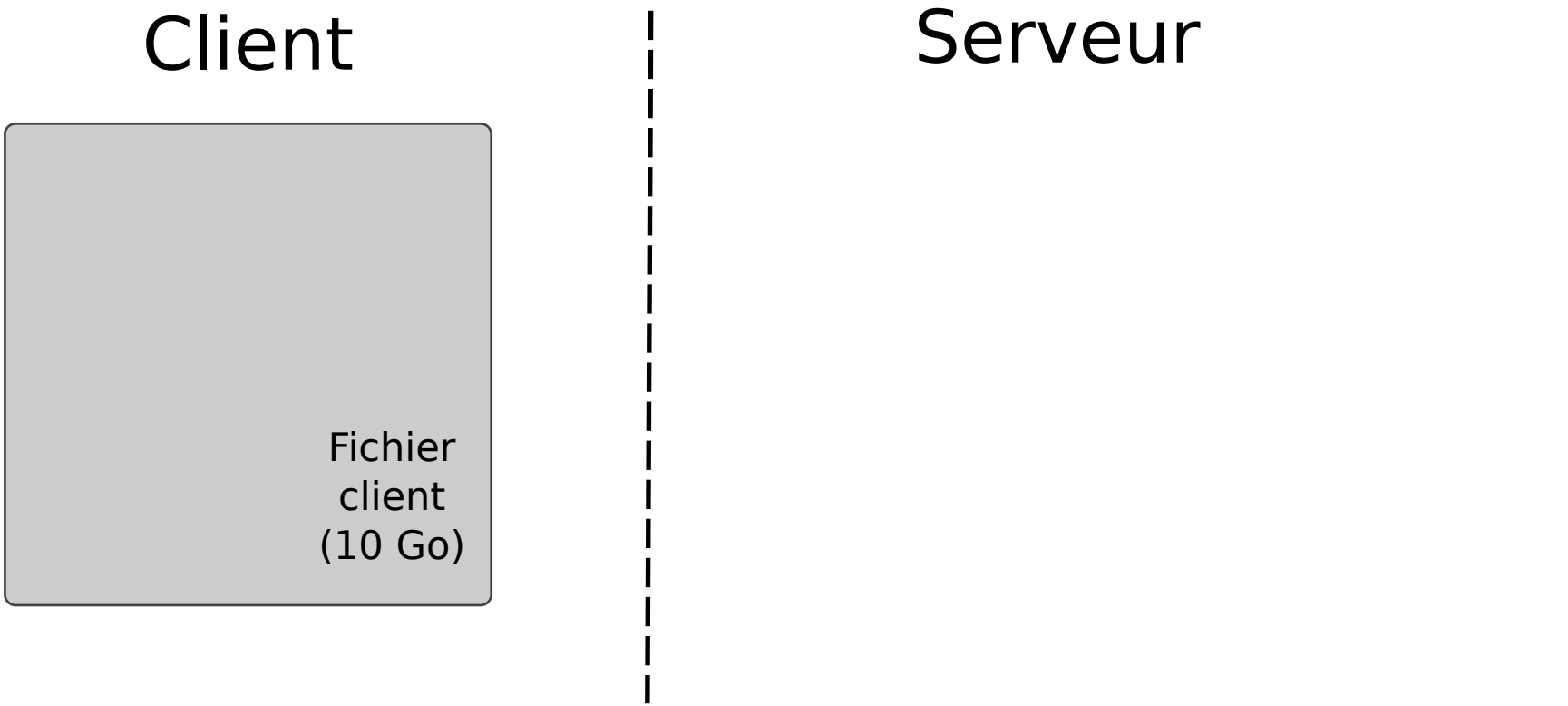
Preuves de récupérabilité fondées sur des codes localement décodables

Julien Lavauzelle

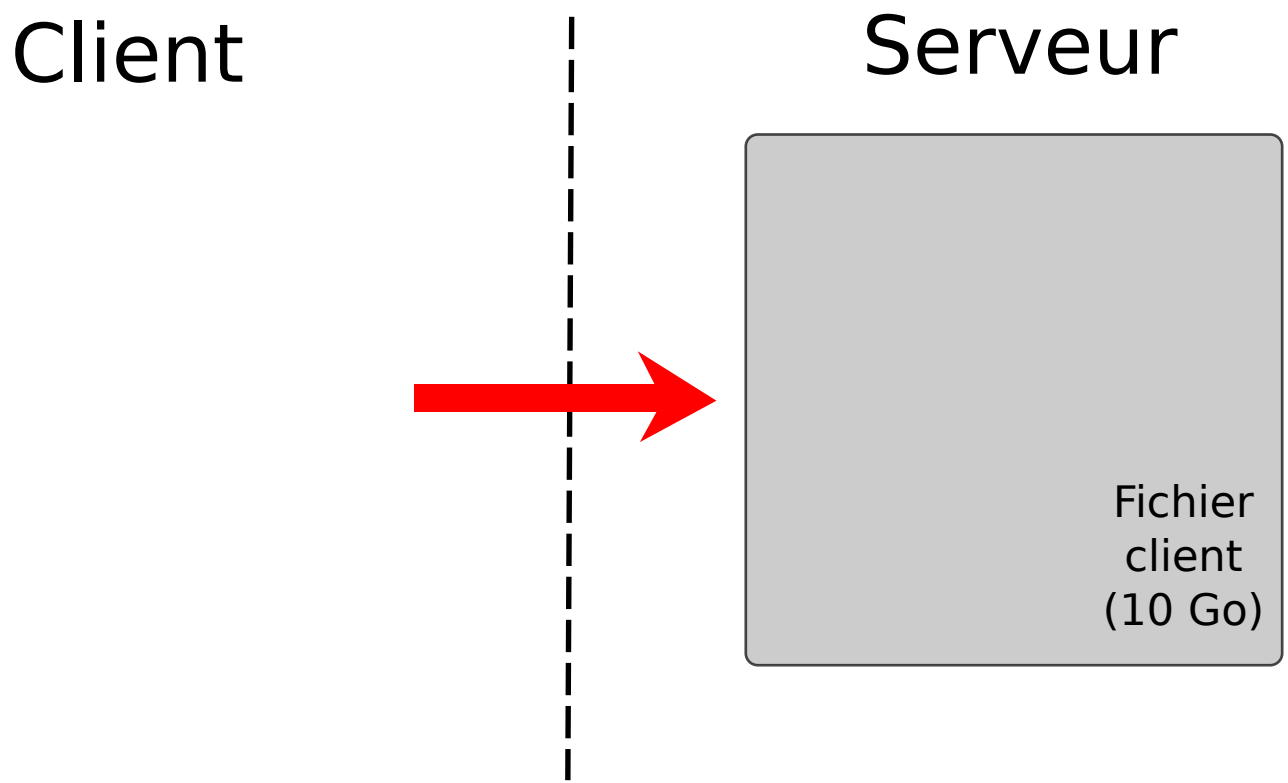
équipe GRACE
LIX & INRIA Saclay

Journées Codes & Cryptographie, La Londe-les-Maures
8 octobre 2015

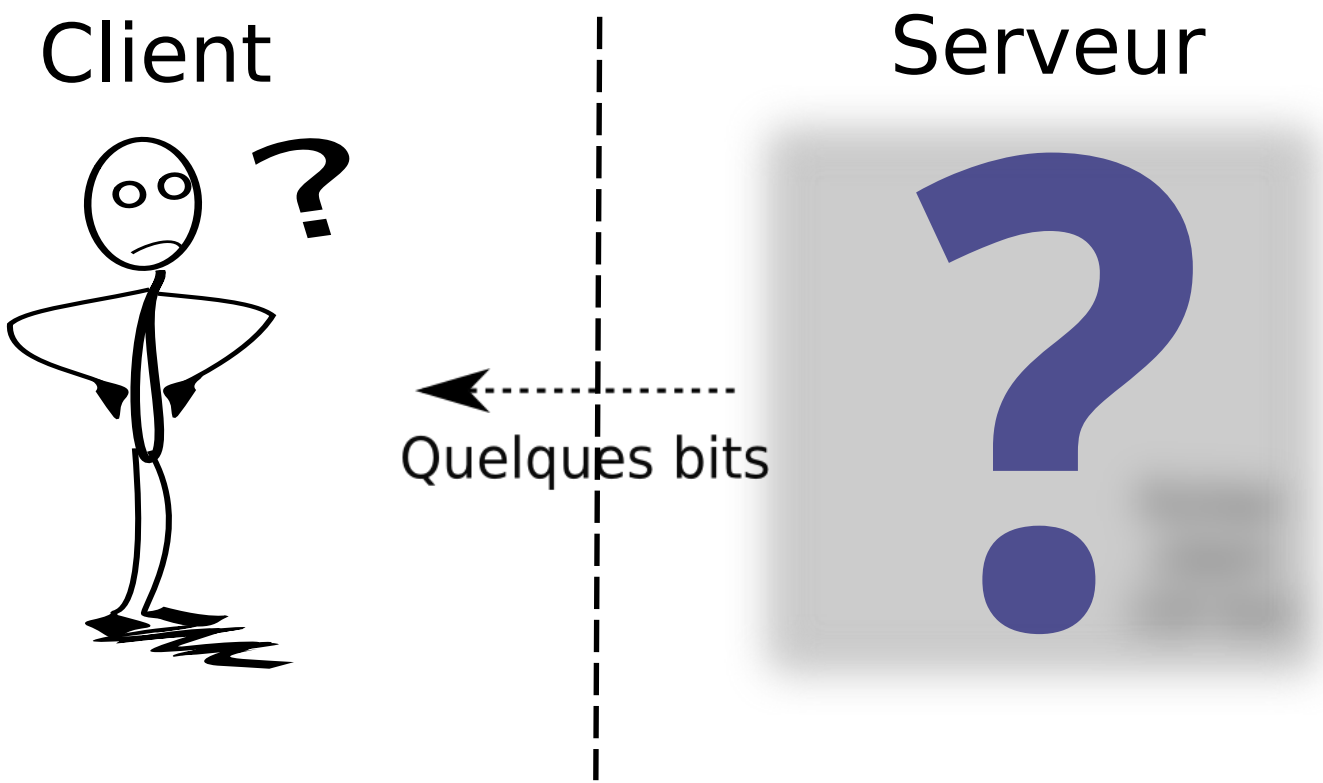
Problématique



Problématique



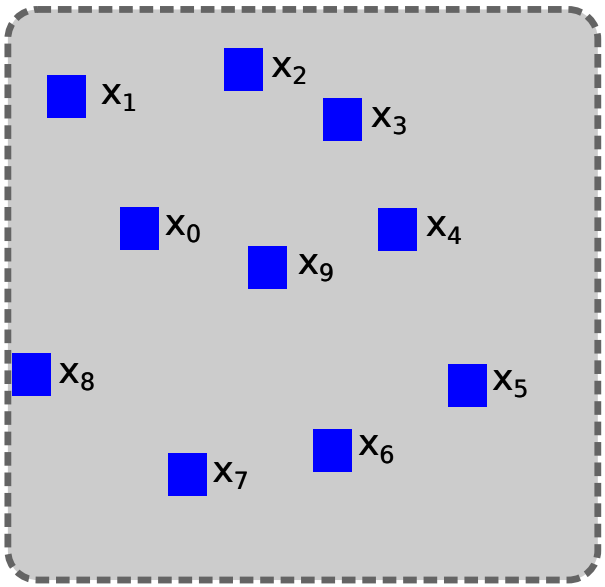
Problématique



Le fichier est-il encore extractible ?

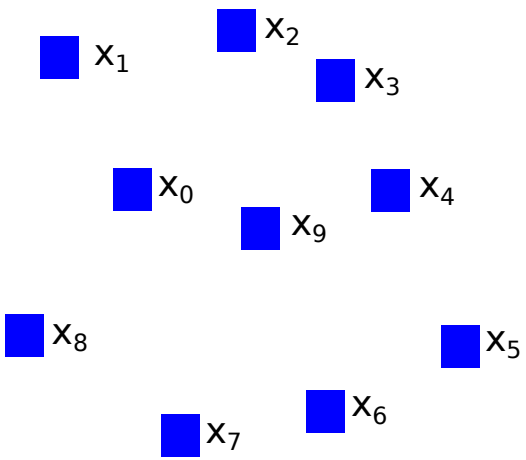
Un exemple

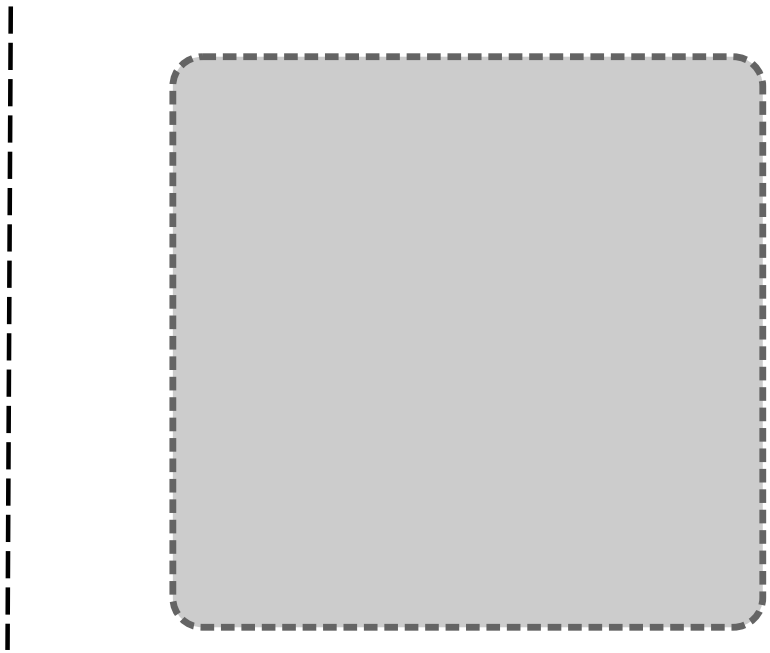
A. Juels, B. Kaliski Jr., *PORs : Proofs of Retrievability for large files.* in Proceedings of the 2007 ACM Conference on Computer and Communications Security, Alexandria, USA, 2007.



Un exemple

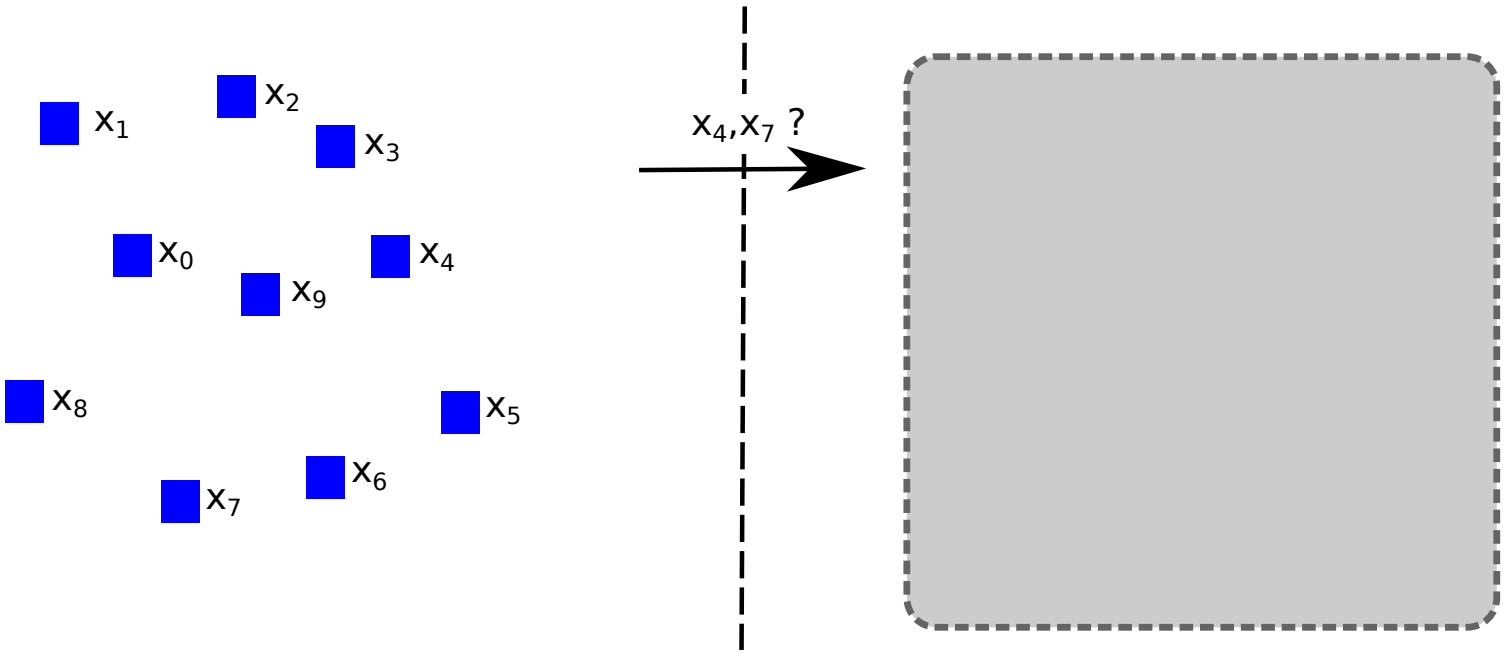
A. Juels, B. Kaliski Jr., *PORs : Proofs of Retrievability for large files.* in Proceedings of the 2007 ACM Conference on Computer and Communications Security, Alexandria, USA, 2007.





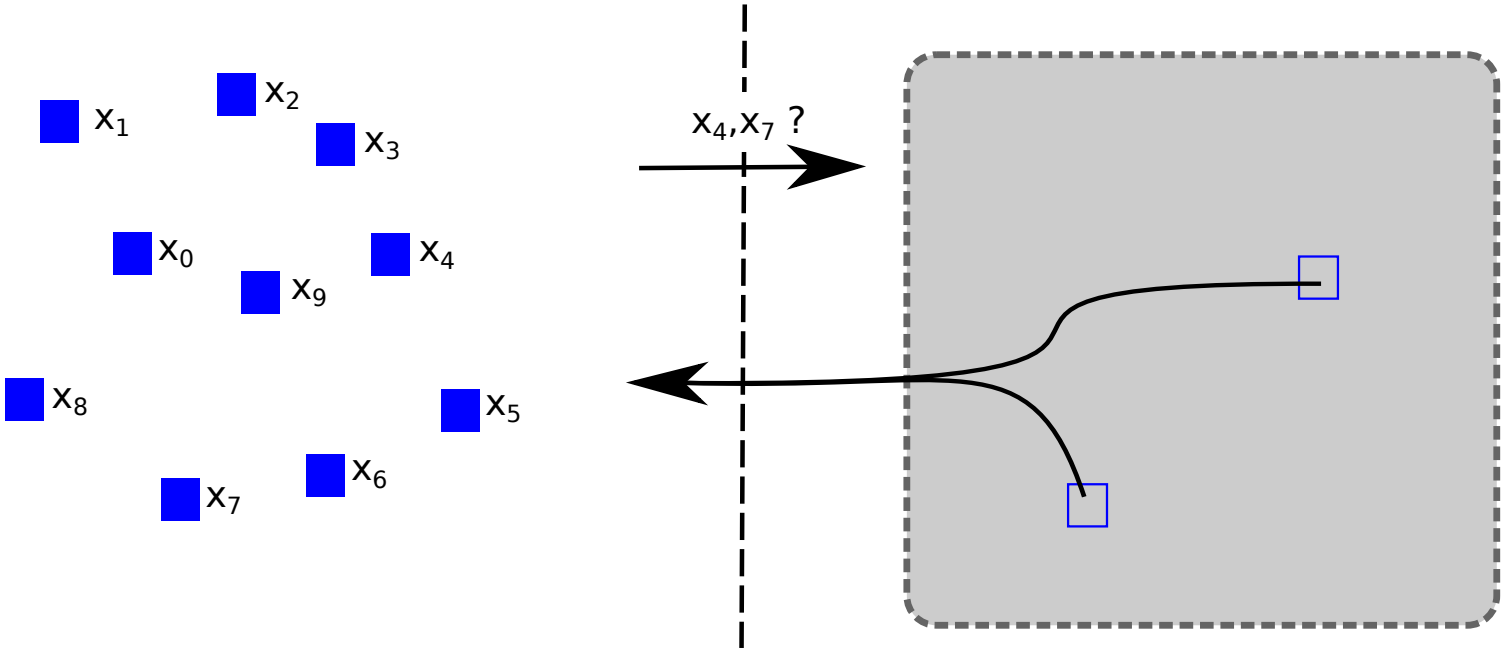
Un exemple

A. Juels, B. Kaliski Jr., *PORs : Proofs of Retrievability for large files.* in Proceedings of the 2007 ACM Conference on Computer and Communications Security, Alexandria, USA, 2007.



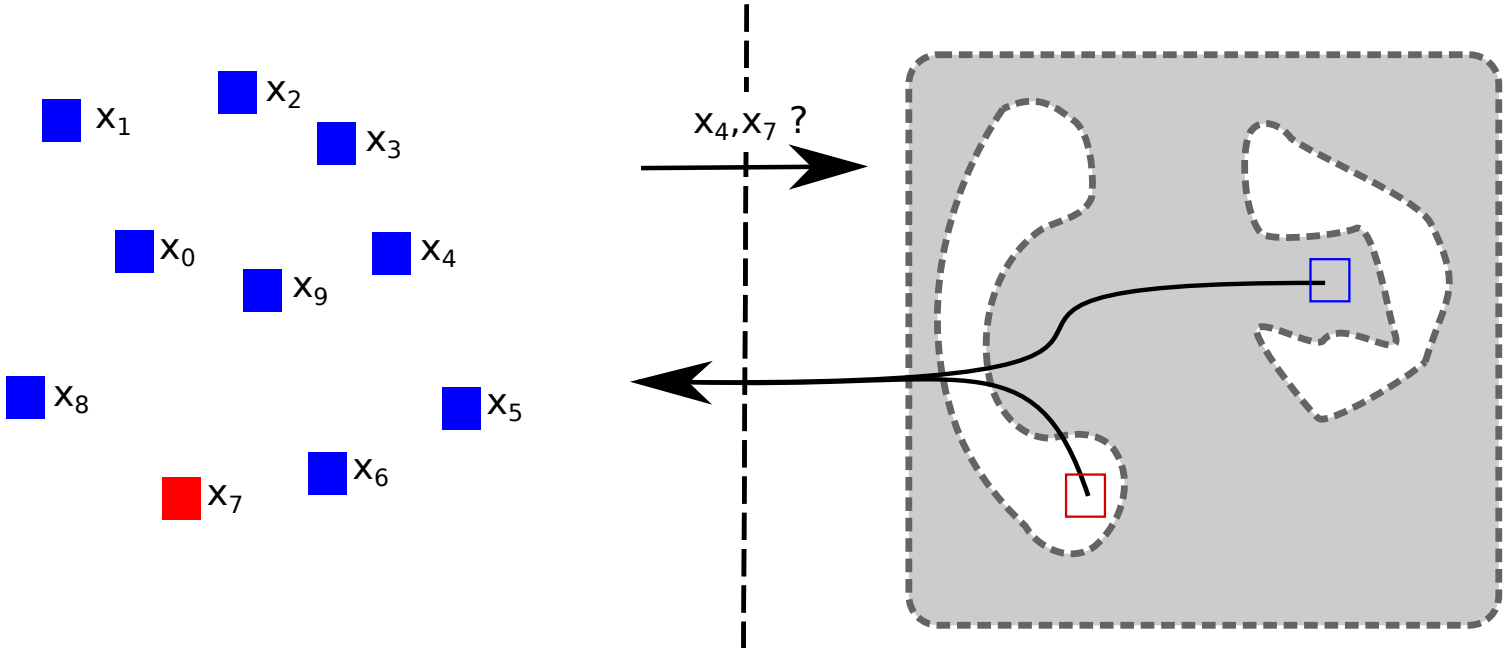
Un exemple

A. Juels, B. Kaliski Jr., *PORs : Proofs of Retrievability for large files.* in Proceedings of the 2007 ACM Conference on Computer and Communications Security, Alexandria, USA, 2007.



Un exemple

A. Juels, B. Kaliski Jr., *PORs : Proofs of Retrievability for large files.* in Proceedings of the 2007 ACM Conference on Computer and Communications Security, Alexandria, USA, 2007.



Définition (preuve de récupérabilité)

Preuve de récupérabilité (Proof of Retrievability, PoR) = ensemble de trois procédures :

- **Initialisation** [*Client*] :

$$F \mapsto \textbf{Init}(F) = (\tilde{F}, \mathtt{data}_F)$$

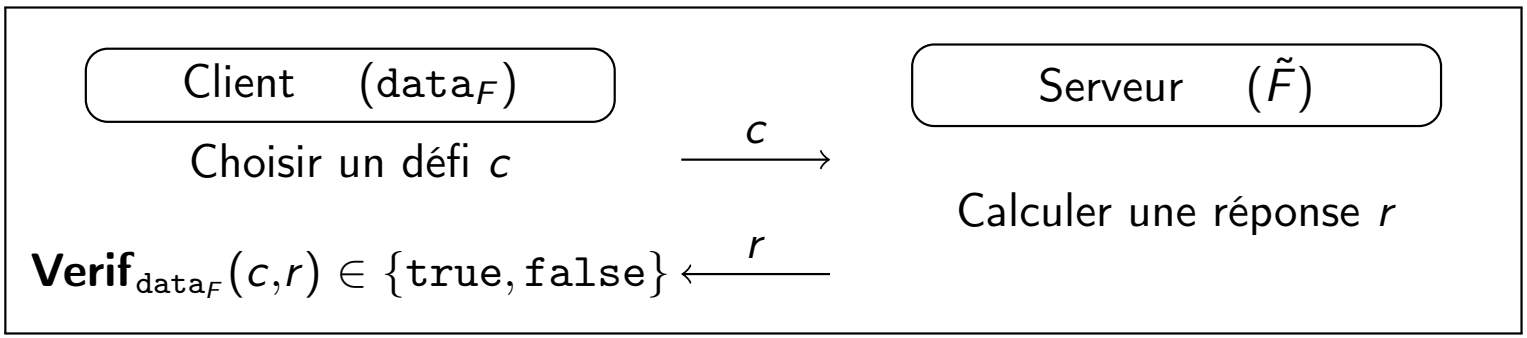
Définition (preuve de récupérabilité)

Preuve de récupérabilité (Proof of Retrievability, PoR) = ensemble de trois procédures :

- **Initialisation** [*Client*] :

$$F \mapsto \textbf{Init}(F) = (\tilde{F}, \mathtt{data}_F)$$

- **Vérification** [*Client* $\longleftrightarrow$ *Serveur*] :



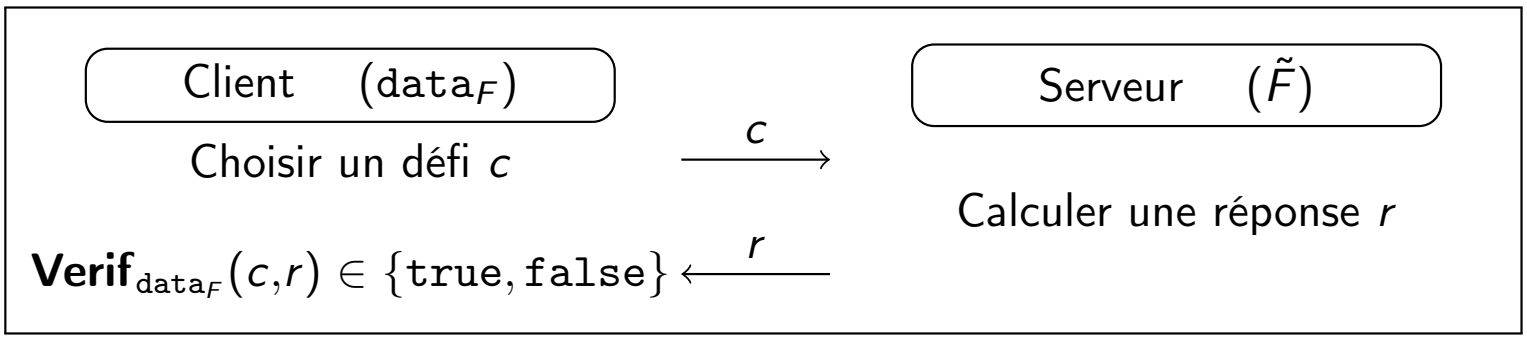
Définition (preuve de récupérabilité)

Preuve de récupérabilité (Proof of Retrievability, PoR) = ensemble de trois procédures :

- **Initialisation** [*Client*] :

$$F \mapsto \textbf{Init}(F) = (\tilde{F}, \mathtt{data}_F)$$

- **Vérification** [*Client* $\longleftrightarrow$ *Serveur*] :



- **Extraction** [*Client* $\longleftrightarrow$ *Serveur*]. On veut :

$$\textbf{Extract}(\mathtt{data}_F) = F$$

Modèle de sécurité

Définition (ε-adversaire). Soit $\varepsilon \in [0,1]$, $\mathcal{P}$ une PoR et X la distribution des défis. Un *ε-adversaire* $\mathcal{A}$ pour $\mathcal{P}$ est un algorithme tel que, pour tout fichier F :

$$\mathbb{P}_{x \sim X}\left[\mathbf{Verif}_{\mathtt{data}_F}(x, \mathcal{A}(\tilde{F}, x)) = \mathtt{false}\right] < \varepsilon\,.$$

Modèle de sécurité

Définition (ε-adversaire). Soit $\varepsilon \in [0,1]$, $\mathcal{P}$ une PoR et X la distribution des défis. Un *ε-adversaire* $\mathcal{A}$ pour $\mathcal{P}$ est un algorithme tel que, pour tout fichier F :

$$\mathbb{P}_{x \sim X}\left[\textbf{Verif}_{\texttt{data}_F}(x, \mathcal{A}(\tilde{F}, x)) = \texttt{false}\right] < \varepsilon\,.$$

Définition (Sécurité d'une PoR). Soient $\varepsilon, \rho \in [0,1]$. Une PoR est dite (ε, ρ)-sûre si, pour tout ε-adversaire $\mathcal{A}$ et pour tout fichier F :

$$\mathbb{P}\left[\textbf{Extract}(\texttt{data}_F) = F\right] \geq \rho\,,$$

où la probabilité est prise selon l'aléa de l'adversaire et des tirages effectués durant l'extraction.

Objectifs

- Communication faible
- Redondance serveur et stockage client faibles
- Complexité faible
- Nombre illimité de vérifications

Vers une vérification structurelle

Inconvénients de JK'07 :

- vérifications limitées ;
- stockage client non négligeable.

Vers une vérification structurelle

Inconvénients de JK'07 :

- vérifications limitées ;
- stockage client non négligeable.

Solution proposée :

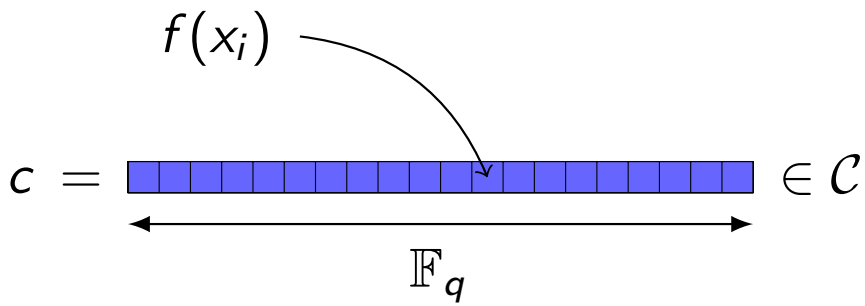
- vérifier la **structure** du fichier plutôt que la valeur de symboles.

Codes correcteurs

$\mathbb{F}_q = \{x_1, \ldots, x_q\}$ corps fini.

Exemple : code de Reed-Solomon (longueur $n = q$).

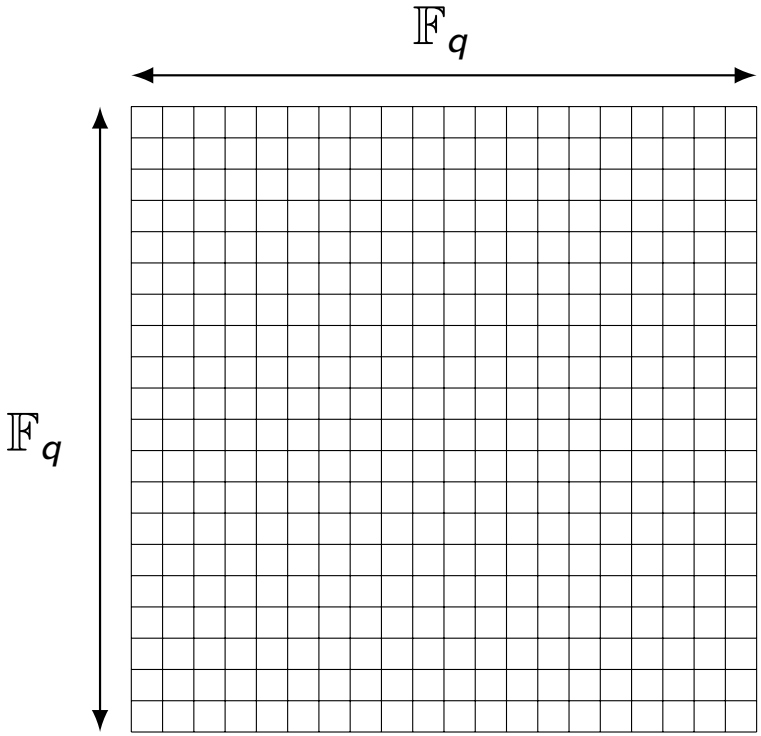
$$\begin{aligned} \mathcal{C} = \mathrm{RS}_q(k) &= \{(f(x_1), \ldots, f(x_q)), f \in \mathbb{F}_q[X], \deg f < k\} \\ &\simeq \{f \in \mathbb{F}_q[X], \deg f < k\} \end{aligned}$$



Relèvement de code affine-invariant

Alan Guo, Swastik Kopparty, Madhu Sudan, *New Affine-Invariant Codes from Lifting* in Proceedings of Innovations in Theoretical Computer Science, ITCS '13, Berkeley, USA, 2013.

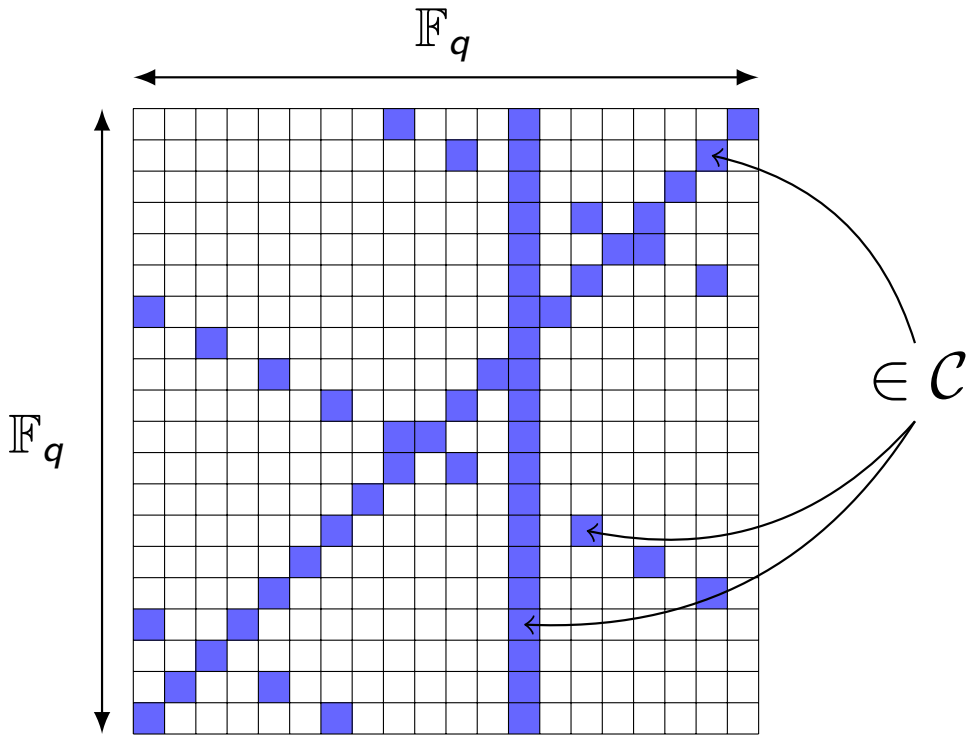
$$\mathcal{L} = \mathrm{Lift}_m(\mathcal{C}) = \{g \in \mathbb{F}_q[X_1, \ldots, X_m] \quad \}$$



Relèvement de code affine-invariant

Alan Guo, Swastik Kopparty, Madhu Sudan, *New Affine-Invariant Codes from Lifting* in Proceedings of Innovations in Theoretical Computer Science, ITCS '13, Berkeley, USA, 2013.

$$\mathcal{L} = \mathrm{Lift}_m(\mathcal{C}) = \{g \in \mathbb{F}_q[X_1, \ldots, X_m], \forall \text{ droite affine } \ell, g_{|\ell} \in \mathcal{C}\}$$



Quelques propriétés du relèvement

$$\mathcal{L} = \mathrm{Lift}_m(\mathcal{C}) = \{g : \mathbb{F}_q^m \to \mathbb{F}_q, \forall \text{ droite affine } \ell, g_{|\ell} \in \mathcal{C}\}$$

- La construction nécessite $\mathcal{C}$ affine-invariant.

Quelques propriétés du relèvement

$$\mathcal{L} = \mathrm{Lift}_m(\mathcal{C}) = \{g : \mathbb{F}_q^m \to \mathbb{F}_q, \forall \text{ droite affine } \ell, g_{|\ell} \in \mathcal{C}\}$$

- La construction nécessite $\mathcal{C}$ affine-invariant.
- **Exemple :** Soit $k < q$ et $\mathcal{C} = \mathrm{RS}_q(k)$. On a :

$$\mathrm{RM}_q(m,k) \quad \subseteq \quad \mathrm{Lift}_m(\mathcal{C}) \quad \subseteq \quad \underbrace{\mathcal{C} \otimes \cdots \otimes \mathcal{C}}_{m \text{ fois}}$$

Quelques propriétés du relèvement

$$\mathcal{L} = \mathrm{Lift}_m(\mathcal{C}) = \{g : \mathbb{F}_q^m \to \mathbb{F}_q, \forall \text{ droite affine } \ell, g_{|\ell} \in \mathcal{C}\}$$

- La construction nécessite $\mathcal{C}$ affine-invariant.
- **Exemple :** Soit $k < q$ et $\mathcal{C} = \mathrm{RS}_q(k)$. On a :

$$\mathrm{RM}_q(m,k) \quad \subseteq \quad \mathrm{Lift}_m(\mathcal{C}) \quad \subseteq \quad \underbrace{\mathcal{C} \otimes \cdots \otimes \mathcal{C}}_{m \text{ fois}}$$

Si $\mathbf{q = 2^r}$, $\mathbf{k = q - 1}$ (code de parité) et $\mathbf{m = 2}$ (relèvement plan) :

$$\forall i,j \in [0, q-1], \quad X^i Y^j \in \mathrm{Lift}_2(\mathcal{C}) \iff i \oplus j \neq q-1$$

$$\dim \mathrm{Lift}_2(\mathcal{C}) = 4^r - 3^r \quad \Rightarrow \quad R = 1 - \left(\frac{3}{4}\right)^r$$

Propriétés locales

$$\mathcal{L} = \mathrm{Lift}_m(\mathcal{C}) = \{g : \mathbb{F}_q^m \to \mathbb{F}_q, \forall \text{ droite affine } \ell, g_{|\ell} \in \mathcal{C}\}$$

Soit $\delta = \frac{d_{\min}(\mathcal{C})}{n(\mathcal{C})}$.

$\mathcal{L}$ est **localement corrigible** : il existe un algorithme probabiliste $\mathcal{D}^y(x)$ t.q. :

- $\mathcal{D}$ fait q requêtes sur $y : \mathbb{F}_q^m \to \mathbb{F}_q$
- pour $c \in \mathcal{L}$, si $\mathrm{d}(y, c) < \frac{\delta}{3} q^m$, alors $\forall x \in \mathbb{F}_q^m, \mathbb{P}\left[\mathcal{D}^y(x) = c_i\right] \geq \frac{2}{3}$

LiftPoR : Initialisation et vérification

- Initialisation :

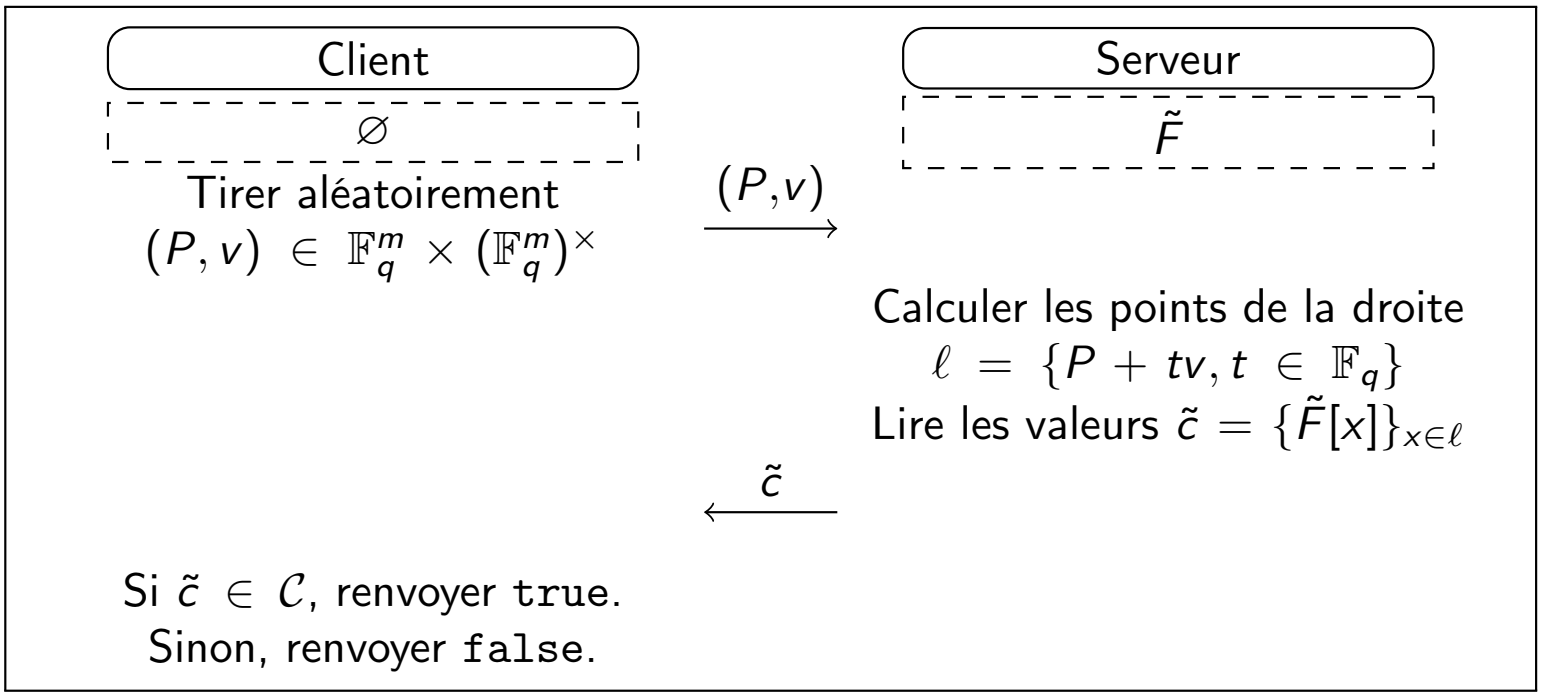
$$\textbf{Init}(F) = \begin{cases} \texttt{data}_F & = \varnothing \\ \tilde{F} & = \mathrm{Enc}_{\mathcal{L}}(F) \end{cases}$$

LiftPoR : Initialisation et vérification

- Initialisation :

$$\textbf{Init}(F) = \begin{cases} \texttt{data}_F & = \varnothing \\ \tilde{F} & = \mathrm{Enc}_{\mathcal{L}}(F) \end{cases}$$

- Vérification :

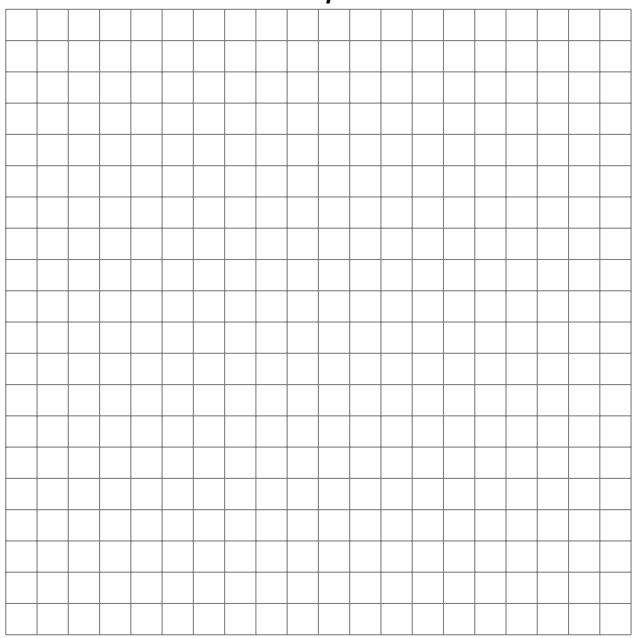


LiftPoR : Extraction

Entrée : $\varnothing$
Sortie : Le fichier $\tilde{F}$

1. Initialisation
$\texttt{Tab} \leftarrow [\perp]_{\mathbb{F}_q^m}$ % $\perp$ *désigne un effacement*
$N_e = q^m$ % *le nombre d'effacements restantes dans* Tab

$\texttt{Tab} : \mathbb{F}_q^m \rightarrow \mathbb{F}_q$



LiftPoR : Extraction

Entrée : $\varnothing$
Sortie : Le fichier $\tilde{F}$

1. Initialisation

$\texttt{Tab} \leftarrow [\perp]_{\mathbb{F}_q^m}$ % $\perp$ *désigne un effacement*

$N_e = q^m$ % *le nombre d'effacements restantes dans* Tab

2. Interaction client-serveur

Tant que $N_e > \texttt{bound}$ **faire**

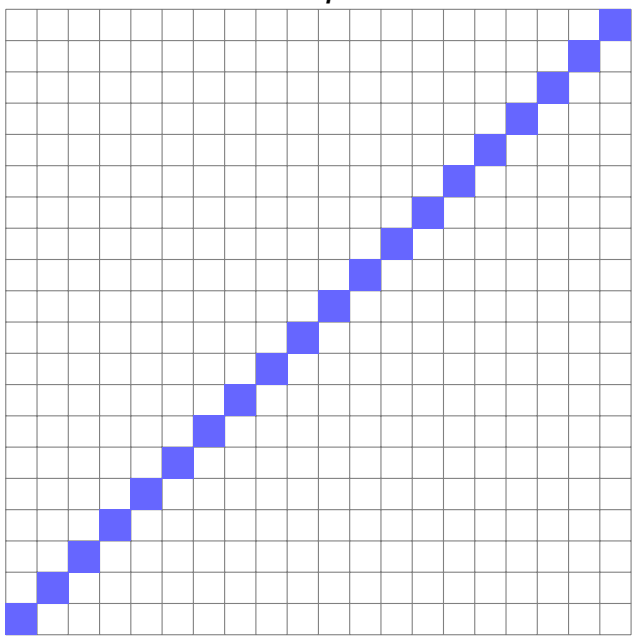
  Tirer une *nouvelle* droite $\ell \in \mathbb{F}_q^m$ et effectuer la vérification associée à ℓ. On note $\tilde{c}$ la réponse du serveur.

  Si $\textbf{Verif}_{\texttt{data}_F}(\ell, \tilde{c}) = \texttt{true}$ **alors**

    Avec $\tilde{c}$, mettre à jour les valeurs $\texttt{Tab}[x], x \in \ell$.

    Décroître N_e du nombre d'effacements éliminés dans Tab.

$\texttt{Tab} : \mathbb{F}_q^m \rightarrow \mathbb{F}_q$



LiftPoR : Extraction

Entrée : $\varnothing$
Sortie : Le fichier $\tilde{F}$

1. Initialisation

$\texttt{Tab} \leftarrow [\perp]_{\mathbb{F}_q^m}$ % $\perp$ *désigne un effacement*

$N_e = q^m$ % *le nombre d'effacements restantes dans* Tab

2. Interaction client-serveur

Tant que $N_e > \texttt{bound}$ **faire**

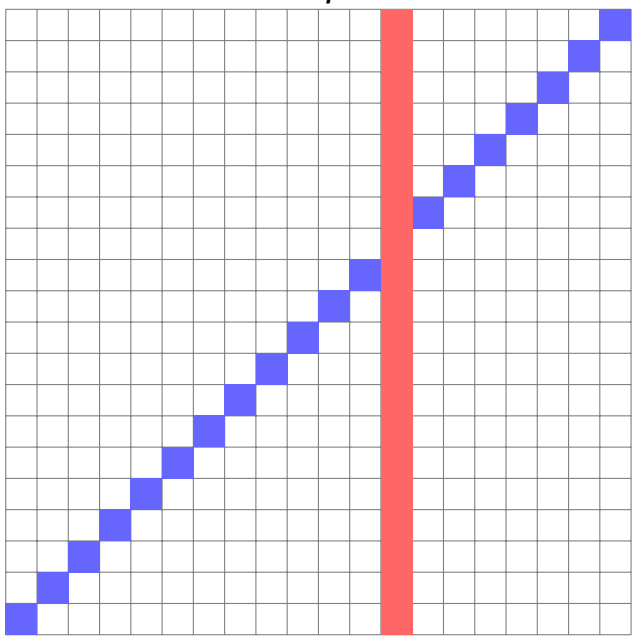
  Tirer une *nouvelle* droite $\ell \in \mathbb{F}_q^m$ et effectuer la vérification associée à ℓ. On note $\tilde{c}$ la réponse du serveur.

  Si $\textbf{Verif}_{\texttt{data}_F}(\ell, \tilde{c}) = \texttt{true}$ **alors**

    Avec $\tilde{c}$, mettre à jour les valeurs $\texttt{Tab}[x], x \in \ell$.

    Décroître N_e du nombre d'effacements éliminés dans Tab.

$\texttt{Tab} : \mathbb{F}_q^m \rightarrow \mathbb{F}_q$



LiftPoR : Extraction

Entrée : $\varnothing$
Sortie : Le fichier $\tilde{F}$

1. Initialisation

$\texttt{Tab} \leftarrow [\perp]_{\mathbb{F}_q^m}$ % $\perp$ *désigne un effacement*

$N_e = q^m$ % *le nombre d'effacements restantes dans* Tab

2. Interaction client-serveur

Tant que $N_e > \texttt{bound}$ **faire**

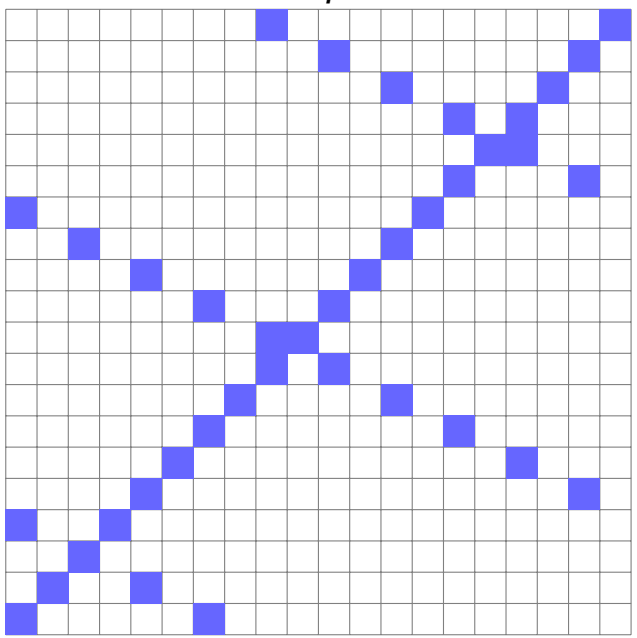
  Tirer une *nouvelle* droite $\ell \in \mathbb{F}_q^m$ et effectuer la vérification associée à ℓ. On note $\tilde{c}$ la réponse du serveur.

  Si $\textbf{Verif}_{\texttt{data}_F}(\ell, \tilde{c}) = \texttt{true}$ **alors**

    Avec $\tilde{c}$, mettre à jour les valeurs $\texttt{Tab}[x], x \in \ell$.

    Décroître N_e du nombre d'effacements éliminés dans Tab.

$\texttt{Tab} : \mathbb{F}_q^m \rightarrow \mathbb{F}_q$



LiftPoR : Extraction

Entrée : $\varnothing$
Sortie : Le fichier $\tilde{F}$

1. Initialisation

$\texttt{Tab} \leftarrow [\perp]_{\mathbb{F}_q^m}$ % $\perp$ *désigne un effacement*

$N_e = q^m$ % *le nombre d'effacements restantes dans* Tab

2. Interaction client-serveur

Tant que $N_e > \texttt{bound}$ **faire**

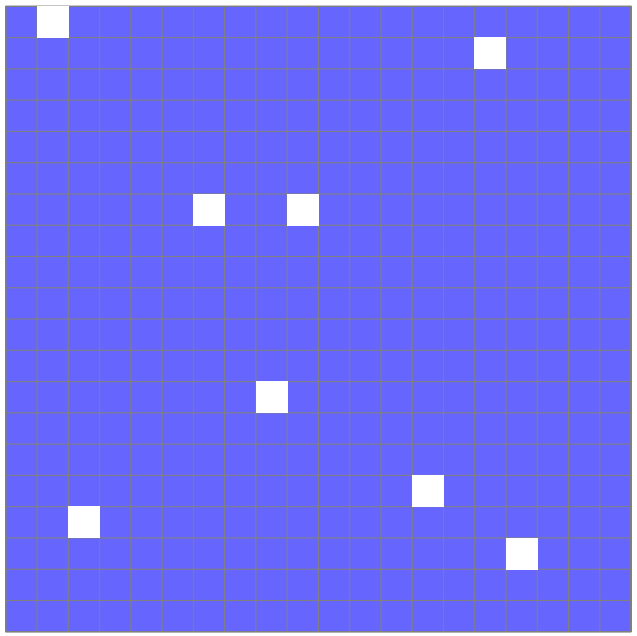
Tirer une *nouvelle* droite $\ell \in \mathbb{F}_q^m$ et effectuer la vérification associée à ℓ. On note $\tilde{c}$ la réponse du serveur.

Si $\textbf{Verif}_{\texttt{data}_F}(\ell, \tilde{c}) = \texttt{true}$ **alors**

Avec $\tilde{c}$, mettre à jour les valeurs $\texttt{Tab}[x], x \in \ell$.

Décroître N_e du nombre d'effacements éliminés dans Tab.

$\texttt{Tab} : \mathbb{F}_q^m \rightarrow \mathbb{F}_q$



LiftPoR : Extraction

Entrée : $\varnothing$
Sortie : Le fichier $\tilde{F}$

1. Initialisation

$\texttt{Tab} \leftarrow [\perp]_{\mathbb{F}_q^m}$ % $\perp$ *désigne un effacement*

$N_e = q^m$ % *le nombre d'effacements restantes dans* Tab

2. Interaction client-serveur

Tant que $N_e > \texttt{bound}$ **faire**

- Tirer une *nouvelle* droite $\ell \in \mathbb{F}_q^m$ et effectuer la vérification associée à ℓ. On note $\tilde{c}$ la réponse du serveur.
- **Si** $\textbf{Verif}_{\texttt{data}_F}(\ell, \tilde{c}) = \texttt{true}$ **alors**
  - Avec $\tilde{c}$, mettre à jour les valeurs $\texttt{Tab}[x], x \in \ell$.
  - Décroître N_e du nombre d'effacements éliminés dans Tab.

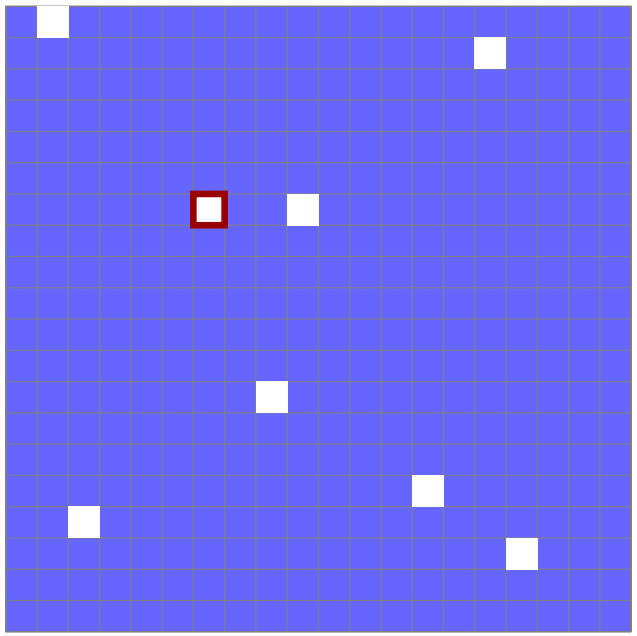
3. Décodage

Pour tous $x \in \mathbb{F}_q^m$ *tels que* $\texttt{Tab}[x] = \perp$ **faire**

- Trouver une droite ℓ_x passant par x telle que $\#\{y \in \ell_x, \texttt{Tab}[y] = \perp\} < d$
- Décoder le mot $\texttt{Tab}_{|\ell_x}$ avec l'algorithme de décodage du code $\mathcal{C}$, puis mettre à jour Tab avec le résultat.

Renvoyer Tab.

$\texttt{Tab} : \mathbb{F}_q^m \rightarrow \mathbb{F}_q$



LiftPoR : Extraction

Entrée : $\varnothing$
Sortie : Le fichier $\tilde{F}$

1. Initialisation

$\texttt{Tab} \leftarrow [\perp]_{\mathbb{F}_q^m}$ % $\perp$ *désigne un effacement*

$N_e = q^m$ % *le nombre d'effacements restantes dans* Tab

2. Interaction client-serveur

Tant que $N_e > \texttt{bound}$ **faire**

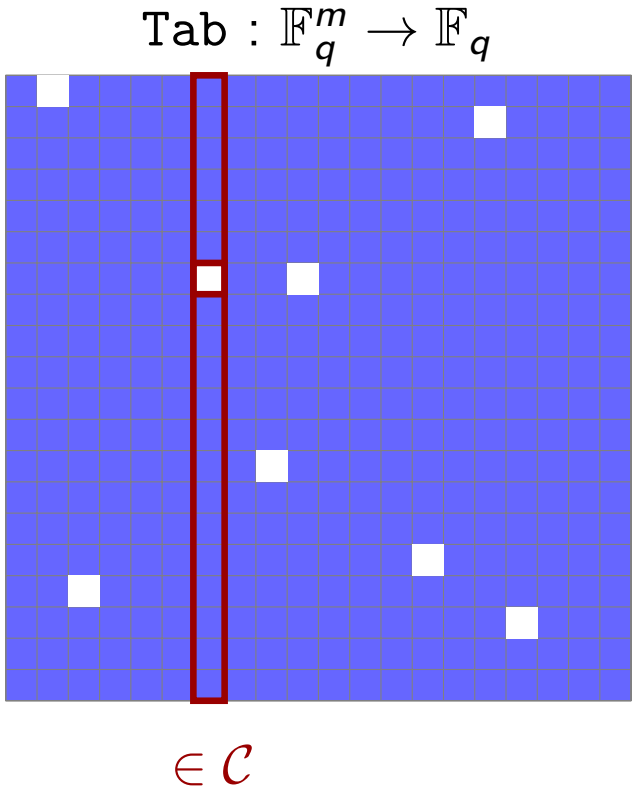
- Tirer une *nouvelle* droite $\ell \in \mathbb{F}_q^m$ et effectuer la vérification associée à ℓ. On note $\tilde{c}$ la réponse du serveur.
- **Si** $\textbf{Verif}_{\texttt{data}_F}(\ell, \tilde{c}) = \texttt{true}$ **alors**
  - Avec $\tilde{c}$, mettre à jour les valeurs $\texttt{Tab}[x], x \in \ell$.
  - Décroître N_e du nombre d'effacements éliminés dans Tab.

3. Décodage

Pour tous $x \in \mathbb{F}_q^m$ *tels que* $\texttt{Tab}[x] = \perp$ **faire**

- Trouver une droite ℓ_x passant par x telle que $\#\{y \in \ell_x, \texttt{Tab}[y] = \perp\} < d$
- Décoder le mot $\texttt{Tab}_{|\ell_x}$ avec l'algorithme de décodage du code $\mathcal{C}$, puis mettre à jour Tab avec le résultat.

Renvoyer Tab.



LiftPoR : Extraction

Entrée : $\varnothing$
Sortie : Le fichier $\tilde{F}$

1. Initialisation

$\texttt{Tab} \leftarrow [\perp]_{\mathbb{F}_q^m}$ % $\perp$ *désigne un effacement*

$N_e = q^m$ % *le nombre d'effacements restantes dans* Tab

2. Interaction client-serveur

Tant que $N_e > \texttt{bound}$ **faire**

- Tirer une *nouvelle* droite $\ell \in \mathbb{F}_q^m$ et effectuer la vérification associée à ℓ. On note $\tilde{c}$ la réponse du serveur.
- **Si** $\textbf{Verif}_{\texttt{data}_F}(\ell, \tilde{c}) = \texttt{true}$ **alors**
  - Avec $\tilde{c}$, mettre à jour les valeurs $\texttt{Tab}[x], x \in \ell$.
  - Décroître N_e du nombre d'effacements éliminés dans Tab.

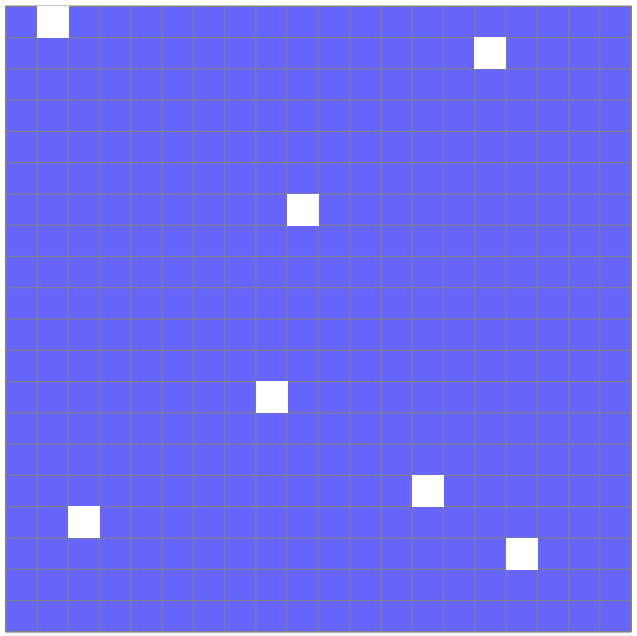
3. Décodage

Pour tous $x \in \mathbb{F}_q^m$ *tels que* $\texttt{Tab}[x] = \perp$ **faire**

- Trouver une droite ℓ_x passant par x telle que $\#\{y \in \ell_x, \texttt{Tab}[y] = \perp\} < d$
- Décoder le mot $\texttt{Tab}_{|\ell_x}$ avec l'algorithme de décodage du code $\mathcal{C}$, puis mettre à jour Tab avec le résultat.

Renvoyer Tab.

$\texttt{Tab} : \mathbb{F}_q^m \rightarrow \mathbb{F}_q$



LiftPoR : Extraction

Entrée : $\varnothing$
Sortie : Le fichier $\tilde{F}$

1. Initialisation

$\texttt{Tab} \leftarrow [\perp]_{\mathbb{F}_q^m}$ % $\perp$ *désigne un effacement*

$N_e = q^m$ % *le nombre d'effacements restantes dans* Tab

2. Interaction client-serveur

Tant que $N_e > \texttt{bound}$ **faire**

- Tirer une *nouvelle* droite $\ell \in \mathbb{F}_q^m$ et effectuer la vérification associée à ℓ. On note $\tilde{c}$ la réponse du serveur.
- **Si** $\textbf{Verif}_{\texttt{data}_F}(\ell, \tilde{c}) = \texttt{true}$ **alors**
  - Avec $\tilde{c}$, mettre à jour les valeurs $\texttt{Tab}[x], x \in \ell$.
  - Décroître N_e du nombre d'effacements éliminés dans Tab.

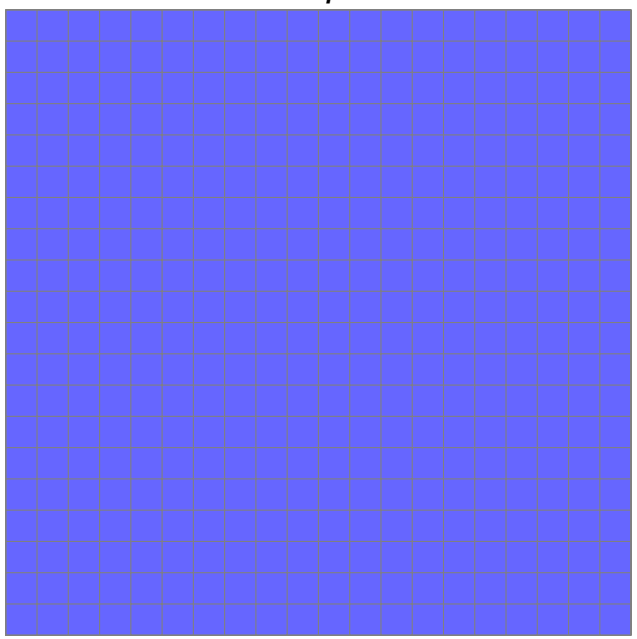
3. Décodage

Pour tous $x \in \mathbb{F}_q^m$ *tels que* $\texttt{Tab}[x] = \perp$ **faire**

- Trouver une droite ℓ_x passant par x telle que $\#\{y \in \ell_x, \texttt{Tab}[y] = \perp\} < d$
- Décoder le mot $\texttt{Tab}_{|\ell_x}$ avec l'algorithme de décodage du code $\mathcal{C}$, puis mettre à jour Tab avec le résultat.

Renvoyer Tab.

$\texttt{Tab} : \mathbb{F}_q^m \rightarrow \mathbb{F}_q$



Analyse de sécurité

$\#\{$ droites affines $\subseteq \mathbb{F}_q^m\} = \Lambda = q^{m-1}\frac{q^m-1}{q-1}$

Lemme. La réunion de $\Lambda/2$ droites affines distinctes couvre au moins une fraction $1 - 1/q$ de $\mathbb{F}_q^m$.

Proposition. Si $d_{\min}(\mathcal{C}) \geq 2$, on peut décoder une fraction $1/q$ d'effacements sur un mot de $\mathbf{Lift}_m(\mathcal{C})$.

Analyse de sécurité

$\#\{$ droites affines $\subseteq \mathbb{F}_q^m\} = \Lambda = q^{m-1}\frac{q^m-1}{q-1}$

Lemme. La réunion de $\Lambda/2$ droites affines distinctes couvre au moins une fraction $1 - 1/q$ de $\mathbb{F}_q^m$.

Proposition. Si $d_{\min}(\mathcal{C}) \geq 2$, on peut décoder une fraction $1/q$ d'effacements sur un mot de $\mathbf{Lift}_m(\mathcal{C})$.

Soit $\mathcal{P}$ une **LiftPoR**, avec $m \geq 2$ et $d_{\min}(\mathcal{C}) \geq 2$. Alors, pour tout $\varepsilon < 1/2$, $\mathcal{P}$ est $(\varepsilon, 1)$-sûre.

Analyse de sécurité

$\#\{$ droites affines $\subseteq \mathbb{F}_q^m\} = \Lambda = q^{m-1}\frac{q^m-1}{q-1}$

Lemme. La réunion de $\Lambda/2$ droites affines distinctes couvre au moins une fraction $1 - 1/q$ de $\mathbb{F}_q^m$.

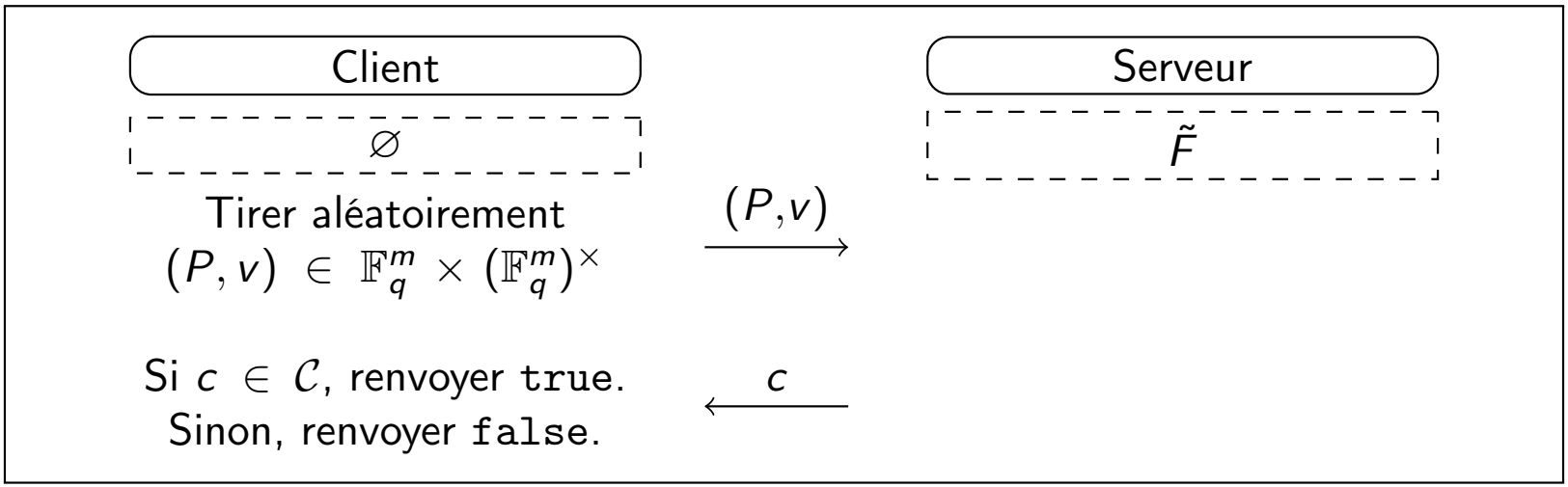
Proposition. Si $d_{\min}(\mathcal{C}) \geq 2$, on peut décoder une fraction $1/q$ d'effacements sur un mot de $\textbf{Lift}_m(\mathcal{C})$.

Soit $\mathcal{P}$ une **LiftPoR**, avec $m \geq 2$ et $d_{\min}(\mathcal{C}) \geq 2$. Alors, pour tout $\varepsilon < 1/2$, $\mathcal{P}$ est $(\varepsilon, 1)$-sûre.

Vraiment ?

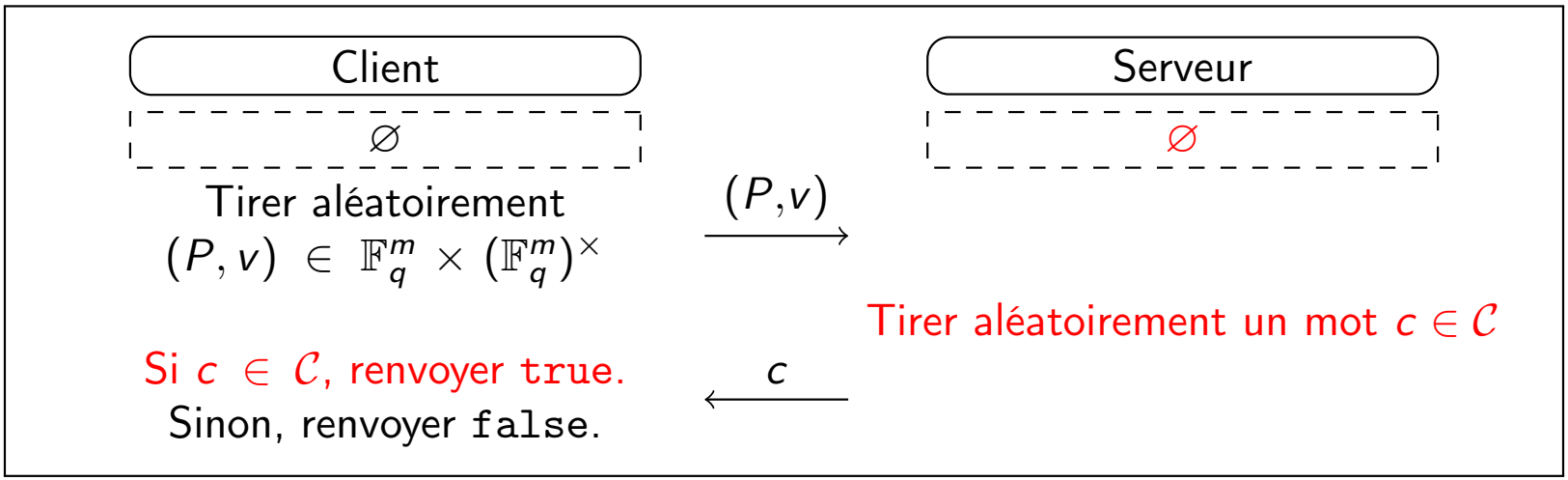
Une attaque

Vérification



Une attaque

Vérification



N'importe quel mot de $\mathcal{C}$ est accepté !

Amélioration

Comment interdire ces **mots parasites** avec grande probabilité ?

Amélioration

Comment interdire ces **mots parasites** avec grande probabilité ?

Idée : Corréler chaque symbole à sa position.

$$\tilde{F}_x = \phi_x(\mathrm{Enc}_{\mathcal{L}}(F)_x) \quad \text{avec } \forall x \in \mathbb{F}_q^m,\ \phi_x \text{ secrète et inversible}$$

Amélioration

Comment interdire ces **mots parasites** avec grande probabilité ?

Idée : Corréler chaque symbole à sa position.

$$\tilde{F}_x = \phi_x(\mathrm{Enc}_{\mathcal{L}}(F)_x) \quad \text{avec } \forall x \in \mathbb{F}_q^m,\ \phi_x \text{ secrète et inversible}$$

En pratique, on peut chiffrer :

- **Suggestion :** Chiffrement par bloc en mode compteur, sur des petits clairs ($\log q$ bits).

LiftPoR : les modifications

- Initialisation, vérification : penser à chiffrer, déchiffrer si besoin.
- Extraction :
  - Ajouter une phase de vote pour amplifier la difficulté du serveur à faire accepter des réponses incorrectes (paramètre $\gamma \in [0,1]$).
  - Conséquences : augmentation de la complexité (légère) et de la mémoire utilisée (forte).

Théorème

Soit $\mathcal{P}$ la **LiftPoR** de paramètres q, m, $d = d_{\min}(\mathcal{C}) \geq 2$ et $\gamma \in [0,1]$. Alors, pour tout $\varepsilon < (1-\gamma)/2$, la preuve $\mathcal{P}$ est $(\varepsilon, 1-2^{-\lambda})$-sûre avec :

$$\lambda = \frac{q^m - 1}{q-1}\Big(\gamma(d-1)\log(q-1) - 1\Big) - m\log q = \tilde{O}(\gamma q^{m-1} d)\,.$$

Grandeurs théoriques et comparaisons

$|F|$: taille du fichier $\quad$ $|\kappa|$: taille de la clé maître

$R = \frac{|F|}{q^m \log q}$: taux du code $\quad$ $R^* = \frac{1}{R} - 1$: redondance

Grandeur	**Quantité exacte**	**Quantité asymptotique** $(\lvert F\rvert \to \infty)$
Stockage (client)	$\lvert\kappa\rvert$	$O(1)$
Stockage (serveur)	$\left(\frac{1}{R} - 1\right)\lvert F\rvert$	$O(\lvert F\rvert)$
Comm. cl $\to$ se.	$2m \log q$	$O(\log \lvert F\rvert)$
Comm. se. $\to$ cl.	$q \log q$	$O(\lvert F\rvert^{1/m})$

Grandeurs théoriques et comparaisons

$|F|$: taille du fichier $\quad$ $|\kappa|$: taille de la clé maître

$R = \frac{|F|}{q^m \log q}$: taux du code $\quad$ $R^* = \frac{1}{R} - 1$: redondance

Grandeur	**Quantité exacte**	**Quantité asymptotique** $(\lvert F\rvert \to \infty)$
Stockage (client)	$\lvert\kappa\rvert$	$O(1)$
Stockage (serveur)	$\left(\frac{1}{R} - 1\right)\lvert F\rvert$	$O(\lvert F\rvert)$
Comm. cl $\to$ se.	$2m \log q$	$O(\log \lvert F\rvert)$
Comm. se. $\to$ cl.	$q \log q$	$O(\lvert F\rvert^{1/m})$

Construction	**Bowers *et. al*** (2010, JK'07 amélioré)	**Shacham-Waters** $(2008 - 13)$	**Notre travail**
Usage illimité	Non (N utilisations)	Oui	Oui
Stockage client	$\lvert\kappa\rvert$	$\lvert F\rvert^{\beta} + \lvert\kappa\rvert$	$\lvert\kappa\rvert$
Stockage serveur	$\alpha N + R^*\lvert F\rvert$	$\frac{\lvert F\rvert^{1-\beta}}{R} + R^*\lvert F\rvert$	$R^*\lvert F\rvert$
Coût de communication	$\lvert\kappa\rvert + \alpha$	$\lvert F\rvert^{\beta} + \frac{\lvert\kappa\rvert}{R^*}$	$\left(\frac{\lvert F\rvert}{R}\right)^{1/m}$
Remarques	$\alpha \simeq 2^8$	$\beta \in]0,1[$	$m \in \{2,3\}$

Instanciation et implantation

Instance		Résultats				
m	q	$\|F\|$ (bits)	$1/R - 1$ (redondance)	comm. (bits) $\mathcal{C} \to \mathcal{S}$	comm. (bits) $\mathcal{S} \to \mathcal{C}$	comm./$\|F\|$
2	64	3367	0.217	24	384	0.121
2	4096	16245775	0.033	48	49152	0.0030
3	512	79837411	0.681	54	4608	5.84×10^{-5}
4	128	49578831	4.414	56	896	1.92×10^{-5}

Implantation ($m = 2, d_{\min}(\mathcal{C}) = 2$) avec SageMath (Python).

- Complexité en temps sous-quadratique pour toutes les phases, mais ...
- ... constantes importantes pour l'extraction : < 10 kbits / s, même sans chiffrement.

$\Rightarrow$ à optimiser (évaluation multi-points, FFT, etc.)

Conclusion

- Construction d'une preuve de récupérabilité à vérification structurelle :
  - faible stockage (surtout client)
  - communication relativement faible
  - implantable, à optimiser
- Perspectives :
  - utilisation d'autres codes localement décodables/testables ?
  - généralisation ?