proofs for remote storage

Proving **deletion of data**:

- **Proof of Secure Erasure (PoSE)**[2] [*One-time computable self-erasing functions*, Dziembowski, Kazan, Wichs TCC'11]

Proving that some **space/time** is invested:

- **Proof of Space (PoS)** [*Proofs of Space*, Dziembowski, Faust, Kolmogorov, Pietrzak, CRYPTO'15]

Proving **robust** storage:

- **Proof of replication (PoReP)**, *e.g.* in FileCoin
- With public audit: **public incompressible encodings (PIE)** [Cecchetti, Miers, Juels, IACR eprint'18]

[2] originally introduced by Perito and Tsudik, *Secure code update for embedded devices via proofs of secure erasure* (ESORICS 2010)

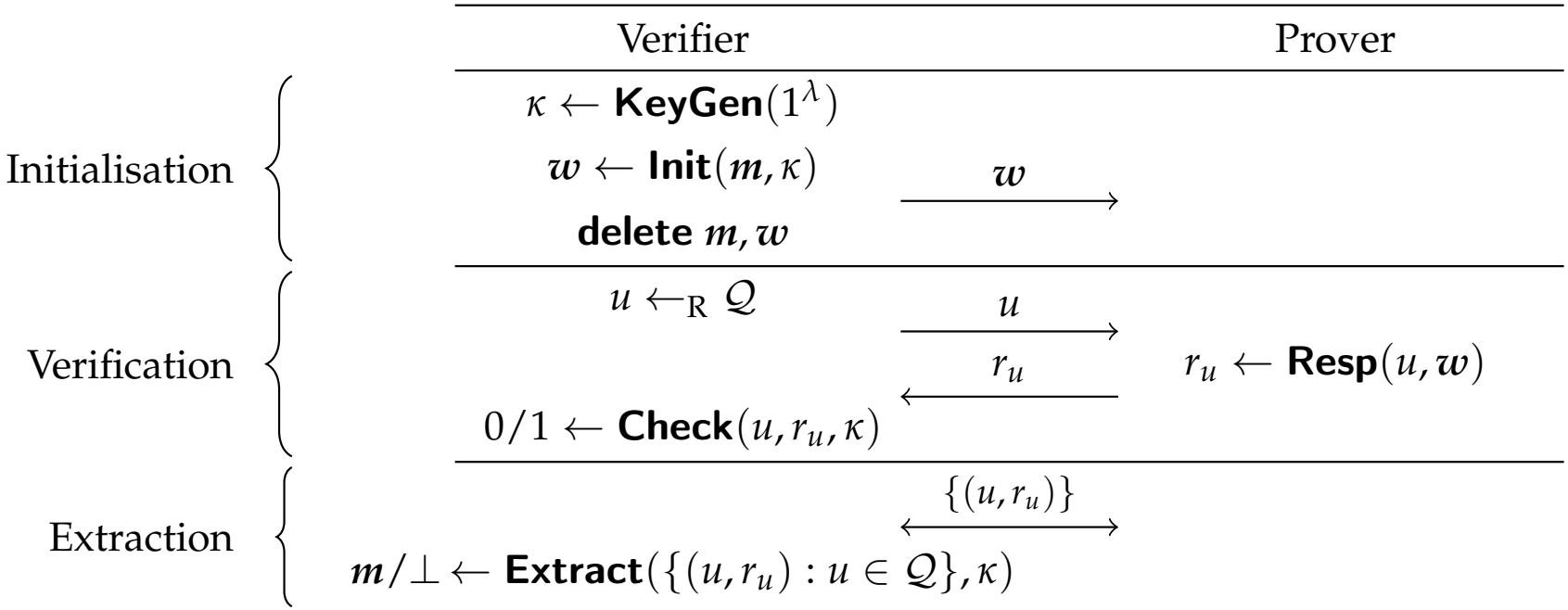
1. Proofs-of-* for secure remote storage

2. A generic construction of proof-of-retrievability
   - Model and definition
   - A generic construction of PoR
   - Some instances

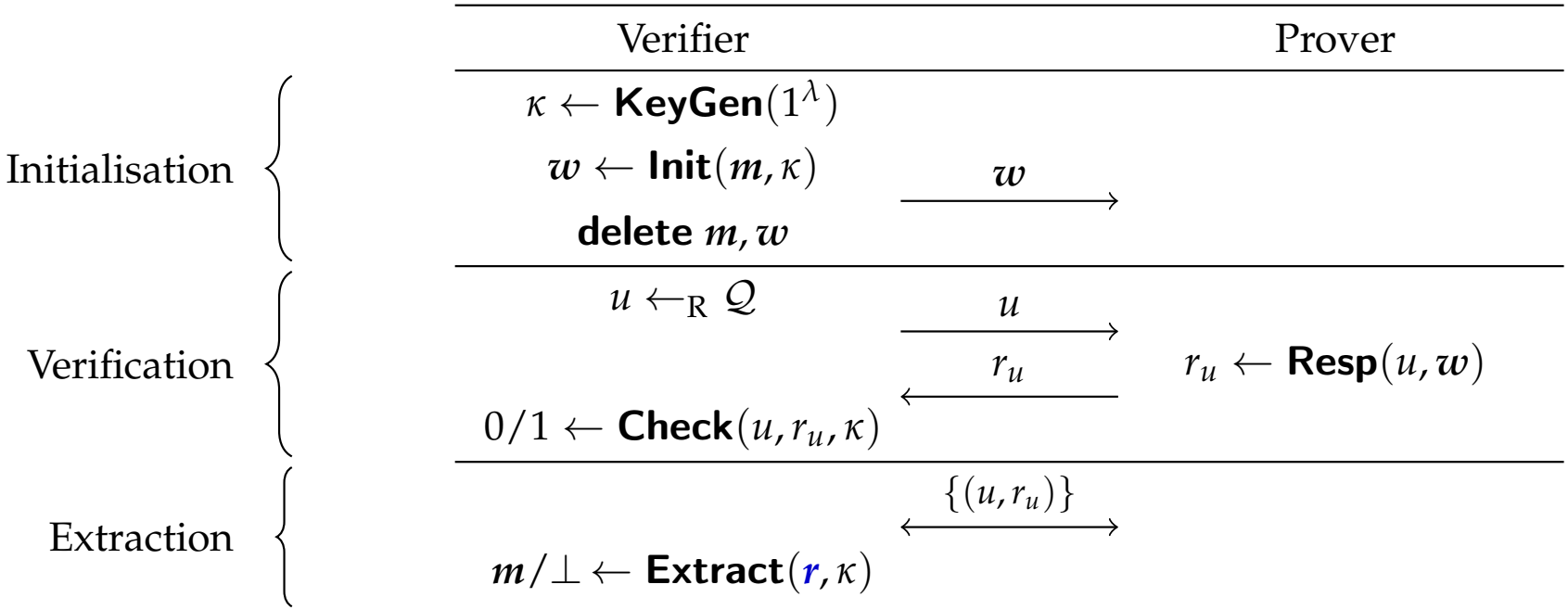
1. Proofs-of-* for secure remote storage

2. A generic construction of proof-of-retrievability
   - Model and definition
   - A generic construction of PoR
   - Some instances

Formal definitions for PoRs



Formal definitions for PoRs



Hypothesis (following [Paterson, Stinson, Upadhyay, J. Math. Crypto.'13]): response algorithm **Resp** is non-adaptive and deterministic.

$\Rightarrow$ One can consider the **response word** $\boldsymbol{r} = (r_u : u \in \mathcal{Q})$

Security model

For some response word $\boldsymbol{r}$ and secret data κ, we define the **success**:

$$\mathrm{succ}(\boldsymbol{r}, \kappa) := \Pr_{u \leftarrow_R \mathcal{Q}} \left(\mathbf{Check}(u, r_u, \kappa) = 1 \right) .$$

Security model

For some response word r and secret data κ, we define the **success**:

$$\mathrm{succ}(r, \kappa) := \Pr_{u \leftarrow_R \mathcal{Q}} \left(\mathsf{Check}(u, r_u, \kappa) = 1 \right) .$$

Soundness. A PoR is (ε, τ)-sound if for every prover r,

$$\Pr \left(\begin{array}{c} \mathsf{Extract}(r, \kappa) \neq m \\ \text{and} \\ \mathrm{succ}(r, \kappa) \geq 1 - \varepsilon \end{array} \middle| \begin{array}{l} m \leftarrow_R \mathcal{M} \\ \kappa \leftarrow_R \mathsf{KeyGen}(1^\lambda) \\ w \leftarrow \mathsf{Init}(m, \kappa) \\ r \leftarrow \mathsf{Resp}(\cdot, w) \end{array} \right) \leq \tau .$$

Security model

For some response word r and secret data κ, we define the **success**:

$$\mathrm{succ}(r, \kappa) := \Pr_{u \leftarrow_R \mathcal{Q}} \left(\mathsf{Check}(u, r_u, \kappa) = 1 \right) .$$

Soundness. A PoR is (ε, τ)-sound if for every prover r,

$$\Pr \left(\begin{array}{c} \mathsf{Extract}(r, \kappa) \neq m \\ \text{and} \\ \mathrm{succ}(r, \kappa) \geq 1 - \varepsilon \end{array} \middle| \begin{array}{l} m \leftarrow_R \mathcal{M} \\ \kappa \leftarrow_R \mathsf{KeyGen}(1^\lambda) \\ w \leftarrow \mathsf{Init}(m, \kappa) \\ r \leftarrow \mathsf{Resp}(\cdot, w) \end{array} \right) \leq \tau .$$

Goal: $\tau \ll 1$, for constant $\varepsilon > 0$.

1. Proofs-of-* for secure remote storage

2. A generic construction of proof-of-retrievability
   - Model and definition
   - A generic construction of PoR
   - Some instances

Outline

Our main goals

- sublinear communication complexity for the verification
- low additional storage
- few computation during the verification step (*e.g.* **Resp** and **Check**)
- analysable/quantifiable soundness

Outline

Our main goals

- sublinear communication complexity for the verification
- low additional storage
- few computation during the verification step (*e.g.* **Resp** and **Check**)
- analysable/quantifiable soundness

Overview of the solution

Let $\mathcal{C} \subseteq \mathbb{F}_q^n$ with "many well-distributed low-weight parity-check equations".

- Initialisation:
  1. encode the file $\boldsymbol{m}$ into a codeword $\boldsymbol{c} \in \mathcal{C}$
  2. apply symbol-wise $\mathbb{F}_q$-permutations to codeword $\boldsymbol{c}$, leading to a file $\boldsymbol{w} \in \mathbb{F}_q^n$
- User verifies that $\boldsymbol{w}$ satisfies permuted low-weight parity-check equations induced by $\mathcal{C}$

Verification structures

- $\mathcal{C} \subseteq \mathbb{F}_q^n$ a code
- $\mathcal{Q}$ a set of ℓ-subsets of $[1, n]$

- $R : \quad \mathcal{Q} \times \mathbb{F}_q^n \quad \to \quad \mathbb{F}_q^\ell$
  $\qquad\qquad (u, \boldsymbol{w}) \quad \mapsto \quad \boldsymbol{w}_{|u}$

Verification structures

- $\mathcal{C} \subseteq \mathbb{F}_q^n$ a code
- $\mathcal{Q}$ a set of ℓ-subsets of $[1, n]$

- $R : \begin{array}{ccc} \mathcal{Q} \times \mathbb{F}_q^n & \to & \mathbb{F}_q^\ell \\ (u, \boldsymbol{w}) & \mapsto & \boldsymbol{w}_{|u} \end{array}$

Let $V : \mathcal{Q} \times \mathbb{F}_q^\ell \to \mathbb{F}_q^s$ for some $s \geq 1$. We say that $(\mathcal{Q}, V)$ is a **verification structure** for $\mathcal{C}$ if:

1. for all $i \in [1, n]$, there exists $u \in \mathcal{Q}$ such that $i \in u$;
2. for all $u \in \mathcal{Q}$, the map $\boldsymbol{a} \mapsto V(u, R(u, \boldsymbol{a}))$ is surjective and

$$\forall \boldsymbol{c} \in \mathcal{C}, V(u, R(u, \boldsymbol{c})) = \mathbf{0}\,.$$

Verification structures

- $\mathcal{C} \subseteq \mathbb{F}_q^n$ a code
- $\mathcal{Q}$ a set of ℓ-subsets of $[1, n]$

- $R : \begin{array}{ccc} \mathcal{Q} \times \mathbb{F}_q^n & \to & \mathbb{F}_q^\ell \\ (u, \boldsymbol{w}) & \mapsto & \boldsymbol{w}_{|u} \end{array}$

Let $V : \mathcal{Q} \times \mathbb{F}_q^\ell \to \mathbb{F}_q^s$ for some $s \geq 1$. We say that $(\mathcal{Q}, V)$ is a **verification structure** for $\mathcal{C}$ if:

1. for all $i \in [1, n]$, there exists $u \in \mathcal{Q}$ such that $i \in u$;
2. for all $u \in \mathcal{Q}$, the map $\boldsymbol{a} \mapsto V(u, R(u, \boldsymbol{a}))$ is surjective and

$$\forall \boldsymbol{c} \in \mathcal{C}, V(u, R(u, \boldsymbol{c})) = \mathbf{0}\,.$$

Notation:

- V is the **verification map** for $\mathcal{C}$, and $\mathcal{Q}$ is a **query set** for $\mathcal{C}$.
- $R(\boldsymbol{w}) := (R(u, \boldsymbol{w}) : u \in \mathcal{Q}) \in (\mathbb{F}_q^\ell)^{\mathcal{Q}}$,
- $R(\mathcal{C}) := \{R(\boldsymbol{c}), \boldsymbol{c} \in \mathcal{C}\}$ is called the **response code** of $\mathcal{C}$.

Example

Let $\mathcal{C} \subseteq \mathbb{F}_2^7$ the binary $[7,3,4]$ **Hadamard code**. $n = 7, \ell = 3, s = 1$

(non full-rank) $H = \begin{pmatrix} 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 1 & 1 \\ 0 & 1 & 0 & 0 & 1 & 1 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 & 1 \\ 0 & 0 & 1 & 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 & 0 & 1 \end{pmatrix}$

Example

Let $\mathcal{C} \subseteq \mathbb{F}_2^7$ the binary $[7,3,4]$ **Hadamard code**. $n = 7, \ell = 3, s = 1$

(non full-rank) $H = \begin{pmatrix} 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 1 & 1 \\ 0 & 1 & 0 & 0 & 1 & 1 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 & 1 \\ 0 & 0 & 1 & 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 & 0 & 1 \end{pmatrix}$

$$\mathcal{Q} = \{\{1,2,3\},\{1,4,5\},\{1,6,7\},\{2,5,6\},\{2,4,7\},\{3,4,6\},\{3,5,7\}\}.$$

Example

Let $\mathcal{C} \subseteq \mathbb{F}_2^7$ the binary $[7,3,4]$ **Hadamard code**. $n = 7, \ell = 3, s = 1$

(non full-rank) $H = \begin{pmatrix} 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 1 & 1 \\ 0 & 1 & 0 & 0 & 1 & 1 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 & 1 \\ 0 & 0 & 1 & 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 & 0 & 1 \end{pmatrix}$

$\mathcal{Q} = \{\{1,2,3\},\{1,4,5\},$
$\{1,6,7\},\{2,5,6\},\{2,4,7\},$
$\{3,4,6\},\{3,5,7\}\}$.

Verification map $V : \mathcal{Q} \times \mathbb{F}_2^3 \to \mathbb{F}_2$ defined by $V(u, \boldsymbol{b}) = \sum_{i=1}^{3} b_i$.

Example

Let $\mathcal{C} \subseteq \mathbb{F}_2^7$ the binary $[7,3,4]$ **Hadamard code**. $n = 7, \ell = 3, s = 1$

(non full-rank) $H = \begin{pmatrix} 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 1 & 1 \\ 0 & 1 & 0 & 0 & 1 & 1 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 & 1 \\ 0 & 0 & 1 & 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 & 0 & 1 \end{pmatrix}$

$$\mathcal{Q} = \{\{1,2,3\},\{1,4,5\},\{1,6,7\},\{2,5,6\},\{2,4,7\},\{3,4,6\},\{3,5,7\}\}.$$

Verification map $V : \mathcal{Q} \times \mathbb{F}_2^3 \to \mathbb{F}_2$ defined by $V(u, \boldsymbol{b}) = \sum_{i=1}^{3} b_i$.

The **response word** $\boldsymbol{r} = R(\boldsymbol{c}) \in (\mathbb{F}_2^3)^7$ is:

$$\boldsymbol{r} = \left(\begin{pmatrix} c_1 \\ c_2 \\ c_3 \end{pmatrix}, \begin{pmatrix} c_1 \\ c_4 \\ c_5 \end{pmatrix}, \begin{pmatrix} c_1 \\ c_6 \\ c_7 \end{pmatrix}, \begin{pmatrix} c_2 \\ c_5 \\ c_6 \end{pmatrix}, \begin{pmatrix} c_2 \\ c_4 \\ c_7 \end{pmatrix}, \begin{pmatrix} c_3 \\ c_4 \\ c_6 \end{pmatrix}, \begin{pmatrix} c_3 \\ c_5 \\ c_7 \end{pmatrix} \right).$$

The PoR construction

- **Key generation:**

$$\kappa := \sigma = (\sigma_1, \ldots, \sigma_n) \leftarrow_{\mathrm{R}} \mathfrak{S}(\mathbb{F}_q)^n$$

The PoR construction

- **Key generation:**

$$\kappa := \sigma = (\sigma_1, \ldots, \sigma_n) \leftarrow_{\mathrm{R}} \mathfrak{S}(\mathbb{F}_q)^n$$

- **Initialisation:**

$$\mathsf{Init}(m, \sigma): \quad m \in \mathbb{F}_q^k \mapsto c \in \mathcal{C} \mapsto w = \sigma(c) \in \mathbb{F}_q^n$$

The PoR construction

- **Key generation:**

$$\kappa := \sigma = (\sigma_1, \ldots, \sigma_n) \leftarrow_{\mathrm{R}} \mathfrak{S}(\mathbb{F}_q)^n$$

- **Initialisation:**

$$\mathsf{Init}(m, \sigma): \quad m \in \mathbb{F}_q^k \mapsto c \in \mathcal{C} \mapsto w = \sigma(c) \in \mathbb{F}_q^n$$

- **Verification:**
  1. Challenge $u = \{u_1, \ldots, u_\ell\} \leftarrow_{\mathrm{R}} \mathcal{Q}$.
  2. The prover must send back $r_u := w_{|u} \in \mathbb{F}_q^\ell$.
  3. $$\mathsf{Check}(u, r_u, \sigma) := \begin{cases} 1 & \text{if } V(u, \sigma^{-1}(r_u)) = 0 \\ 0 & \text{otherwise.} \end{cases}$$

The PoR construction

- **Key generation:**

$$\kappa := \sigma = (\sigma_1, \ldots, \sigma_n) \leftarrow_{\mathrm{R}} \mathfrak{S}(\mathbb{F}_q)^n$$

- **Initialisation:**

$$\mathsf{Init}(m, \sigma): \quad m \in \mathbb{F}_q^k \mapsto c \in \mathcal{C} \mapsto w = \sigma(c) \in \mathbb{F}_q^n$$

- **Verification:**
  1. Challenge $u = \{u_1, \ldots, u_\ell\} \leftarrow_{\mathrm{R}} \mathcal{Q}$.
  2. The prover must send back $r_u := w_{|u} \in \mathbb{F}_q^\ell$.
  3. $$\mathsf{Check}(u, r_u, \sigma) := \begin{cases} 1 & \text{if } V(u, \sigma^{-1}(r_u)) = 0 \\ 0 & \text{otherwise.} \end{cases}$$

- **Extraction:** Given $r = (r_u : u \in \mathcal{Q}) \in (\mathbb{F}_q^\ell)^{\mathcal{Q}}$, the verifier runs a **decoding algorithm for the response code** $R(\sigma(\mathcal{C}))$ and outputs the result.

Analysis

Given the word $\boldsymbol{r} = (\boldsymbol{r}_u : u \in \mathcal{Q})$, one can define:

$$\begin{cases} V(u, \sigma^{-1}(\boldsymbol{r}_u)) \neq 0 & \rightarrow \quad \text{an erasure} \\ V(u, \sigma^{-1}(\boldsymbol{r}_u)) = 0 \text{ but } \boldsymbol{r}_u \neq \boldsymbol{w}_{|u} & \rightarrow \quad \text{an error} \end{cases}$$

Analysis

Given the word $\boldsymbol{r} = (\boldsymbol{r}_u : u \in \mathcal{Q})$, one can define:

$$\begin{cases} V(u, \sigma^{-1}(\boldsymbol{r}_u)) \neq 0 & \rightarrow \quad \text{an erasure} \\ V(u, \sigma^{-1}(\boldsymbol{r}_u)) = 0 \text{ but } \boldsymbol{r}_u \neq \boldsymbol{w}_{|u} & \rightarrow \quad \text{an error} \end{cases}$$

If $E = |\{\text{erasures}\}|$ and $B = |\{\text{errors}\}|$, then

Extraction **succeeds** if $E + 2B < \Delta$, where Δ is a threshold for error-and-erasure decoding on the code $R(\mathcal{C})$.

Analysis

Given the word $\boldsymbol{r} = (\boldsymbol{r}_u : u \in \mathcal{Q})$, one can define:

$$\begin{cases} V(u, \sigma^{-1}(\boldsymbol{r}_u)) \neq 0 & \rightarrow \quad \text{an erasure} \\ V(u, \sigma^{-1}(\boldsymbol{r}_u)) = 0 \text{ but } \boldsymbol{r}_u \neq \boldsymbol{w}_{|u} & \rightarrow \quad \text{an error} \end{cases}$$

If $E = |\{\text{erasures}\}|$ and $B = |\{\text{errors}\}|$, then

Extraction **succeeds** if $E + 2B < \Delta$, where Δ is a threshold for error-and-erasure decoding on the code $R(\mathcal{C})$.

$\Rightarrow$ We need to estimate

$$\Pr\left(\begin{array}{c} \mathsf{Extract}(\boldsymbol{r}, \kappa) \neq \boldsymbol{m} \\ \text{and} \\ \text{succ}(\boldsymbol{r}, \kappa) \geq 1 - \varepsilon \end{array} \middle| \begin{array}{l} \boldsymbol{m} \leftarrow_{\mathrm{R}} \mathcal{M} \\ \kappa \leftarrow_{\mathrm{R}} \mathsf{KeyGen}(1^{\lambda}) \\ \boldsymbol{w} \leftarrow \mathsf{Init}(\boldsymbol{m}, \kappa) \\ \boldsymbol{r} \leftarrow \mathsf{Resp}(\boldsymbol{w}) \end{array}\right)$$

Analysis

Given the word $\boldsymbol{r} = (\boldsymbol{r}_u : u \in \mathcal{Q})$, one can define:

$$\begin{cases} V(u, \sigma^{-1}(\boldsymbol{r}_u)) \neq 0 & \rightarrow \quad \text{an erasure} \\ V(u, \sigma^{-1}(\boldsymbol{r}_u)) = 0 \text{ but } \boldsymbol{r}_u \neq \boldsymbol{w}_{|u} & \rightarrow \quad \text{an error} \end{cases}$$

If $E = |\{\text{erasures}\}|$ and $B = |\{\text{errors}\}|$, then

Extraction **succeeds** if $E + 2B < \Delta$, where Δ is a threshold for error-and-erasure decoding on the code $R(\mathcal{C})$.

$\Rightarrow$ We need to estimate

$$\Pr\left[\begin{array}{c} E + 2B \geq \Delta \\ \text{and} \\ E \leq \varepsilon |\mathcal{Q}| \end{array} \middle| \begin{array}{l} \boldsymbol{m} \leftarrow_{\mathrm{R}} \mathcal{M} \\ \kappa \leftarrow_{\mathrm{R}} \mathbf{KeyGen}(1^\lambda) \\ \boldsymbol{w} \leftarrow \mathbf{Init}(\boldsymbol{m}, \kappa) \\ \boldsymbol{r} \leftarrow \mathbf{Resp}(\boldsymbol{w}) \end{array}\right]$$

Main theorem

Notation.

- $N = |\mathcal{Q}|$ the length of $R(\mathcal{C})$,
- $\delta = \Delta/N$ its relative error-and-erasure decoding capability.

Theorem (simplified). Assume $(\star)$ and define $\varepsilon_0 = \delta\frac{1-\alpha}{1+\alpha}$. Then, for every $\varepsilon < \varepsilon_0$, the PoR scheme associated to $\mathcal{C}$ and $(\mathcal{Q}, V)$ is (ε, τ)-sound, where

$$\tau = \frac{4}{N(1+\alpha)^2(\varepsilon_0 - \varepsilon)^2},$$

where:

- $\alpha \ll 1$ is a bound on the proba that a random answer is accepted
- $(\star)$ $\quad \min\{d^{\perp}(\mathcal{C}_{|u}), u \in \mathcal{Q}\} > \max\{|u \cap v|, u \neq v \in \mathcal{Q}\}$.

Our parameters

Storage needs:

- **prover** stores $(n-k)\log q$ additional bits, where $\mathcal{C} \subseteq \mathbb{F}_q^n$ has dimension k
- **verifier** stores an n-tuple of permutations σ

Our parameters

Storage needs:

- **prover** stores $(n-k)\log q$ additional bits, where $\mathcal{C} \subseteq \mathbb{F}_q^n$ has dimension k
- **verifier** stores ~~an n-tuple of permutations σ~~
  a key κ for generating an n-tuple of suitable PRPs

Our parameters

Storage needs:

- **prover** stores $(n-k)\log q$ additional bits, where $\mathcal{C} \subseteq \mathbb{F}_q^n$ has dimension k
- **verifier** stores a key κ for generating an n-tuple of suitable PRPs

Computational needs:

- **prover** only needs to read symbols (no additional computation)
- **verifier** inverts ℓ permutations $\sigma_{|u}$, and computes $\boldsymbol{r}_u \mapsto V(u, \boldsymbol{r}_u) \in \mathbb{F}_q^s$.

Our parameters

Storage needs:

- **prover** stores $(n-k)\log q$ additional bits, where $\mathcal{C} \subseteq \mathbb{F}_q^n$ has dimension k
- **verifier** stores a key κ for generating an n-tuple of suitable PRPs

Computational needs:

- **prover** only needs to read symbols (no additional computation)
- **verifier** inverts ℓ permutations $\sigma_{|u}$, and computes $\boldsymbol{r}_u \mapsto V(u, \boldsymbol{r}_u) \in \mathbb{F}_q^s$.

$\Rightarrow$ **Our goal:** find a code $\mathcal{C} \subseteq \mathbb{F}_q^n$ and $(\mathcal{Q}, V)$ such that

- $\ell := |u| \ll n$, for $u \in \mathcal{Q}$
- $\max\{|v \cap v'|, v \neq v' \in \mathcal{Q}\} < d^{\perp}(\mathcal{C}_{|u})$, for every $u \in \mathcal{Q}$
- $\mathcal{C}$ has large dimension
- $R(\mathcal{C})$ has large relative minimum distance δ and large length N

1. Proofs-of-* for secure remote storage

2. A generic construction of proof-of-retrievability
   - Model and definition
   - A generic construction of PoR
   - Some instances

Instance 1: tensor product of codes

$\mathcal{A} \subseteq \mathbb{F}_q^\ell$ a code of small length. Its s-fold **tensor product** is:

$$\mathcal{A}^{\otimes s} := \left\{ (a_{i_1}^{(1)} \dots a_{i_s}^{(s)} : \boldsymbol{i} \in [1,\ell]^s) \;\middle|\; \forall 1 \leq j \leq s,\, \boldsymbol{a}^{(j)} \in \mathcal{A} \right\} \subseteq \mathbb{F}_q^{\ell^s}$$

Instance 1: tensor product of codes

$\mathcal{A} \subseteq \mathbb{F}_q^\ell$ a code of small length. Its s-fold **tensor product** is:

$$\mathcal{A}^{\otimes s} := \left\{ (a_{i_1}^{(1)} \dots a_{i_s}^{(s)} : \boldsymbol{i} \in [1,\ell]^s) \;\middle|\; \forall 1 \leq j \leq s,\, \boldsymbol{a}^{(j)} \in \mathcal{A} \right\} \subseteq \mathbb{F}_q^{\ell^s}$$

Verification structure $(\mathcal{Q}, V)$:

- $\mathcal{Q} = \{$"axis-parallel lines" $L \subset [1,\ell]^s\}$,
- $V(L, \boldsymbol{b}) = H\boldsymbol{b}$, where H is a parity-check matrix for $\mathcal{A}$.

Instance 1: tensor product of codes

$\mathcal{A} \subseteq \mathbb{F}_q^\ell$ a code of small length. Its s-fold **tensor product** is:

$$\mathcal{A}^{\otimes s} := \left\{ (a_{i_1}^{(1)} \dots a_{i_s}^{(s)} : \boldsymbol{i} \in [1,\ell]^s) \;\middle|\; \forall 1 \le j \le s,\, \boldsymbol{a}^{(j)} \in \mathcal{A} \right\} \subseteq \mathbb{F}_q^{\ell^s}$$

Verification structure $(\mathcal{Q}, V)$**:**

- $\mathcal{Q} = \{$"axis-parallel lines" $L \subset [1,\ell]^s\}$,
- $V(L, \boldsymbol{b}) = H\boldsymbol{b}$, where H is a parity-check matrix for $\mathcal{A}$.

Proposition. If $\mathcal{A} \subseteq \mathbb{F}_q^\ell$ is an MDS code of distance d, then

$$d_{\min}(R(\mathcal{A}^{\otimes s})) = sd^{s-1}\,.$$

Instance 1: tensor product of codes

$\mathcal{A} \subseteq \mathbb{F}_q^\ell$ a code of small length. Its s-fold **tensor product** is:

$$\mathcal{A}^{\otimes s} := \left\{ (a_{i_1}^{(1)} \dots a_{i_s}^{(s)} : \boldsymbol{i} \in [1,\ell]^s) \;\middle|\; \forall 1 \le j \le s,\, \boldsymbol{a}^{(j)} \in \mathcal{A} \right\} \subseteq \mathbb{F}_q^{\ell^s}$$

Verification structure $(\mathcal{Q}, V)$**:**

- $\mathcal{Q} = \{$"axis-parallel lines" $L \subset [1,\ell]^s\}$,
- $V(L, \boldsymbol{b}) = H\boldsymbol{b}$, where H is a parity-check matrix for $\mathcal{A}$.

Proposition. If $\mathcal{A} \subseteq \mathbb{F}_q^\ell$ is an MDS code of distance d, then

$$d_{\min}(R(\mathcal{A}^{\otimes s})) = sd^{s-1}\,.$$

Theorem $[\gamma > 0, d = \gamma\ell, \ell \to \infty]$. A PoR based on $\mathcal{A}^{\otimes s}$ is (ε, τ)-sound for every $\varepsilon < \gamma^s$, where

$$\tau = \mathcal{O}\left(\frac{1}{s(\gamma\ell)^s}\right) = \mathcal{O}(1/n)\,.$$

Instance 2: code based on $AG(2, q)$

Let $A := \mathbb{F}_q^2$ and $\mathcal{Q} := \{\text{affine lines } L \subset A\}$.

$$\mathcal{C}_q := \{\boldsymbol{c} \in \mathbb{F}_q^A, \forall L \in \mathcal{Q}, \sum_{x \in L} c_x = 0\} \subseteq \mathbb{F}_q^A$$

Instance 2: code based on $AG(2,q)$

Let $A := \mathbb{F}_q^2$ and $\mathcal{Q} := \{\text{affine lines } L \subset A\}$.

$$\mathcal{C}_q := \{\boldsymbol{c} \in \mathbb{F}_q^A, \forall L \in \mathcal{Q}, \sum_{x \in L} c_x = 0\} \subseteq \mathbb{F}_q^A$$

Verification structure $(\mathcal{Q}, V)$:

- query set $\mathcal{Q}$ defined above
- for $\boldsymbol{b} \in \mathbb{F}_q^q \simeq \mathbb{F}_q^L$, define $V(L, \boldsymbol{b}) = \sum_{i=1}^{q} b_i$

Instance 2: code based on $AG(2,q)$

Let $A := \mathbb{F}_q^2$ and $\mathcal{Q} := \{\text{affine lines } L \subset A\}$.

$$\mathcal{C}_q := \{\boldsymbol{c} \in \mathbb{F}_q^A, \forall L \in \mathcal{Q}, \sum_{x \in L} c_x = 0\} \subseteq \mathbb{F}_q^A$$

Verification structure $(\mathcal{Q}, V)$**:**

- query set $\mathcal{Q}$ defined above
- for $\boldsymbol{b} \in \mathbb{F}_q^q \simeq \mathbb{F}_q^L$, define $V(L, \boldsymbol{b}) = \sum_{i=1}^{q} b_i$

Prop. Response code $R(\mathcal{C})$ has relative minimum distance $\delta \geq 1 - 2/q$.

Instance 2: code based on $AG(2,q)$

Let $A := \mathbb{F}_q^2$ and $\mathcal{Q} := \{\text{affine lines } L \subset A\}$.

$$\mathcal{C}_q := \{\boldsymbol{c} \in \mathbb{F}_q^A, \forall L \in \mathcal{Q}, \sum_{x \in L} c_x = 0\} \subseteq \mathbb{F}_q^A$$

Verification structure $(\mathcal{Q}, V)$**:**

- query set $\mathcal{Q}$ defined above
- for $\boldsymbol{b} \in \mathbb{F}_q^q \simeq \mathbb{F}_q^L$, define $V(L, \boldsymbol{b}) = \sum_{i=1}^{q} b_i$

Prop. Response code $R(\mathcal{C})$ has relative minimum distance $\delta \geq 1 - 2/q$.

Theorem $[q \to \infty]$. A PoR based on $\mathcal{C}_q$ is (ε, τ) sound for every $\varepsilon < 1 - o(1)$, where

$$\tau = \mathcal{O}\left(\frac{1}{(1-\varepsilon)q^2}\right) = \mathcal{O}(1/n)\,.$$

Instance 2: code based on $AG(2,q)$

Let $A := \mathbb{F}_q^2$ and $\mathcal{Q} := \{\text{affine lines } L \subset A\}$.

$$\mathcal{C}_q := \{\boldsymbol{c} \in \mathbb{F}_q^A, \forall L \in \mathcal{Q}, \sum_{x \in L} c_x = 0\} \subseteq \mathbb{F}_q^A$$

Verification structure $(\mathcal{Q}, V)$**:**

- query set $\mathcal{Q}$ defined above
- for $\boldsymbol{b} \in \mathbb{F}_q^q \simeq \mathbb{F}_q^L$, define $V(L, \boldsymbol{b}) = \sum_{i=1}^{q} b_i$

Prop. Response code $R(\mathcal{C})$ has relative minimum distance $\delta \geq 1 - 2/q$.

Theorem $[q \to \infty]$. A PoR based on $\mathcal{C}_q$ is (ε, τ) sound for every $\varepsilon < 1 - o(1)$, where

$$\tau = \mathcal{O}\left(\frac{1}{(1-\varepsilon)q^2}\right) = \mathcal{O}(1/n)\,.$$

Prop [from Hamada'68]. If $q = 2^e$, then the code $\mathcal{C}_q$ has rate

$$1 - \mathcal{O}\left(n^{-1+\frac{\log_3(2)}{2}}\right).$$

Numerical parameters

Using the code $\mathcal{C}_q$ with $q = 2^{13}$

- initial file of **size** $\simeq$ 106 MB,
- **server storage**: $+2.4\%$ storage overhead,
- **client storage**: a few bits for κ,
- **communication rate**: $1.25 \cdot 10^{-4}$ of the initial file size.

Conclusion

We presented:

- a generic construction of PoR based on codes with locality
- low computation, low storage

Possible further works:

- better instances of codes and verification structures
- other features (*e.g.* dynamic PoR)