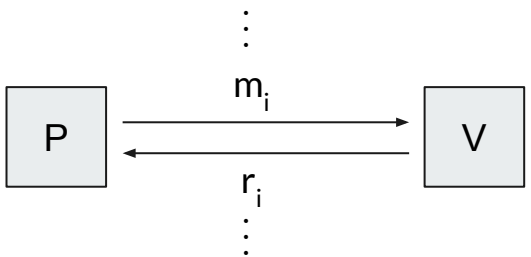
Linear-Time Zero-Knowledge Arguments in Practice

Contents

Proof System Models

- Interactive proofs (IPs)

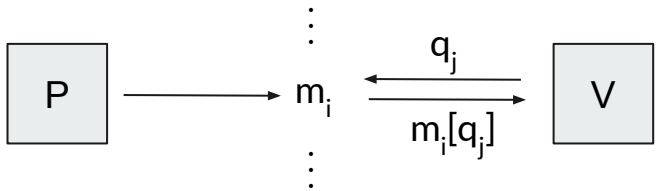
  Verifier exchanges ***k***-rounds of messages with a prover, and then either accepts or rejects the statements.



Proof System Models

- Interactive oracle proofs (IOPs)

  Verifier has oracle access to the prover's messages and may query them on a few positions probabilistically (rather than having to read the proof string in full).

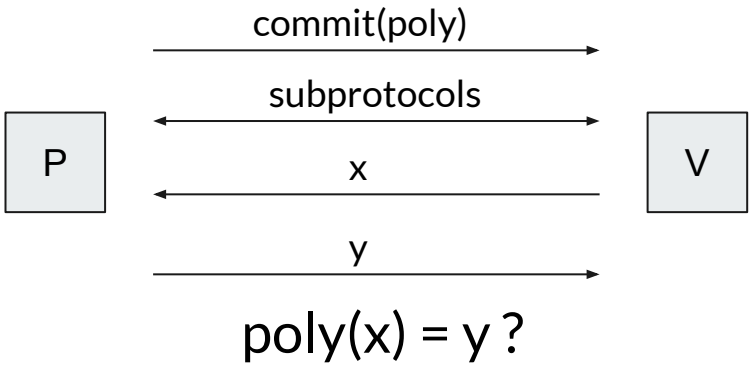


Polynomial Commitment Scheme

- Many interesting real-world statements can be reduced into a polynomial and the proof of such a statement can be converted to an evaluation of a specific point.
- Polynomial commitments can be used to compile polynomial-query IOPs into arguments.
- Linear time polynomial commitment schemes can imply a linear-time proof system.

Polynomial Commitment Scheme

- Prover commits to a secret polynomial and convinces the verifier that the evaluation result of the committed secret polynomial is correct after several rounds of communication later.



Polynomial Commitment Scheme

- Zero-knowledge

  Verifier learns nothing extra about the polynomial except the evaluation result.

  Zero-knowledge polynomial commitment schemes can imply a Zero-knowledge proof system.

- Linear time

  If the polynomial has **N** coefficients, prover runs in **O(N)** time.

  Linear-time polynomial commitment schemes can imply a linear-time proof system.

Linear Code

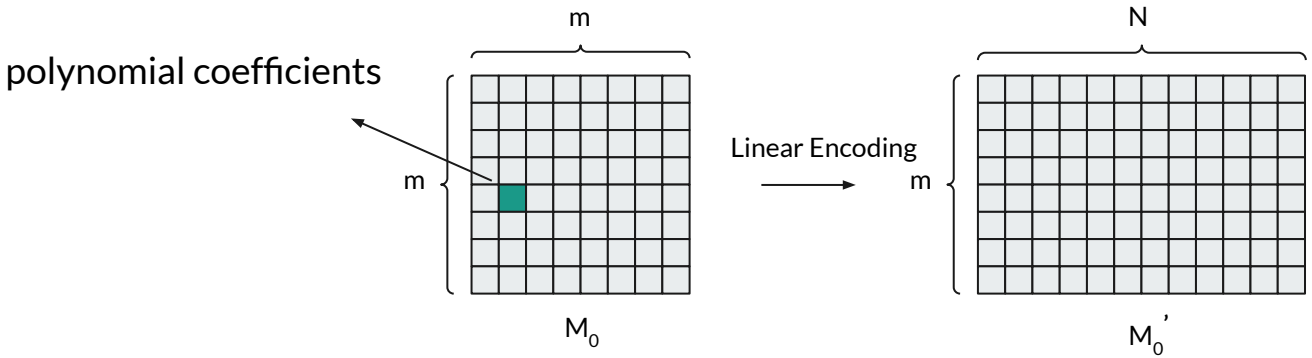
- The encoding function of a linear code will map a message in F^m to a codeword in F^N
- Any linear combination of codewords is also a codeword.
- Relative Distance: the minimum distance between any two valid codeword divided by code length N.

Linear Code in Proof Systems

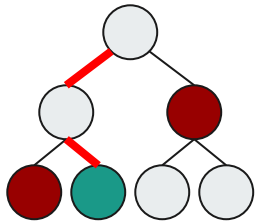
- Linear codes are essential in many proof systems.
- They rely on special families of linear codes, whose structures and properties influence the overall performance of the proof systems.
- Our polynomial commitment scheme uses linear code.

2-dimensional Polynomial Commitment in Brakedown

Commitment Phase:



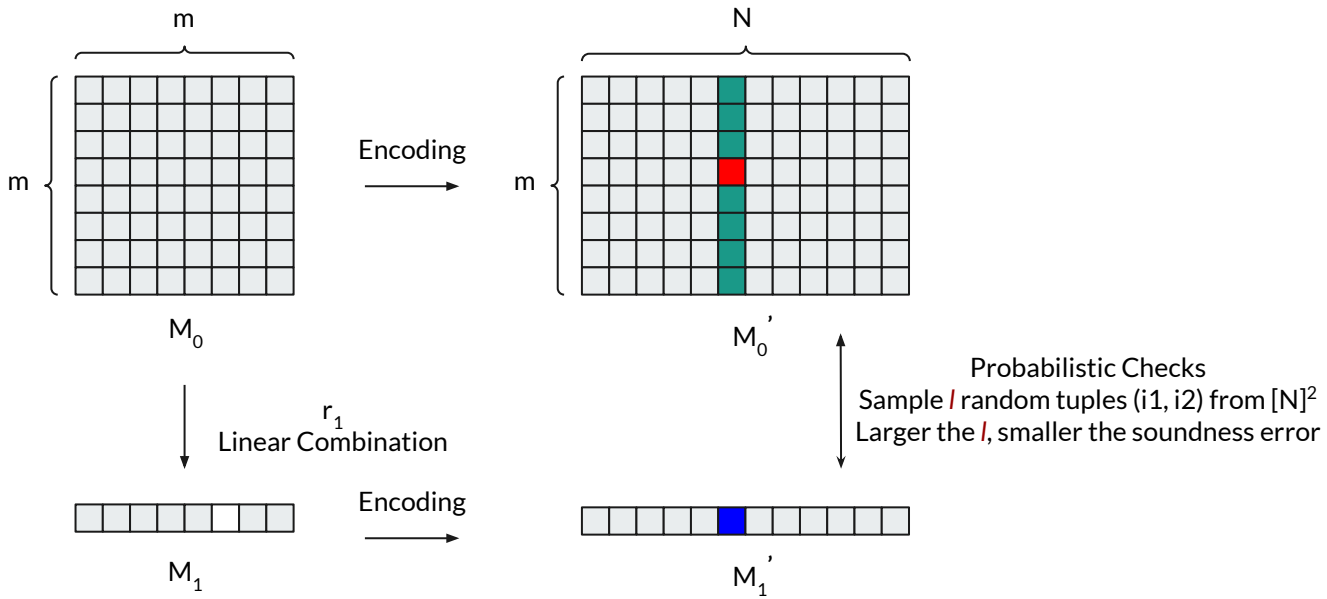
IOP Model
Merkle Tree Commitment



2-dimensional Polynomial Commitment in Brakedown

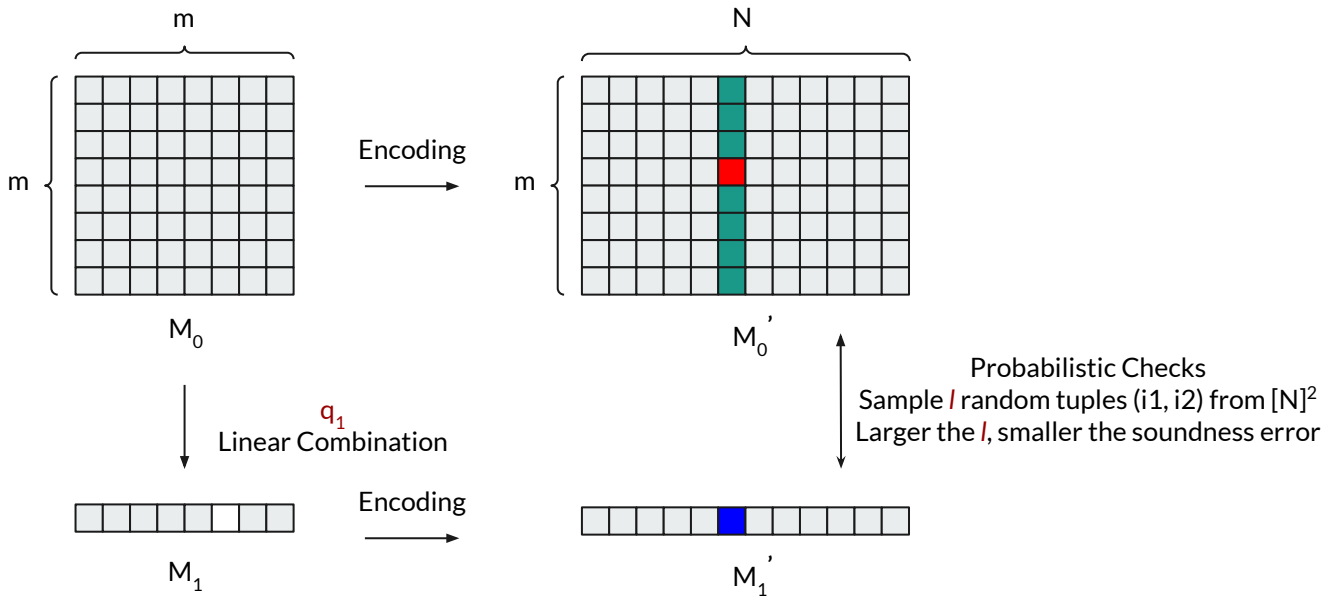
Testing Phase:

(Proximity Test)



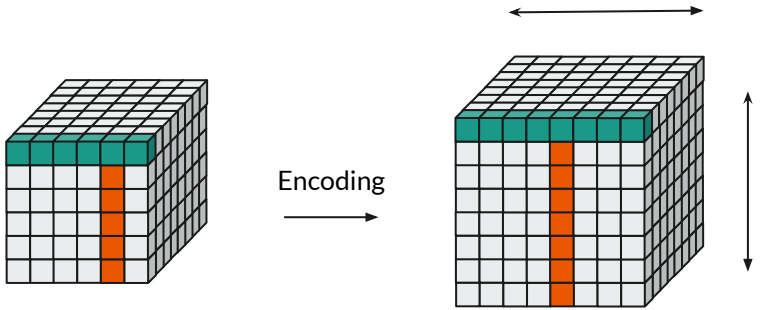
2-dimensional Polynomial Commitment in Brakedown

Evaluation Phase:



t-dimensional (t=3) Polynomial Commitment [BCG20, BCL22]

Commitment Phase:

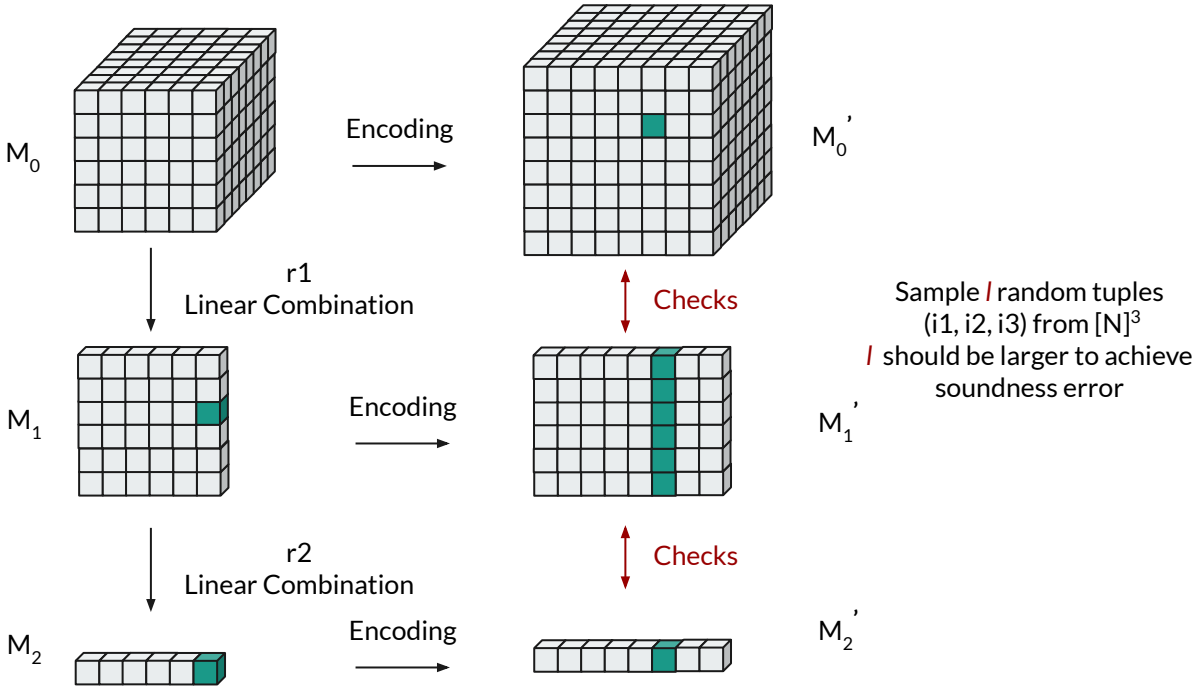


Tensor Code
Relative distance:Δ^t

t-dimensional (t=3) Polynomial Commitment

Testing Phase:

(Proximity Test)



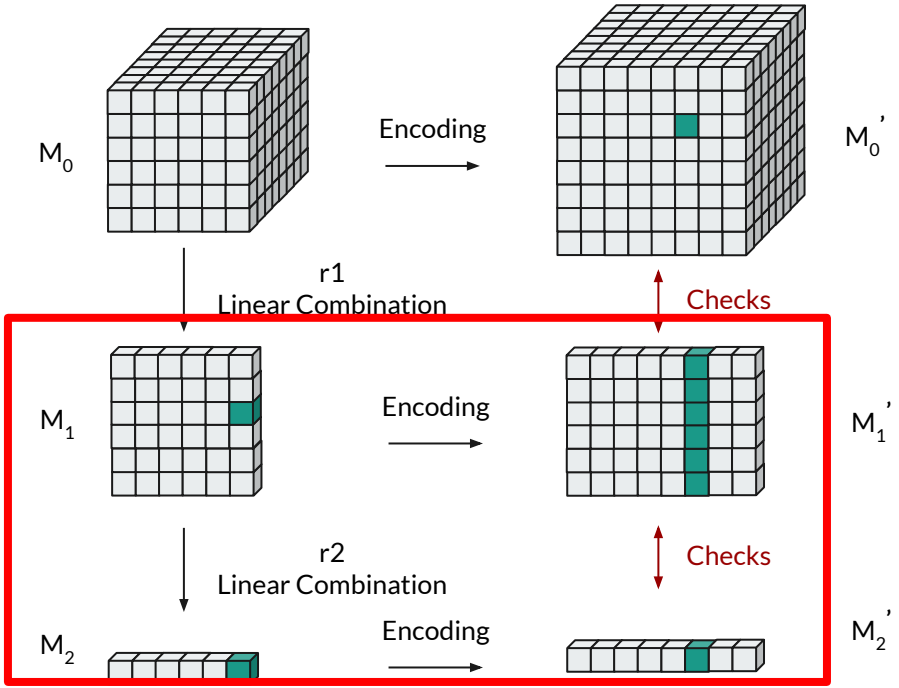
Sample *l* random tuples (i1, i2, i3) from $[N]^3$

l should be larger to achieve soundness error

t-dimensional (t=3) Polynomial Commitment

Testing Phase:

(Proximity Test)

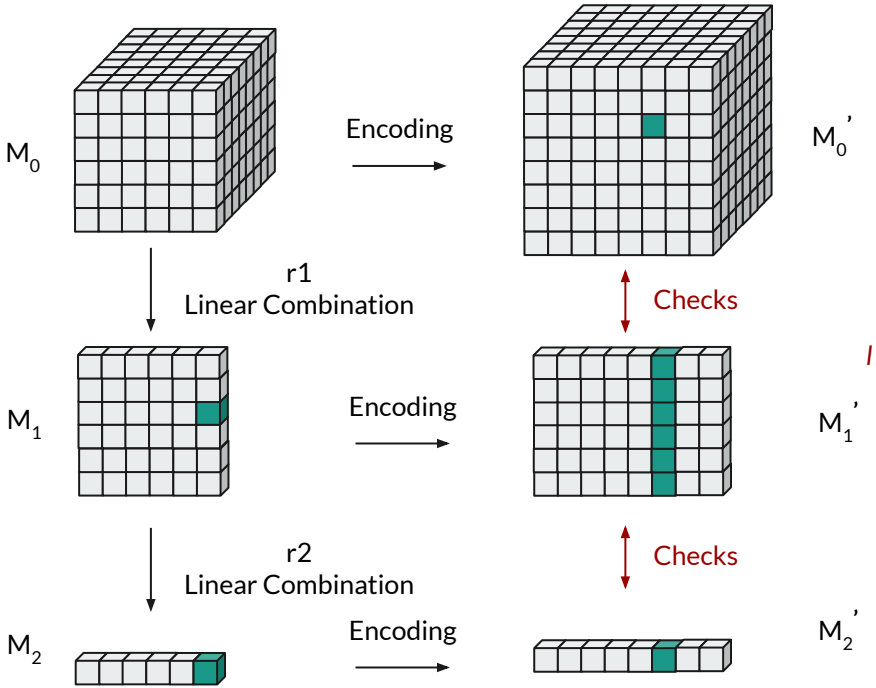


Sample *l* random tuples (i1, i2, i3) from $[N]^3$
l should be larger to achieve soundness error

t-dimensional (t=3) Polynomial Commitment

Testing Phase:

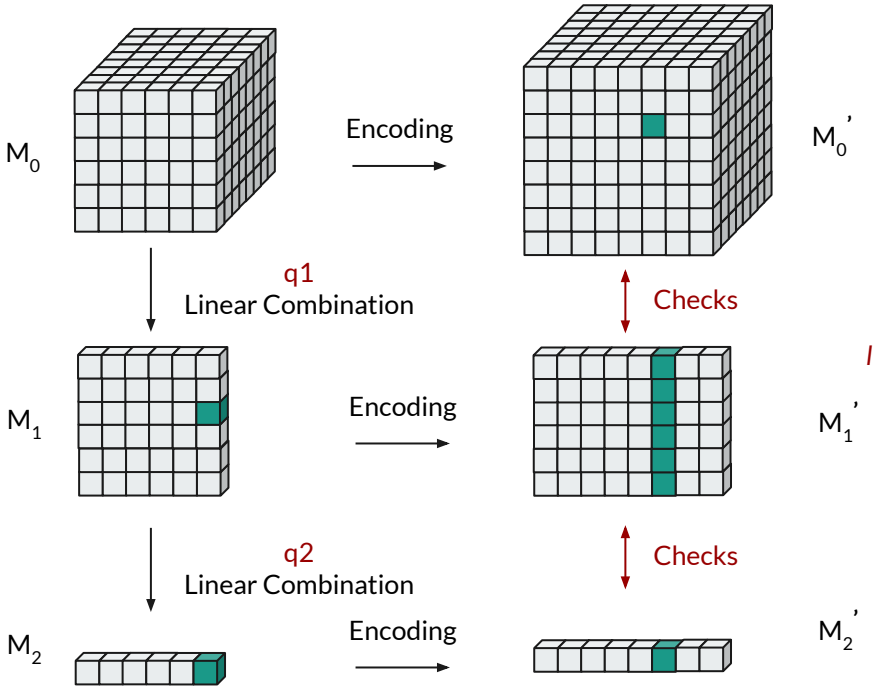
(Proximity Test)



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

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.

Conclusion

- 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.

Zero-Knowledge Polynomial Commitment

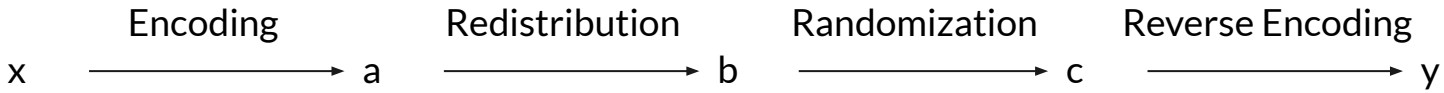
- **Zero-Knowledge Linear Code**
- Simple Modified Construction

Zero-Knowledge Linear Code

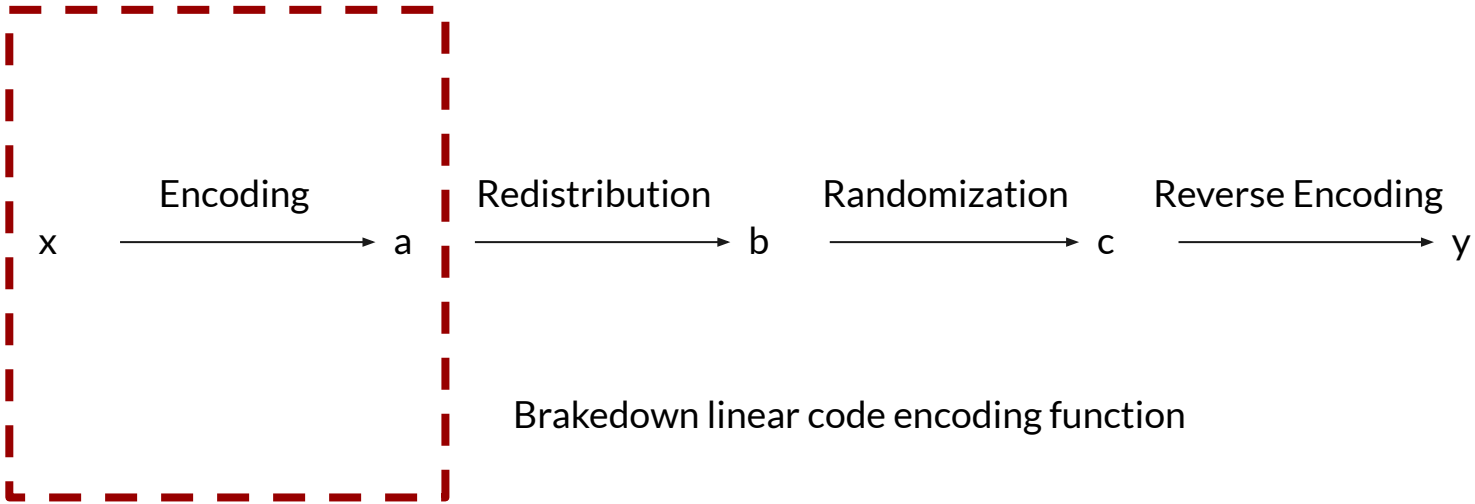
- Zero-knowledge:

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

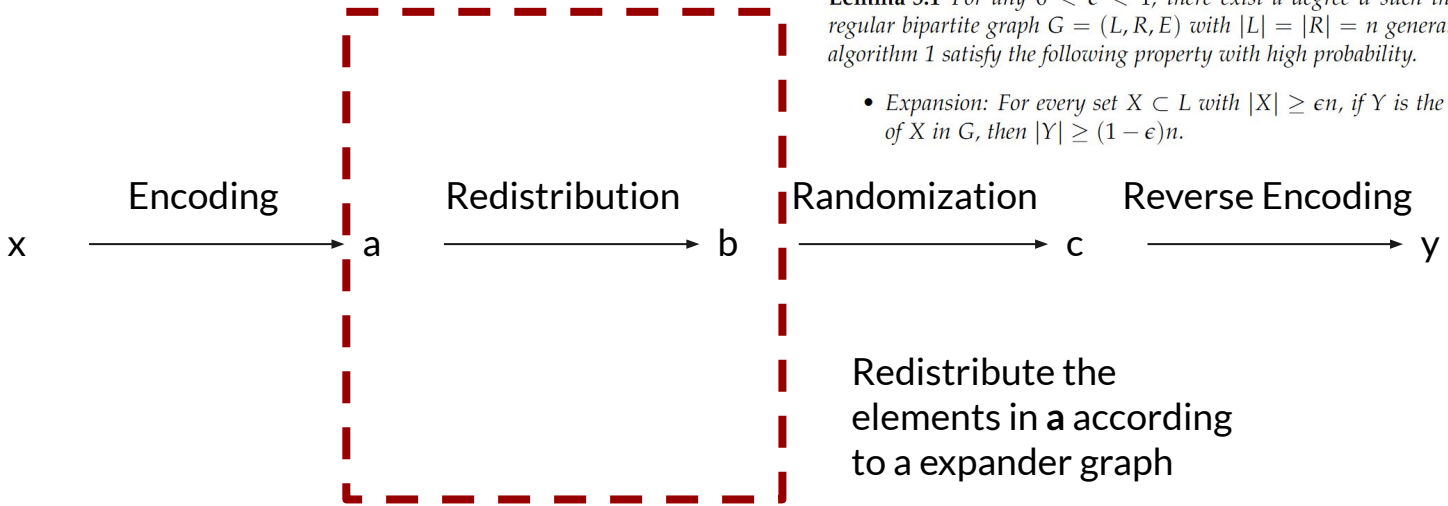
General Transformation [DI14][BCL22]
From Linear Code to Zero-knowledge Linear Code



Step 1: Encoding



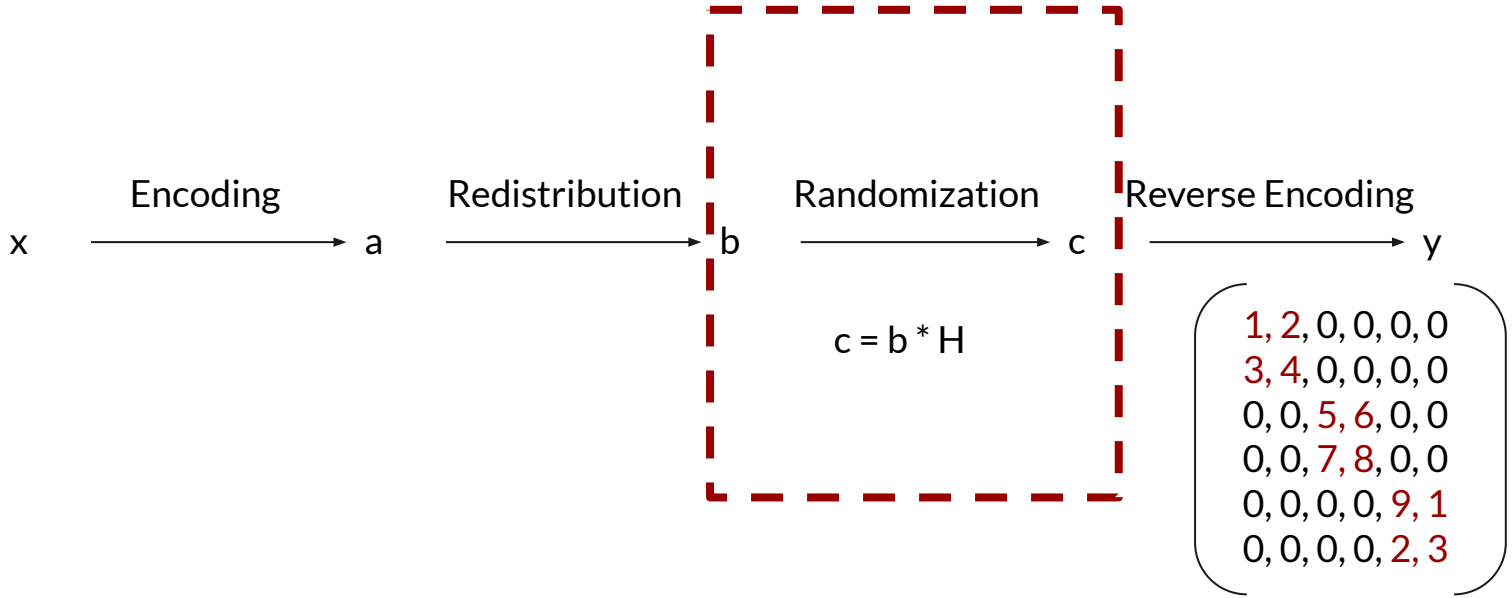
Step 2: Redistribution



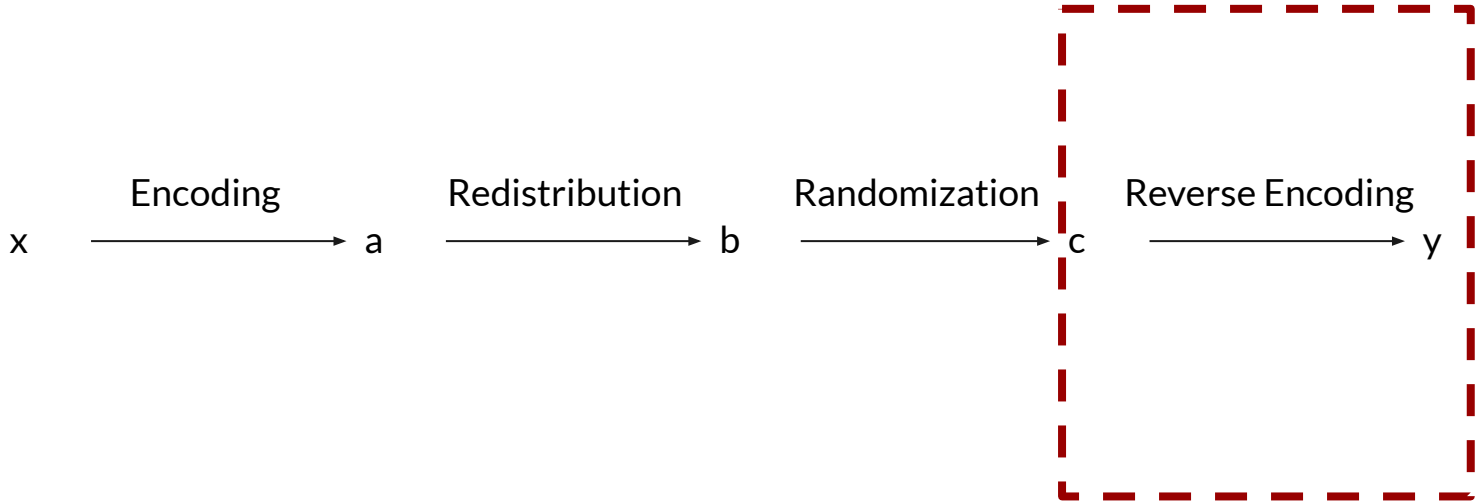
Lemma 5.1 *For any $0 < \epsilon < 1$, there exist a degree d such that a random d-regular bipartite graph $G = (L, R, E)$ with $|L| = |R| = n$ generated according to algorithm 1 satisfy the following property with high probability.*

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

Step 3: Randomization



Step 4: Reverse Encoding



Brakedown Linear Code

Randomness Extractor

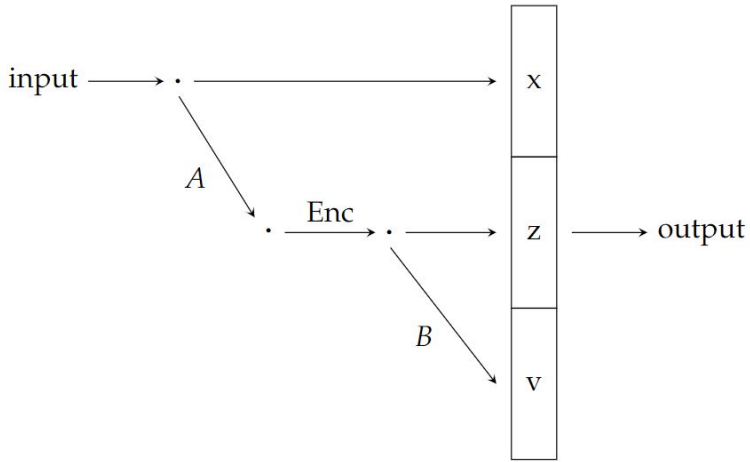


Figure 5.1: Linear Code

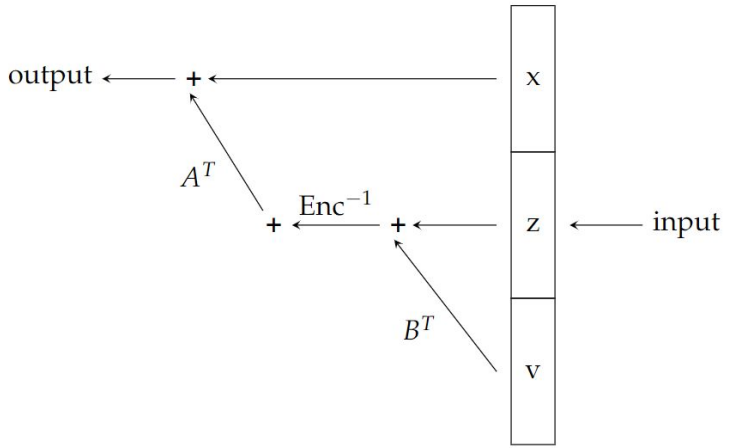
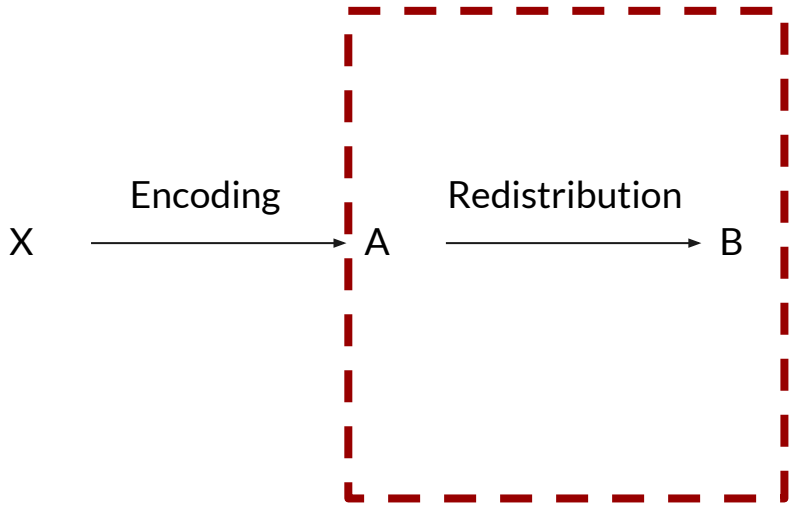


Figure 5.2: Reversed Linear Code

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 $|Y| \geq (1 - \epsilon)n$.*

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

Probability: a random graph is not a expander graph

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

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

Transforming Linear Code

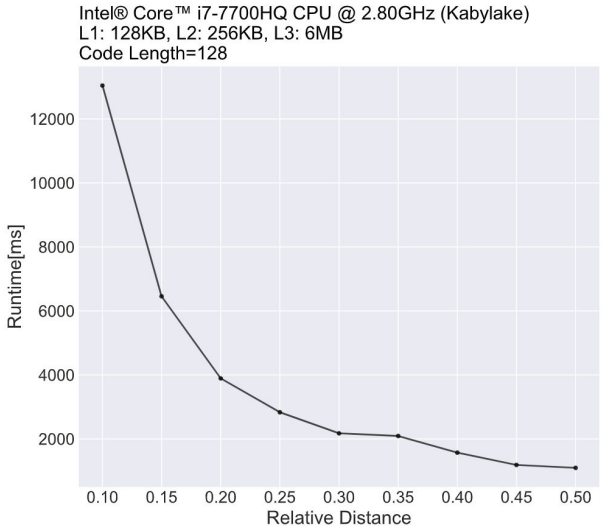


Figure 5.3: Runtime of Redistribution and Randomization Step

Transforming Linear Code

Relative distance in Brakedown:
0.02 ~ 0.07

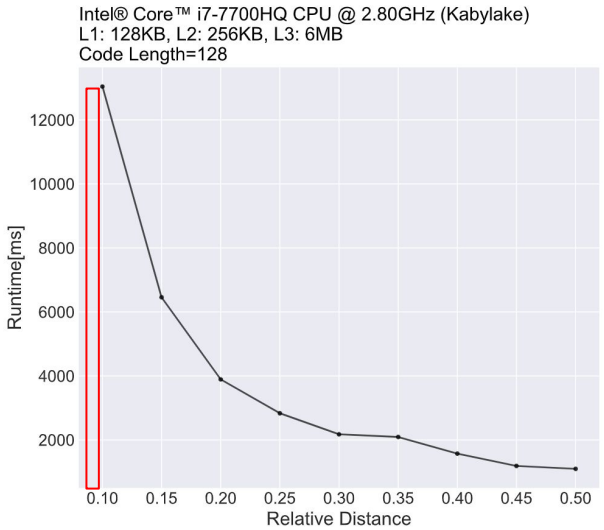


Figure 5.3: Runtime of Redistribution and Randomization Step

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

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

- Inefficient in practice
- Hard to implement

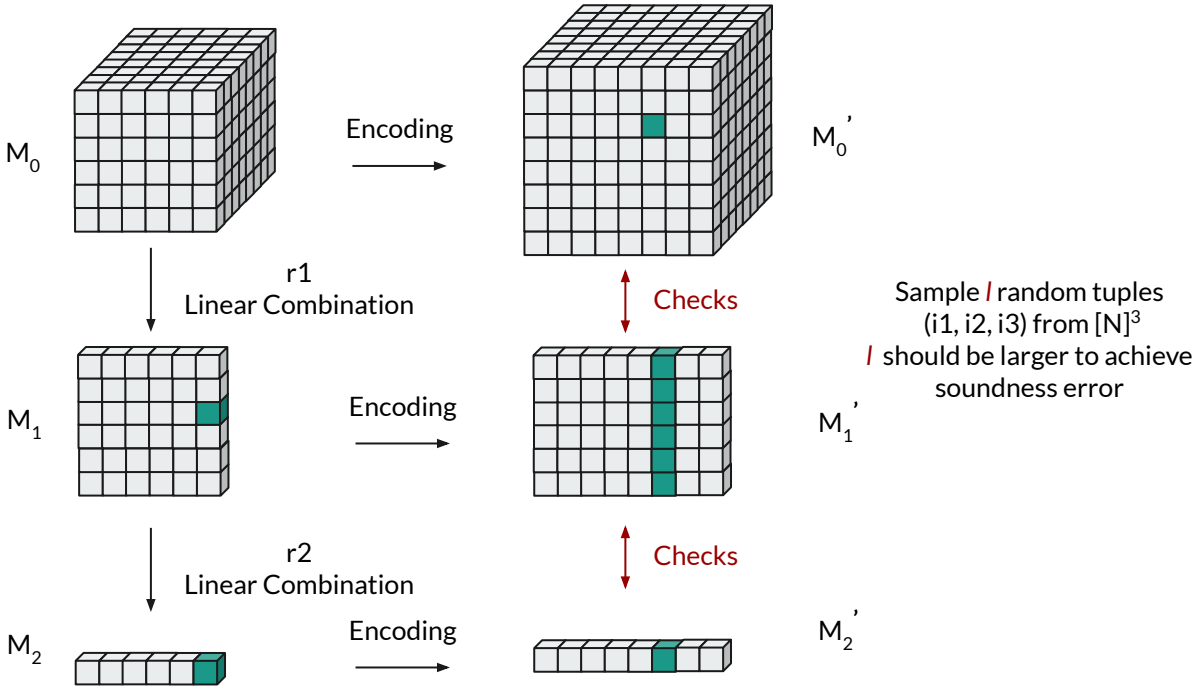
Zero-Knowledge Polynomial Commitment

- Zero-Knowledge Linear Code
- **Simple Modified Construction**

t-dimensional (t=3) Polynomial Commitment

Testing Phase:

(Proximity Test)

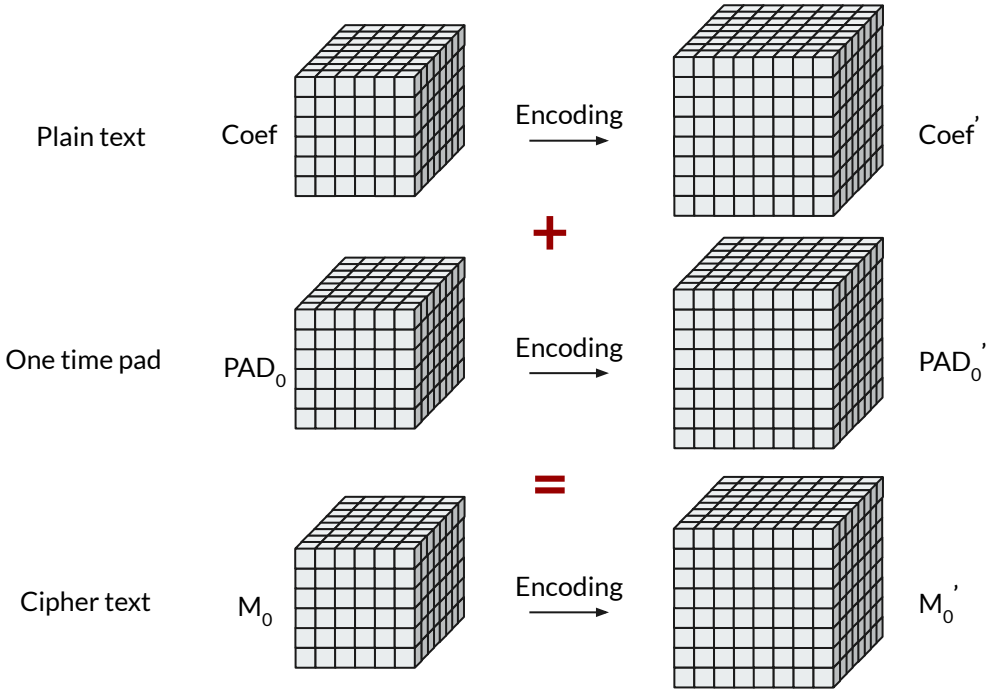


Sample *l* random tuples (i1, i2, i3) from $[N]^3$
l should be larger to achieve soundness error

Simple Zero-Knowledge Construction

Commitment Phase:

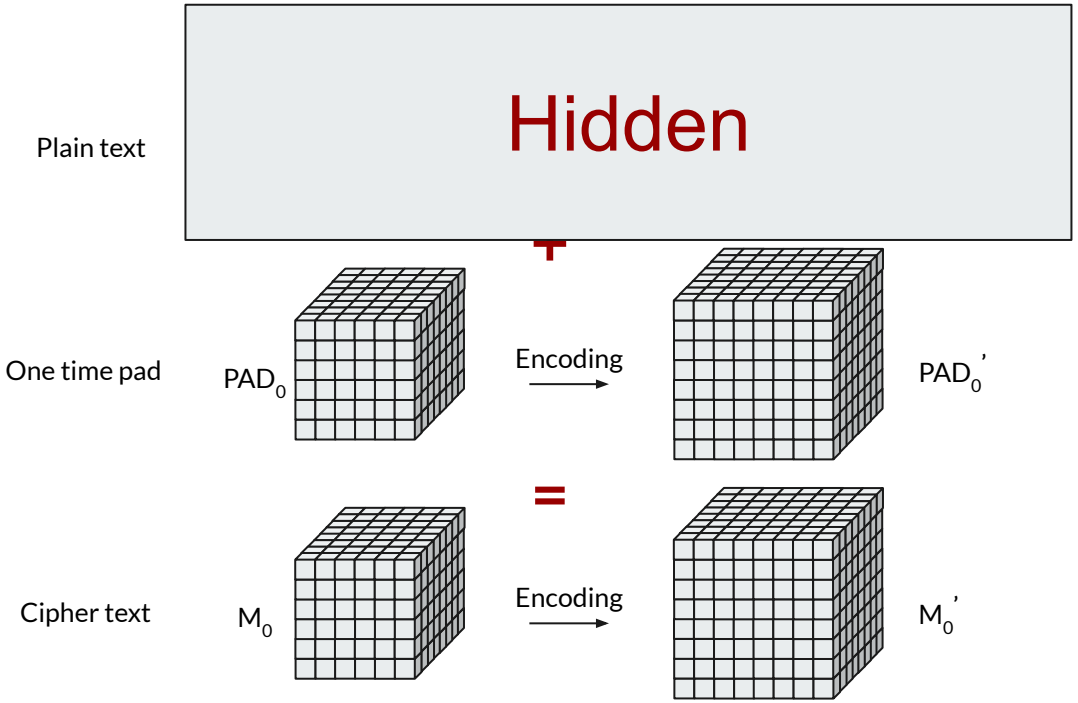
[BCGGHJ17][XZS22]



Simple Zero-Knowledge Construction

Testing Phase:

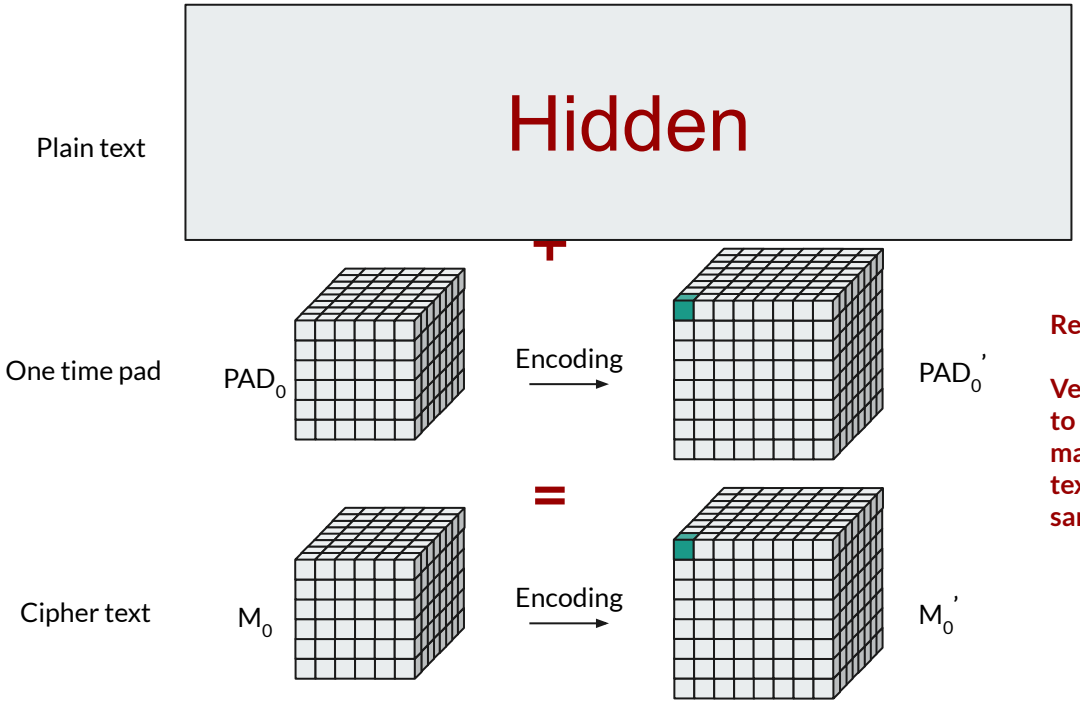
[BCGGHJ17][XZS22]



Simple Zero-Knowledge Construction

Testing Phase:

[BCGGHJ17][XZS22]



Restriction:

Verifier is not allowed to query the pad matrix and the cipher text matrix at the same position.

Attack?

Linear code:
[a b] -> [a b a]

Plain text	Coef	[w1 w2]	Encoding →	[w1 w2 w3]	Coef'
			+		
One time pad	PAD_0	[a1 a2]	Encoding →	[a1 a2 a3]	PAD_0'
			=		
Cipher text	M_0	[b1 b2]	Encoding →	[b1 b2 b3]	M_0'

Attack?

Linear code:
[a b] -> [a b a]

Plain text	Coef	[w1 w2]	Encoding →	[w1 w2 w3]	Coef'
			+		
One time pad	PAD_0	[a1 a2]	Encoding →	[a1 a2 a3]	PAD_0'
			=		
Cipher text	M_0	[b1 b2]	Encoding →	[b1 b2 b3]	M_0'

w1 = b1 - a1
= b3 - a1

A possible fix

Definition 4.6 (l-query independent codeword) *A linear code $\mathcal{C}$ is l-query independent if all adversaries $\mathcal{A}$ cannot distinguish a random value from an entry in the codeword when the adversary $\mathcal{A}$ can make l queries to the codeword. And the adversary $\mathcal{A}$ can choose the challenge position I' himself. Formally speaking, the following probability should be negligible:*

$$\left| Pr\left(\begin{array}{c} b = b' : \\ c \xleftarrow{\$} \mathcal{C} \\ b \xleftarrow{\$} \{0,1\} \\ t_0 \xleftarrow{\$} \mathbb{F} \\ (I_1, \cdots, I_l) \leftarrow \mathcal{A} \\ \mathcal{A} \leftarrow c[I_1], \cdots, c[I_l] \\ I' \leftarrow \mathcal{A} \\ t_1 \leftarrow c[I'] \\ b' \leftarrow \mathcal{A}(t_b, \mathcal{C}) \end{array} \right) - \frac{1}{2} \right|$$

A possible fix

Lemma 4.7 *A linear code $\mathcal{C}$ is l-query independent if and only if any $l+1$ columns of the generator matrix G are linearly independent of each other.*

Simple Zero-Knowledge Construction

- Simple to understand
- Easy to implement
- Only increase the prover time/verifier time/proof size by a factor of 2

- Have restrictions on the queries the verifier can make
- Require l-query-independent property from the linear code

Conclusions

- Using the method similar to one-time-pad encryption, we can add zero-knowledge property into the polynomial commitment scheme if the underlying linear code satisfying the l-query-independent property. And it is practical and efficient compared with the alternative approach.

LWE (Learning with error)

- LWE problem is one of the fundamental lattice problems upon which lots of the lattice-based cryptography rests.
- LWE was first introduced by Regev in 2009, whose main result is a quantum reduction from lattice problems (GAPSVP and SIVP) to LWE problem.
- LWE states that for a tuple **(A, u)** it is hard to find a small **s** and a small **e** such that
  - **u = As + e**
- The protocol makes use of polynomials and linear codes.

LWE (Learning with error)

- The original protocol use Reed-Solomon linear code, whereas we use the Brakedown linear code. Brakedown linear code can encode a message in linear time, which is more efficient than Reed-Solomon linear code.
- One message in the original protocol is only a subset of the one in our protocol. Our version reduces the computation required by the verifier at the cost of increasing communication complexity / proof size. Also, our security proof could be cleaner and simpler.

Thanks