Linear-Time Zero-Knowledge Arguments in Practice

Zero-Knowledge Arguments

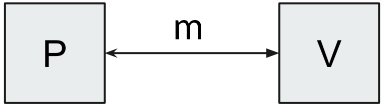
Argument

$$\boxed{P} \xleftrightarrow{\;m\;} \boxed{V}$$

$f(x) = y$?

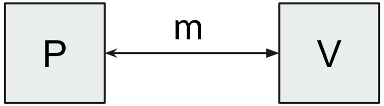
Zero-Knowledge Arguments

Argument



$f(x) = y$?

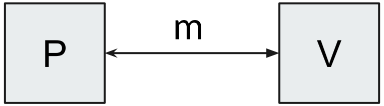
Polynomial Commitment



$p(x) = y$?

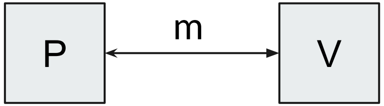
Zero-Knowledge Arguments

Argument



$f(x) = y$?

Polynomial Commitment



$p(x) = y$?

- ZK property: verifier learns nothing extra about f/p
- Linear time: If polynomial p has N coefficients, prover runs in O(N) time.

Contents

- Low-dimensional (t = 2) Polynomial Commitment in *Brakedown*
- High-dimensional (t = 3, 4, 5, …) Polynomial Commitment [BCG20, BCL22]
- Zero-Knowledge Polynomial Commitment
    - Simple Modified Construction
    - Zero-Knowledge Linear Code

Polynomial Commitment

A Polynomial Commitment consists of three algorithms:

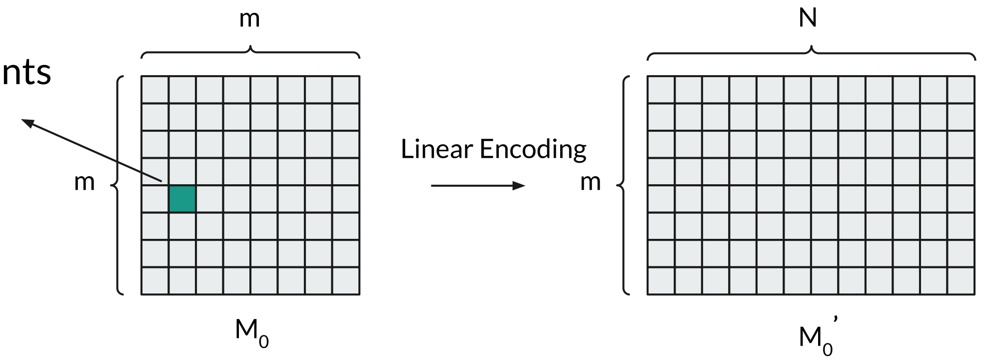
- PC.COMMIT$(\phi(\cdot))$:*the algorithm outputs a commitment* $\mathcal{R}$ *of the polynomial* $\phi(\cdot)$.
- PC.PROVE$(\phi, x, \mathcal{R})$: *given an evaluation point* x*, the algorithm outputs a tuple* $(x, \phi(x), \pi_x)$*, where* π_x *is the proof.*
- PC.VERIFY$(\pi_x, x, \phi(x), \mathcal{R})$:*given* $\pi_x, x, \phi(x), \mathcal{R}$*, the algorithm checks if* $\phi(x)$ *is the correct evaluation. The algorithm outputs* ACCEPT *or* REJECT.

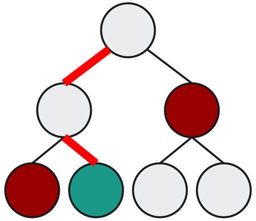
2-dimensional Polynomial Commitment in Brakedown

Commitment Phase:

IOP Model
Merkle Tree Commitment

polynomial coefficients





- Encoding Function: $F^m \rightarrow F^N$
- Relative Distance: the minimum distance between any two valid codeword divided by code length N

Polynomial Commitment

A Polynomial Commitment consists of three algorithms:

- PC.COMMIT($\phi(\cdot)$):*the algorithm outputs a commitment* $\mathcal{R}$ *of the polynomial* $\phi(\cdot)$.
- PC.PROVE($\phi, x, \mathcal{R}$): *given an evaluation point* x, *the algorithm outputs a tuple* $(x, \phi(x), \pi_x)$, *where* π_x *is the proof.*
- PC.VERIFY($\pi_x, x, \phi(x), \mathcal{R}$):*given* $\pi_x, x, \phi(x), \mathcal{R}$, *the algorithm checks if* $\phi(x)$ *is the correct evaluation. The algorithm outputs* ACCEPT *or* REJECT.

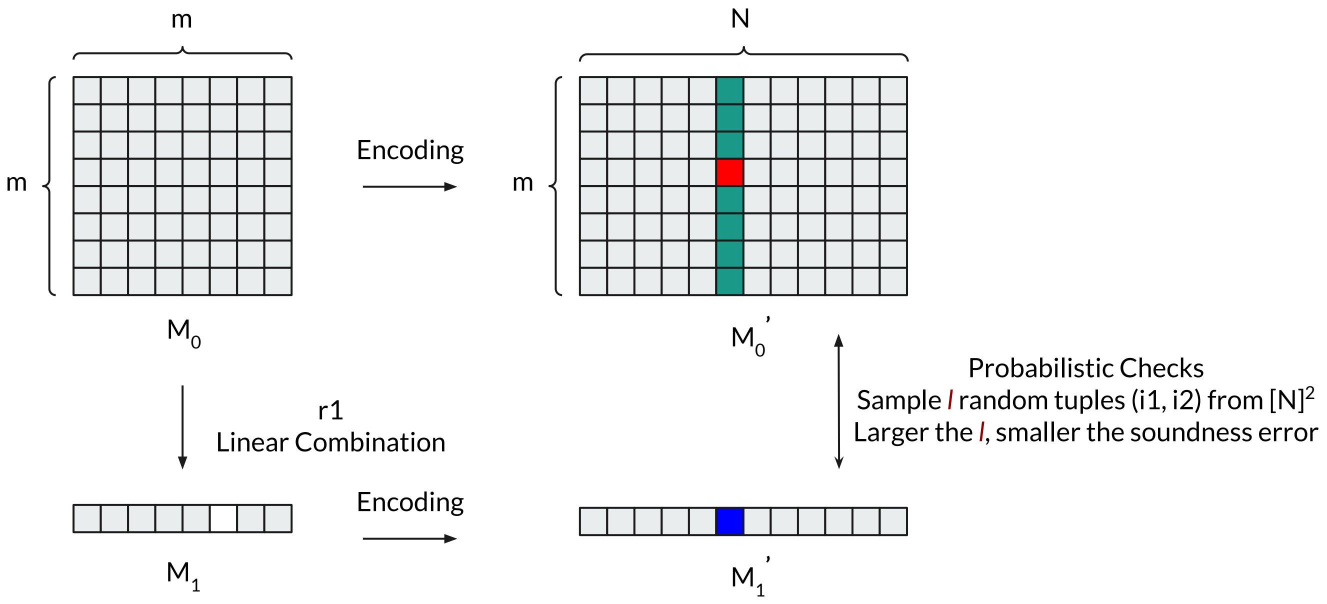
Polynomial Commitment

A Polynomial Commitment consists of three algorithms:

- PC.COMMIT$(\phi(\cdot))$:*the algorithm outputs a commitment* $\mathcal{R}$ *of the polynomial* $\phi(\cdot)$.
- PC.PROVE$(\phi, x, \mathcal{R})$: *given an evaluation point* x, *the algorithm outputs a tuple* $(x, \phi(x), \pi_x)$, *where* π_x *is the proof.*
- PC.VERIFY$(\pi_x, x, \phi(x), \mathcal{R})$:*given* $\pi_x, x, \phi(x), \mathcal{R}$, *the algorithm checks if* $\phi(x)$ *is the correct evaluation. The algorithm outputs* ACCEPT *or* REJECT.

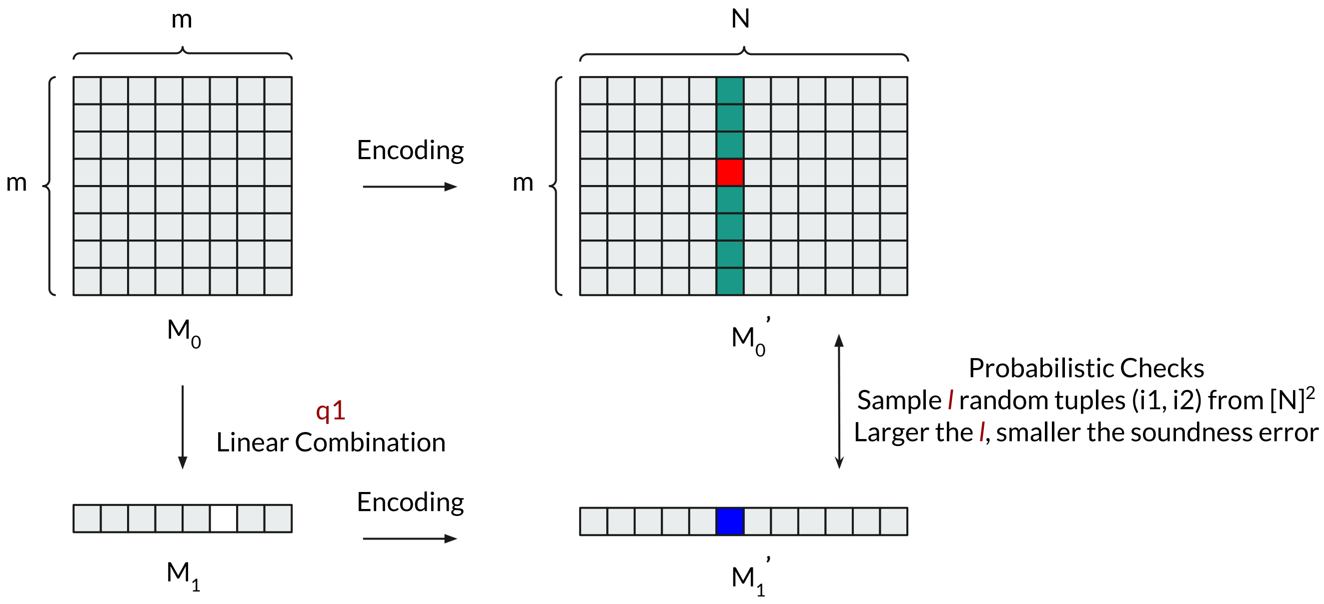
2-dimensional Polynomial Commitment in Brakedown

Testing Phase:



2-dimensional Polynomial Commitment in Brakedown

Evaluation Phase:

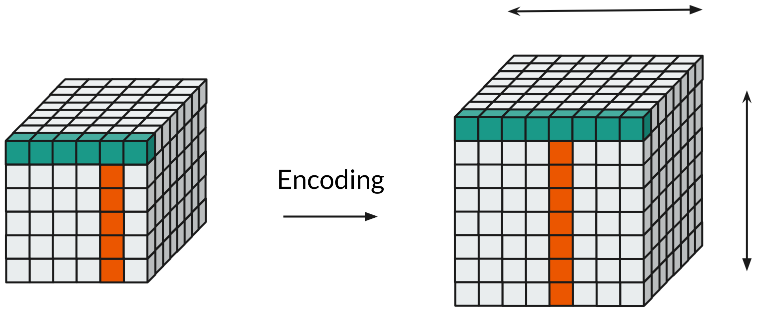


Contents

- Low-dimensional (t = 2) Polynomial Commitment in *Brakedown*
- **High-dimensional (t = 3, 4, 5, …) Polynomial Commitment [BCG20, BCL22]**
- Zero-Knowledge Polynomial Commitment
    - Simple Modified Construction
    - Zero-Knowledge Linear Code

t-dimensional (t=3) Polynomial Commitment

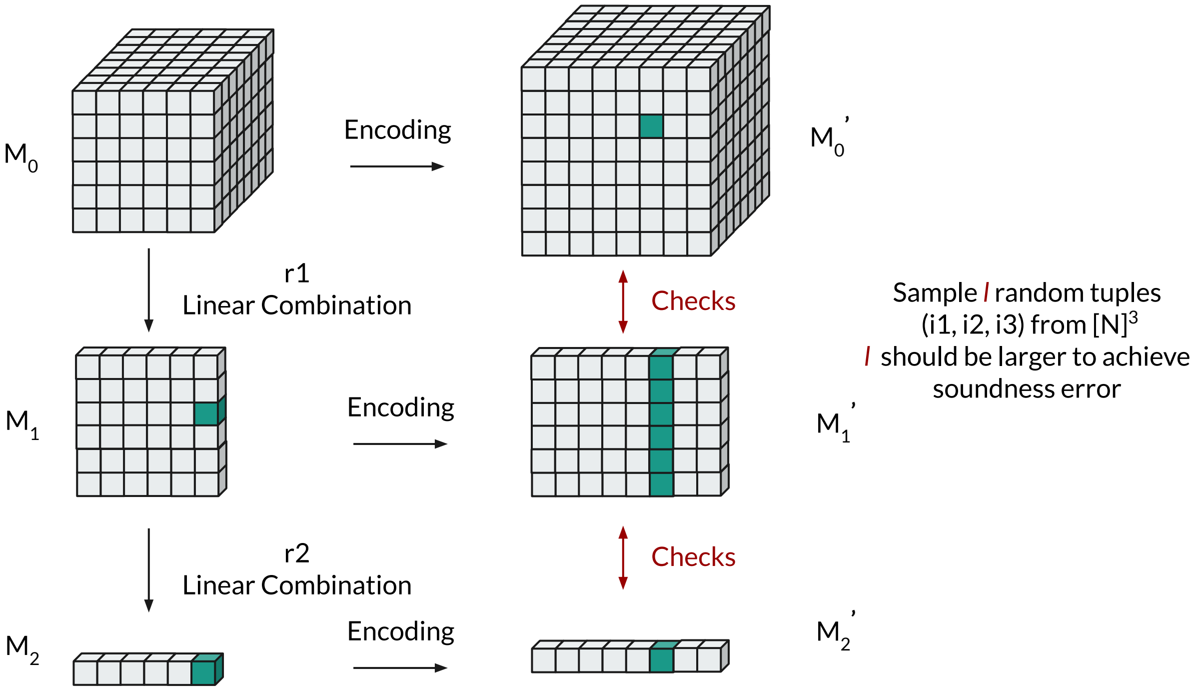
Commitment Phase:



Tensor Code
Relative distance:Δ^t

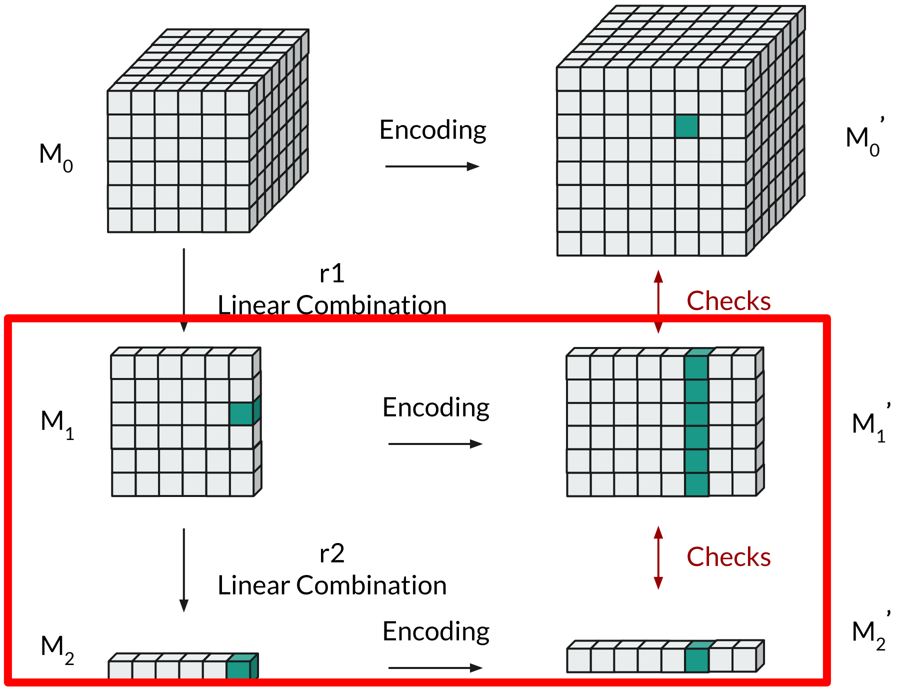
t-dimensional (t=3) Polynomial Commitment

Testing Phase:



t-dimensional (t=3) Polynomial Commitment

Testing Phase:

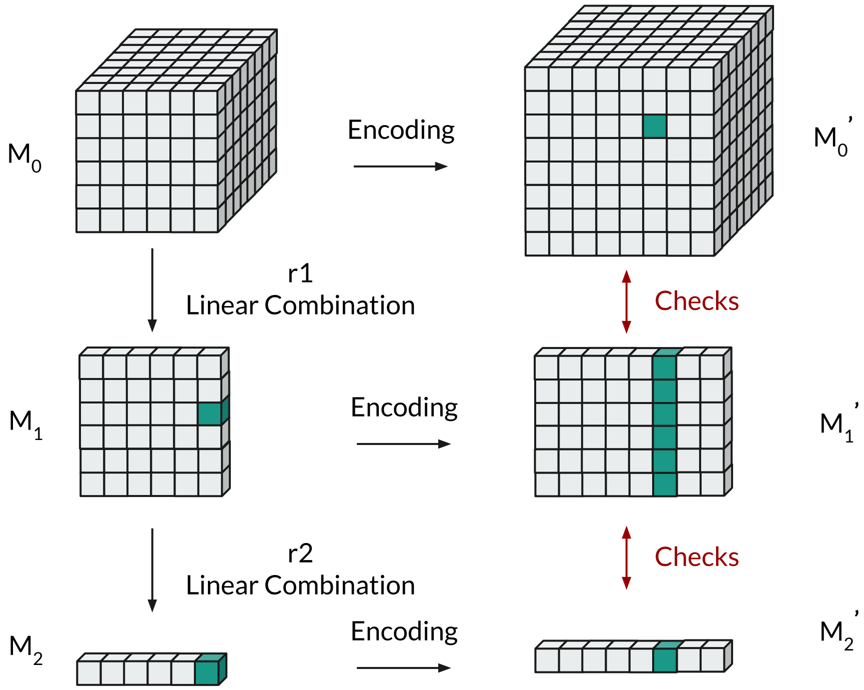


Sample l random tuples (i1, i2, i3) from $[N]^3$

l should be larger to achieve soundness error

t-dimensional (t=3) Polynomial Commitment

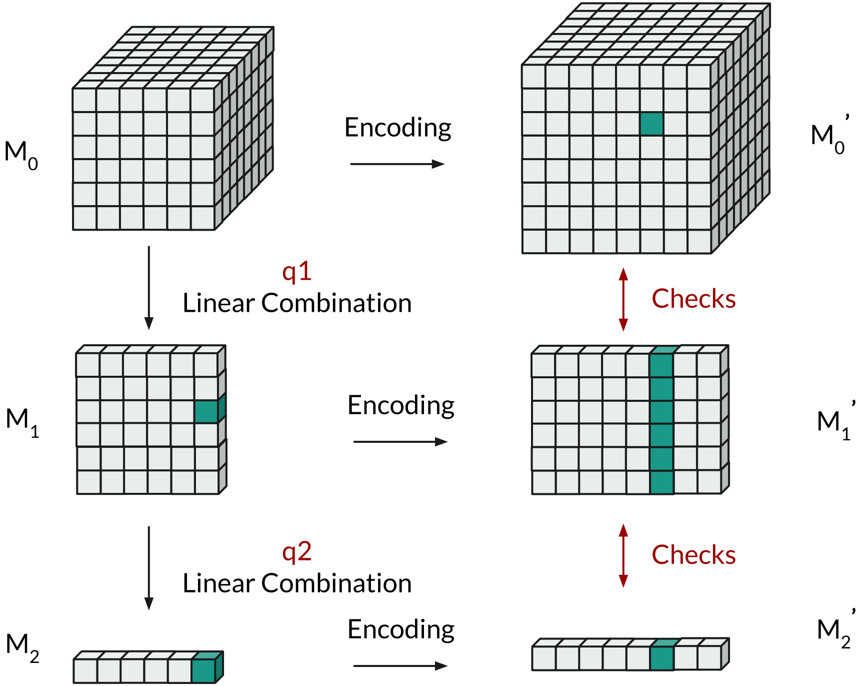
Testing Phase:



Sample *l* random tuples (i1, i2, i3) from $[N]^3$
l should be larger to achieve soundness error

t-dimensional (t=3) Polynomial Commitment

Evaluation Phase:



Sample *l* random tuples (i1, i2, i3) from $[N]^3$

l should be larger to achieve soundness error

Benchmark

Dimension	Message Length	Code Length	Commit Time [ms]	Verify Time [ms]	Soundness Error	Communication Complexity [Field Element]
2	1024	1762	41737	3057	0.37	1206579
3	101	174	99642	623	1.76	235621
4	32	56	153558	204	1.98	114701

Table 2.1: Runtime of polynomial commitment scheme with 2^{20} coefficients, 1 threads, linear code with relative distance 0.07, and 1000 test tuples.

Dimension	Message Length	Code Length	Commit Time [ms]	Verify Time [ms]	Soundness Error	Communication Complexity [Field Element]
2	1024	1762	10048	776	0.37	1206579
3	101	174	24314	165	1.76	235621
4	32	56	37961	63	1.98	114701

Table 2.2: Runtime of polynomial commitment scheme with 2^{20} coefficients, 8 threads, linear code with relative distance 0.07, and 1000 test tuples.

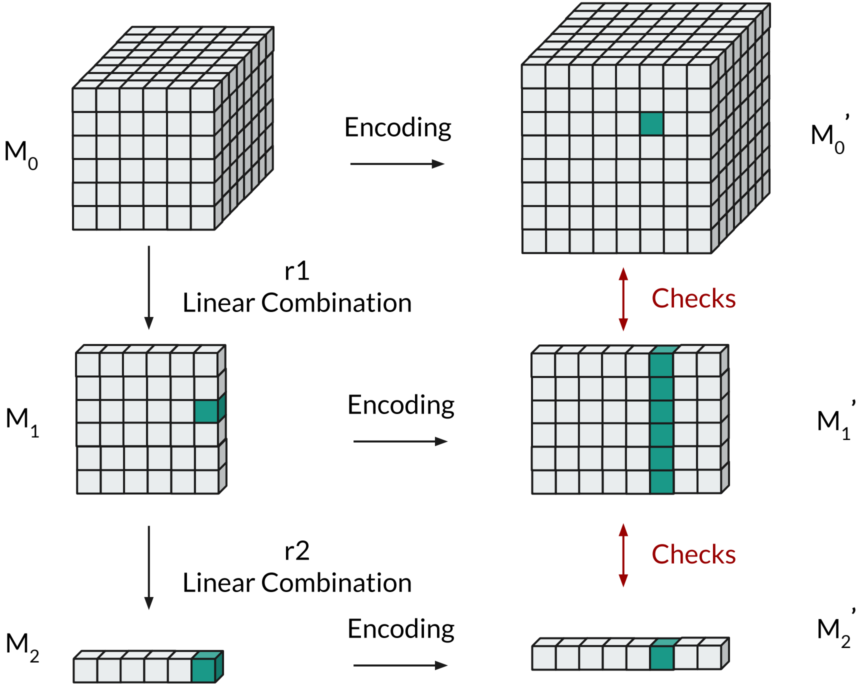
Contents

- Low-dimensional (t = 2) Polynomial Commitment in *Brakedown*
- High-dimensional (t = 3, 4, 5, …) Polynomial Commitment [BCG20, BCL22]
- Zero-Knowledge Polynomial Commitment
  - **Simple Modified Construction**
  - Zero-Knowledge Linear Code

t-dimensional (t=3) Polynomial Commitment

Testing Phase:

(Proximity Test)

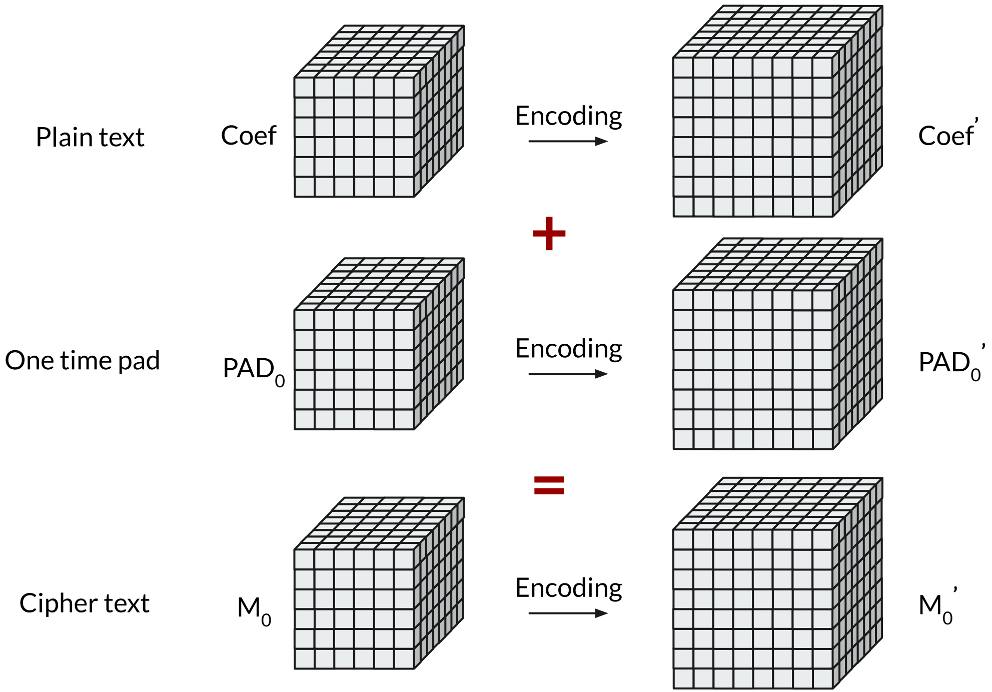


Sample *l* random tuples (i1, i2, i3) from $[N]^3$
l should be larger to achieve soundness error

Simple Zero-Knowledge Construction

Commitment Phase:

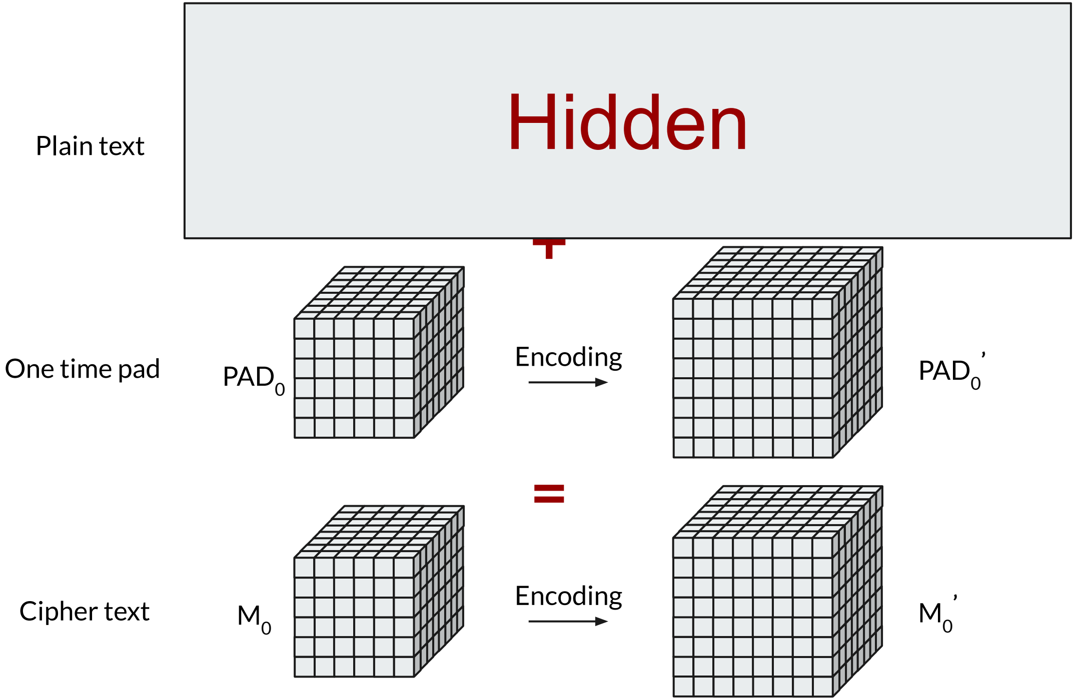
[BCGGHJ17][XZS22]



Simple Zero-Knowledge Construction

Testing Phase:

[BCGGHJ17][XZS22]



Simple Zero-Knowledge Construction

- Simple to understand
- Easy to implement
- Do not require any special property from the linear code

- Have restrictions on the queries the verifier can make
- Increase the prover time/verifier time/proof size by a factor of 2

Contents

- Low-dimensional (t = 2) Polynomial Commitment in *Brakedown*
- High-dimensional (t = 3, 4, 5, …) Polynomial Commitment [BCG20, BCL22]
- Zero-Knowledge Polynomial Commitment
    - Simple Modified Construction
    - **Zero-Knowledge Linear Code**

Zero-Knowledge Linear Code

- Encoding Function: F^k -> F^n
- Linearity: any linear combination of codeword is also a codeword
- Distance: the distance between any two valid codeword
- **Zero-knowledge:**

  It looks random, if only access a few positions in the codeword

General Transformation [DI14][BCL22]
From Linear Code to Zero-knowledge Linear Code

$$x \xrightarrow{\text{Encoding}} a \xrightarrow{\text{Redistribution}} b \xrightarrow{\text{Randomization}} c \xrightarrow{\text{Reverse Encoding}} y$$

Step 1: Encoding

$$x \xrightarrow{\text{Encoding}} a \xrightarrow{\text{Redistribution}} b \xrightarrow{\text{Randomization}} c \xrightarrow{\text{Reverse Encoding}} y$$

Brakedown linear code encoding function

Step 2: Redistribution

Lemma 5.1 *For any $0 < \epsilon < 1$, there exist a degree d such that a random d-regular bipartite graph $G = (L, R, E)$ with $|L| = |R| = n$ generated according to algorithm 1 satisfy the following property with high probability.*

- *Expansion: For every set $X \subset L$ with $|X| \geq \epsilon n$, if Y is the set of neighbors of X in G, then $|Y| \geq (1 - \epsilon)n$.*

$$x \xrightarrow{\text{Encoding}} a \xrightarrow{\text{Redistribution}} b \xrightarrow{\text{Randomization}} c \xrightarrow{\text{Reverse Encoding}} y$$

Redistribute the elements in **a** according to a expander graph

Step 3: Randomization

$$x \xrightarrow{\text{Encoding}} a \xrightarrow{\text{Redistribution}} b \xrightarrow{\text{Randomization}} c \xrightarrow{\text{Reverse Encoding}} y$$

$$c = b * H$$

$$\begin{pmatrix} 1, 2, 0, 0, 0, 0 \\ 3, 4, 0, 0, 0, 0 \\ 0, 0, 5, 6, 0, 0 \\ 0, 0, 7, 8, 0, 0 \\ 0, 0, 0, 0, 9, 1 \\ 0, 0, 0, 0, 2, 3 \end{pmatrix}$$

Step 4: Reverse Encoding

$$x \xrightarrow{\text{Encoding}} a \xrightarrow{\text{Redistribution}} b \xrightarrow{\text{Randomization}} c \xrightarrow{\text{Reverse Encoding}} y$$

Brakedown Linear Code

Randomness Extractor

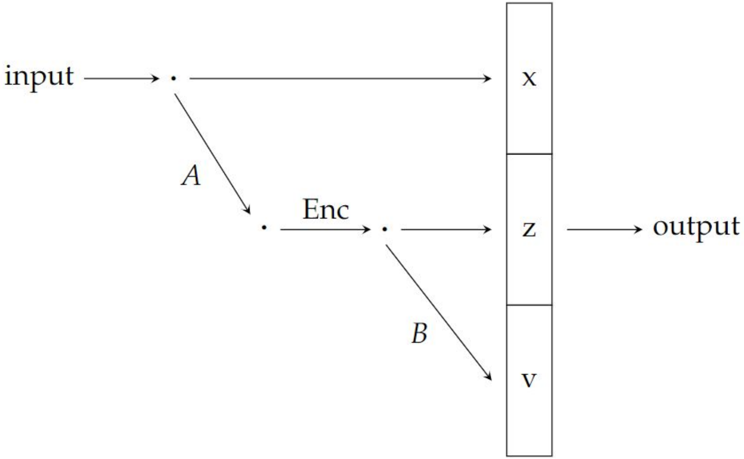


Figure 5.1: Linear Code

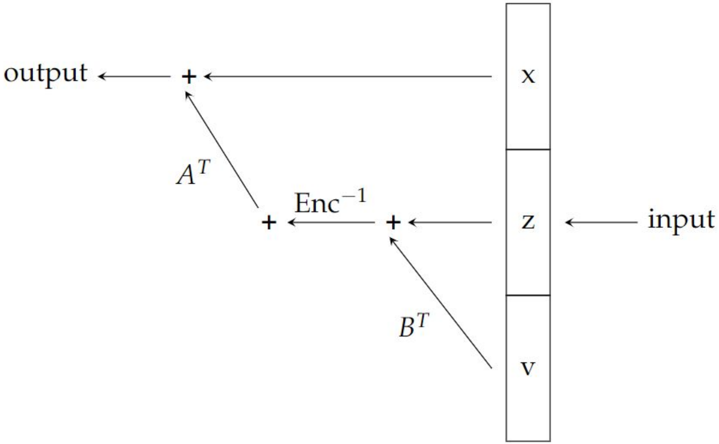


Figure 5.2: Reversed Linear Code

Efficiency Bottleneck

Lemma 5.1 *For any $0 < \epsilon < 1$, there exist a degree d such that a random d-regular bipartite graph $G = (L, R, E)$ with $|L| = |R| = n$ generated according to algorithm 1 satisfy the following property with high probability.*

- *Expansion: For every set $X \subset L$ with $|X| \geq \epsilon n$, if Y is the set of neighbors of X in G, then $|Y| \geq (1 - \epsilon)n$.*

$$X \xrightarrow{\text{Encoding}} A \xrightarrow{\text{Redistribution}} B \xrightarrow{\text{Randomization}} C \xrightarrow{\text{Reverse Encoding}} Y$$

How to generate expander graph?

- Explicit Construction: slow, hard to find
- **Random graph**

Sample a random graph

Algorithm 1: Random d-regular Bipartite Graph Generation

```
Data: n ≥ 0, d <= n
Result: A random d-regular bipartite graph G = (L, R, E) with
        |L| = |R| = n
L ← a set of n nodes;
R ← a set of n nodes;
E ← ∅;
P ← [1, 2, ⋯ , n];
for i in 1, 2, ⋯ , d do
    Permute P randomly ;        /* sample a perfect matching */
    for j in 1, 2, ⋯ , n do
        E ← E ∪ (L_j, R_{P_j}) ;
    end
end
return (L, R, E)
```

Probability: a random graph is not a expander graph

$$
\begin{aligned}
P_3 &= \binom{n}{\epsilon n}(P_2)^{\epsilon n} \\
&= \binom{n}{\epsilon n}\left(\frac{n-1}{n}\right)^{d\epsilon^2 n^2} \\
&\le 2^{nH(\frac{\epsilon n}{n})}\left(\frac{n-1}{n}\right)^{d\epsilon^2 n^2} \qquad \binom{n}{k} \le 2^{nH(\frac{k}{n})} \\
&= 2^{nH(\epsilon)}\left(\left(1-\frac{1}{n}\right)^{n}\right)^{d\epsilon^2 n} \\
&\le 2^{nH(\epsilon)}\left(\frac{1}{e}\right)^{d\epsilon^2 n} \qquad \left(1-\frac{1}{x}\right)^{x} \le \frac{1}{e} \text{ for } x \ge 1 \text{ (lemma A.2)} \\
&= \left(e^{H(\epsilon)\ln 2 - d\epsilon^2}\right)^{n}
\end{aligned}
$$

For example, if $\epsilon = 0.05$, $n = 5000$, $p = 2^{-256}$, then degree d need to be greater than 93.60.

Transforming Linear Code

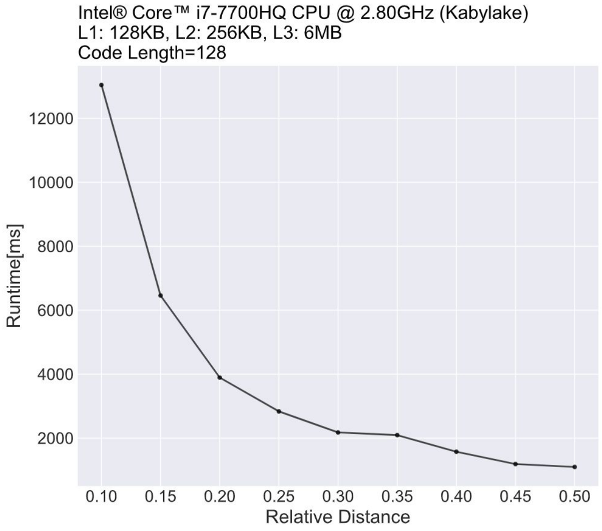


Figure 5.3: Runtime of Redistribution and Randomization Step

Transforming Linear Code

Relative distance in Brakedown:
0.02 ~ 0.07

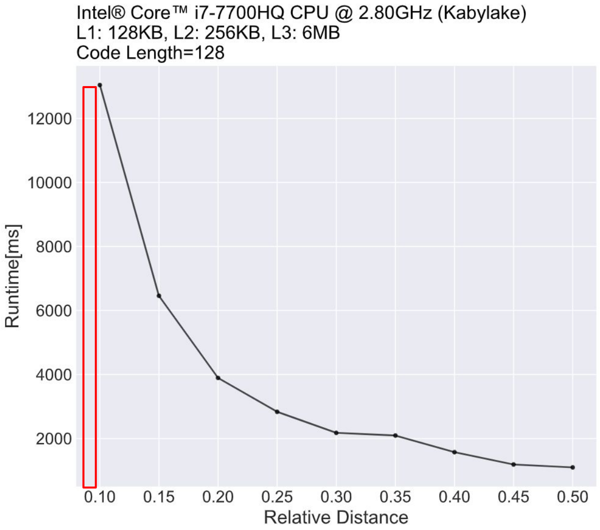


Figure 5.3: Runtime of Redistribution and Randomization Step

General Transformation [DI14]
From Linear Code to Zero-knowledge Linear Code

- Simple to understand
- No restrictions on verifier queries
- Zero-Knowledge Property
- Better Distance Property

- Require zero-knowledge property from the linear code
- Inefficient in practice
- Hard to implement

Conclusions

- High dimension polynomial commitment scheme is not worth using unless we can improve the relative distance of these linear code used in the constructions. However, improving relative distance seems to be a difficult task.
- Using the method similar to one-time-pad encryption, we can add zero-knowledge property into the polynomial commitment scheme. And it is practical and efficient compared with the alternative approach.

Remaining Goals

- Find a better way to generate a good expander graph
- Apply this commitment scheme to a real system and benchmark the performance

Thanks