RUB

RUHR-UNIVERSITÄT BOCHUM

Improved Low-Memory Subset Sum and LPN Algorithms via Multiple Collision

January 2019, Nancy

Claire Delaplace, Andre Esser and Alexander May

About Me

Claire Delaplace: Postdoc researcher

- Ruhr University Bochum, Germany
- Team: Cryptology and IT-Security
- Scientific supervisor: Alexander May

About Me

Claire Delaplace: Postdoc researcher

- Ruhr University Bochum, Germany
- Team: Cryptology and IT-Security
- Scientific supervisor: Alexander May

Before that...

- University of Rennes, IRISA. Team EMSEC
- University of Lille, CRIStAL. Team CFHP
- PhD supervisors: Pierre-Alain Fouque & Charles Bouillaguet
- Thesis: Linear Algebra Algorithm for Cryptography

RUB

Research Topic

Attacking Underlying Cryptographic Problems

Research Topic

Attacking Underlying Cryptographic Problems

- Generalised Birthday Problem ([B**D**F2018] + 2 in submission)
- ECDLP ([**D**M19] + 1 in submission)
- LWE variants ([B**D**FK17,B**D**EFT18])
- Sparse Linear Algebra ([B**D**16,B**D**V17])
- Subset Sum & LPN ([**D**EM19])

Research Topic

Attacking Underlying Cryptographic Problems

- Generalised Birthday Problem ([B**D**F2018] + 2 in submission)
- ECDLP ([**D**M19] + 1 in submission)
- LWE variants ([B**D**FK17,B**D**EFT18])
- Sparse Linear Algebra ([B**D**16,B**D**V17])
- Subset Sum & LPN ([**D**EM19])

Motivations

Post-Quantum Cryptography

- Popular families of schemes: Lattices & Codes based
- Subset-sum & LPN related to Lattices & Codes
- Better algo for subset-sum & LPN $\stackrel{?}{\Rightarrow}$ Better algo for Lattices & Codes

Motivations

Post-Quantum Cryptography

- Popular families of schemes: Lattices & Codes based
- Subset-sum & LPN related to Lattices & Codes
- Better algo for subset-sum & LPN $\overset{?}{\Rightarrow}$ Better algo for Lattices & Codes

Main drawback

HUGE amount of memory these attacks require
$\Rightarrow$ Need for time-memory trade-offs

Motivations

Post-Quantum Cryptography

- Popular families of schemes: Lattices & Codes based
- Subset-sum & LPN related to Lattices & Codes
- Better algo for subset-sum & LPN $\stackrel{?}{\Rightarrow}$ Better algo for Lattices & Codes

Main drawback

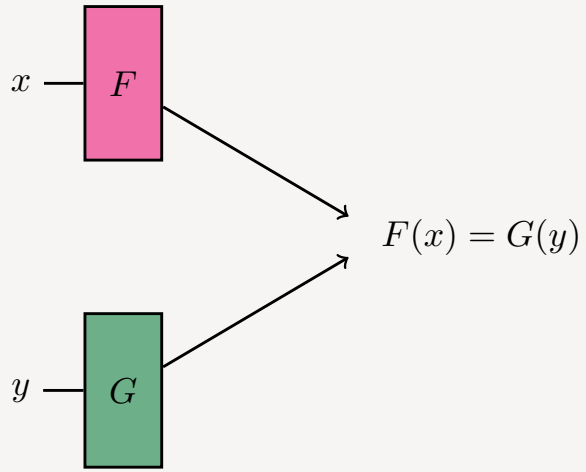
HUGE amount of memory these attacks require
$\Rightarrow$ Need for time-memory trade-offs

This work

New time-memory trade-offs for subset-sum & LPN
Main tool: Parallel Collision Search algorithm [vOW99]

Collisions Search

Given: $F, G : \mathbb{F}_2^n \to \mathbb{F}_2^n$ with uniformly random outputs
Goal: Find $x, y \in \mathbb{F}_2^n$ s.t. $F(x) = G(y)$

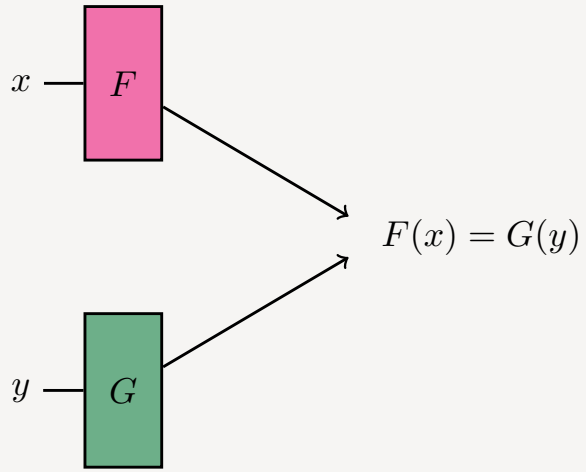


Birthday Paradox

Recovering one collision:
Time: $\mathcal{O}\left(2^{\frac{n}{2}}\right)$

Collisions Search

Given: $F, G : \mathbb{F}_2^n \to \mathbb{F}_2^n$ with uniformly random outputs
Goal: Find $x, y \in \mathbb{F}_2^n$ s.t. $F(x) = G(y)$



Birthday Paradox

Recovering one collision:
Time: $\mathcal{O}\left(2^{\frac{n}{2}}\right)$

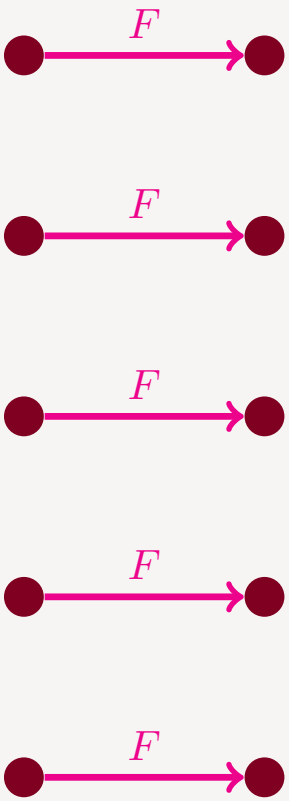
Searching for 2^m collisions

- 2^m Birthday method: Time $\mathcal{O}\left(2^{m+\frac{n}{2}}\right)$
- Parallel Collision Search [vOW99]: Time $\tilde{\mathcal{O}}\left(2^{\frac{m+n}{2}}\right)$

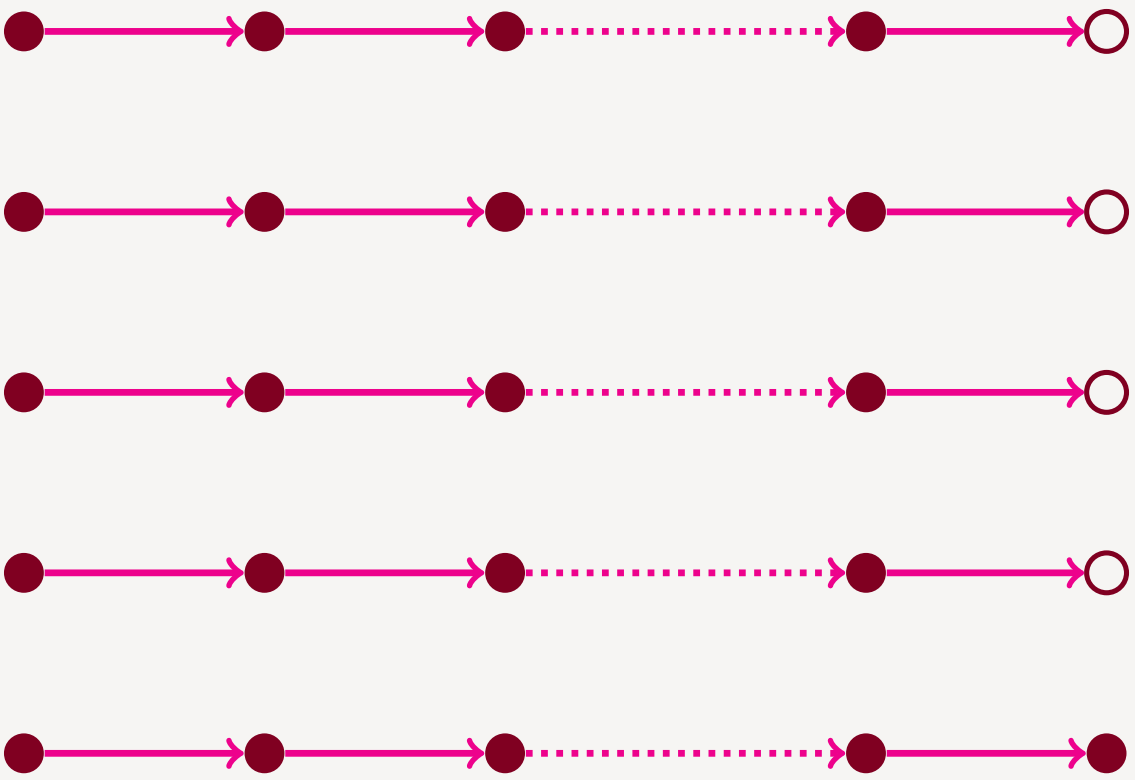
PCS: High level Idea

-
-
-
-
-

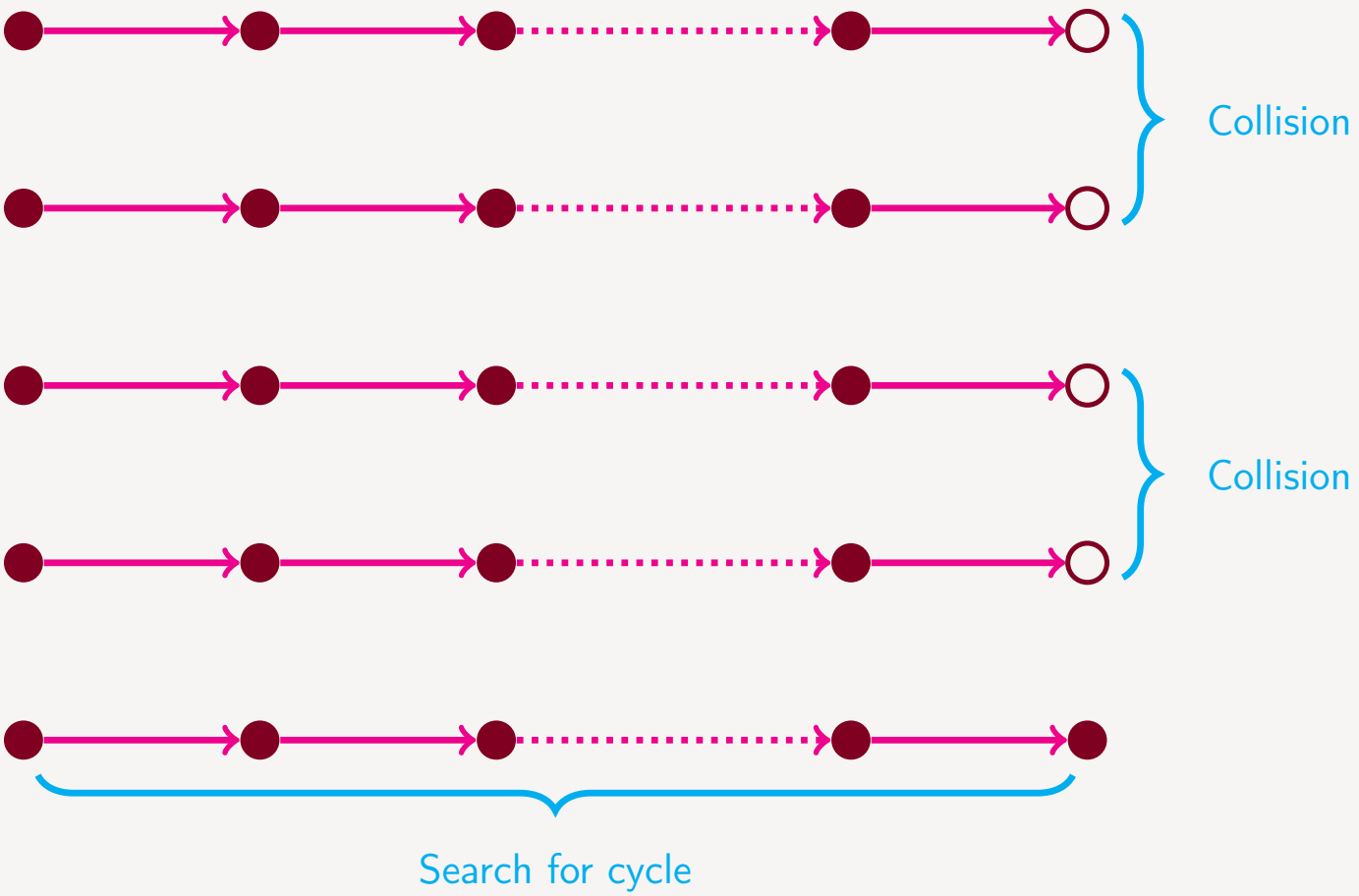
PCS: High level Idea



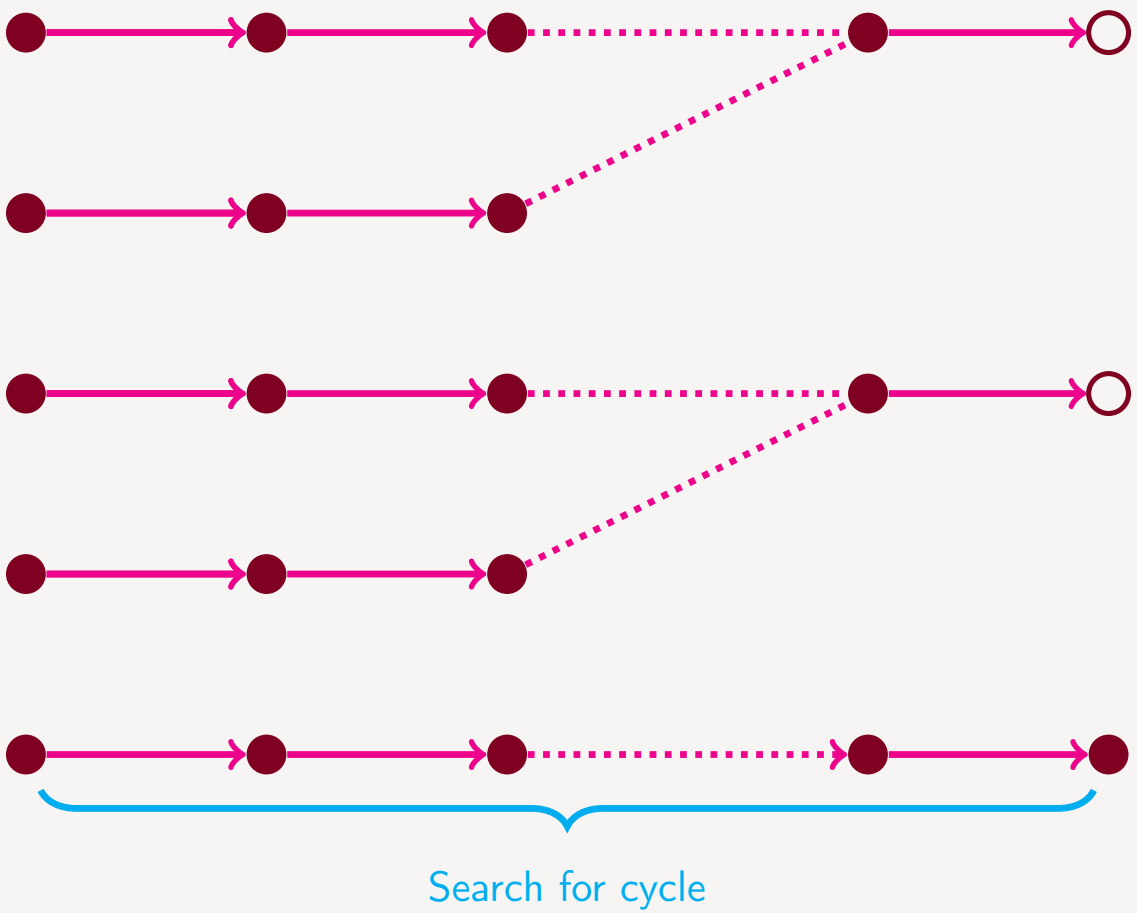
PCS: High level Idea



PCS: High level Idea

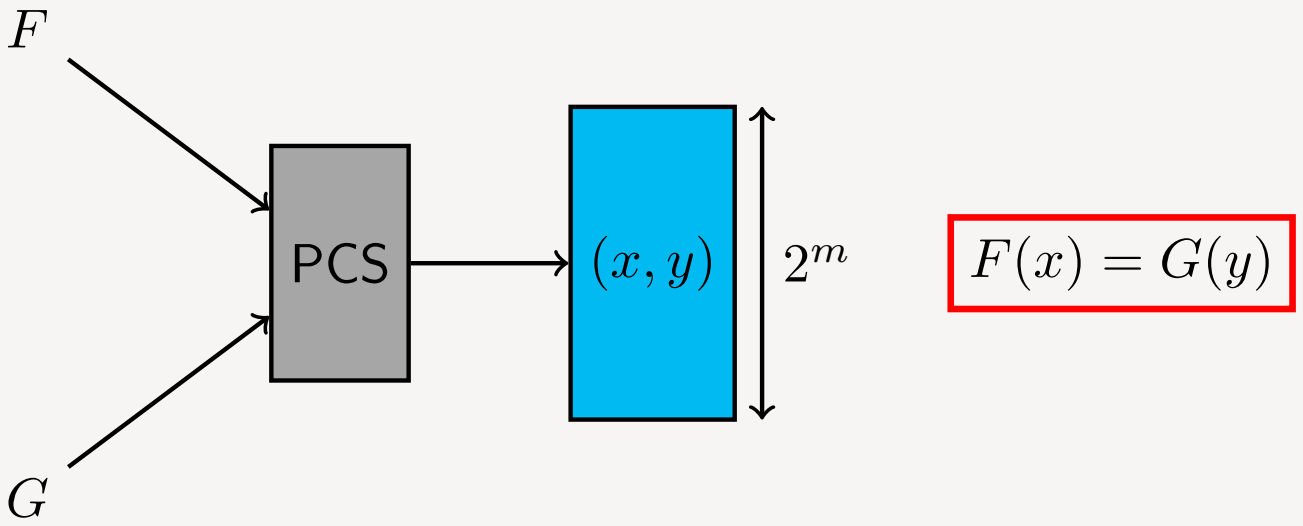


PCS: High level Idea



PCS in a Nutshell

Given: $F, G : \mathbb{F}_2^n \to \mathbb{F}_2^n$ with uniformly random outputs
Goal: Find 2^m $(x, y) \in (\mathbb{F}_2^n)^2$ s.t. $F(x) = G(y)$



$$T = \tilde{\mathcal{O}}\left(2^{\frac{n+m}{2}}\right) \qquad M = \tilde{\mathcal{O}}\left(2^m\right)$$

Random Subset-Sum (RSS) Problem

Definition

- $\mathbf{a} = (a_1 \dots a_n) \in (\mathbb{Z}_{2^n})^n$
- $\mathbf{e} = (e_1 \dots e_n) \in \{0,1\}^n \quad wt(\mathbf{e}) = \frac{n}{2}$ *unknown*
- $t = \langle \mathbf{a}, \mathbf{e} \rangle \mod 2^n$

GOAL: Given $(\mathbf{a}, t)$ find $\mathbf{e} \in \{0,1\}^n$ such that $\langle \mathbf{a}, \mathbf{e} \rangle = t$

Random Subset-Sum (RSS) Problem

Definition

- $\mathbf{a} = (a_1 \dots a_n) \in (\mathbb{Z}_{2^n})^n$
- $\mathbf{e} = (e_1 \dots e_n) \in \{0,1\}^n \quad wt(\mathbf{e}) = \frac{n}{2} \quad$ *unknown*
- $t = \langle \mathbf{a}, \mathbf{e} \rangle \mod 2^n$

GOAL: Given $(\mathbf{a}, t)$ find $\mathbf{e} \in \{0,1\}^n$ such that $\langle \mathbf{a}, \mathbf{e} \rangle = t$

Our Work

Two new algorithms

- SS-PCS Better than previous work for $M < 2^{0.02n}$
- SS-PCS$_4$ Better than previous work for $2^{0.13n} < M < 2^{0.2n}$

Previous Work

- MitM (Folklore algorithm): $T = M = 2^{\frac{n}{2}}$

Previous Work

- MitM (Folklore algorithm): $T = M = 2^{\frac{n}{2}}$
- [SS81] Schroeppel-Shamir 4-list Algorithm: $T = 2^{\frac{n}{2}}, \ M = 2^{\frac{n}{4}}$

Previous Work

- MitM (Folklore algorithm): $T = M = 2^{\frac{n}{2}}$
- [SS81] Schroeppel-Shamir 4-list Algorithm: $T = 2^{\frac{n}{2}}, \ M = 2^{\frac{n}{4}}$
- [H-GJ10] Representation Technique. $T = M \simeq 2^{0.337n}$

Previous Work

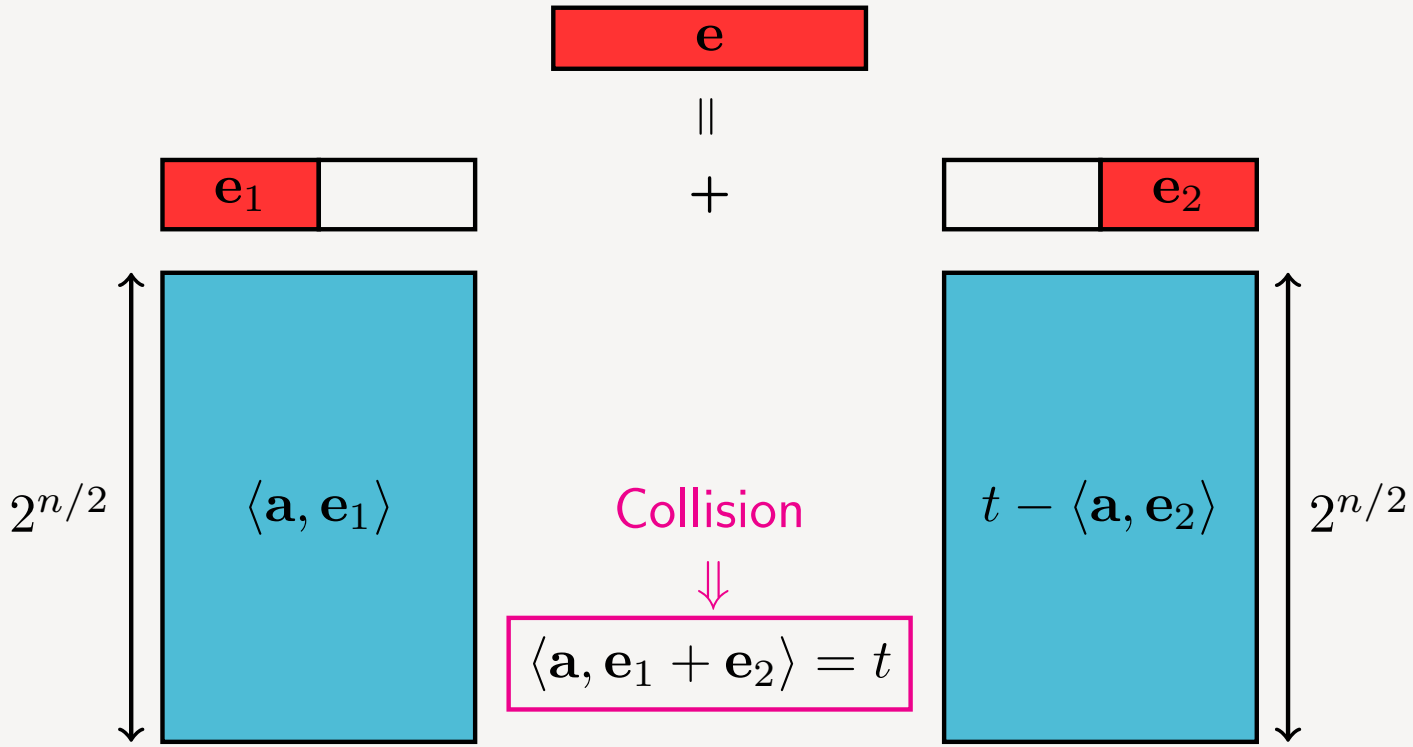
- MitM (Folklore algorithm): $T = M = 2^{\frac{n}{2}}$
- [SS81] Schroeppel-Shamir 4-list Algorithm: $T = 2^{\frac{n}{2}}, \ M = 2^{\frac{n}{4}}$
- [H-GJ10] Representation Technique. $T = M \simeq 2^{0.337n}$
- [BCJ11]
  - Improvement of [H-GJ10]: $T = M \simeq 2^{0.291n}$
  - Memoryless algorithm: $T \simeq 2^{0.71n}$

Previous Work

- MitM (Folklore algorithm): $T = M = 2^{\frac{n}{2}}$
- [SS81] Schroeppel-Shamir 4-list Algorithm: $T = 2^{\frac{n}{2}}, \ M = 2^{\frac{n}{4}}$
- [H-GJ10] Representation Technique. $T = M \simeq 2^{0.337n}$
- [BCJ11]
  - Improvement of [H-GJ10]: $T = M \simeq 2^{0.291n}$
  - Memoryless algorithm: $T \simeq 2^{0.71n}$
- [DDKS12] Best $2^{0.01n} \leq M < 2^{0.17n}$

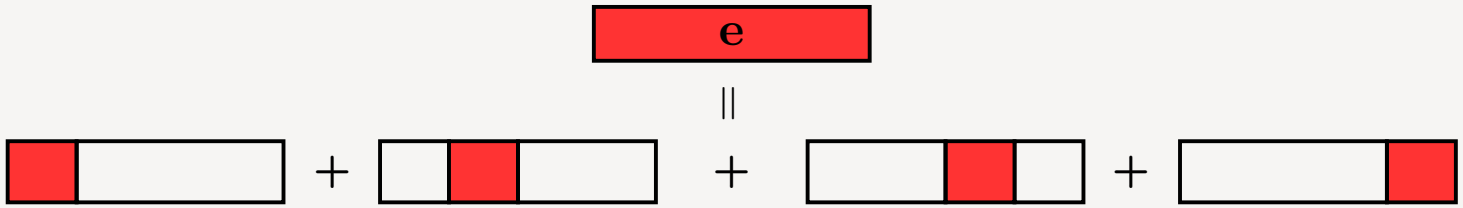
MitM Algorithm

Goal: Find $\mathbf{e}$ s.t. $\langle \mathbf{a}, \mathbf{e} \rangle = t$



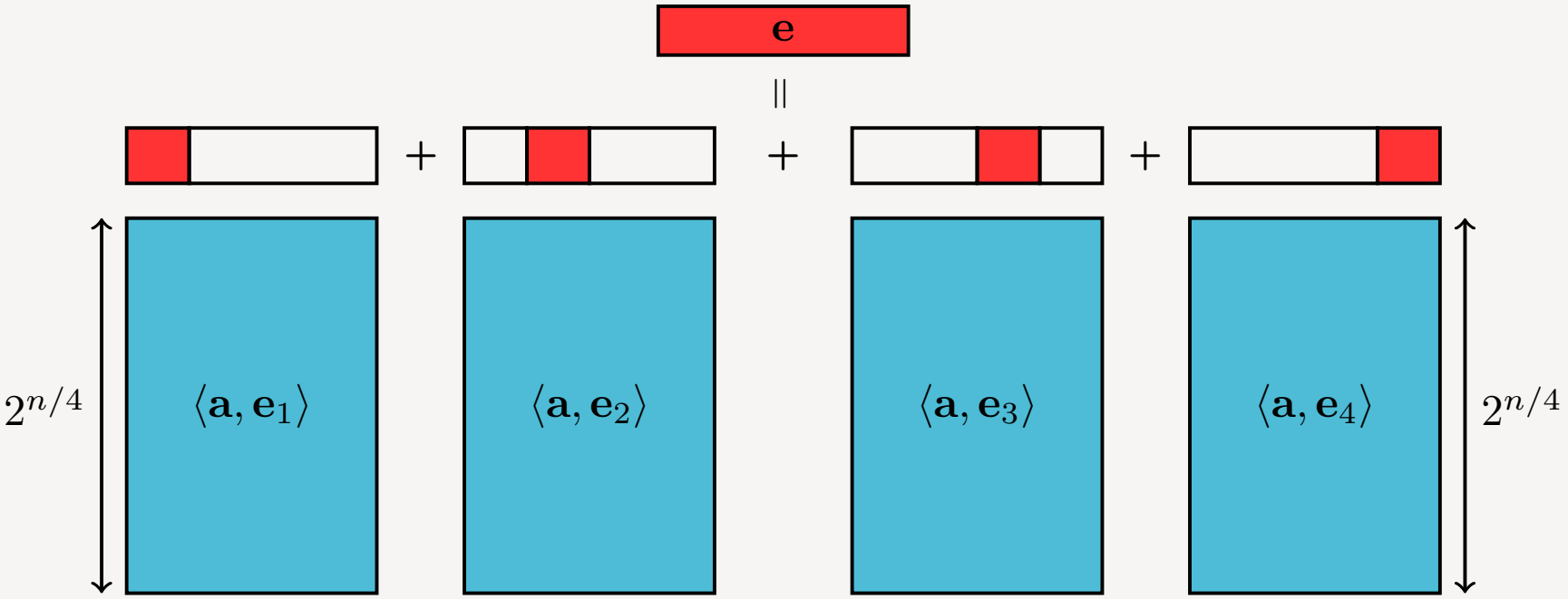
Schroeppel-Shamir [SS81]

Goal: Find $\mathbf{e}$ s.t. $\langle \mathbf{a}, \mathbf{e} \rangle = t$



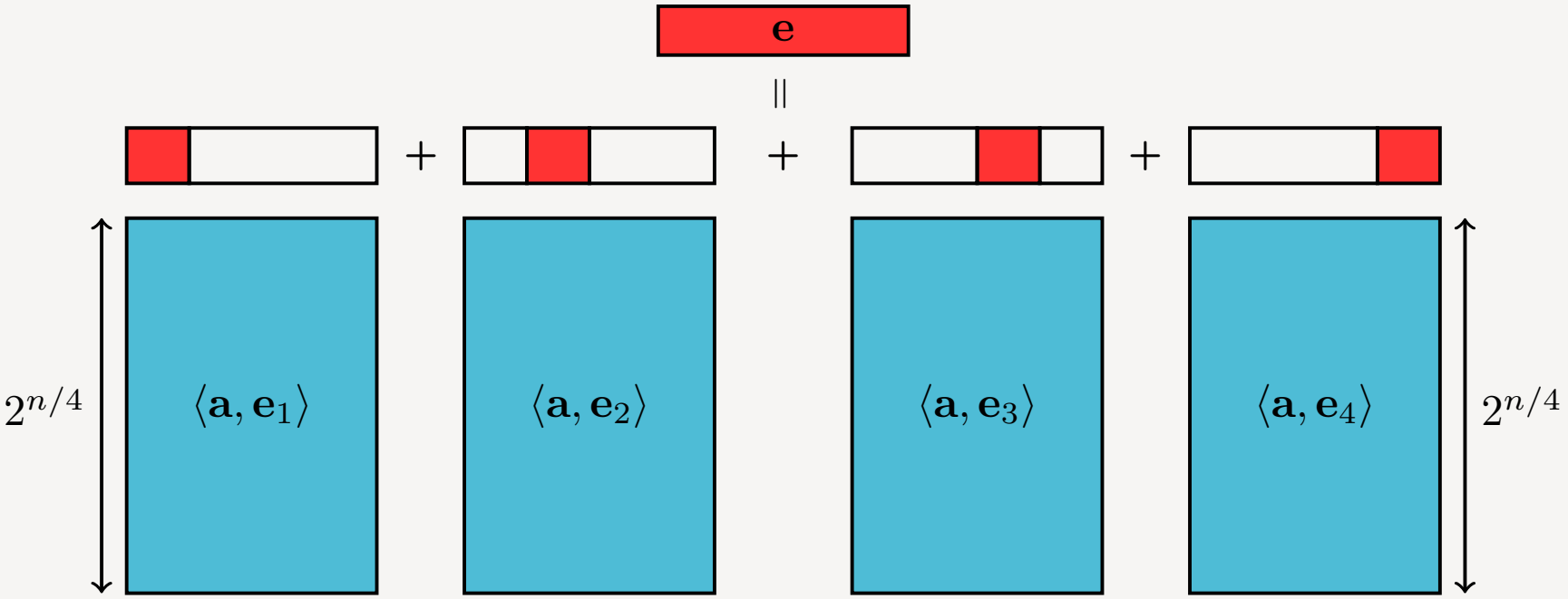
Schroeppel-Shamir [SS81]

Goal: Find $\mathbf{e}$ s.t. $\langle \mathbf{a}, \mathbf{e} \rangle = t$



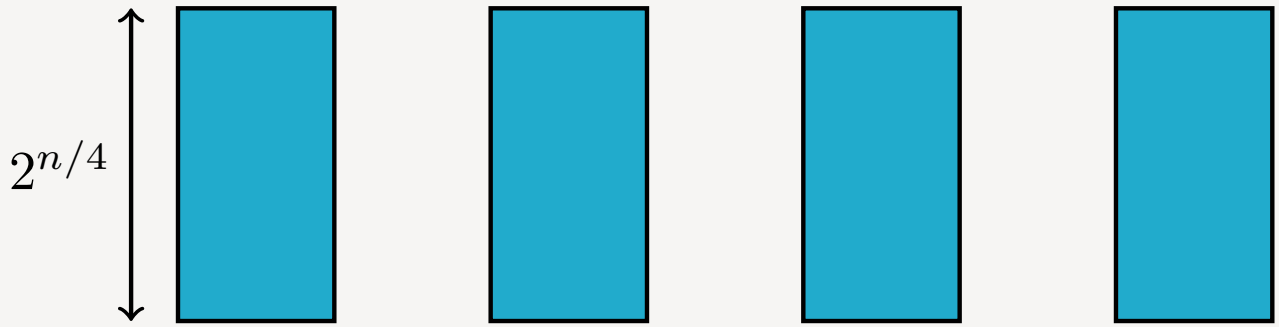
Schroeppel-Shamir [SS81]

Goal: Find $\mathbf{e}$ s.t. $\langle \mathbf{a}, \mathbf{e} \rangle = t$

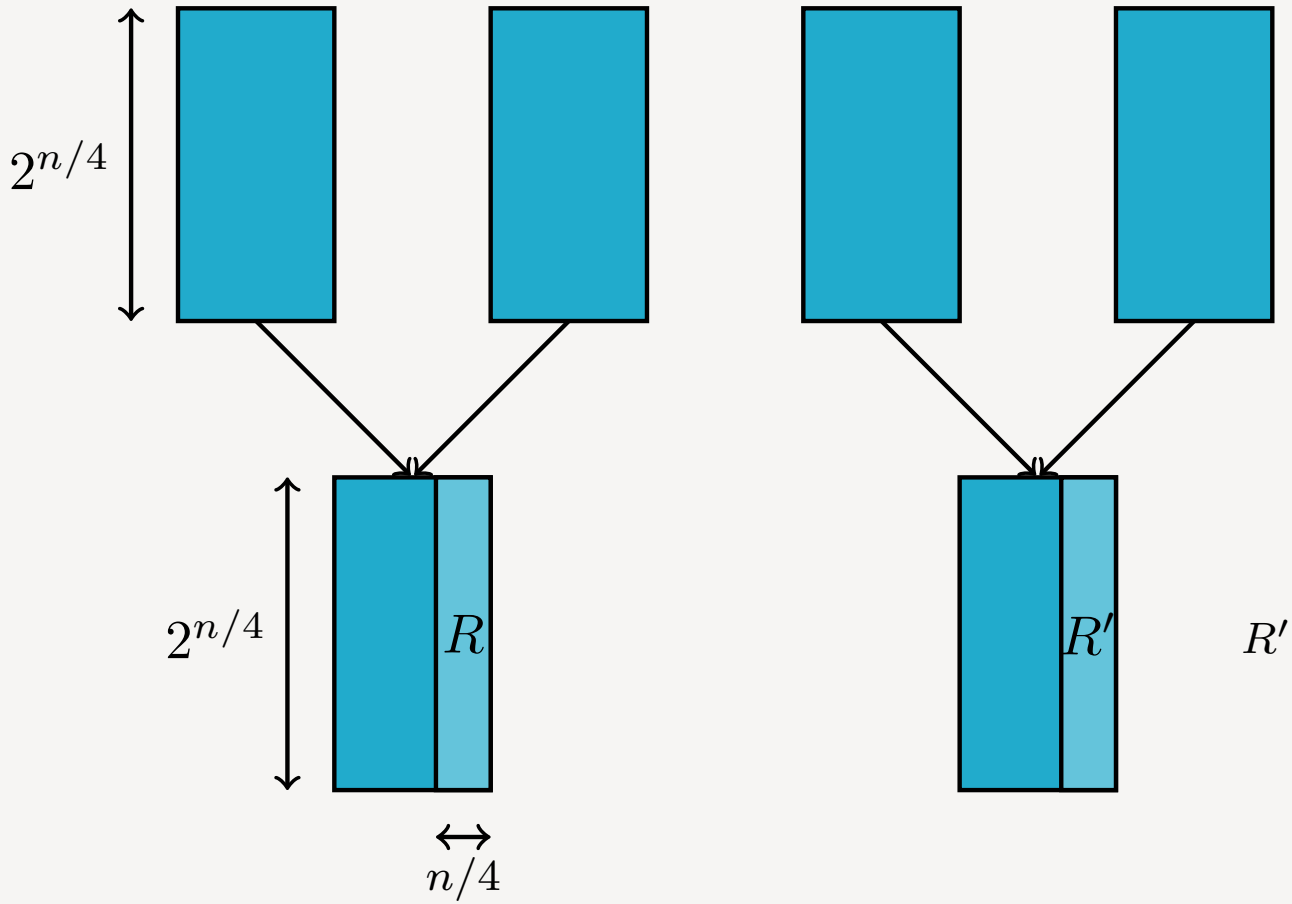


$t_1 \in L_1, \ldots t_4 \in L_4$ s.t. $\sum_i t_i = t$ $\Rightarrow$ $\langle \mathbf{a}, \mathbf{e}_1 + \cdots + \mathbf{e}_4 \rangle = t$

Schroeppel-Shamir 4-list Algorithm

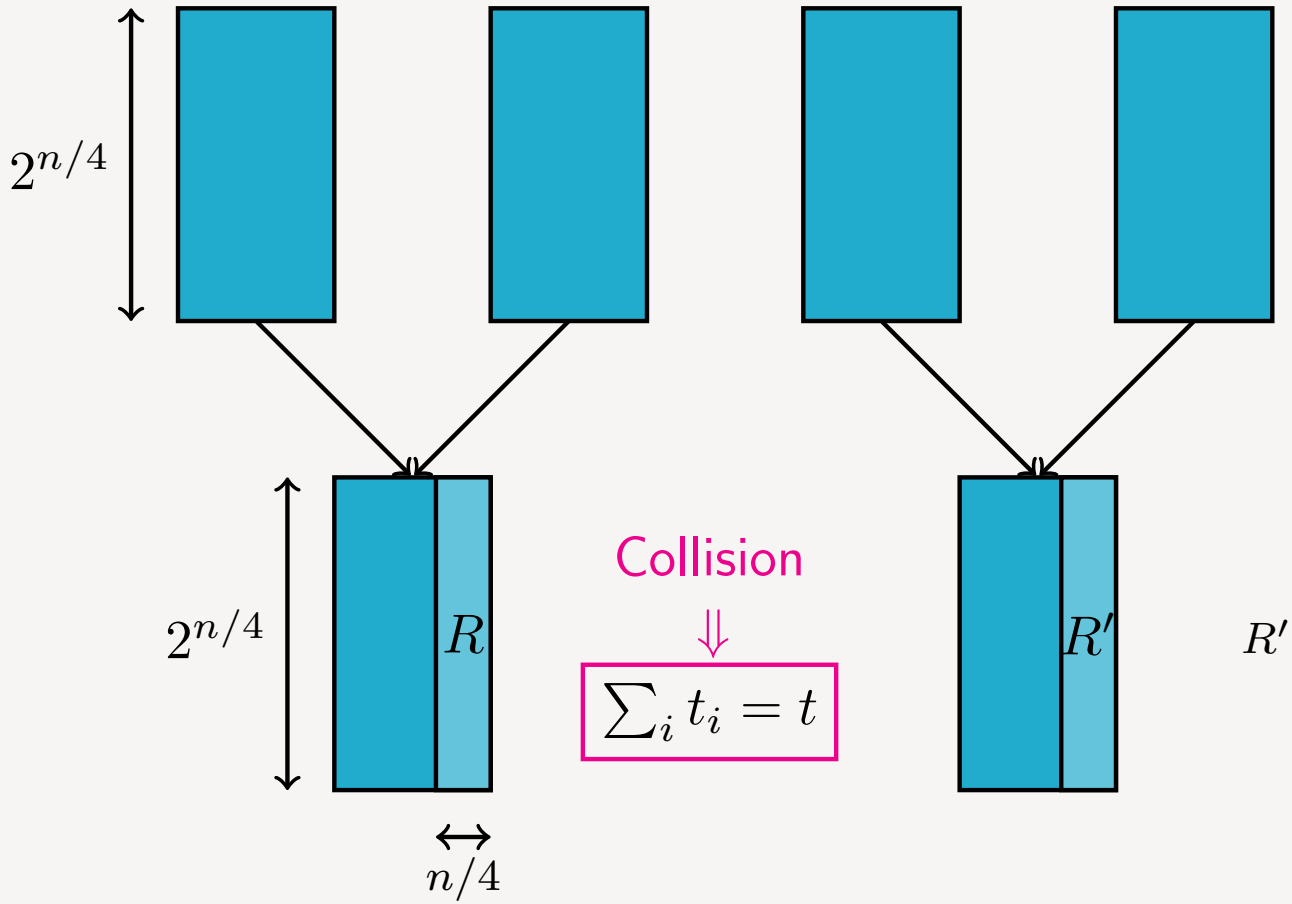


Schroeppel-Shamir 4-list Algorithm

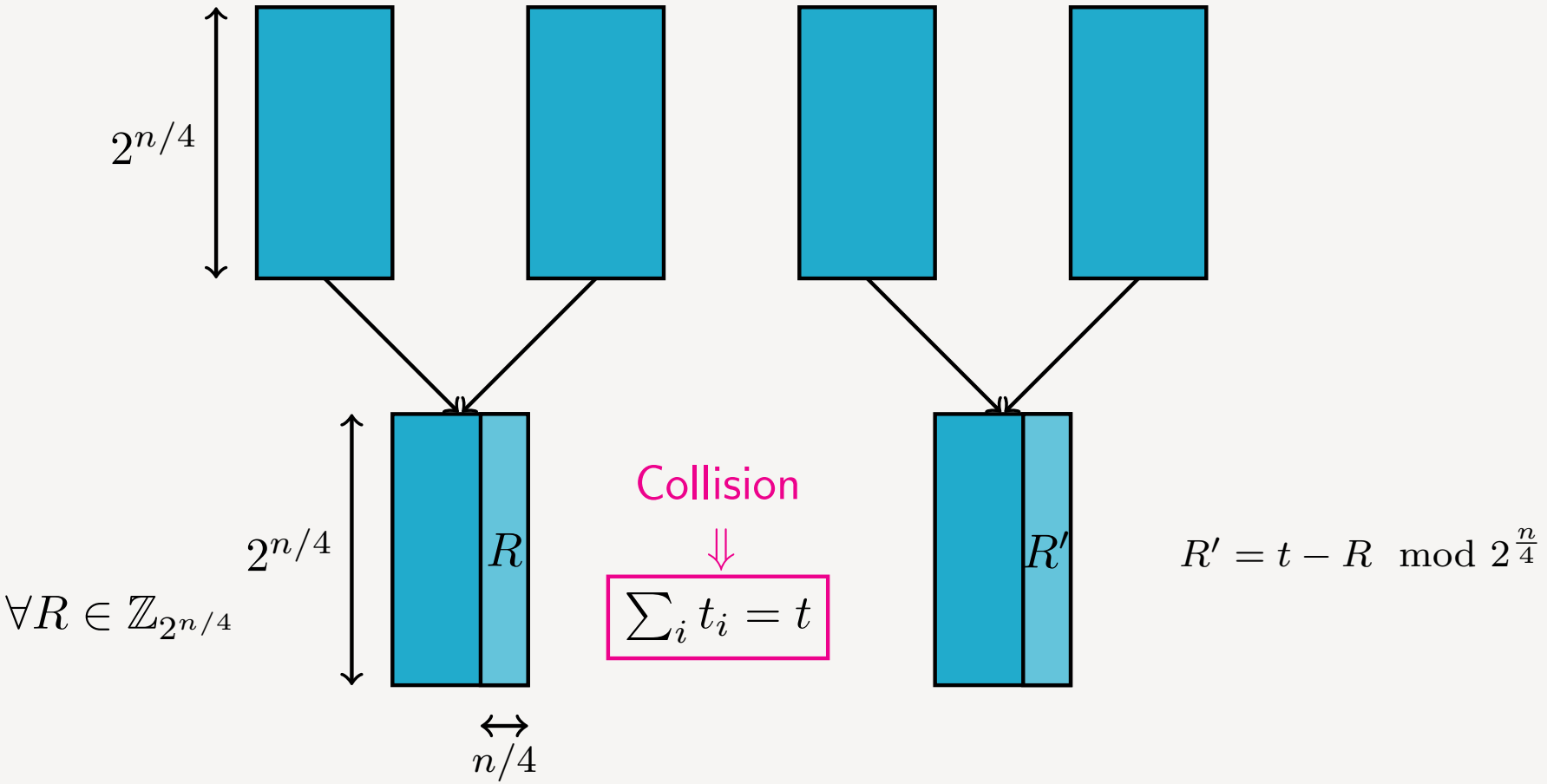


$$R' = t - R \mod 2^{\frac{n}{4}}$$

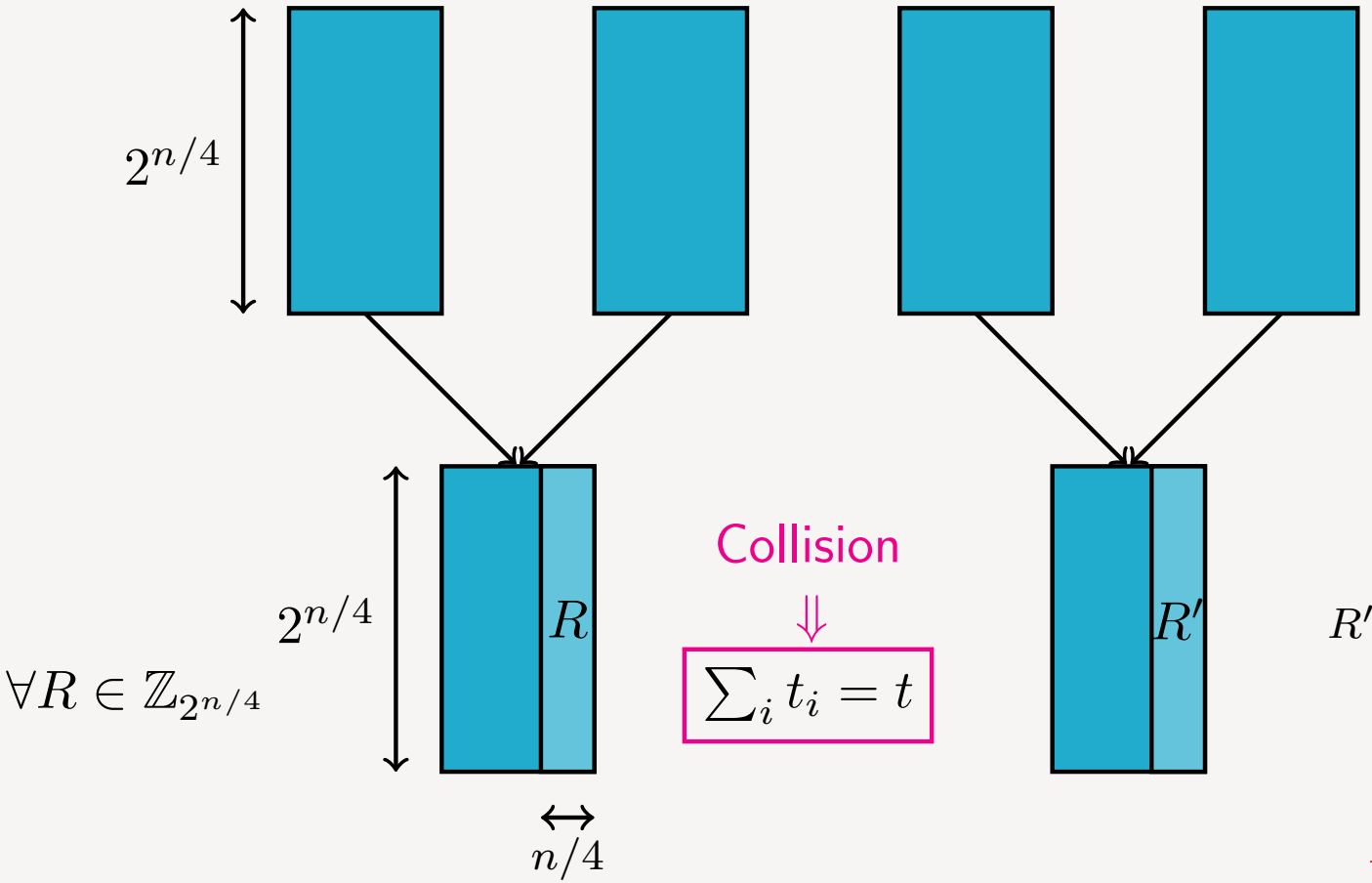
Schroeppel-Shamir 4-list Algorithm



Schroeppel-Shamir 4-list Algorithm



Schroeppel-Shamir 4-list Algorithm



Representations

Representation of $\mathbf{e}$: $(\mathbf{e}_1 \dots \mathbf{e}_k) \quad wt(\mathbf{e}_i) = \frac{n}{2k} \ \forall i \quad \sum \mathbf{e}_i = \mathbf{e}$

Example

$$\mathbf{e}_1 = (10001000) \qquad \mathbf{e}'_1 = (10000001)$$
$$\mathbf{e}_2 = (01000001) \qquad \mathbf{e}'_2 = (01001000)$$

$(\mathbf{e}_1, \mathbf{e}_2)$ and $(\mathbf{e}'_1, \mathbf{e}'_2)$: representations of $\mathbf{e} = (11001001)$

Representations

Representation of $\mathbf{e}$: $(\mathbf{e}_1 \dots \mathbf{e}_k)$ $\quad wt(\mathbf{e}_i) = \frac{n}{2k}\ \forall i \quad \sum \mathbf{e}_i = \mathbf{e}$

Example

$$\mathbf{e}_1 = (10001000) \qquad \mathbf{e}'_1 = (10000001)$$
$$\mathbf{e}_2 = (01000001) \qquad \mathbf{e}'_2 = (01001000)$$

$(\mathbf{e}_1, \mathbf{e}_2)$ and $(\mathbf{e}'_1, \mathbf{e}'_2)$: representations of $\mathbf{e} = (11001001)$

Important remark

$\mathbf{e} \in \{0,1\}^n$, $wt(\mathbf{e}) = n/2$

There are $\binom{n/2}{n/4} \approx 2^{n/2}$ representations $(\mathbf{e}_1, \mathbf{e}_2)$ of $\mathbf{e}$

Representation Technique: Needles and Haystack

Subset-sum

Find $\mathbf{e} \in \{0,1\}^n$ s.t. $\langle \mathbf{a}, \mathbf{e} \rangle = t \mod 2^n$

Representation Technique: Needles and Haystack

Representation Technique [H-GJ10]

Find $(\mathbf{e}_1, \mathbf{e}_2) \in \{0,1\}^n \times \{0,1\}^n$ s.t. $\langle \mathbf{a}, \mathbf{e}_1 + \mathbf{e}_2 \rangle = t \mod 2^n$

Representation Technique: Needles and Haystack

Representation Technique [H-GJ10]

Find $(\mathbf{e}_1, \mathbf{e}_2) \in \{0,1\}^n \times \{0,1\}^n$ s.t. $\langle \mathbf{a}, \mathbf{e}_1 + \mathbf{e}_2 \rangle = t \mod 2^n$

Without Rep.

search space: $\binom{n}{n/2} \approx 2^n$

solutions: 1

With Rep.

search space: $\binom{n}{n/4}^2 \approx 2^{1.623n}$ ☹

solutions: $\binom{n/2}{n/4} \approx 2^{n/2}$ ☺

BCJ Memoryless Algorithm [BCJ11]

- $wt(\mathbf{x}) = \frac{n}{4}$
- $g(\mathbf{x}) = \langle \mathbf{a}, \mathbf{x} \rangle \mod 2^r, \quad 2^r \approx \binom{n}{n/4}$
- $g_t(\mathbf{x}) = t - g(\mathbf{x}) \mod 2^r$

BCJ Memoryless Algorithm [BCJ11]

- $wt(\mathbf{x}) = \frac{n}{4}$
- $g(\mathbf{x}) = \langle \mathbf{a}, \mathbf{x} \rangle \mod 2^r, \quad 2^r \approx \binom{n}{n/4}$
- $g_t(\mathbf{x}) = t - g(\mathbf{x}) \mod 2^r$

$$g(\mathbf{x}) = g_t(\mathbf{y})$$

$$\Updownarrow$$

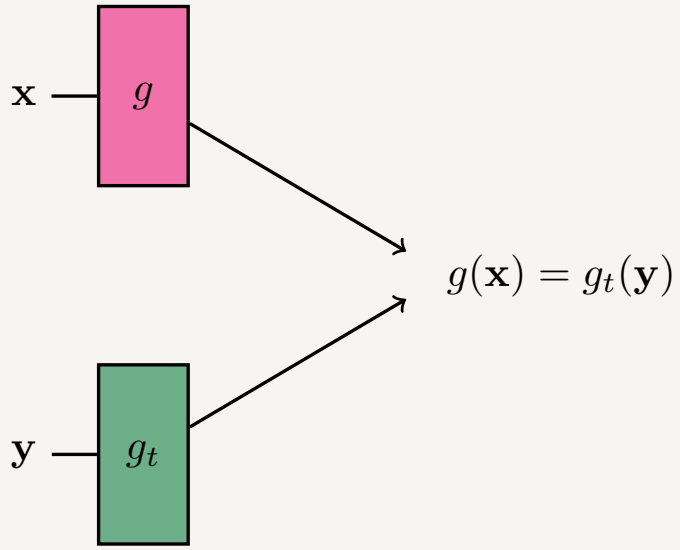
$$\langle \mathbf{a}, \mathbf{x} + \mathbf{y} \rangle = t \mod 2^r$$

BCJ Memoryless Algorithm [BCJ11]

- $wt(\mathbf{x}) = \frac{n}{4}$
- $g(\mathbf{x}) = \langle \mathbf{a}, \mathbf{x} \rangle \mod 2^r, \quad 2^r \approx \binom{n}{n/4}$
- $g_t(\mathbf{x}) = t - g(\mathbf{x}) \mod 2^r$

$$g(\mathbf{x}) = g_t(\mathbf{y}) \iff \langle \mathbf{a}, \mathbf{x} + \mathbf{y} \rangle = t \mod 2^r$$

BCJ Memoryless Algorithm



- Search a collision between g and g_t
- If $\mathbf{x} + \mathbf{y} \in \{0,1\}^n$ and $\langle \mathbf{a}, \mathbf{x}+\mathbf{y} \rangle = t \mod 2^n$ return $\mathbf{x} + \mathbf{y}$
- Else restart

BCJ Memoryless Algorithm [BCJ11]

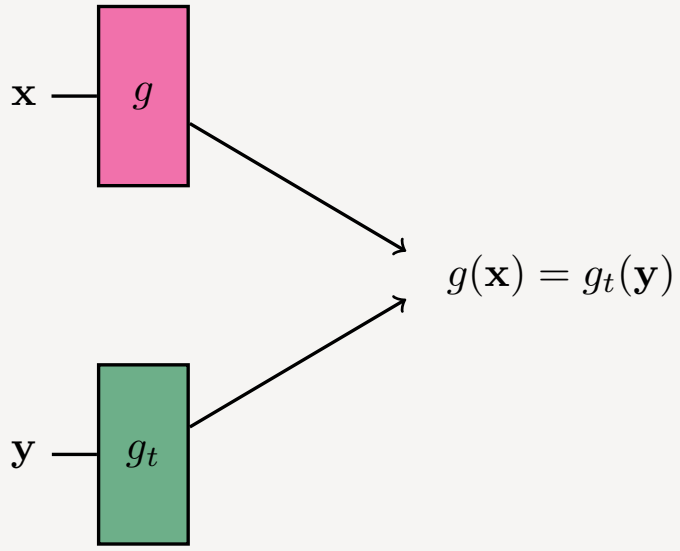
- $wt(\mathbf{x}) = \frac{n}{4}$
- $g(\mathbf{x}) = \langle \mathbf{a}, \mathbf{x} \rangle \mod 2^r, \quad 2^r \approx \binom{n}{n/4}$
- $g_t(\mathbf{x}) = t - g(\mathbf{x}) \mod 2^r$

$$g(\mathbf{x}) = g_t(\mathbf{y})$$

$$\Updownarrow$$

$$\langle \mathbf{a}, \mathbf{x} + \mathbf{y} \rangle = t \mod 2^r$$

BCJ Memoryless Algorithm



- Search a collision between g and g_t
- If $\mathbf{x} + \mathbf{y} \in \{0,1\}^n$ and $\langle \mathbf{a}, \mathbf{x}+\mathbf{y} \rangle = t \mod 2^n$ return $\mathbf{x} + \mathbf{y}$
- Else restart

$$\frac{\#\text{coll.}}{\#\text{rep.}} \approx 2^{r-n/2}$$

BCJ Memoryless Algorithm [BCJ11]

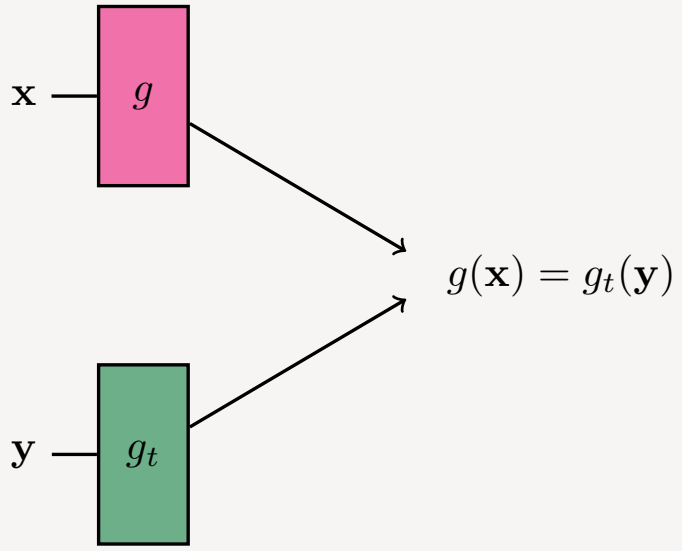
- $wt(\mathbf{x}) = \frac{n}{4}$
- $g(\mathbf{x}) = \langle \mathbf{a}, \mathbf{x} \rangle \mod 2^r, \quad 2^r \approx \binom{n}{n/4}$
- $g_t(\mathbf{x}) = t - g(\mathbf{x}) \mod 2^r$

$$g(\mathbf{x}) = g_t(\mathbf{y})$$

$$\Updownarrow$$

$$\langle \mathbf{a}, \mathbf{x} + \mathbf{y} \rangle = t \mod 2^r$$

BCJ Memoryless Algorithm



- Search a collision between g and g_t
- If $\mathbf{x} + \mathbf{y} \in \{0,1\}^n$ and $\langle \mathbf{a}, \mathbf{x}+\mathbf{y} \rangle = t \mod 2^n$ return $\mathbf{x} + \mathbf{y}$
- Else restart $\frac{\#\text{coll.}}{\#\text{rep.}} \approx 2^{r-n/2}$

$$T = \tilde{\mathcal{O}}\left(2^{0.717n}\right)$$

First Contribution: SS-PCS

- $wt(\mathbf{x}) = wt(\mathbf{y}) = \frac{n}{4}$
- $g(\mathbf{x}) = \langle \mathbf{a}, \mathbf{x} \rangle \mod 2^r, \quad 2^r \approx \binom{n}{n/4}$
- $g_t(\mathbf{y}) = t - g(\mathbf{y}) \mod 2^r$

$$g(\mathbf{x}) = g_t(\mathbf{y})$$

$$\Updownarrow$$

$$\langle \mathbf{a}, \mathbf{x} + \mathbf{y} \rangle = t \mod 2^r$$

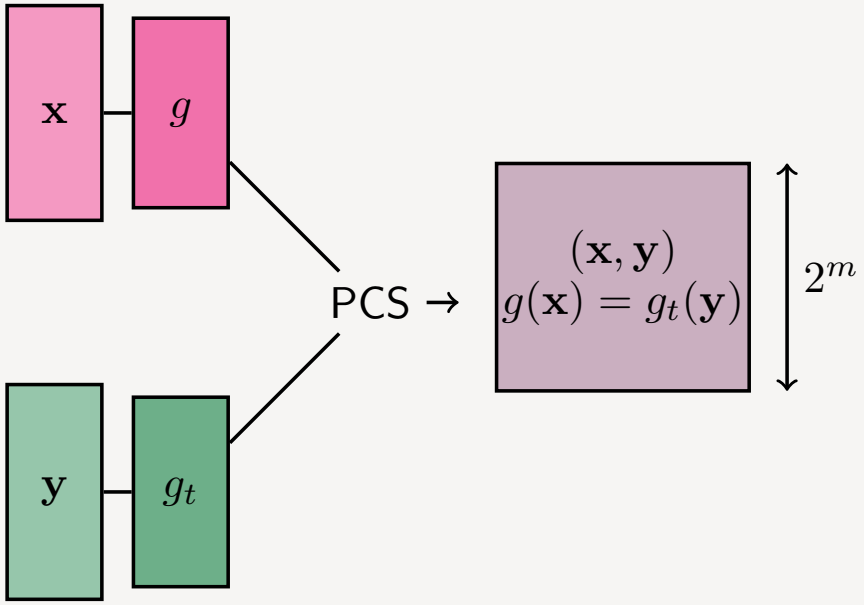
First Contribution: SS-PCS

- $wt(\mathbf{x}) = wt(\mathbf{y}) = \frac{n}{4}$
- $g(\mathbf{x}) = \langle \mathbf{a}, \mathbf{x} \rangle \mod 2^r, \quad 2^r \approx \binom{n}{n/4}$
- $g_t(\mathbf{y}) = t - g(\mathbf{y}) \mod 2^r$

$$g(\mathbf{x}) = g_t(\mathbf{y})$$

$$\Updownarrow$$

$$\langle \mathbf{a}, \mathbf{x} + \mathbf{y} \rangle = t \mod 2^r$$



- Generate 2^m collisions
- If $\exists\ \mathbf{x} + \mathbf{y} \in \{0,1\}^n$ and $\langle \mathbf{a}, \mathbf{x}+\mathbf{y} \rangle = t \mod 2^n$ return $\mathbf{x} + \mathbf{y}$
- Else Restart

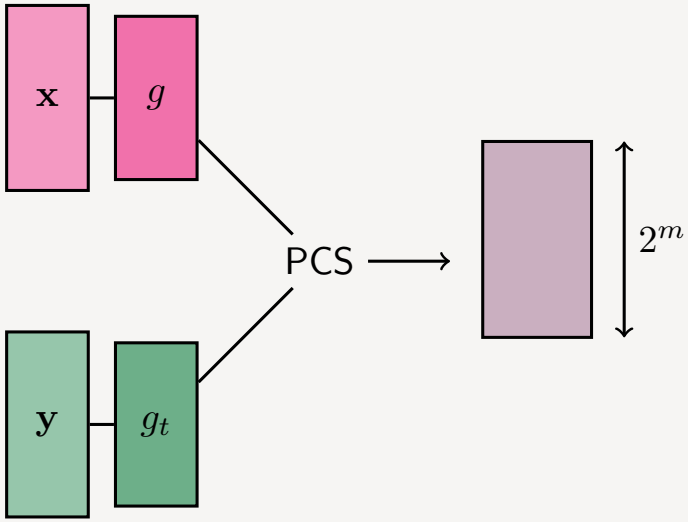
SS-PCS: Complexity

- $wt(\mathbf{x}) = wt(\mathbf{y}) = \frac{n}{4}$
- $g(\mathbf{x}) = \langle \mathbf{a}, \mathbf{x} \rangle \mod 2^r, \quad r \approx 0.811n$
- $g_t(\mathbf{y}) = t - g(\mathbf{y}) \mod 2^r$

$$g(\mathbf{x}) = g_t(\mathbf{y})$$

$$\Updownarrow$$

$$\langle \mathbf{a}, \mathbf{x} + \mathbf{y} \rangle = t \mod 2^r$$



- Generate 2^m collisions
- If $\exists\ \mathbf{x} + \mathbf{y} \in \{0,1\}^n$ and $\langle \mathbf{a}, \mathbf{x} + \mathbf{y} \rangle = t \mod 2^r$ return $\mathbf{x} + \mathbf{y}$
- Else Restart

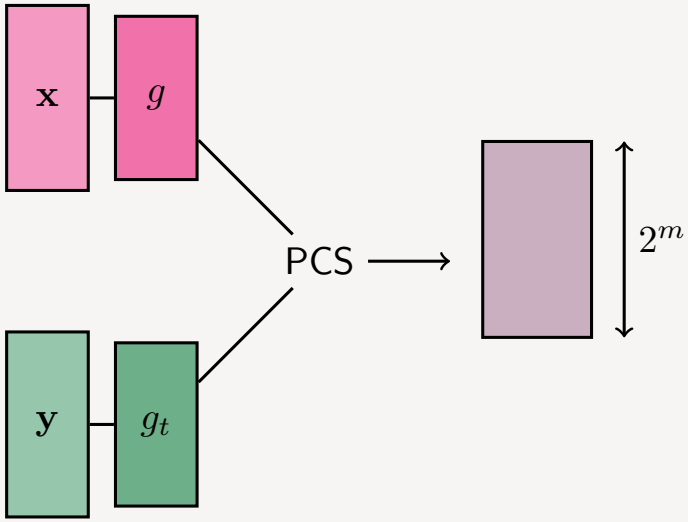
SS-PCS: Complexity

- $wt(\mathbf{x}) = wt(\mathbf{y}) = \frac{n}{4}$
- $g(\mathbf{x}) = \langle \mathbf{a}, \mathbf{x} \rangle \mod 2^r, \quad r \approx 0.811n$
- $g_t(\mathbf{y}) = t - g(\mathbf{y}) \mod 2^r$

$$g(\mathbf{x}) = g_t(\mathbf{y})$$

$$\Updownarrow$$

$$\langle \mathbf{a}, \mathbf{x} + \mathbf{y} \rangle = t \mod 2^r$$



- Generate 2^m collisions $\quad \tilde{\mathcal{O}}\left(2^{\frac{r+m}{2}}\right)$
- If $\exists\ \mathbf{x} + \mathbf{y} \in \{0,1\}^n$ and $\langle \mathbf{a}, \mathbf{x} + \mathbf{y} \rangle = t \mod 2^r$ return $\mathbf{x} + \mathbf{y}$
- Else Restart

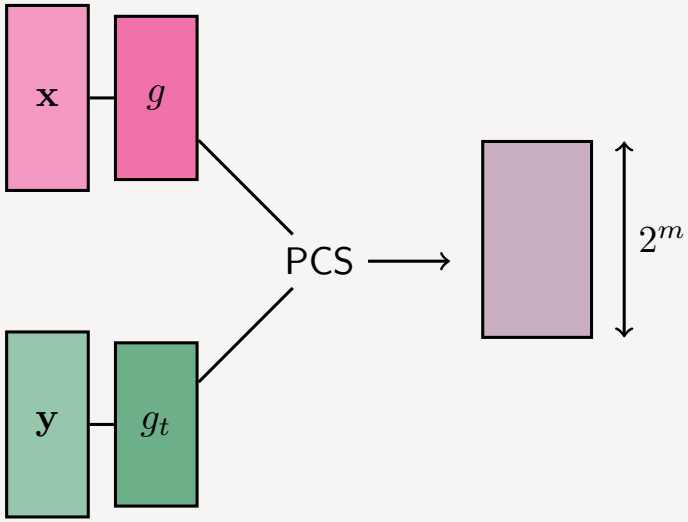
SS-PCS: Complexity

- $wt(\mathbf{x}) = wt(\mathbf{y}) = \frac{n}{4}$
- $g(\mathbf{x}) = \langle \mathbf{a}, \mathbf{x} \rangle \mod 2^r, \quad r \approx 0.811n$
- $g_t(\mathbf{y}) = t - g(\mathbf{y}) \mod 2^r$

$$g(\mathbf{x}) = g_t(\mathbf{y})$$

$$\Updownarrow$$

$$\langle \mathbf{a}, \mathbf{x} + \mathbf{y} \rangle = t \mod 2^r$$



- Generate 2^m collisions $\quad \tilde{\mathcal{O}}\left(2^{\frac{r+m}{2}}\right)$
- If $\exists\ \mathbf{x} + \mathbf{y} \in \{0,1\}^n$ and $\langle \mathbf{a}, \mathbf{x} + \mathbf{y} \rangle = t \mod 2^r$ return $\mathbf{x} + \mathbf{y}$
- Else Restart $\quad \tilde{\mathcal{O}}\left(2^{r-\frac{n}{2}-m}\right)$

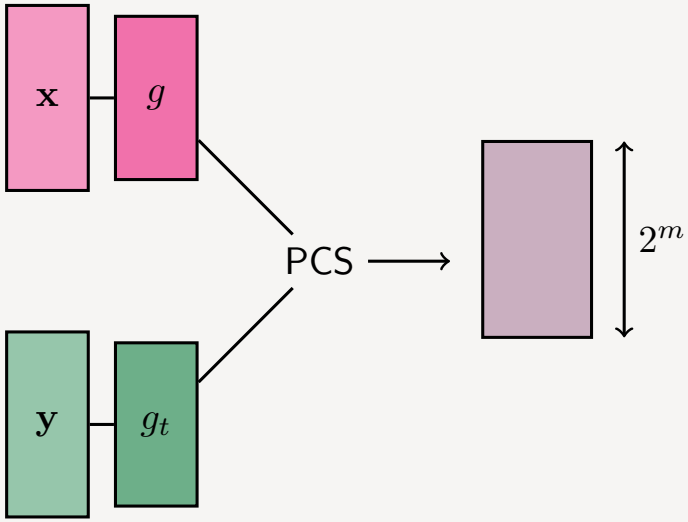
SS-PCS: Complexity

- $wt(\mathbf{x}) = wt(\mathbf{y}) = \frac{n}{4}$
- $g(\mathbf{x}) = \langle \mathbf{a}, \mathbf{x} \rangle \mod 2^r, \quad r \approx 0.811n$
- $g_t(\mathbf{y}) = t - g(\mathbf{y}) \mod 2^r$

$$g(\mathbf{x}) = g_t(\mathbf{y})$$

$$\Updownarrow$$

$$\langle \mathbf{a}, \mathbf{x} + \mathbf{y} \rangle = t \mod 2^r$$



- Generate 2^m collisions $\quad \tilde{\mathcal{O}}\left(2^{\frac{r+m}{2}}\right)$
- If $\exists\ \mathbf{x} + \mathbf{y} \in \{0,1\}^n$ and $\langle \mathbf{a}, \mathbf{x} + \mathbf{y} \rangle = t \mod 2^r$ return $\mathbf{x} + \mathbf{y}$
- Else Restart $\quad \tilde{\mathcal{O}}\left(2^{r-\frac{n}{2}-m}\right)$

Time: $\tilde{\mathcal{O}}\left(2^{0.717n-\frac{m}{2}}\right)$

Space: $\tilde{\mathcal{O}}\left(2^m\right)$

SS-PCS in a Nutshell

Assumptions

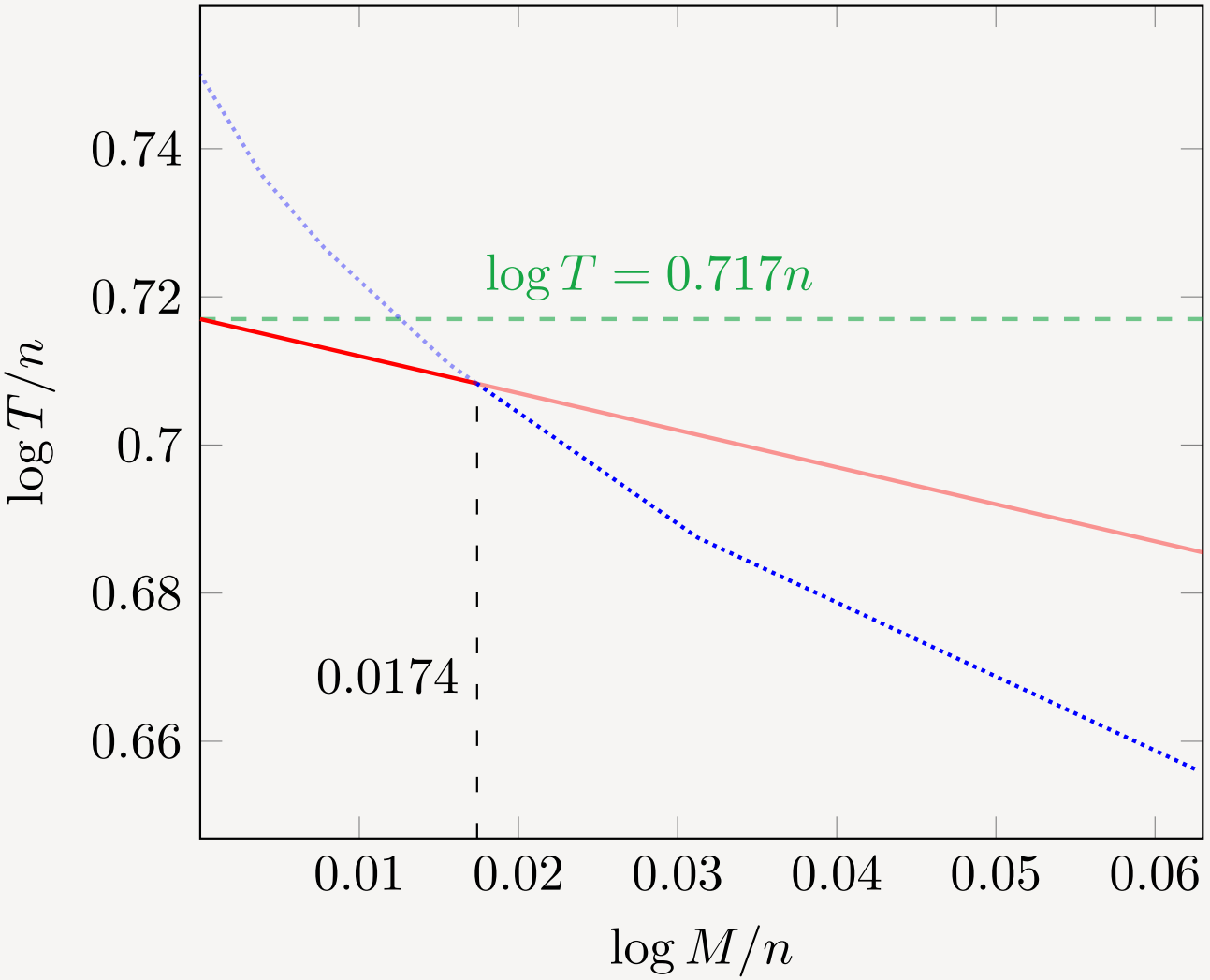
- PCS behaves with g, g_t as with independent random functions
- PCS returns uniformly random collisions

Experimentally checked

SS-PCS...

- requires $\tilde{\mathcal{O}}(2^m)$ space
- takes times $\tilde{\mathcal{O}}\left(2^{0.717n-\frac{m}{2}}\right)$
- is better than previous work when $m \leq 0.0174n$
- is the BCJ memoryless algorithm when $m = 0$

SS-PCS trade-off



Second Contribution: Set-up

- $wt(\mathbf{x}) = wt(\mathbf{y}) = \frac{n}{16}$
- Random $R_1, R_2, R_3 \in \mathbb{Z}_{2^s}$ and $R_4 = t - (R_1 + R_2 + R_3)$ $\quad 2^s \approx \binom{n}{n/16}$
- $f(\mathbf{x}) = \langle \mathbf{a}, \mathbf{x} \rangle \mod 2^s$
- $f_{R_i}(\mathbf{y}) = R_i - f(\mathbf{y}) \mod 2^s$

Second Contribution: Set-up

- $wt(\mathbf{x}) = wt(\mathbf{y}) = \frac{n}{16}$
- Random $R_1, R_2, R_3 \in \mathbb{Z}_{2^s}$ and $R_4 = t - (R_1 + R_2 + R_3)$ $\quad 2^s \approx \binom{n}{n/16}$
- $f(\mathbf{x}) = \langle \mathbf{a}, \mathbf{x} \rangle \mod 2^s$
- $f_{R_i}(\mathbf{y}) = R_i - f(\mathbf{y}) \mod 2^s$

$$f(\mathbf{x}_i) = f_{R_i}(\mathbf{y}_i) \iff \langle \mathbf{a}, \mathbf{x}_i + \mathbf{y}_i \rangle = R_i \mod 2^s$$

Second Contribution: Set-up

- $wt(\mathbf{x}) = wt(\mathbf{y}) = \frac{n}{16}$
- Random $R_1, R_2, R_3 \in \mathbb{Z}_{2^s}$ and $R_4 = t - (R_1 + R_2 + R_3)$ $\quad 2^s \approx \binom{n}{n/16}$
- $f(\mathbf{x}) = \langle \mathbf{a}, \mathbf{x} \rangle \mod 2^s$
- $f_{R_i}(\mathbf{y}) = R_i - f(\mathbf{y}) \mod 2^s$

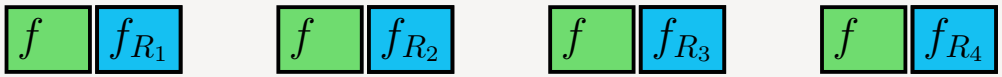
$$f(\mathbf{x}_i) = f_{R_i}(\mathbf{y}_i) \iff \langle \mathbf{a}, \mathbf{x}_i + \mathbf{y}_i \rangle = R_i \mod 2^s$$

$$\Downarrow$$

$$\langle \mathbf{a}, \mathbf{x}_1 + \mathbf{y}_1 + \cdots + \mathbf{x}_4 + \mathbf{y}_4 \rangle = t \mod 2^s$$

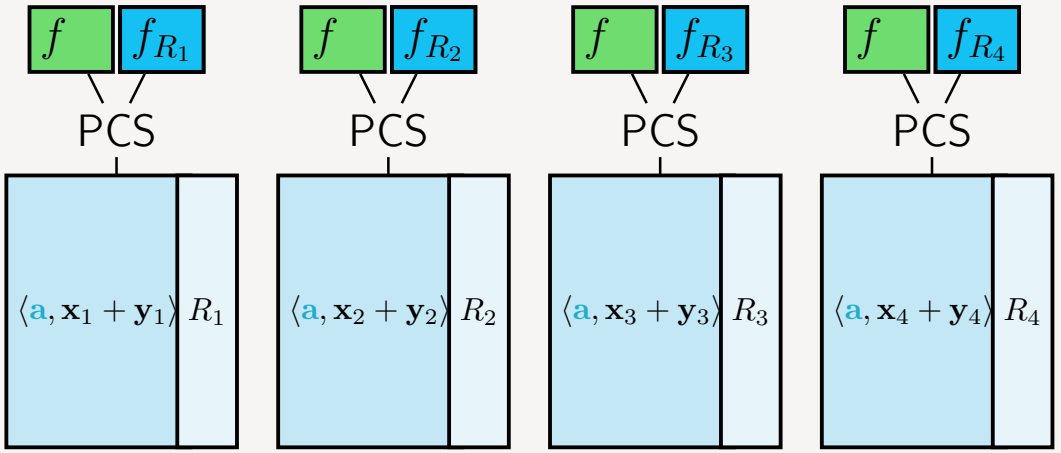
Second Contribution: SS-PCS$_4$

$$f(\mathbf{x}) = \langle \mathbf{a}, \mathbf{x} \rangle \mod 2^s \qquad f_{R_i}(\mathbf{x}) = R_i - f(\mathbf{x}) \mod 2^s$$



Second Contribution: SS-PCS$_4$

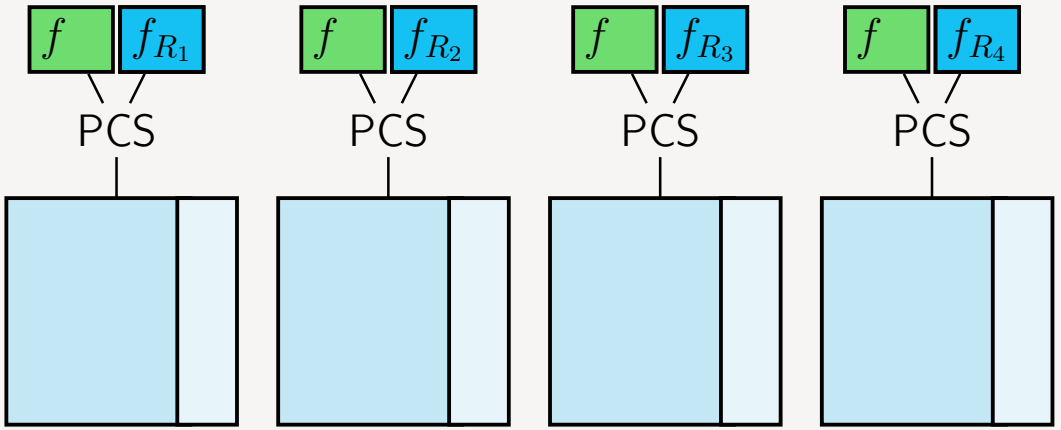
$$f(\mathbf{x}) = \langle \mathbf{a}, \mathbf{x} \rangle \mod 2^s \qquad f_{R_i}(\mathbf{x}) = R_i - f(\mathbf{x}) \mod 2^s$$



- Find 2^m collisions between f and f_{R_i}

Second Contribution: SS-PCS$_4$

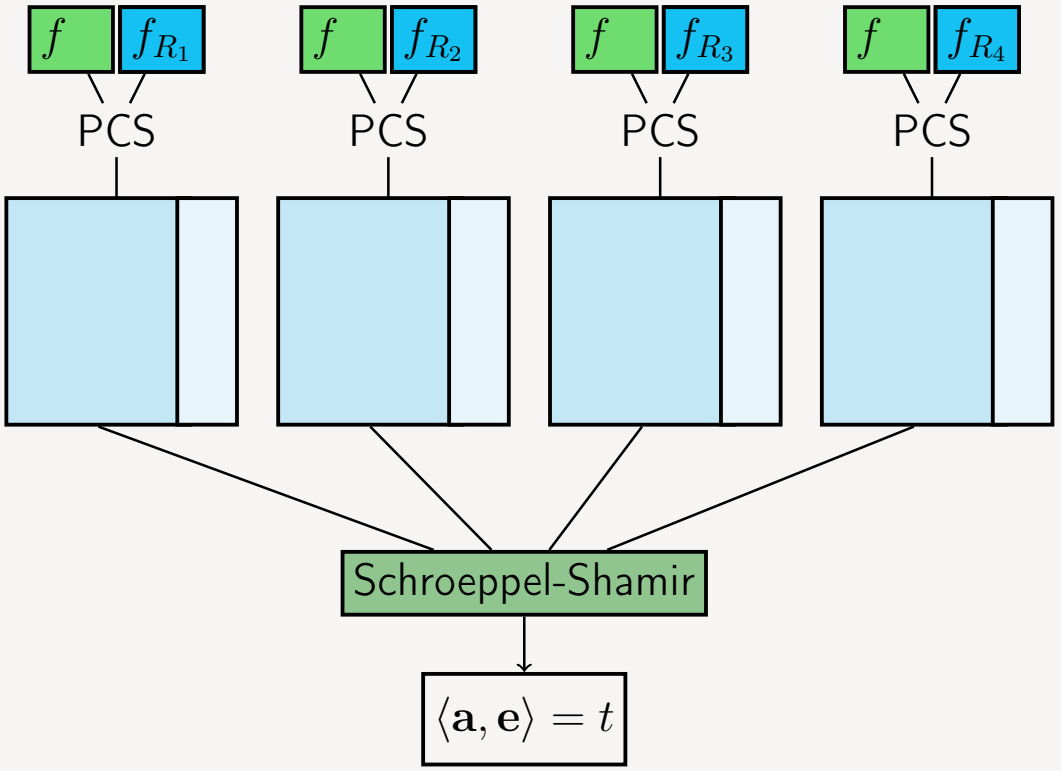
$$f(\mathbf{x}) = \langle \mathbf{a}, \mathbf{x} \rangle \mod 2^s \qquad f_{R_i}(\mathbf{x}) = R_i - f(\mathbf{x}) \mod 2^s$$



- Find 2^m collisions between f and f_{R_i}
- Discard inconsistent ones

Second Contribution: SS-PCS$_4$

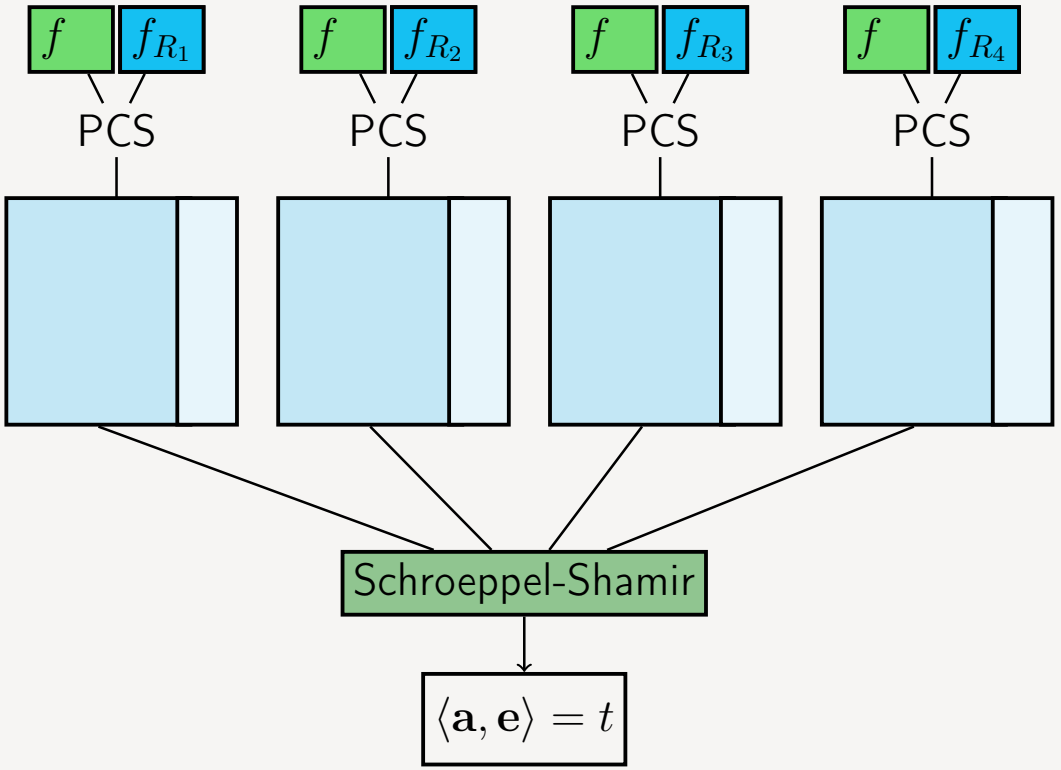
$$f(\mathbf{x}) = \langle \mathbf{a}, \mathbf{x} \rangle \mod 2^s \qquad f_{R_i}(\mathbf{x}) = R_i - f(\mathbf{x}) \mod 2^s$$



- Find 2^m collisions between f and f_{R_i}
- Discard inconsistent ones
- Search for $t_1 + t_2 + t_3 + t_4 = t$

Second Contribution: SS-PCS$_4$

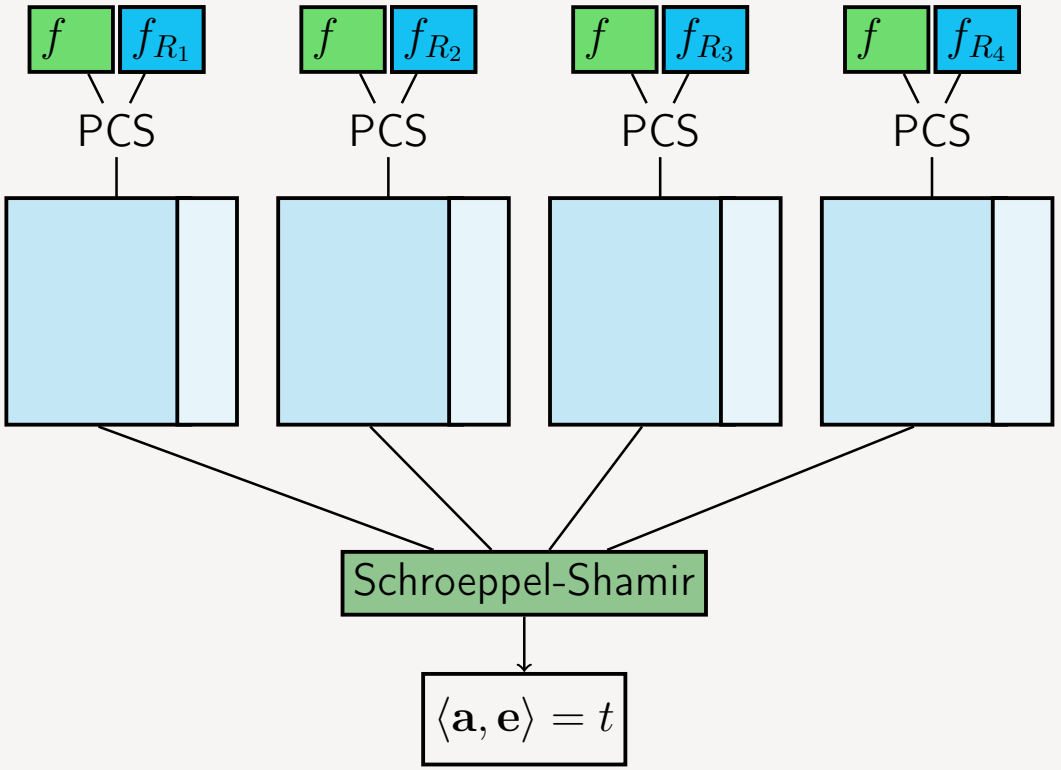
$$f(\mathbf{x}) = \langle \mathbf{a}, \mathbf{x} \rangle \mod 2^s \qquad f_{R_i}(\mathbf{x}) = R_i - f(\mathbf{x}) \mod 2^s$$



- Find 2^m collisions between f and f_{R_i}
- Discard inconsistent ones
- Search for $t_1 + t_2 + t_3 + t_4 = t$
- If $\exists\ \mathbf{e} = \sum \mathbf{x}_i + \mathbf{y}_i \in \{0,1\}^n$ return $\mathbf{e}$

Second Contribution: SS-PCS$_4$

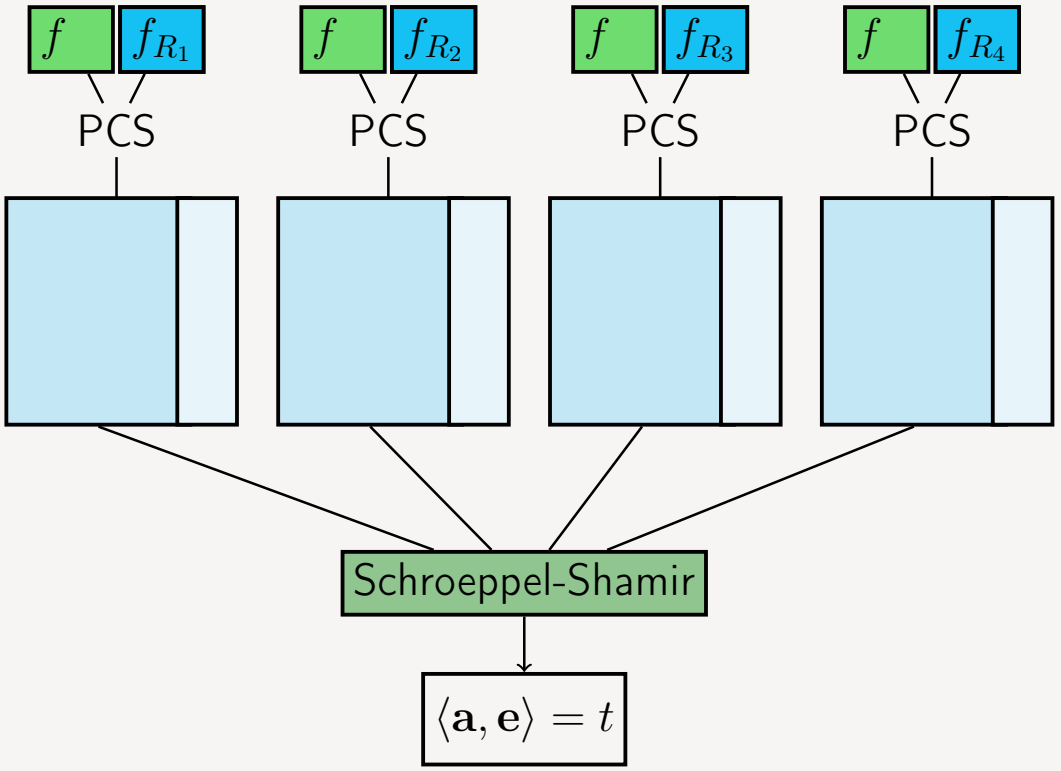
$$f(\mathbf{x}) = \langle \mathbf{a}, \mathbf{x} \rangle \mod 2^s \qquad f_{R_i}(\mathbf{x}) = R_i - f(\mathbf{x}) \mod 2^s$$



- Find 2^m collisions between f and f_{R_i}
- Discard inconsistent ones
- Search for $t_1 + t_2 + t_3 + t_4 = t$
- If $\exists\ \mathbf{e} = \sum \mathbf{x}_i + \mathbf{y}_i \in \{0,1\}^n$ return $\mathbf{e}$
- Else restart
  - with same R_i

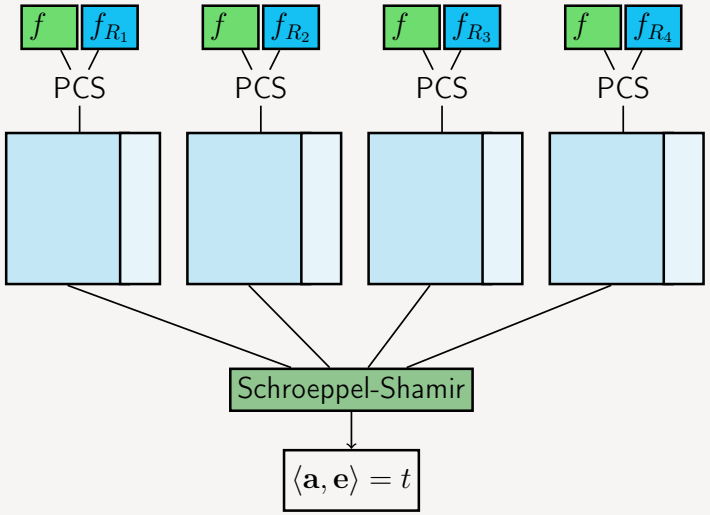
Second Contribution: SS-PCS$_4$

$$f(\mathbf{x}) = \langle \mathbf{a}, \mathbf{x} \rangle \mod 2^s \qquad f_{R_i}(\mathbf{x}) = R_i - f(\mathbf{x}) \mod 2^s$$



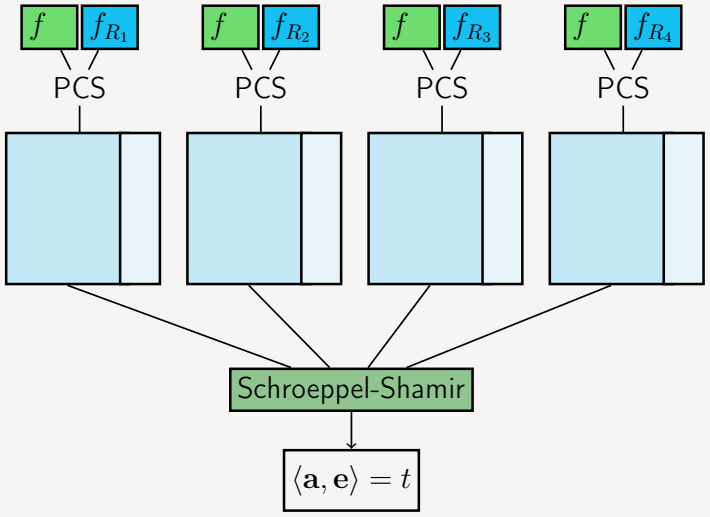
- Find 2^m collisions between f and f_{R_i}
- Discard inconsistent ones
- Search for $t_1 + t_2 + t_3 + t_4 = t$
- If $\exists\ \mathbf{e} = \sum \mathbf{x}_i + \mathbf{y}_i \in \{0,1\}^n$ return $\mathbf{e}$
- Else restart
  - with same R_i
  - with different R_i

SS-PCS$_4$: Complexity



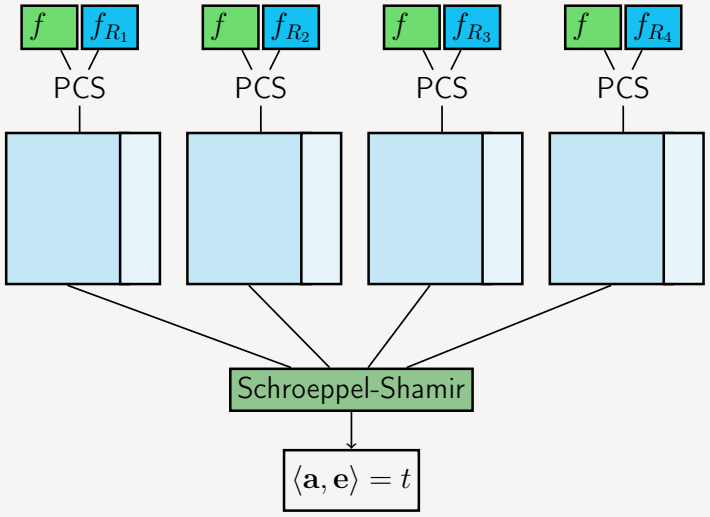
- Find 2^m collisions between f and f_{R_i}
- Discard inconsistent ones
- Search for $t_1 + t_2 + t_3 + t_4 = t$
- If $\exists\ \mathbf{e} \in \{0,1\}^n$ return $\mathbf{e}$
- Else restart
  - Same R_i
  - New R_i

SS-PCS$_4$: Complexity



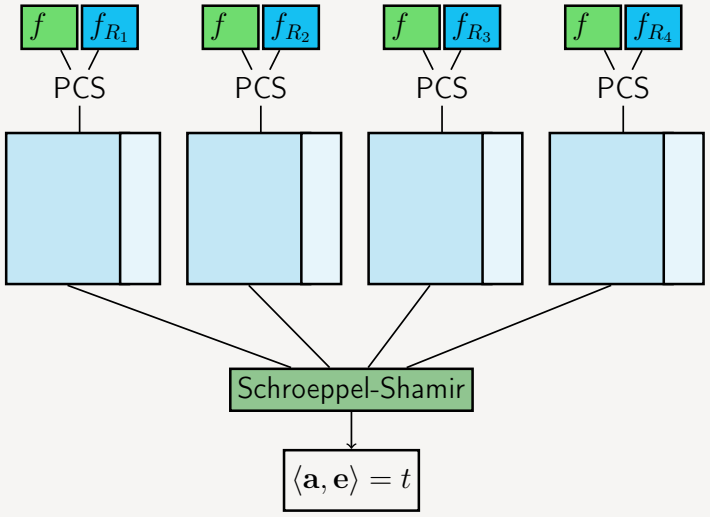
- Find 2^m collisions between f and f_{R_i} $\tilde{\mathcal{O}}\left(2^{\frac{s+m}{2}}\right)$
- Discard inconsistent ones
- Search for $t_1+t_2+t_3+t_4 = t$
- If $\exists\ \mathbf{e} \in \{0,1\}^n$ return $\mathbf{e}$
- Else restart
  - Same R_i
  - New R_i

SS-PCS$_4$: Complexity



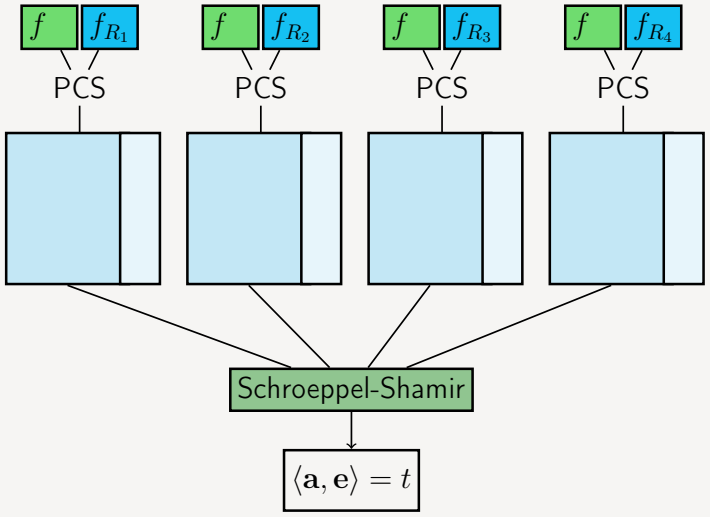
- Find 2^m collisions between f and f_{R_i} $\tilde{\mathcal{O}}\left(2^{\frac{s+m}{2}}\right)$
- Discard inconsistent ones $\tilde{\mathcal{O}}\left(2^m\right)$
- Search for $t_1+t_2+t_3+t_4 = t$
- If $\exists\ \mathbf{e} \in \{0,1\}^n$ return $\mathbf{e}$
- Else restart
  - Same R_i
  - New R_i

SS-PCS$_4$: Complexity



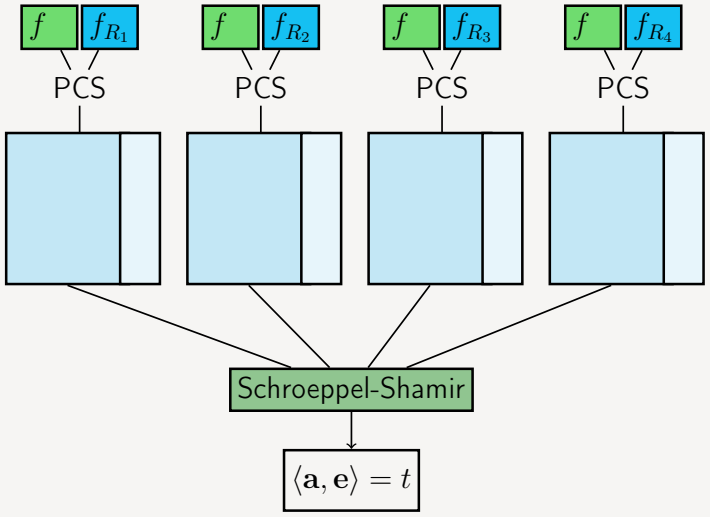
- Find 2^m collisions between f and f_{R_i} $\tilde{\mathcal{O}}\left(2^{\frac{s+m}{2}}\right)$
- Discard inconsistent ones $\tilde{\mathcal{O}}\left(2^m\right)$
- Search for $t_1+t_2+t_3+t_4 = t$ $\tilde{\mathcal{O}}\left(2^{2m'}\right)$
- If $\exists\ \mathbf{e} \in \{0,1\}^n$ return $\mathbf{e}$
- Else restart
  - Same R_i
  - New R_i

SS-PCS$_4$: Complexity



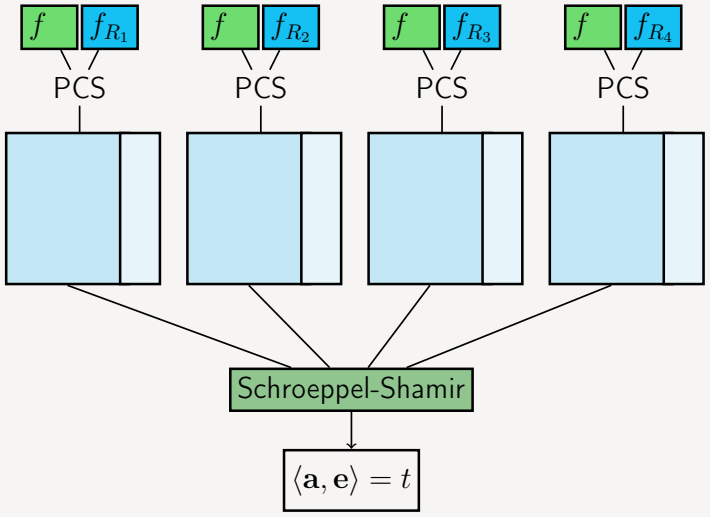
- Find 2^m collisions between f and f_{R_i} $\tilde{\mathcal{O}}\left(2^{\frac{s+m}{2}}\right)$
- Discard inconsistent ones $\tilde{\mathcal{O}}\left(2^m\right)$
- Search for $t_1+t_2+t_3+t_4=t$ $\tilde{\mathcal{O}}\left(2^{2m'}\right)$
- If $\exists\ \mathbf{e} \in \{0,1\}^n$ return $\mathbf{e}$
- Else restart
  - Same R_i $\tilde{\mathcal{O}}\left(2^{4s-\frac{1}{2}-4m}\right) = P[\text{Rep.}|R_i \text{ good}]^{-1}$
  - New R_i $\tilde{\mathcal{O}}\left(2^{3s-1}\right) = P[R_i \text{ good}]^{-1}$

SS-PCS$_4$: Complexity



- Find 2^m collisions between f and f_{R_i} $\tilde{\mathcal{O}}\left(2^{\frac{s+m}{2}}\right)$
- Discard inconsistent ones $\tilde{\mathcal{O}}\left(2^m\right)$
- Search for $t_1+t_2+t_3+t_4 = t$ $\tilde{\mathcal{O}}\left(2^{2m'}\right)$
- If $\exists\ \mathbf{e} \in \{0,1\}^n$ return $\mathbf{e}$
- Else restart $\tilde{\mathcal{O}}\left(2^{7s-4m-3\frac{n}{2}}\right)$
  - Same R_i
  - New R_i

SS-PCS$_4$: Complexity



- Find 2^m collisions between f and f_{R_i} $\tilde{\mathcal{O}}\left(2^{\frac{s+m}{2}}\right)$
- Discard inconsistent ones $\tilde{\mathcal{O}}\left(2^m\right)$
- Search for $t_1+t_2+t_3+t_4 = t$ $\tilde{\mathcal{O}}\left(2^{2m'}\right)$
- If $\exists\ \mathbf{e} \in \{0,1\}^n$ return $\mathbf{e}$
- Else restart $\tilde{\mathcal{O}}\left(2^{7s-4m-3\frac{n}{2}}\right)$
  - Same R_i
  - New R_i

$$s \approx \log \binom{n}{n/16} \approx 0.337n,\ m < 0.21n,$$
$$m' \approx m - 0.006n$$

Time: $\tilde{\mathcal{O}}\left(2^{0.849n-2m}\right)$

Space: $\tilde{\mathcal{O}}\left(2^m\right)$

In a Nutshell

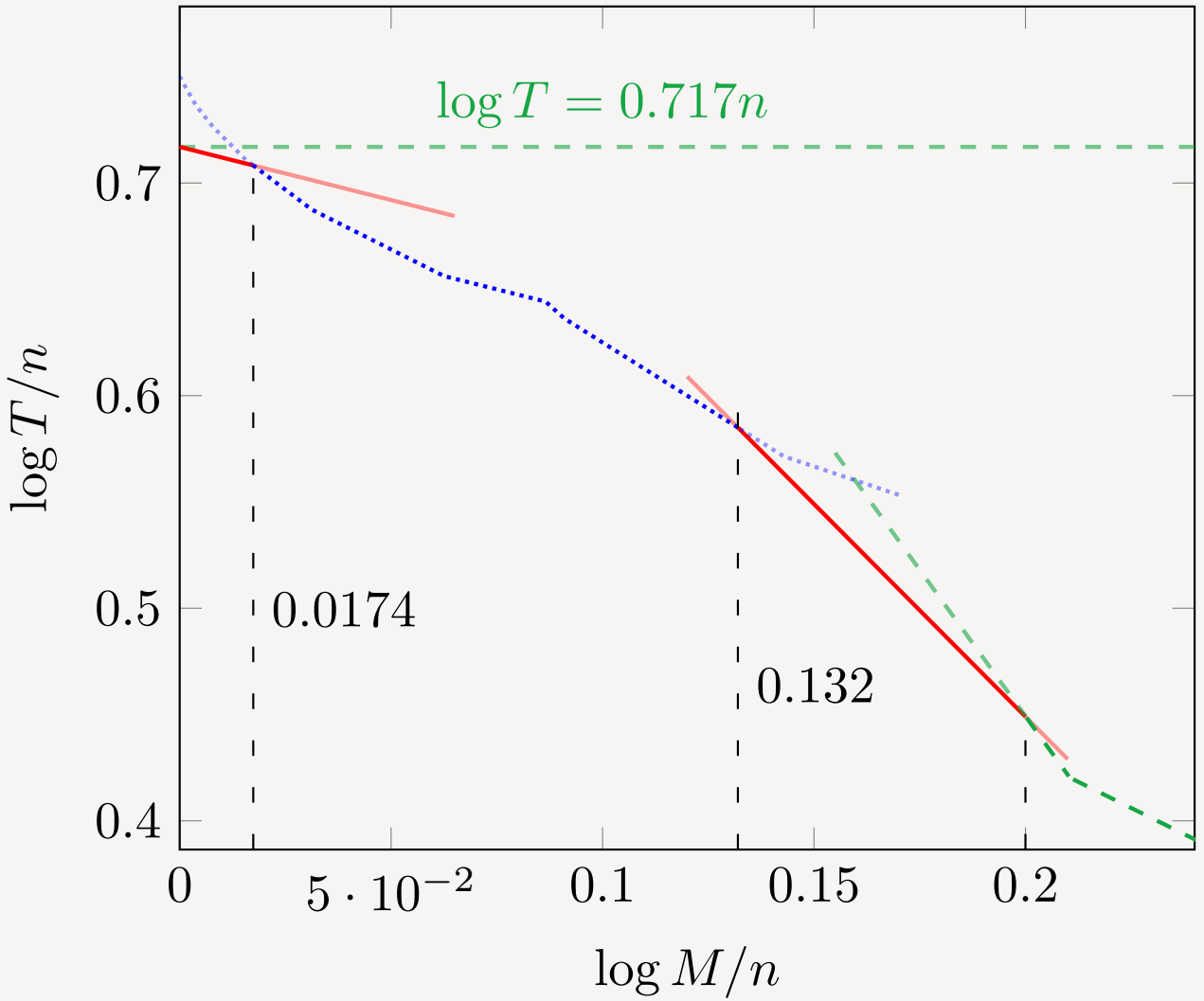
Assumptions

- PCS behaves with f, f_{R_i} as with independent random functions
- PCS returns uniformly random collisions
- $\langle \mathbf{a}, \mathbf{e}_i \rangle \mod 2^s$ are independently and uniformly distributed.

SS-PCS$_4$...

- requires $\tilde{\mathcal{O}}\left(2^m\right)$ memory
- find a solution in expected time $\tilde{\mathcal{O}}\left(2^{0.849n-2m}\right)$, $m \leq 0.21n$.
- is better than previous work for $0.132n \leq m \leq 0.2n$

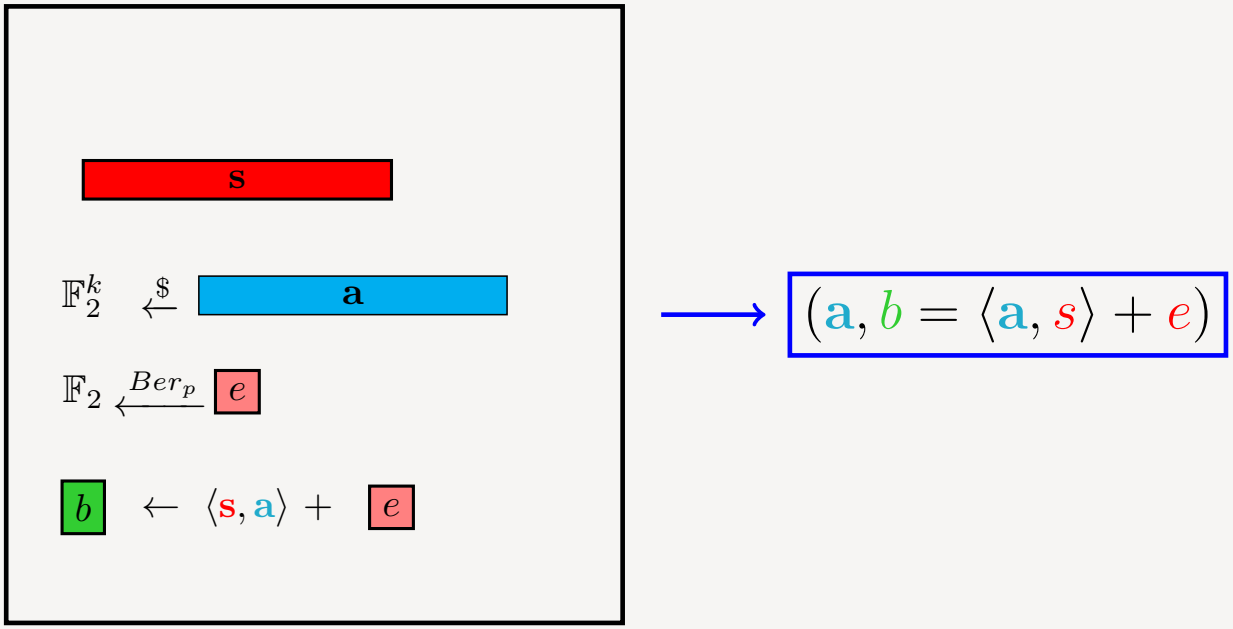
New Trade-offs



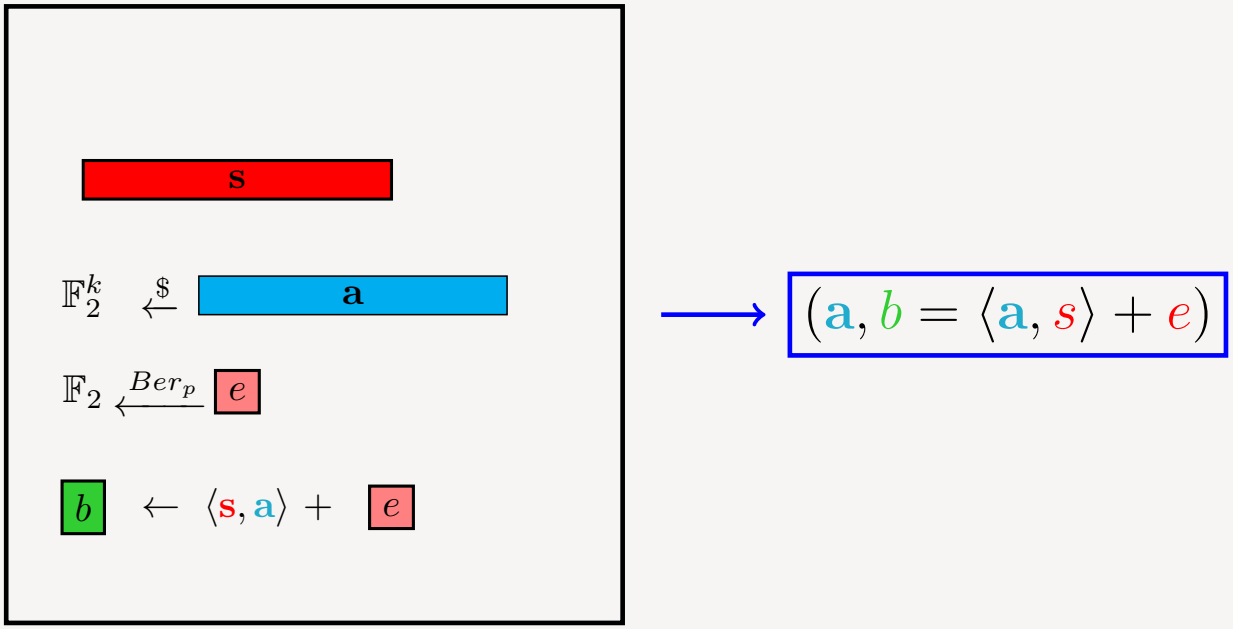
1 Application 1: Random Subset-Sum Problem

2 Application 2: LPN

Learning Parity with Noise



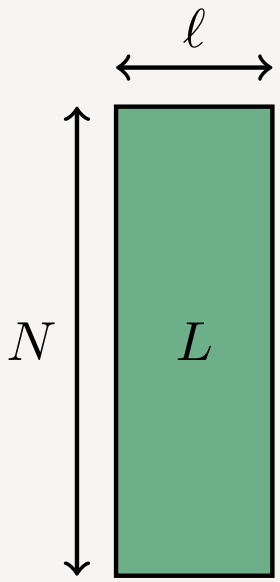
Learning Parity with Noise



(Search)LPN

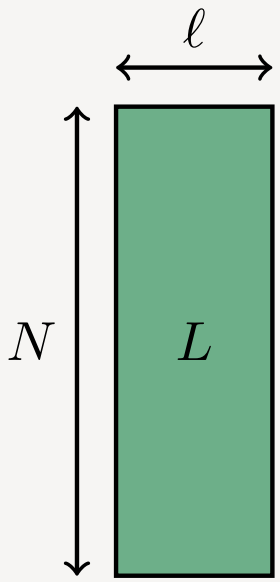
- $\mathbf{s}$ unknown
- $p < \frac{1}{2}$ $e = 1$ with probability p
- GOAL: Recover $\mathbf{s}$ given access to LPN oracle

c-sum Problem [EHKMS18]



- Single-solution: $(x_1, \ldots, x_c) \in L^c : x_1 + \cdots + x_c = 0$
- Solution: a set of N distinct single-solutions

c-sum Problem [EHKMS18]



- Single-solution: $(x_1, \ldots, x_c) \in L^c : x_1 + \cdots + x_c = 0$
- Solution: a set of N distinct single-solutions

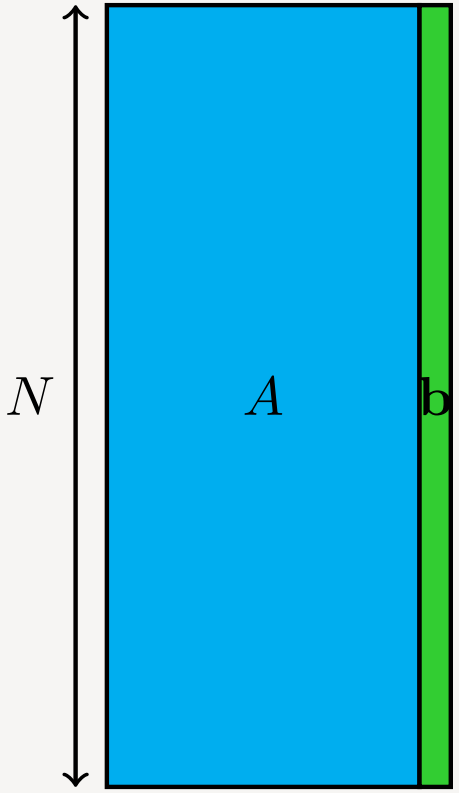
Reduction

- $\ell = \frac{k \log c}{(1-\varepsilon) \log k}$
- $\log N \geq \frac{\ell + c \log c + 1}{c}$
- Assuming c-sums behave as independent
- c-sum Pb solved in time T and memory M

LPN solved in time $T^{1+o(1)}$ and memory $M^{1+o(1)}$ using $N^{1+o(1)}$ samples

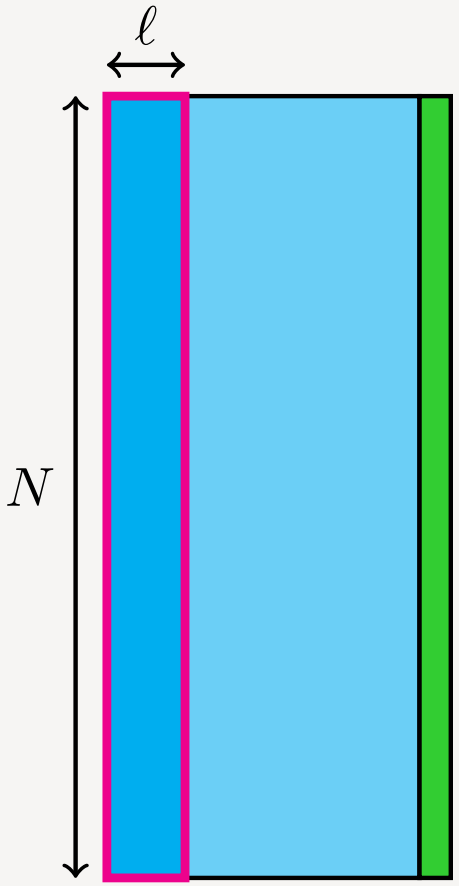
Solving LPN with c-sum

GOAL: Solve $A\mathbf{s} + \mathbf{e} = b$



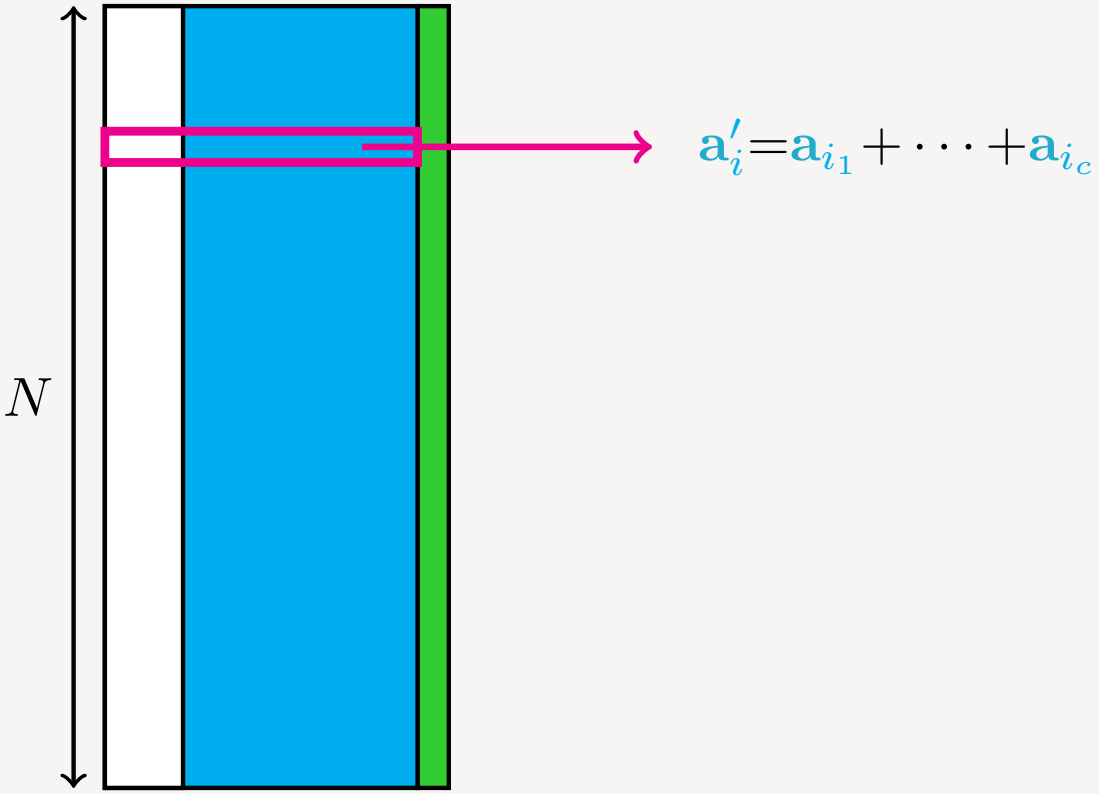
Solving LPN with c-sum

GOAL: Solve $A\mathbf{s} + \mathbf{e} = b$



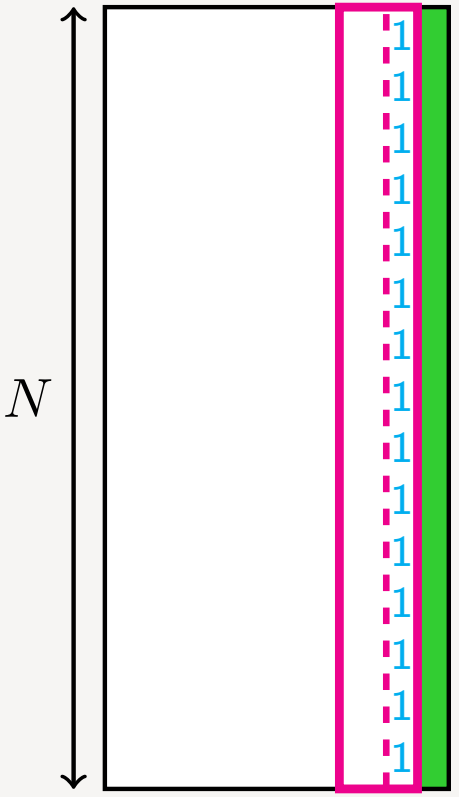
Solving LPN with c-sum

GOAL: Solve $A\mathbf{s} + \mathbf{e} = b$



Solving LPN with c-sum

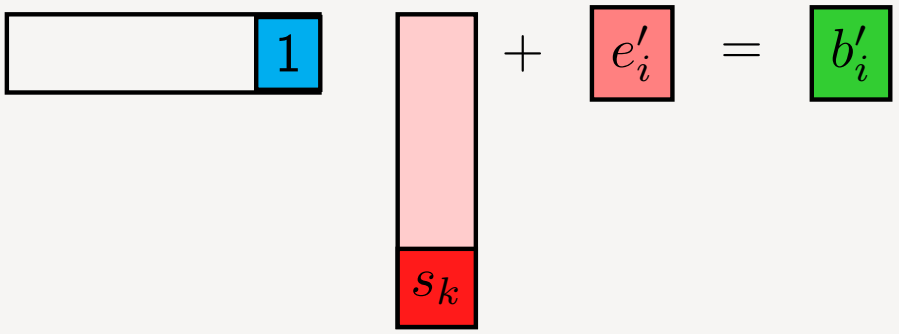
GOAL: Solve $A\mathbf{s} + \mathbf{e} = b$



Solving LPN with c-sum

GOAL: Solve $A\mathbf{s} + \mathbf{e} = b$

- $e'_i = \sum_{j=1}^{t} e_{i_j}$
- Pilling-up Lemma: $e'_i = 1$ w.p. $p' = \frac{1}{2} - \frac{1}{2}(1-2p)^t$



Solving LPN with c-sum

GOAL: Solve $A\mathbf{s} + \mathbf{e} = b$

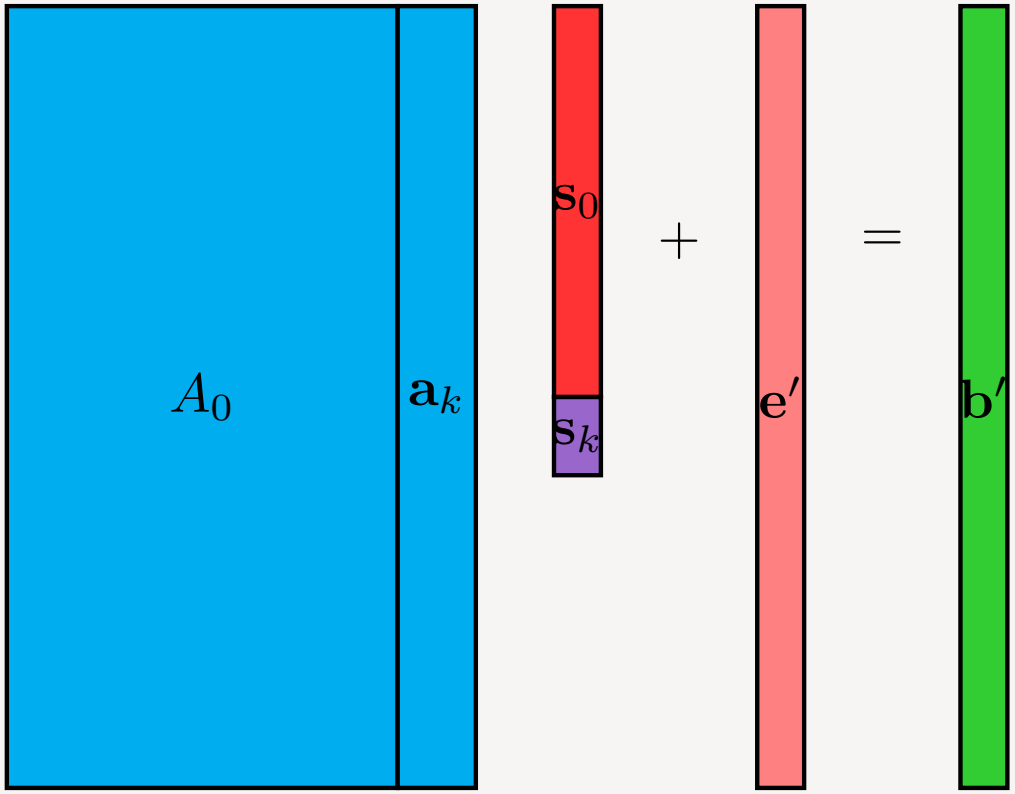
- $e'_i = \sum_{j=1}^{t} e_{i_j}$
- Pilling-up Lemma: $e'_i = 1$ w.p. $p' = \frac{1}{2} - \frac{1}{2}(1-2p)^t$
- Majority vote

$$s_k + e'_i = b'_i$$

Solving LPN with c-sum

GOAL: Solve $A\mathbf{s} + \mathbf{e} = b$

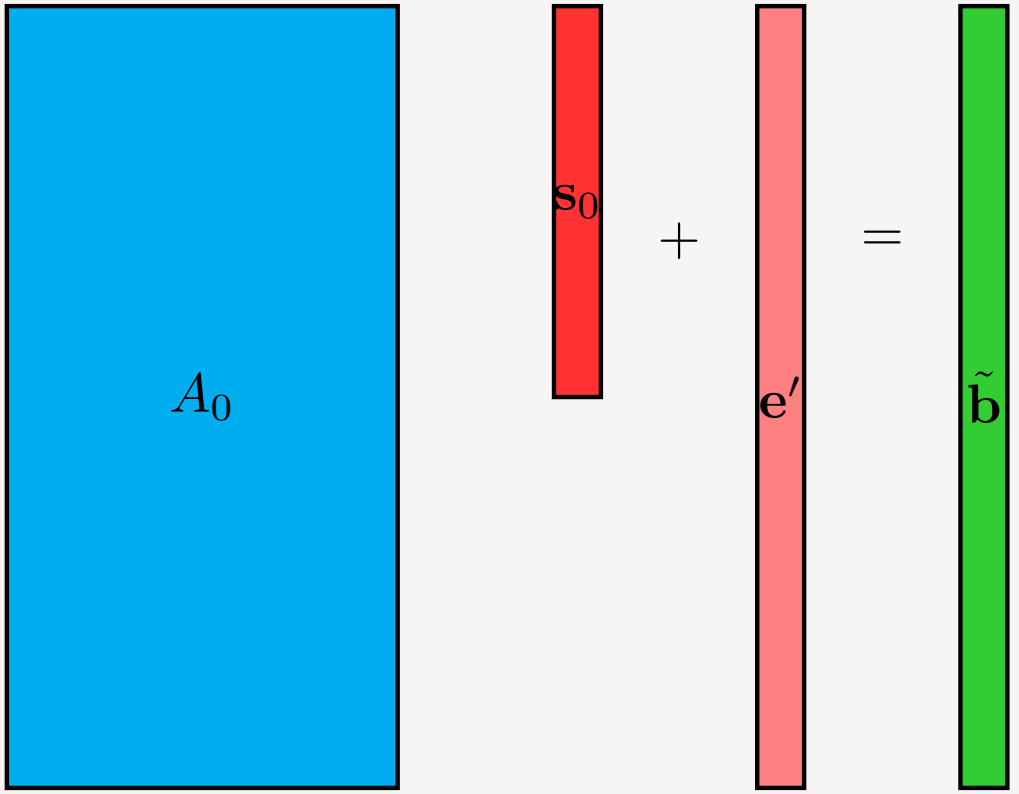
- $e'_i = \sum_{j=1}^{t} e_{i_j}$
- Pilling-up Lemma: $e'_i = 1$ w.p. $p' = \frac{1}{2} - \frac{1}{2}(1-2p)^t$
- Majority vote



Solving LPN with c-sum

GOAL: Solve $A\mathbf{s} + \mathbf{e} = b$

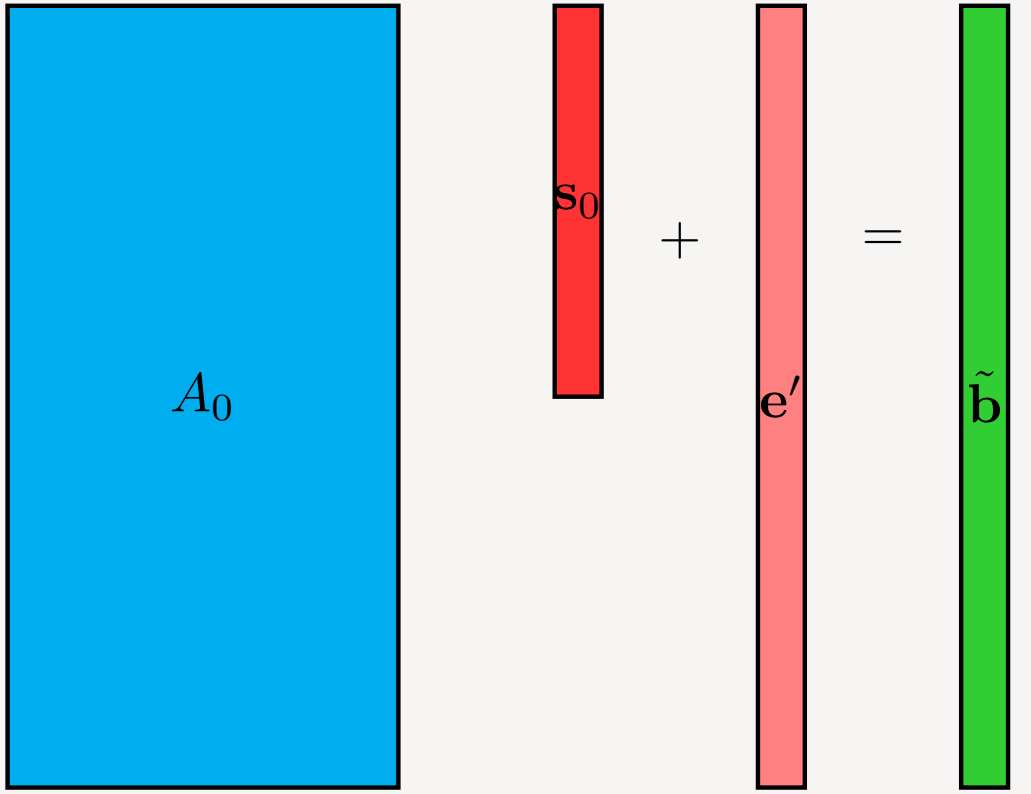
- $e'_i = \sum_{j=1}^{t} e_{i_j}$
- Pilling-up Lemma: $e'_i = 1$ w.p. $p' = \frac{1}{2} - \frac{1}{2}(1-2p)^t$
- Majority vote
- Re-iterate



Solving LPN with c-sum

GOAL: Solve $A\mathbf{s} + \mathbf{e} = b$

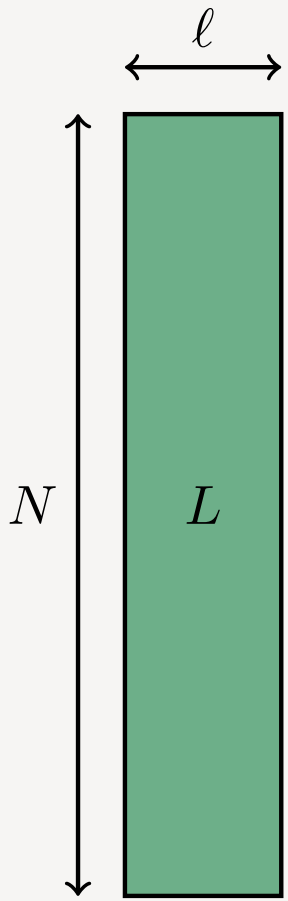
- $e'_i = \sum_{j=1}^{t} e_{i_j}$
- Pilling-up Lemma: $e'_i = 1$ w.p. $p' = \frac{1}{2} - \frac{1}{2}(1-2p)^t$
- Majority vote
- Re-iterate



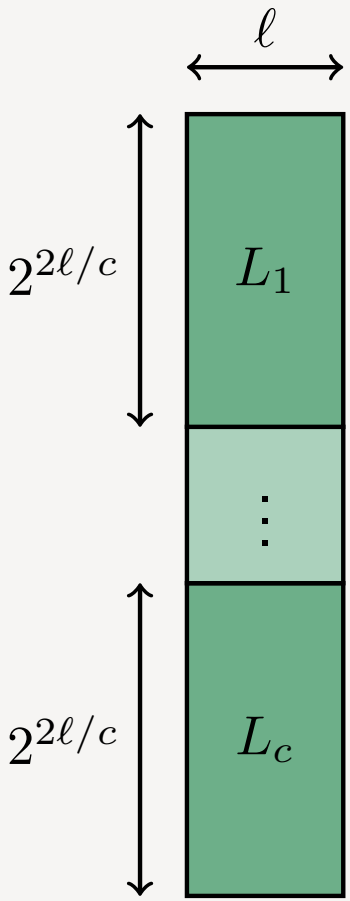
Contribution

[EHKMS18]: c-sum via Dissection [DDKS12]
Our work: c-sum via PCS

New `c-sum-PCS` Algorithm

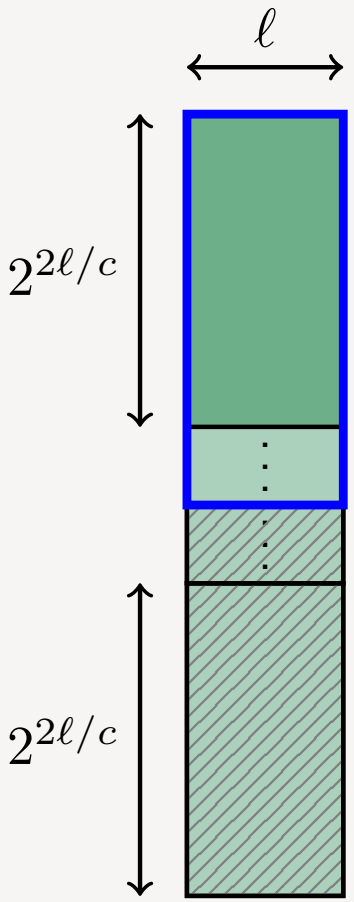


New c-sum-PCS Algorithm



$$f_0, f_1 : (\mathbb{Z}_{2^{2\ell/c}})^{c/2} \to \mathbb{F}_2^{\ell}$$

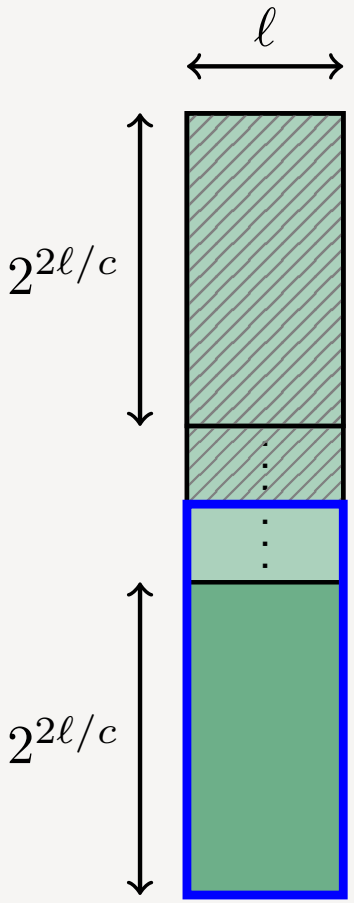
New c-sum-PCS Algorithm



$$f_0, f_1 : (\mathbb{Z}_{2^{2\ell/c}})^{c/2} \to \mathbb{F}_2^{\ell}$$

- $f_0(i_1, \ldots i_{c/2}) = L_1[i_1] + \cdots + L_{c/2}[i_{c/2}]$

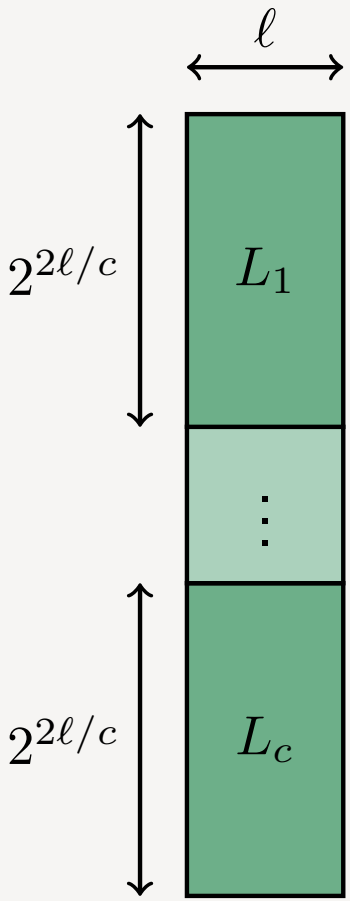
New c-sum-PCS Algorithm



$$f_0, f_1 : (\mathbb{Z}_{2^{2\ell/c}})^{c/2} \to \mathbb{F}_2^{\ell}$$

- $f_0(i_1, \ldots i_{c/2}) = L_1[i_1] + \cdots + L_{c/2}[i_{c/2}]$
- $f_1(i_{1+c/2}, \ldots, i_c) = L_{1+c/2}[i_{c/2+1}] + \cdots + L_c[i_c]$

New c-sum-PCS Algorithm



$$f_0, f_1 : (\mathbb{Z}_{2^{2\ell/c}})^{c/2} \to \mathbb{F}_2^\ell$$

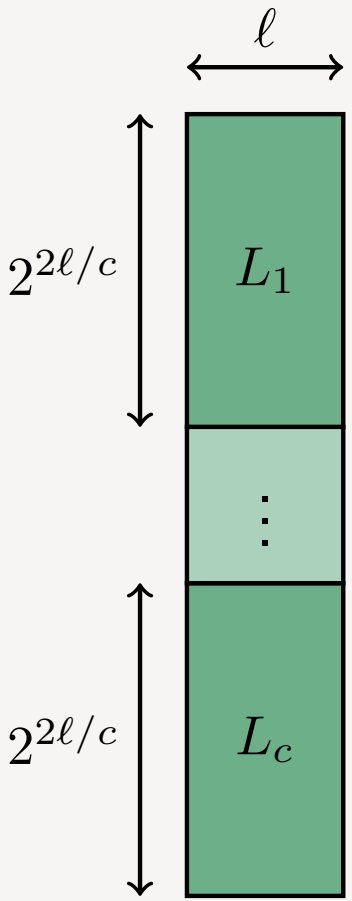
- $f_0(i_1, \ldots i_{c/2}) = L_1[i_1] + \cdots + L_{c/2}[i_{c/2}]$
- $f_1(i_{1+c/2}, \ldots, i_c) = L_{1+c/2}[i_{c/2+1}] + \cdots + L_c[i_c]$

$$f_0(i_1, \ldots, i_{c/2}) = f_1(i_{c/2+1}, \ldots, i_c)$$

$$\Downarrow$$

$$L_1[i_1] + \cdots + L_c[i_c] = 0$$

New c-sum-PCS Algorithm



$$f_0, f_1 : (\mathbb{Z}_{2^{2\ell/c}})^{c/2} \to \mathbb{F}_2^\ell$$

- $f_0(i_1, \ldots i_{c/2}) = L_1[i_1] + \cdots + L_{c/2}[i_{c/2}]$
- $f_1(i_{1+c/2}, \ldots, i_c) = L_{1+c/2}[i_{c/2+1}] + \cdots + L_c[i_c]$

$$f_0(i_1, \ldots, i_{c/2}) = f_1(i_{c/2+1}, \ldots, i_c)$$

$$\Downarrow$$

$$L_1[i_1] + \cdots + L_c[i_c] = 0$$

Find $N = c2^{2\ell/c}$ collisions between f_0 and f_1 with PCS

Complexity analysis

Assumption

Given a list L the c-sums $(x_1, \ldots, x_c) \in L^c$ behave as independent

- *Proved for* $c = 2$ [DRX17]
- *Experimentally checked for* $c = 4, c = 7$ [EHKMS18]

Complexity analysis

Assumption

Given a list L the c-sums $(x_1, \ldots, x_c) \in L^c$ behave as independent

- *Proved for* $c = 2$ [DRX17]
- *Experimentally checked for* $c = 4, c = 7$ [EHKMS18]

`c-sum-PCS`...

- Solves the c-sum problem in a list of size $c2^{2\ell/c}$
- Requires $M = \tilde{\mathcal{O}}\left(2^{2\ell/c}\right)$ memory and $T = \tilde{\mathcal{O}}\left(2^{(\frac{1}{2}+\frac{1}{c})\ell}\right)$ time

Complexity analysis

Assumption

Given a list L the c-sums $(x_1, \ldots, x_c) \in L^c$ behave as independent

- *Proved for* $c = 2$ [DRX17]
- *Experimentally checked for* $c = 4, c = 7$ [EHKMS18]

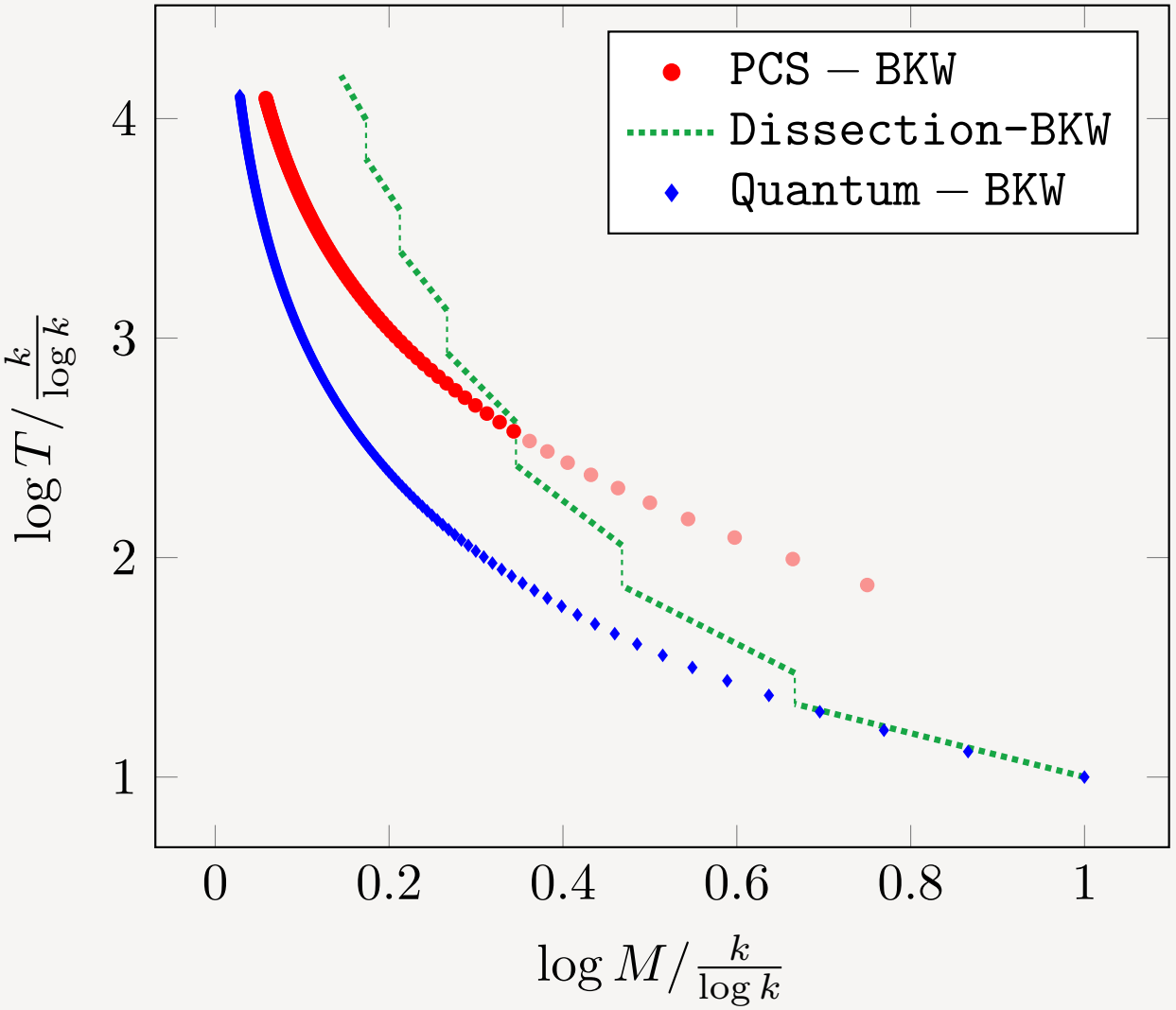
`c-sum-PCS`...

- Solves the c-sum problem in a list of size $c2^{2\ell/c}$
- Requires $M = \tilde{\mathcal{O}}\left(2^{2\ell/c}\right)$ memory and $T = \tilde{\mathcal{O}}\left(2^{(\frac{1}{2}+\frac{1}{c})\ell}\right)$ time

`PCS-BKW`...

- Solves LPN in time $T = 2^{(\frac{1}{2}+\frac{1}{c})\log c \frac{k}{\log k}(1+\varepsilon)}$ and memory $M = 2^{\frac{2}{c}\log c \frac{k}{\log k}(1+\varepsilon)}$
- Is faster than [EHKMS18] for $M < 2^{0.35\frac{k}{\log k}}$

New Trade-offs



Conclusion

This Work proposes

- Two new low-memory algorithms for subset-sum using PCS
  - `SS-PCS`: Based on BCJ memoryless algorithm works best for $M < 2^{0.02n}$
  - `SS-PCS`$_4$: works best for $2^{0.13n} < M < 2^{0.2n}$
- `PCS-BKW`: A new low-memory algorithm for LPN using PCS
  - Follows the idea of [EHKMS18]
  - Improve the c-sum routine
  - Works best for $M < 2^{0.35 \frac{k}{\log k}}$

Conclusion

This Work proposes

- Two new low-memory algorithms for subset-sum using PCS
  - `SS-PCS`: Based on BCJ memoryless algorithm works best for $M < 2^{0.02n}$
  - `SS-PCS`$_4$: works best for $2^{0.13n} < M < 2^{0.2n}$
- `PCS-BKW`: A new low-memory algorithm for LPN using PCS
  - Follows the idea of [EHKMS18]
  - Improve the c-sum routine
  - Works best for $M < 2^{0.35 \frac{k}{\log k}}$

What's next?

- Improving these trade-offs
- Applying similar ideas to other problem (e.g. Decoding, Lattice)?

Conclusion

This Work proposes

- Two new low-memory algorithms for subset-sum using PCS
  - `SS-PCS`: Based on BCJ memoryless algorithm works best for $M < 2^{0.02n}$
  - `SS-PCS`$_4$: works best for $2^{0.13n} < M < 2^{0.2n}$
- `PCS-BKW`: A new low-memory algorithm for LPN using PCS
  - Follows the idea of [EHKMS18]
  - Improve the c-sum routine
  - Works best for $M < 2^{0.35 \frac{k}{\log k}}$

What's next?

- Improving these trade-offs
- Applying similar ideas to other problem (e.g. Decoding, Lattice)?

Thank you for your time!