



Eidgenössische Technische Hochschule Zürich  
Swiss Federal Institute of Technology Zurich

# Linear-Time Zero-Knowledge Arguments in Practice

Master Thesis

Ran Liao

October 15, 2022

Advisors: Prof. Dr. Kenny Paterson, Dr. Jonathan Bootle

Applied Cryptography Group  
Institute of Information Security  
Department of Computer Science, ETH Zürich



---

## Abstract

Interactive zero-knowledge proofs allow an untrusted prover to convince a skeptical verifier that a statement is true without revealing any further information about why the statement is true. The polynomial commitment scheme is the key component of many proof systems, which allows the prover to commit to a polynomial, and later reveal the evaluation of the polynomial at a given point, while allowing the verifier can check that the evaluation is correct. After years of research, many schemes with linear prover time have been proposed, however, there is little work on their concrete efficiency and performance.

In this thesis, we want to investigate the concrete efficiency of those polynomial commitment schemes, especially in high-dimensional situations. We implement the protocol in Rust, benchmark the performance and analyze the result. Additionally, we would investigate various ways to add zero-knowledge property into the polynomial commitment scheme and research their advantages and limitations.

We conclude that the efficiency and the soundness error of high dimensional polynomial commitment schemes are not acceptable to be used in practice because of the lack of linear code that is both efficient in encoding and provides a large relative distance guarantee. And we can add zero-knowledge property into the polynomial commitment schemes at the cost of increasing prover time, verifier time, and proof size by roughly a factor of two.



---

# Contents

---

<b>Contents</b>	<b>iii</b>
<b>1 Preliminaries</b>	<b>1</b>
1.1 Combinatorics . . . . .	1
1.2 Interactive Oracle Proof . . . . .	1
1.3 Polynomial . . . . .	2
1.4 Linear Code . . . . .	3
<b>2 Polynomial Commitment</b>	<b>5</b>
2.1 Notation . . . . .	5
2.2 Polynomial Commitment for $t = 3$ . . . . .	6
2.3 Polynomial Commitment for Arbitrary $t$ . . . . .	7
2.3.1 Protocol . . . . .	7
2.4 Analysis . . . . .	8
2.5 Implementation Details . . . . .	8
2.5.1 Merkle Tree Commitment . . . . .	8
2.5.2 Parallelism . . . . .	8
2.6 Benchmark . . . . .	9
2.6.1 Runtime . . . . .	9
2.6.2 Soundness Error . . . . .	10
<b>3 Simple Zero-Knowledge Polynomial Commitment</b>	<b>11</b>
3.1 Protocol . . . . .	11
3.2 Formal Description . . . . .	13
3.2.1 Notation . . . . .	13
3.2.2 Testing Phase (Proximity Test) . . . . .	13
3.2.3 Testing Phase Completeness . . . . .	14
3.2.4 Testing Phase Soundness . . . . .	15
3.2.5 Testing Phase Zero-Knowledge . . . . .	15

<b>4</b>	<b>Brakedown Linear Code</b>	<b>19</b>
4.1	Notation . . . . .	19
4.2	Construction . . . . .	19
4.3	Theoretical Limits for Relative Distance . . . . .	20
<b>5</b>	<b>Zero-Knowledge Linear Code</b>	<b>23</b>
5.1	Random $d$ -regular Bipartite Graph . . . . .	23
5.2	Expander Graph . . . . .	24
5.3	Reversed Linear Code . . . . .	27
5.4	Construction . . . . .	27
	5.4.1 Redistribution . . . . .	27
	5.4.2 Randomization . . . . .	28
	5.4.3 Reverse Encoding . . . . .	28
5.5	Performance . . . . .	28
5.6	Improvement . . . . .	29
<b>6</b>	<b>Zero-Knowledge Proofs for LWE</b>	<b>31</b>
6.1	Introduction . . . . .	31
<b>A</b>	<b>Mathematical Preliminaries</b>	<b>33</b>
	<b>Bibliography</b>	<b>35</b>

## Chapter 1

---

# Preliminaries

---

### 1.1 Combinatorics

**Definition 1.1 ( $d$ -regular Graph)** A graph  $G = (V, E)$  is  $d$ -regular if every vertex in  $V$  has degree  $d$ .

**Definition 1.2 (Set)**  $[n]$  is the shorthand for the set  $\{i : 1 \leq i \leq n\}$

### 1.2 Interactive Oracle Proof

**Definition 1.3 (Relation)** An *relation*  $R$  is a set of pairs  $(\mathbb{X}, \mathbb{W})$  where  $\mathbb{X}$  is the instance and  $\mathbb{W}$  is the witness. The corresponding **language**  $L(R)$  is the set of instances  $\mathbb{X}$  for which there exists a witness  $\mathbb{W}$  such that  $(\mathbb{X}, \mathbb{W}) \in R$ .

**Definition 1.4 (Point Query Interactive Oracle Proof (IOP))** An *Interactive Oracle Proof (IOP)* is defined by a pair of interactive randomized algorithms  $\text{IOP} = (P, V)$ , where  $P$  denotes the prover and  $V$  the verifier. The number of rounds of interaction is called the **round complexity** of the system. During a single round the prover sends a message to which the verifier is given oracle access, and the verifier responds with a message to the prover. The **proof length** is the sum of lengths of all messages sent by the prover. Specifically, the prover is allowed to send a large array message  $\pi$  to the verifier, and the verifier is allowed to query this message  $\pi$  at position  $I$ . Then the prover will response with  $\pi[I]$ . The **query complexity** of the protocol is the number of entries read by  $V$  from the various prover message and for each query only elements in a single point will be responded. We denote by  $\langle P \leftrightarrow V \rangle(\mathbb{X}, \mathbb{W})$  the output of  $V$  after interacting with  $P$  on instance  $\mathbb{X}$  and witness  $\mathbb{W}$ ; this output is either **ACCEPT** or **REJECT**.

An interactive oracle proof  $\text{IOP} = (P, V)$  for a relation  $R$  has completeness 1 and soundness error  $\epsilon$  if the following holds.

1. **Completeness.** For every pair  $(\mathbb{X}, \mathbb{W}) \in R$ , the probability that  $P(\mathbb{X}, \mathbb{W})$  convinces  $V(\mathbb{X})$  to accept is 1.
2. **Soundness.** For every instance  $\mathbb{X} \notin L(R)$  and malicious prover  $\tilde{P}$ , the probability that  $\tilde{P}$  convince  $V(\mathbb{X})$  to accept is at most  $\epsilon$ .

**Definition 1.5 (Interactive Oracle Proof of Proximity (IOPP))** An *Interactive Oracle Proof of Proximity (IOPP)* is defined by a pair of interactive randomized algorithms  $\text{IOPP} = (P, V)$ , where  $P$  denotes the prover and  $V$  the verifier. The number of rounds of interaction is called the **round complexity** of the system. During a single round the prover sends a message to which the verifier is given oracle access, and the verifier responds with a message to the prover. The **proof length** is the sum of lengths of all messages sent by the prover. The **query complexity** of the protocol is the number of entries read by  $V$  from the various prover message. We denote by  $\langle P \leftrightarrow V \rangle(\mathbb{X}, \mathbb{W})$  the output of  $V$  after interacting with  $P$  on instance  $\mathbb{X}$  and witness  $\mathbb{W}$ ; this output is either **ACCEPT** or **REJECT**. The protocol's goal is to show that a particular string is close to a valid witness. An interactive oracle proof of proximity  $\text{IOPP} = (P, V)$  for a relation  $R$  has completeness 1 and soundness error  $\epsilon$  with distance function  $\Delta$  if the following holds.

1. **Completeness.** For every pair  $(\mathbb{X}, \mathbb{W}) \in R$ , the probability that  $P(\mathbb{X}, \mathbb{W})$  convinces  $V^{\mathbb{W}}(\mathbb{X})$  to accept is 1.
2. **Soundness.** For every instance  $\mathbb{X} \notin L(R)$  and malicious prover  $\tilde{P}$ , the probability that  $\tilde{P}$  convince  $V^{\mathbb{W}}(\mathbb{X})$  to accept is at most  $\epsilon(\Delta(\mathbb{W}, R|_{\mathbb{X}}))$ . Here the soundness error  $\epsilon$  is a function of the  $\Delta$ -distance of  $\mathbb{W}$  to the set of valid witnesses  $R|_{\mathbb{X}} := \{\mathbb{W}' | (\mathbb{X}, \mathbb{W}') \in R\}$ .

### 1.3 Polynomial

**Definition 1.6 (Monomials of Polynomial)** A polynomial  $g$  over  $\mathbb{F}$  is an expression consisting of a sum of **monomials** where each monomial is the product of a constant (from  $\mathbb{F}$ ) and powers of one or more variables (which take values from  $\mathbb{F}$ ); all arithmetic is performed over  $\mathbb{F}$ .

**Definition 1.7 (Degree of Polynomial)** The degree of a monomial is the sum of the exponents of variables in the monomial; the (total) degree of a polynomial  $g$  is the maximum degree of any monomial in  $g$ . Furthermore, the degree of a polynomial  $g$  in a particular variable  $x_i$  is the maximum exponent that  $x_i$  takes in any of the monomials in  $g$ .

**Definition 1.8 (Multivariate / Univariate Polynomial)** A **multivariate** polynomial is a polynomial with more than one variable; otherwise it is called a **univariate** polynomial.



**Definition 1.9 (Multilinear Polynomial)** A *multivariate* polynomial is called a **multilinear** polynomial if the degree of the polynomial in each variable is at most one.

**Definition 1.10 (Polynomial Commitment)** A **Polynomial Commitment** consists of three algorithms:

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

**Completeness.** Let  $\mathcal{R} \leftarrow \text{PC.COMMIT}(\phi(\cdot))$  be the commitment of a polynomial. Any output  $(x, \phi(x), \pi_x)$  generated by an honest prover using algorithm  $\text{PC.PROVE}(\phi, x, \mathcal{R})$ , will be successfully verified by algorithm  $\text{PC.VERIFY}(\pi_x, x, \phi(x), \mathcal{R})$ .

$$\Pr \left( \begin{array}{l} \mathcal{R} \leftarrow \text{PC.COMMIT}(\phi(\cdot)) : \\ (x, \phi(x), \pi_x) = \text{PC.PROVE}(\phi, x, \mathcal{R}) \wedge \\ \text{PC.VERIFY}(\pi_x, x, \phi(x), \mathcal{R}) = \text{ACCEPT} \end{array} \right) = 1$$

**Soundness.** For all adversaries  $\mathcal{A}$ , the soundness error  $\epsilon$  is defined to be the following probability:

$$\epsilon = \Pr \left( \begin{array}{l} (\mathcal{R}, (\phi(\cdot), \pi_x), (\phi'(\cdot), \pi'_x)) \leftarrow \mathcal{A}(x) : \\ \text{PC.VERIFY}(\pi_x, x, \phi(x), \mathcal{R}) = \text{ACCEPT} \wedge \\ \text{PC.VERIFY}(\pi'_x, x, \phi'(x), \mathcal{R}) = \text{ACCEPT} \wedge \\ \phi(x) \neq \phi'(x) \end{array} \right)$$

**Zero-knowledge.** At the end of the protocol, the verifier will know the evaluation result of the polynomial at some evaluation points, but nothing more than that.

## 1.4 Linear Code

**Definition 1.11 (Linear Code)** If  $\mathbb{F}$  is a field and  $C \subset \mathbb{F}^n$  is a subspace of  $\mathbb{F}^n$  then  $C$  is said to be a linear code.

The weight of a codeword is the number of its elements that are nonzero and the distance between two codewords is the **Hamming distance** between them, that is, the number of elements in which they differ. The distance  $d$  of the linear code is the minimum weight of its nonzero codewords, or

equivalently, the **minimum distance** between distinct codewords. A linear code of length  $n$ , dimension  $k$ , and distance  $d$  is called an  $[n, k, d]$  code.

As  $C$  is a subspace, there exists a basis  $c_1, c_2, \dots, c_k$  where  $k$  is the dimension of the subspace. Any codeword can be expressed as the linear combination of these basis vectors. We can write these vectors in matrix form as the rows of a  $k \times n$  matrix. Such a matrix is called a **generator matrix**.

Additionally, any linear combination of valid codeword is also a valid codeword, which is the **linearity property**. Formally speaking, given  $x_1, x_2, \dots, x_n$  are valid codeword, and given  $r_1, r_2, \dots, r_n$  are some constants,  $x' = r_1x_1 + r_2x_2 + \dots + r_nx_n$  is also a valid codeword.

**Definition 1.12 (Tensor Product Code)** *The tensor product code  $C^{\otimes t}$  is the linear code in  $\mathbb{F}^{n^t}$  with message length  $k^t$ , block length  $n^t$  and distance  $d^t$  where any axis-parallel line of elements is in  $C$ .*

**Definition 1.13 (Relation  $R_{\otimes}$ )** *The relation  $R_{\otimes}$  is the set of tuples*

$$(\mathbb{X}, \mathbb{W}) = ((\mathbb{F}, C, l, q, t), (c_0^0, \{c_1^{(s)}\}_s, \dots, \{c_{t-1}^{(s)}\}_s))$$

*such that  $c_0^0 \in (C^{\otimes t})^l$  and for all  $r \in [t-1]$  and  $s \in [q]$ , we have  $c_r^{(s)} \in (C^{\otimes t-r})^k$ .*

**Definition 1.14 (Distance  $\Delta_{\otimes}$ )** *Let  $\mathbb{W} = (c_0^0, \{c_1^{(s)}\}_s, \dots, \{c_{t-1}^{(s)}\}_s)$  be such that  $c_0^{(0)} \in \mathbb{F}^{l \cdot n^t}$  and, for all  $r \in [t-1]$  and  $s \in [q]$ , we have  $c_r^{(s)} \in \mathbb{F}^{k \cdot n^{t-r}}$ . Given  $\mathbb{X} = (\mathbb{F}, C, l, q, t)$ , the  $\Delta_{\otimes}$  distance of  $\mathbb{W}$  to  $R_{\otimes}|_{\mathbb{X}}$  is*

$$\Delta_{\otimes}(\mathbb{W}, R_{\otimes}|_{\mathbb{X}}) := \max\{\Delta_0, \Delta_1, \dots, \Delta_{t-1}\}$$

*where  $\Delta_0 := \Delta(c_0^{(0)}, C^{\otimes t})$  and  $\forall r \in [t-1], \Delta_r := \Delta(\{c_r^{(s)}\}_s, C^{\otimes t-r})$ .*

## Chapter 2

---

# Polynomial Commitment

---

In this chapter, we present a general polynomial commitment scheme in the language of IOP for arbitrary dimension  $t$ . The scheme is an extension of the polynomial commitment scheme for  $t = 2$  described in [5]. We first extend the scheme to the  $t = 3$  situation so that readers can have a good intuition on how it works. Then we generalize it to arbitrary  $t$  with detailed analysis available.

### 2.1 Notation

Let  $g$  be a multilinear polynomial with  $n$  coefficients. For simplicity we assume that  $n = m^t$  for some integer  $m$ . And let  $u$  denote the coefficient vector of  $g$  in the Lagrange basis, which means  $u$  represents all evaluations of  $g$  over inputs in hypercube  $\{0,1\}^{\log n}$ . We can rearrange  $u$  to be a  $\underbrace{n^{\frac{1}{t}} \times n^{\frac{1}{t}} \times \cdots \times n^{\frac{1}{t}}}_{t \text{ times}}$  matrix, such that we can index entries in this matrix easily by elements from set  $[m]^t$ .

Let  $N = \rho^{-1} \cdot m$  and  $\text{Enc}: \mathbb{F}^m \rightarrow \mathbb{F}^N$  represent the encoding function of a linear code with a constant rate  $\rho > 0$  and a constant minimum relative distance  $\gamma > 0$ .

Let  $\text{Enc}_i(M)$  denote the function that encode every stripes in the  $i$ th dimension of matrix  $M$  using encoding function  $\text{Enc}$ . For example,  $\text{Enc}_1(M)$  will encode each column of a  $n \times n$  matrix and produce a  $N \times m$  matrix.

**Lemma 2.1 (Polynomial Evaluation [5])** *For an  $l$ -variate multilinear polynomial  $g$  represented in the Lagrange basis via a vector  $u \in \mathbb{F}^n$  where  $2^l = n$ , given an evaluation point  $x \in \mathbb{F}^l$ ,  $g(x)$  can be evaluated using the following tensor product identity:*

$$g(x) = \langle (x_1, 1 - x_1) \otimes (x_2, 1 - x_2) \otimes \cdots \otimes (x_l, 1 - x_l), u \rangle$$

And for any  $1 \leq t \leq l$ , there always exist vectors  $q_1, q_2, \dots, q_t \in \mathbb{F}^{n^{\frac{1}{t}}}$  such that the following holds:

$$(x_1, 1 - x_1) \otimes (x_2, 1 - x_2) \otimes \dots \otimes (x_l, 1 - x_l) = q_1 \otimes q_2 \otimes \dots \otimes q_t$$

## 2.2 Polynomial Commitment for $t = 3$

### Commitment Phase.

Let  $M_0 = u$  and  $M'_0 = \text{Enc}_1(\text{Enc}_2(M_0)) \in \mathbb{F}^{N \times N \times m}$ . Send  $M'_0$  to the verifier.

### Testing Phase.

The testing phase consists of 2 rounds, with each round reducing the dimension by 1.


In round 1, the verifier will send a random value  $r_1 \in \mathbb{F}^m$  to the prover. The prover will compute a linear combination  $M_1 \in \mathbb{F}^{m \times m}$  of the 3rd dimension of matrix  $M_0$ . Namely,  $M_1[i, j] = \sum_{k=1}^m r_1[k] \cdot M_0[i, j, k]$  for  $1 \leq i, j \leq m$ . Let  $M'_1 = \text{Enc}_1(M_1) \in \mathbb{F}^{N \times m}$ . Then the prover sends  $M'_1$  to the verifier.

In round 2, the verifier will send a random value  $r_2 \in \mathbb{F}^m$  to the prover. The prover will compute a linear combination  $M_2 \in \mathbb{F}^m$  of the 2nd dimension of matrix  $M_1$ . Namely,  $M_2[i] = \sum_{k=1}^m r_2[k] \cdot M_1[i, k]$ . Let  $M'_2 = M_2 \in \mathbb{F}^m$ . Then the prover sends  $M'_2$  to the verifier.

Then the verifier will perform a probabilistic check to make sure  $M'_0$ ,  $M'_1$  and  $M'_2$  are consistent with each other. Formally speaking, the verifier will sample  $l$  random tuple  $(j_1, j_2, j_3)$  from space  $[N] \times [N] \times [N]$ . For each tuple  $(j_1, j_2, j_3)$ , let  $M'_1[j_1, *]$  denotes the  $j_1$ -th row in  $M'_1$ . The verifier will check whether the following equation holds:

$$\begin{aligned} \text{Enc}(M'_1[j_1, *])[i_2] &\stackrel{?}{=} \sum_{k=1}^m r_1[k] \cdot M'_0[j_1, j_2, k] \\ \text{Enc}(M'_2[*])[j_1] &\stackrel{?}{=} \sum_{k=1}^m r_2[k] \cdot M'_1[j_1, k] \end{aligned}$$

### Evaluation Phase.

Let  $q_1, q_2, q_3 \in \mathbb{F}^m$  be vectors such that  $g(x) = \langle q_1 \otimes q_2 \otimes q_3, u \rangle$ . The evaluation phase is identical to the testing phase, except that in round  $i$ , the random value  $r_i$  is replaced by  $q_i$ . If all consistency checks passed, then the verifier outputs  $M_3$  as  $g(x)$ . 

## 2.3 Polynomial Commitment for Arbitrary $t$

### 2.3.1 Protocol

#### Commitment Phase.

Let  $M_0 = u$  and  $M'_0 = \text{Enc}_1 \circ \text{Enc}_2 \circ \dots \circ \text{Enc}_{t-1}(M_0) \in \mathbb{F}^{\overbrace{N \times N \times \dots \times N}^{t-1 \text{ times}} \times m}$ . Send  $M'_0$  to the verifier.

#### Testing Phase.

The testing phase consists of  $t - 1$  rounds, with each round reducing the number of dimensions by 1.

In round  $i$ , the verifier will send a random value  $r_i \in \mathbb{F}^m$  to the prover. The prover will compute a linear combination  $M_i \in \mathbb{F}^{\overbrace{m \times m \times \dots \times m}^{t-i \text{ times}}}$  of the last dimension of matrix  $M_{i-1}$ . Namely, for  $1 \leq j_1, j_2, \dots, j_{t-i} \leq m$ :

$$M_i[j_1, j_2, \dots, j_{t-i}] = \sum_{k=1}^m r_i[k] \cdot M_{i-1}[j_1, j_2, \dots, j_{t-i}, k]$$

Let

$$M'_i = \text{Enc}_1 \circ \text{Enc}_2 \circ \dots \circ \text{Enc}_{t-i-1}(M_i) \in \mathbb{F}^{\overbrace{N \times N \times \dots \times N}^{t-i-1 \text{ times}} \times m}$$

Then the prover sends  $M'_i$  to the verifier.

Then the verifier will perform a probabilistic check to make sure  $M'_0, M'_1, M'_2, \dots, M'_{t-1}$  are consistent with each other. Formally speaking, the verifier

will sample  $l$  random tuple  $(j_1, j_2, \dots, j_t)$  from space  $\overbrace{[N] \times [N] \times \dots \times [N]}^{t \text{ times}}$ . For each tuple  $(j_1, j_2, \dots, j_t)$ , the verifier will check whether the following equation holds for every  $i \in [t - 1]$ :

$$\text{Enc}(M'_i[j_1, j_2, \dots, j_{t-i-1}, *])[j_{t-i}] \stackrel{?}{=} \sum_{k=1}^m r_i[k] \cdot M'_{i-1}[j_1, j_2, \dots, j_{t-i}, k]$$

#### Evaluation Phase.

Let  $q_1, q_2, \dots, q_t \in \mathbb{F}^m$  be vectors such that  $g(x) = \langle q_1 \otimes q_2 \otimes \dots \otimes q_t, u \rangle$ . The evaluation phase is identical to the testing phase, except that in round  $i$ , the random value  $r_i$  is replaced by  $q_i$ . If all consistency checks passed, then the verifier outputs  $M'_t$  as  $g(x)$ .

## 2.4 Analysis

We refer to the result in [1] and summarize to the following lemmas.

**Lemma 2.2** *The testing phase (proximity test) has perfect completeness.*

**Lemma 2.3** *The testing phase (proximity test) has soundness error:*

$$\epsilon(\Delta_{\otimes}, t, l) = \frac{d(d^t - 1)}{4(d - 1)|\mathbb{F}|} + (1 - \min\{\frac{\delta^t}{4}, \Delta_{\otimes}\})^l$$

where  $d = \delta \cdot N$ , and  $\delta$  denotes the relative distance.

## 2.5 Implementation Details

### 2.5.1 Merkle Tree Commitment

A Merkle Tree is a data structure that allows one to commit to  $l = 2^d$  messages by a single hash value  $h$ , such that revealing any bit of the message require  $d + 1$  hash values. A Merkle hash tree is presented by a binary tree of depth  $d$  where  $l$  messages elements  $m_1, m_2, \dots, m_l$  are assigned to the leaves of the tree. The values assigned to internal nodes are computed by hashing the value of its two child nodes. To reveal  $m_i$ , we need to reveal  $m_i$  together with the values on the path from  $m_i$  to the root. We denote the algorithm as follows:

1.  $h \leftarrow \text{MERKLE.COMMIT}(m_1, m_2, \dots, m_l)$
2.  $(m_i, \pi_i) \leftarrow \text{MERKLE.OPEN}(m, i)$
3.  $\{\text{ACCEPT}, \text{REJECT}\} \leftarrow \text{MERKLE.VERIFY}(\pi_i, m_i, h)$

Coefficient matrices  $M'_0, M'_1, \dots, M'_{t-1}$  sent by the prover may be replaced by a Merkle tree commitment to that matrix, and each query the verifier makes to a matrix is answered by the prover with Merkle tree authentication path for the answer.



And since each time the verifier will query a strip of elements in a matrix (i.e.  $M'_i[i_1, i_2, \dots, i_{t-i-1}, *]$ ), it is possible to zip such a strip of elements into a single node in Merkle-tree's leaf level to decrease runtime complexity and communication complexity.

### 2.5.2 Parallelism

Most of the computations for the commitment scheme can be done in parallel in a natural fashion. There is little data dependence among them. Therefore, it is possible to run the commitment scheme using multiple threads to increase efficiency significantly for both the prover and the verifier.

## 2.6 Benchmark

### 2.6.1 Runtime

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.

We benchmark the above polynomial commitment scheme on a computer with Intel ® Core™ i7-7700HQ CPU @ 2.80GHz (Kabylake), L1 cache: 128KB, L2 cache: 256KB and L3 cache: 6MB. There are 8 physical CPU cores available on this machine. The runtimes are summarized in the table 2.1 and table 2.2.

Running the polynomial commitment scheme with the same setting, using 8-threads-parallelism can provides approximately a 4x speedup.

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.

As the dimension increases, it is generally require more time to complete the commit phase for the prover. And less time is required to complete the verify phase for the verifier. Also high dimensional polynomial commitment scheme will have less communication complexity. However, since the relative distance is decreasing as the tensor code's dimension is increasing, the soundness error will also increase. In fact, the soundness error for 3-dimensional and 4-dimensional polynomial commitment scheme is higher than 1, which is unusable in practice.

### 2.6.2 Soundness Error

According to lemma 2.3, we can compute the soundness error summarized in the table 2.3.

Dimension	Number of Test Tuples	Code Length	Code Relative Distance	Soundness Error
2	100	1762	0.07	1.66
	1000	1762	0.07	0.37
	100	1762	0.55*	0.0003
3	100	174	0.07	1.97
	1000	174	0.07	1.76
	100	174	0.55*	0.01
4	100	56	0.07	1.99
	1000	56	0.07	1.98
	100	56	0.55*	0.10

**Table 2.3:** Soundness error of polynomial commitment scheme. (\* represents an imaginary linear code with relative distance 0.55)

The theoretically computed soundness error for the setting used in the above benchmark experiment is large, even above 1, making it not usable in practice. The soundness error can be decreased by either increasing the number of tested tuples or by increasing the relative distance of the underlying linear code. However, the soundness error is not sensitive to the number of tested tuples and the length of the code is usually quite limited. Therefore, using a linear code with a large relative distance is the only promising solution here. One of our conclusion would be high dimension polynomial commitment scheme is not worth using unless we can improve the relative distance of these linear codes used in the constructions significantly. However, improving relative distance seems to be a difficult task.



## Chapter 3

---

# Simple Zero-Knowledge Polynomial Commitment

---

In this chapter, we describe a simple method to add the zero-knowledge property to a given polynomial commitment scheme. This method uses random numbers to hide the actual coefficients and it works similarly to one-time pad encryption.

### 3.1 Protocol

#### Commitment Phase.

Let  $M_0 = u$  and  $PAD_0$  be a tensor with dimensions identical to  $M_0$  filled with random elements from  $\mathbb{F}$ . Let

$$M'_0 = \text{Enc}_1 \circ \text{Enc}_2 \circ \cdots \circ \text{Enc}_{t-1}(M_0 \oplus PAD_0) \in \mathbb{F}^{\overbrace{N \times N \times \cdots \times N}^{t-1 \text{ times}} \times m}$$

$$PAD'_0 = \text{Enc}_1 \circ \text{Enc}_2 \circ \cdots \circ \text{Enc}_{t-1}(PAD_0) \in \mathbb{F}^{\overbrace{N \times N \times \cdots \times N}^{t-1 \text{ times}} \times m}$$

where  $\oplus$  denotes elements-wise tensor addition. Send  $M'_0$  and  $PAD'_0$  to the verifier.

#### Testing Phase.

The testing phase consists of  $t - 1$  rounds, with each round reducing the number of dimensions by 1.

In round  $i$ , the verifier will send a random value  $r_i \in \mathbb{F}^m$  to the prover. The prover will compute a linear combination for  $M_i, PAD_i \in \mathbb{F}^{\overbrace{m \times m \times \cdots \times m}^{t-i \text{ times}}}$

### 3. SIMPLE ZERO-KNOWLEDGE POLYNOMIAL COMMITMENT

---

of their last dimension. Namely, for  $1 \leq j_1, j_2, \dots, j_{t-i} \leq m$ :

$$M_i[j_1, j_2, \dots, j_{t-i}] = \sum_{k=1}^m r_i[k] \cdot M_{i-1}[j_1, j_2, \dots, j_{t-i}, k]$$

$$PAD_i[j_1, j_2, \dots, j_{t-i}] = \sum_{k=1}^m r_i[k] \cdot PAD_{i-1}[j_1, j_2, \dots, j_{t-i}, k]$$

Let

$$M'_i = \text{Enc}_1 \circ \text{Enc}_2 \circ \dots \circ \text{Enc}_{t-i-1}(M_i \oplus PAD_i) \in \mathbb{F}^{\overbrace{N \times N \times \dots \times N}^{t-i-1 \text{ times}} \times m}$$

$$PAD'_i = \text{Enc}_1 \circ \text{Enc}_2 \circ \dots \circ \text{Enc}_{t-i-1}(PAD_i) \in \mathbb{F}^{\overbrace{N \times N \times \dots \times N}^{t-i-1 \text{ times}} \times m}$$

where  $\oplus$  denotes element-wise addition.

Then the prover sends  $M'_i$  and  $PAD'_i$  to the verifier.

Then the verifier will perform a probabilistic check to make sure  $M'_0, M'_1, M'_2, \dots, M'_t, PAD'_0, PAD'_1, PAD'_2, \dots, PAD'_t$  are consistent with each other.

Formally speaking, in step 1, the verifier will sample  $l_1$  random tuple  $(j_1, j_2, \dots, j_t)$  from space  $\underbrace{[N] \times [N] \times \dots \times [N]}_{t \text{ times}}$ . Denote this set of tuples as  $L_1$ . For each

sampld tuple  $(j_1, j_2, \dots, j_t)$ , the verifier will check the following equation holds for every  $i \in [t-1]$ .

$$\text{Enc}(M'_i[j_1, j_2, \dots, j_{t-i-1}, *])[j_{t-i}] \stackrel{?}{=} \sum_{k=1}^m r_i[k] \cdot M'_{i-1}[j_1, j_2, \dots, j_{t-i}, k]$$

Then, in step 2, the verifier will sample another  $l_2$  random tuple  $(j'_1, j'_2, \dots, j'_t)$  from space  $\underbrace{[N] \times [N] \times \dots \times [N]}_{t \text{ times}}$  with the restriction that  $j'_k \neq j_k$  for  $\forall (j_1, j_2, \dots, j_t) \in L_1$ . Denote this set of tuples as  $L_2$ . For each sampled tuple  $(j'_1, j'_2, \dots, j'_t)$ , the

verifier will check the following equation holds for every  $1 \leq i \leq t-2$ .

$$\text{Enc}(PAD'_i[j_1, j_2, \dots, j_{t-i-1}, *])[j_{t-i}] \stackrel{?}{=} \sum_{k=1}^m r_i[k] \cdot PAD'_{i-1}[j_1, j_2, \dots, j_{t-i}, k]$$

#### Evaluation Phase.

Let  $q_1, q_2, \dots, q_t \in \mathbb{F}^m$  be vectors such that  $g(x) = \langle q_1 \otimes q_2 \otimes \dots \otimes q_t, u \rangle$ . The evaluation phase is identical to the testing phase, except for the following difference.

- One additional round is required. There are  $t$  rounds in total. In round  $t$ ,  $M'_t$  and  $PAD'_t$  are degenerated to a single value.

- In round  $i$ , the random value  $r_i$  is replaced by  $q_i$ .
- For every sampled tuple  $(j_1, j_2, \dots, j_t)$  in step 1, the following restriction is required,  $j_k \neq j'_k$  for  $\forall (j'_1, j'_2, \dots, j'_t) \in L_2$ .
- For every sampled tuple  $(j'_1, j'_2, \dots, j'_t)$  in step 2, the following restriction is required,  $j'_k \neq j_k$  for  $\forall (j_1, j_2, \dots, j_t) \in L_1$ .

If all consistent checks passed, then the verifier outputs  $M'_t - PAD'_t$  as  $g(x)$ .

## 3.2 Formal Description

### 3.2.1 Notation

#### Fold Operation

Define  $\text{Fold}_i(X, r)$  to be the operation taking a linear combination of  $X$  across the  $i$ -th dimension according to coefficient  $r$ .

Namely, for indexes  $j_1, \dots, j_{i-1}, j_{i+1}, \dots, j_k \geq 1$ :

$$\text{Fold}_i(X, r)[j_1, \dots, j_{i-1}, j_{i+1}, \dots, j_k] = \sum_{k=1}^m r_i[k] \cdot X[j_1, \dots, j_{i-1}, k, j_{i+1}, \dots, j_k]$$

#### Encode Operation

Define  $\text{Enc}_{1, \dots, i}$  be short-hand for  $\text{Enc}_1 \circ \text{Enc}_2 \circ \dots \circ \text{Enc}_i$ .

### 3.2.2 Testing Phase (Proximity Test)

In this section, we describe the testing phase in the above protocol formally in terms of a IOPP (interactive oracle proof of proximity) with point queries for the relation  $R_{\otimes}(\mathbb{F}, C, m, N, t)$  between a prover  $\mathbf{P}$  and a verifier  $\mathbf{V}$ .

The prover  $\mathbf{P}$  takes as input an instance  $\mathbb{X} = (\mathbb{F}, C, m, N, t)$  and witness  $\mathbb{W} = (M'_0, M'_1, \dots, M'_{t-1}, PAD'_0, PAD'_1, \dots, PAD'_{t-1})$ . The verifier  $\mathbf{V}$  takes as input the instance  $\mathbb{X}$ .

#### 1. Interactive phase.

In the beginning,  $\mathbf{P}$  sends the proof message  $M'_0$  and  $PAD'_0$  computed as:

$$\begin{aligned} M_0 &= u \in \mathbb{F}^{m^t} \\ M'_0 &= \text{Enc}_{1, \dots, t-1}(M_0 \oplus PAD_0) \in \mathbb{F}^{N^{t-1} \cdot m} \\ PAD'_0 &= \text{Enc}_{1, \dots, t-1}(PAD_0) \in \mathbb{F}^{N^{t-1} \cdot m} \end{aligned}$$

Note that  $PAD_0$  is a matrix with dimension identical to  $M_0$  filled with random elements from  $\mathbb{F}$ . And  $\oplus$  denotes elements-wise matrix addition.

For each round  $i \in [t - 1]$ :

- **V** sends random challenge message  $r_i \in \mathbb{F}^m$ .
- **P** sends the proof message  $M'_i$  computed as:

$$\begin{aligned} PAD_i &= \mathbf{Fold}_{t-i+1}(PAD_{i-1}, r_i) \in \mathbb{F}^{m^{t-i}} \\ M_i &= \mathbf{Fold}_{t-i+1}(M_{i-1}, r_i) \in \mathbb{F}^{m^{t-i}} \\ M'_i &= \mathbf{Enc}_{1, \dots, t-i-1}(M_i \oplus PAD_i) \in \mathbb{F}^{N^{t-i-1} \cdot m} \\ PAD'_i &= \mathbf{Enc}_{1, \dots, t-i-1}(PAD_i) \in \mathbb{F}^{N^{t-i-1} \cdot m} \end{aligned}$$

## 2. Query phase.

In step 1, the verifier **V** samples  $l_1$  tuples of the form  $(j_1, \dots, j_t)$  in space  $[N]^t$ . Denote this set of tuples as  $L_1$ . The verifier **V** proceeds as follows for each sampled tuple.

For each  $0 \leq i \leq t - 1$ , the verifier **V** will query  $M'_i$  at  $(j_1, \dots, j_{t-i-1}, j_k)$  for each  $j_k \in [m]$ .

Then the verifier **V** will check the following equation for  $i \in [t - 1]$ :

$$\mathbf{Enc}_{t-i}(M'_i)[i_1, \dots, i_{t-i}] \stackrel{?}{=} \mathbf{Fold}_{t-i+1}(M'_{i-1}, r_i)[i_1, \dots, i_{t-i}] \quad (3.1)$$

In step 2, the verifier **V** samples  $l_2$  tuples of the form  $(j'_1, \dots, j'_t)$  in space  $[N]^t$  with the restriction that  $j'_k \neq j_k$  for  $\forall (j_1, j_2, \dots, j_t) \in L_1$ . Denote this set of tuples as  $L_2$ . The verifier **V** proceeds as follows for each sampled tuple.

For each  $0 \leq i \leq t - 1$ , the verifier **V** will query  $PAD'_i$  at  $(j'_1, \dots, j'_{t-i-1}, j'_k)$  for each  $j'_k \in [m]$ .

Then the verifier **V** will check the following equation for  $i \in [t - 2]$ :

$$\mathbf{Enc}_{t-i}(PAD'_i)[i_1, \dots, i_{t-i}] \stackrel{?}{=} \mathbf{Fold}_{t-i+1}(PAD'_{i-1}, r_i)[i_1, \dots, i_{t-i}] \quad (3.2)$$

### 3.2.3 Testing Phase Completeness

**Lemma 3.1** *IOPP = (P, V) has perfect completeness.*

**Proof** We begin by noting that the queries made by **V** suffice to perform the checks in the query phase (see equation 3.1 and 3.2).

Next, observe that the verifier **V** checks the following equation:

$$\mathbf{Enc}_{t-i}(M'_i) \stackrel{?}{=} \mathbf{Fold}_{t-i+1}(M'_{i-1}, r_i)$$

Note that the left side of this equation is equivalent to:

$$\begin{aligned} \text{Enc}_{t-i}(M'_i) &= \text{Enc}_{t-i}(\mathbf{Enc}_{1,\dots,t-i-1}(M_i \oplus \text{PAD}_i)) \\ &= \mathbf{Enc}_{1,\dots,t-i}(M_i \oplus \text{PAD}_i) \\ &= \mathbf{Enc}_{1,\dots,t-i}(\mathbf{Fold}_{t-i+1}(M_{i-1} \oplus \text{PAD}_{i-1}, r_i)) \end{aligned} \quad (3.3)$$

(3.4)

And the right side of this equation is equivalent to:

$$\mathbf{Fold}_{t-i+1}(M'_{i-1}, r_i) = \mathbf{Fold}_{t-i+1}(\mathbf{Enc}_{1,\dots,t-i}(M_{i-1} \oplus \text{PAD}_{i-1}), r_i) \quad (3.5)$$

(3.6)

□

Since both **Fold** and **Enc** operations are linear operation, expression 3.3 and expression 3.5 are equivalent to each other. And similar argument applied to the equation 3.2. The equations checked by the verifier **V** holds.

### 3.2.4 Testing Phase Soundness

**Lemma 3.2**  $\text{IOPP} = (P, V)$  has soundness error at most:

$$\epsilon_{\text{ZK}}(\Delta_{\otimes}, t, l_1, l_2) = \epsilon(\Delta_{\otimes}, t, l_1) + \frac{\epsilon(\Delta_{\otimes}, t, l_2)}{\epsilon(\Delta_{\otimes}, 2, l_2)}$$

**Proof** Equivalently speaking, this protocol performs two proximity test in parallel. One on  $M'_i$  tensor and the other on  $\text{PAD}_i$  tensor.

The soundness error introduced by the first proximity test is  $\epsilon(\Delta_{\otimes}, t, l_1)$ .

The soundness error introduced by the second proximity test is  $\frac{\epsilon(\Delta_{\otimes}, t, l_2)}{\epsilon(\Delta_{\otimes}, 2, l_2)}$ . In a complete proximity test, we use  $E_{\text{last}}$  denote the event that the last round of test is passed. And we use  $E_{\text{other}}$  denote the event that all other tests are passed. The soundness error is the probability the verifier is convinced by a malicious input. The soundness error of a complete proximity test is  $P_t = \epsilon(\Delta_{\otimes}, t, l_2)$ . And it is also the probability where both event  $E_{\text{last}}$  and event  $E_{\text{other}}$  occurs. Therefore,  $P_t = P_{E_{\text{last}}} \cdot P_{E_{\text{other}}}$ . Note that  $P_{E_{\text{last}}}$  is actually the soundness error when  $t = 2$ , namely,  $P_{E_{\text{last}}} = \epsilon(\Delta_{\otimes}, 2, l_2)$ . And  $P_{E_{\text{other}}}$  is the soundness error introduced by the second proximity test here, where the input is malicious and all tests except the last one are passed. Therefore,  $P_{E_{\text{other}}} = \frac{P_t}{P_{E_{\text{last}}}} = \frac{\epsilon(\Delta_{\otimes}, t, l_2)}{\epsilon(\Delta_{\otimes}, 2, l_2)}$ .

### 3.2.5 Testing Phase Zero-Knowledge

**Definition 3.3** A interactive oracle proof of proximity  $\text{IOPP} = (P, V)$  for a relation  $R$  is **perfect zero-knowledge** if there exists a polynomial-time simulator algorithm  $S$  such that, for every  $(\mathbb{X}, \mathbb{W}) \in R$  and choice of verifier randomness  $\rho$ , the random variables  $S^{\text{V}(\mathbb{X}; \rho)}(\mathbb{X})$  and  $\text{View}(P(\mathbb{X}, \mathbb{W}), V(\mathbb{X}; \rho))$  are identically distributed.



**Lemma 3.4**  $IOPP = (P, V)$  is *perfect zero-knowledge*



**Proof** For every  $(\mathbb{X}, \mathbb{W}) \in R_{\otimes}$  and choice of verifier randomness  $\rho$ , we can construct the polynomial-time simulator algorithm **S** as follows:

- Generate matrix  $M_0$  and  $PAD_0$  randomly from field  $\mathbb{F}$ . Then compute  $M'_0$  and  $PAD'_0$  as follows:

$$M'_0 = \mathbf{Enc}_{1, \dots, t-1}(M_0) \in \mathbb{F}^{N^{t-1} \cdot m}$$

$$PAD'_0 = \mathbf{Enc}_{1, \dots, t-1}(PAD_0) \in \mathbb{F}^{N^{t-1} \cdot m}$$

- Then compute  $M'_i$  and  $PAD'_i$  for  $i \in [t-1]$ :

$$PAD_i = \mathbf{Fold}_{t-i+1}(PAD_{i-1}, r_i) \in \mathbb{F}^{m^{t-i}}$$

$$M_i = \mathbf{Fold}_{t-i+1}(M_{i-1}, r_i) \in \mathbb{F}^{m^{t-i}}$$

$$M'_i = \mathbf{Enc}_{1, \dots, t-i-1}(M_i) \in \mathbb{F}^{N^{t-i-1} \cdot m}$$

$$PAD'_i = \mathbf{Enc}_{1, \dots, t-i-1}(PAD_i) \in \mathbb{F}^{N^{t-i-1} \cdot m}$$

If the verifier query  $M'_i$  or  $PAD'_i$  at index  $I = (i_1, i_2, \dots, i_t)$ :

- If  $i_k \leq m$  for  $\forall k \in [t]$ ,

In the simulation world, both  $M'_0[I]$  and  $PAD'_0[I]$  are uniformly random variables. And both  $M'_i[I]$  and  $PAD'_i[I]$  for  $i > 0$  are linear combination of a set of uniformly random variables, which are also uniformly random variables.

In the real world,  $PAD'_0[I]$  is a uniformly random variable by definition.  $M'_0[I]$  is also a uniformly random variable because  $M'_0[I] = u[I] + PAD'_0[I]$ . Similarly, both  $M'_i[I]$  and  $PAD'_i[I]$  for  $i > 0$  are linear combination of a set of uniformly random variables, which are also uniformly random variables.

Therefore, the verifier will see a uniformly distributed random element from  $\mathbb{F}$  both in simulation world and in real world.

- Otherwise,

In the simulation world and in the real world,  $M'_i[I]$  can be determined by a set of random elements in  $M_i$ . Denote this computation equation as **FUNC**, namely,  $M'_i[I] = \mathbf{FUNC}(M_i[I_1], \dots, M_i[I_x])$ . And  $M'_i[I]$  will represent a distribution that is uniquely determined by function **FUNC** and the distribution of variables  $M_i[I_1], \dots, M_i[I_x]$ . Similarly,  $PAD'_i[I]$  will represent a distribution that is uniquely determined by function **FUNC** and the distribution of variables  $PAD_i[I_1], \dots, PAD_i[I_x]$ .

Note that both in simulation world and in the real world,  $M_i[I_1], \dots, M_i[I_x]$  and  $PAD_i[I_1], \dots, PAD_i[I_x]$  will represent uniformly random variables. And since the function  $\text{Func}$  is identical in both cases, distribution of  $M'_i[I]$  and  $PAD'_i[I]$  will be identical in two worlds.

The random variables in  $\mathbf{S}^{\mathbf{V}(\mathbb{X};\rho)}(\mathbb{X})$  and in  $\text{View}(\mathbf{P}(\mathbb{X}, \mathbb{W}), \mathbf{V}(\mathbb{X};\rho))$  are indistinguishable to each other. They are identically distributed.





## Chapter 4

---

# Brakedown Linear Code

---

We use the practical linear code presented in paper [5] to implement and benchmark our polynomial commitment schemes.

### 4.1 Notation

Let  $\alpha$  and  $\beta$  be parameters with no explicit meanings.  $r$  denotes the ratio between the length of codeword and the length of input message.  $\delta$  denotes the relative distance.  $n$  is the length of encoded message. Let  $q$  be a prime power and  $\mathbb{F}_q$  be the field of size  $q$ . And for  $p \in [0, 1]$ , we denote the binary entropy function as  $H(p) = -p \log_2(p) - (1-p) \log_2(1-p)$ . Let  $\mathcal{M}_{n,m,d}$  be a distribution of matrices  $M \in \mathbb{F}^{n \times m}$ , where in each row  $d$  distinct uniformly random elements are assigned uniformly random non-zero elements of  $\mathbb{F}$ .

### 4.2 Construction

The encoding function  $\mathbf{Enc}_n$  works as follows. First we generate a random sparse matrix  $A \leftarrow \mathcal{M}_{n,\alpha n,c_n}$  for

$$c_n = \left\lceil \min \left( \max(1.28\beta n, \beta n + 4), \frac{1}{\beta \log_2 \frac{\alpha}{1.28\beta}} \left( \frac{110}{n} + H(\beta) + \alpha H\left(\frac{1.28\beta}{\alpha}\right) \right) \right) \right\rceil$$

And compute  $y = x \cdot A \in \mathbb{F}^{\alpha n}$ . Then we apply  $\mathbf{Enc}$  function recursively to  $y$ , let  $z = \mathbf{Enc}_{\alpha n}(y) \in \mathbb{F}^{\alpha r n}$ . Finally, we generate a random sparse matrix  $B \leftarrow \mathcal{M}_{\alpha r n, (r-1-r\alpha), d_n}$  for

$$d_n = \left\lceil \min \left( \left( 2\beta + \frac{(r-1) + \frac{110}{n}}{\log_2 q} \right) n, \frac{r\alpha H\left(\frac{\beta}{r}\right) + \mu H\left(\frac{\nu}{\mu}\right) + \frac{110}{n}}{\alpha\beta \log_2 \frac{\mu}{\nu}} \right) \right\rceil$$

$$\mu = r - 1 - r\alpha$$

$$v = \beta + \alpha\beta + 0.03$$

Let  $v = z \cdot B \in \mathbb{F}^{(r-1-r\alpha)n}$  The resulting codeword is the concatenation of  $x, z$  and  $v$ .

$$w = \mathbf{Enc}(x) := \begin{pmatrix} x \\ z \\ v \end{pmatrix} \in \mathbb{F}^{rn}$$

### 4.3 Theoretical Limits for Relative Distance

In Brakedown paper [5], there are a few explicit constrains for parameter  $\alpha$ ,  $\beta$  and  $r$ . And since binary entropy function used in the linear code is only well-defined between 0 and 1, there is also one more implicit constrain. The full list of constrains are as follows,

$$\begin{aligned} 0 < \alpha < 1 \\ 0 < \beta < \frac{\alpha}{1.28} \end{aligned} \tag{4.1}$$

$$r > \frac{1+2\beta}{1-\alpha} > 1 \tag{4.2}$$

$$\delta = \frac{\beta}{r} \tag{4.3}$$

$$\beta + \alpha\beta + 0.03 < r - 1 - r\alpha \tag{4.4}$$

$$\tag{4.5}$$

Combine constrain 4.3 and constrain 4.1, we have,

$$\alpha > 1.28 \cdot \delta \cdot r \tag{4.6}$$

Combine constrain 4.3 and constrain 4.2, we have,

$$\alpha > 1 - 2\delta - \frac{1}{r} \tag{4.7}$$

Combine constrain 4.3 and constrain 4.4, we have,

$$\alpha < \frac{r(1-\delta) - 1.03}{r(1+\delta)} \tag{4.8}$$

To make sure  $\alpha$  has a valid value, we have,

$$\frac{r(1 - \delta) - 1.03}{r(1 + \delta)} > 1.28 \cdot \delta \cdot r \quad (4.9)$$

$$\frac{r(1 - \delta) - 1.03}{r(1 + \delta)} > 1 - 2\delta - \frac{1}{r} \quad (4.10)$$

$$(4.11)$$

Equation 4.9 and equation 4.10 make the maximum possible relative distance  $\delta$  to be around 0.12.



---

## Zero-Knowledge Linear Code

---

In this chapter, we use the construction presented in paper [4] to add zero-knowledge property to a normal linear code through code transformation.

### 5.1 Random $d$ -regular Bipartite Graph

First, we present a algorithm to generate a random  $d$ -regular bipartite graph. To make sure each vertex has degree  $d$ , we can first sample  $d$  random perfect matching for 2 sets of  $n$  vertices. Then take the union of them. Note that it is possible to generate parallel edges. But this should not be a concern for our purpose here. And it can be shown that this happens with low probability.

---

**Algorithm 1:** Random  $d$ -regular Bipartite Graph Generation

---

```

Data:  $n \geq 0, d \leq n$ 
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;
 $E \leftarrow \emptyset$ ;
 $P \leftarrow [1, 2, \dots, n]$ ;
for  $i$  in  $1, 2, \dots, d$  do
    | Permute  $P$  randomly ;           /* sample a perfect matching */
    | for  $j$  in  $1, 2, \dots, n$  do
    | |  $E \leftarrow E \cup (L_j, R_{P_i})$  ;
    | end
end
return  $(L, R, E)$ 

```

---

## 5.2 Expander Graph

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

**Proof** Negating the statement, we can say that the randomly generated graph  $G$  does not satisfy the expansion property if and only if  $\exists S \subseteq L$ ,  $|S| \geq \epsilon n$ ,  $\exists M \subseteq R$ ,  $|M| \geq \epsilon n$  such that there is no edge connecting between set  $S$  and set  $M$ . We bound the probability that this negating statement is true as follows:

For every vertex  $a \in L$  and every vertex  $b \in R$ , the probability that  $a$  and  $b$  are not connected in the random graph  $G$  is:

$$P_1 = \left(\frac{n-1}{n}\right)^d$$

For a set of vertices  $S \subset L$  with  $|S| = s \geq \epsilon n$ , the probability that non of vertices in  $S$  is connected to  $b$  is:

$$P_2 = (P_1)^s = \left(\frac{n-1}{n}\right)^{ds}$$

The probability that there exists at least  $\epsilon n$  vertices in  $R$  are not connected to any vertex in  $S$  is:

$$P_3 = \binom{n}{\epsilon n} (P_2)^{\epsilon n} = \binom{n}{\epsilon n} \left(\frac{n-1}{n}\right)^{ds\epsilon n}$$

For  $0 \leq x \leq 1$ , we denote the binary entropy function to be:

$$H(x) = -x \log_2 x - (1-x) \log_2 (1-x)$$

where we adopt the convention that  $0 \log_2 0 = 0$ .

Then, we take a union bound over all possible sets  $S$ ,

$$\begin{aligned}
 P_4 &= \sum_{s=\epsilon n}^n \binom{n}{s} P_3 \\
 &= \sum_{s=\epsilon n}^n \binom{n}{s} \binom{n}{\epsilon n} \left(\frac{n-1}{n}\right)^{d\epsilon n} \\
 &\leq \sum_{s=\epsilon n}^n \binom{n}{s} \binom{n}{\epsilon n} \left(\frac{n-1}{n}\right)^{d\epsilon^2 n^2} \quad \text{since } s \geq \epsilon n \text{ and } \frac{n-1}{n} < 1 \\
 &\leq \sum_{s=\epsilon n}^n \binom{n}{s} 2^{nH(\frac{\epsilon n}{n})} \left(\frac{n-1}{n}\right)^{d\epsilon^2 n^2} \quad \binom{n}{k} \leq 2^{nH(\frac{k}{n})} \\
 &= \sum_{s=\epsilon n}^n \binom{n}{s} 2^{nH(\epsilon)} \left(1 - \frac{1}{n}\right)^{d\epsilon^2 n} \\
 &\leq \sum_{s=\epsilon n}^n \binom{n}{s} 2^{nH(\epsilon)} \left(\frac{1}{e}\right)^{d\epsilon^2 n} \quad \left(1 - \frac{1}{x}\right)^x \leq \frac{1}{e} \text{ for } x \geq 1 \text{ (lemma A.2)} \\
 &= \sum_{s=\epsilon n}^n \binom{n}{s} (e^{H(\epsilon) \ln 2 - d\epsilon^2})^n \\
 &\leq \sum_{s=0}^n \binom{n}{s} (e^{H(\epsilon) \ln 2 - d\epsilon^2})^n \\
 &= 2^n (e^{H(\epsilon) \ln 2 - d\epsilon^2})^n \quad \sum_{i=0}^n \binom{n}{i} = 2^n \\
 &= (e^{\ln 2 + H(\epsilon) \ln 2 - d\epsilon^2})^n
 \end{aligned} \tag{5.1}$$

□

$P_4$  is the probability that a randomly generated graph  $G$  does not satisfy the expansion property. Suppose we want the failing probability be smaller than  $p$ , let  $(e^{\ln 2 + H(\epsilon) \ln 2 - d\epsilon^2})^n < p$ . By rearranging the above equation, we have  $d > \frac{\ln 2 + H(\epsilon) \ln 2 - \frac{\ln p}{n}}{\epsilon^2}$ .

For example, if  $\epsilon = 0.05$ ,  $n = 5000$ ,  $p = 2^{-256}$ , then degree  $d$  need to be greater than 370.86.

**Lemma 5.2** 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.

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

**Proof** We use the same trick as in lemma 5.1. Negating the statement, we can say that the randomly generated graph  $G$  does not satisfy the expansion

property if and only if for every  $S \subseteq L$ ,  $|S| \geq \epsilon n$ ,  $\exists M \subseteq R$ ,  $|M| > \epsilon n$  such that there is no edge connecting between set  $S$  and set  $M$  with low probability. We bound the probability true as follows:

For every vertex  $a \in L$  and every vertex  $b \in R$ , the probability that  $a$  and  $b$  are not connected in the random graph  $G$  is:

$$P_1 = \left(\frac{n-1}{n}\right)^d$$

For a set of vertices  $S \subset L$  with  $|S| = \epsilon n$ , the probability that non of vertices in  $S$  is connected to  $b$  is:

$$P_2 = (P_1)^{\epsilon n} = \left(\frac{n-1}{n}\right)^{d\epsilon n}$$

The probability that there exists at least  $\epsilon n$  vertices in  $R$  are not connected to any vertex in  $S$  is:

$$\begin{aligned} P_3 &= \binom{n}{\epsilon n} (P_2)^{\epsilon n} \\ &= \binom{n}{\epsilon n} \left(\frac{n-1}{n}\right)^{d\epsilon^2 n^2} \\ &\leq 2^{nH(\frac{\epsilon n}{n})} \left(\frac{n-1}{n}\right)^{d\epsilon^2 n^2} \quad \binom{n}{k} \leq 2^{nH(\frac{k}{n})} \\ &= 2^{nH(\epsilon)} \left(\left(1 - \frac{1}{n}\right)^n\right)^{d\epsilon^2 n} \\ &\leq 2^{nH(\epsilon)} \left(\frac{1}{e}\right)^{d\epsilon^2 n} \quad \left(1 - \frac{1}{x}\right)^x \leq \frac{1}{e} \text{ for } x \geq 1 \text{ (lemma A.2)} \\ &= (e^{H(\epsilon) \ln 2 - d\epsilon^2})^n \end{aligned} \tag{5.2}$$

□

$P_3$  is the probability that a set  $S$  in a randomly generated graph does not satisfy the expansion property. Suppose we want the failing probability be smaller than  $p$ , let  $(e^{H(\epsilon) \ln 2 - d\epsilon^2})^n < p$ . By rearranging the above equation, we have  $d > \frac{H(\epsilon) \ln 2 - \frac{\ln p}{n}}{\epsilon^2}$ .

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

Compared with lemma 5.1, lemma 5.2 produces a much tighter bound by weakening the expansion property. A graph satisfy the expansion property



in lemma 5.2 may not satisfy the expansion property in lemma 5.1. There may exist a set  $S \subset L$  in graph such that the expansion property fails. But lemma 5.2 guarantees that such set is hard to be found. Similar with hash functions, hash collision must exist somewhere, but this collision is hard to be found.

### 5.3 Reversed Linear Code

According to Tellegen's principle [3], it is generally possible to transpose a linear algorithm that performs a matrix-vector product, producing an algorithm that computes the transposed matrix-vector product.

In this section, we transpose the Brakedown linear code to get a reverse encoding algorithm. Figure 5.1 is the construction of Brakedown linear code. And figure 5.2 is the reversed construction.

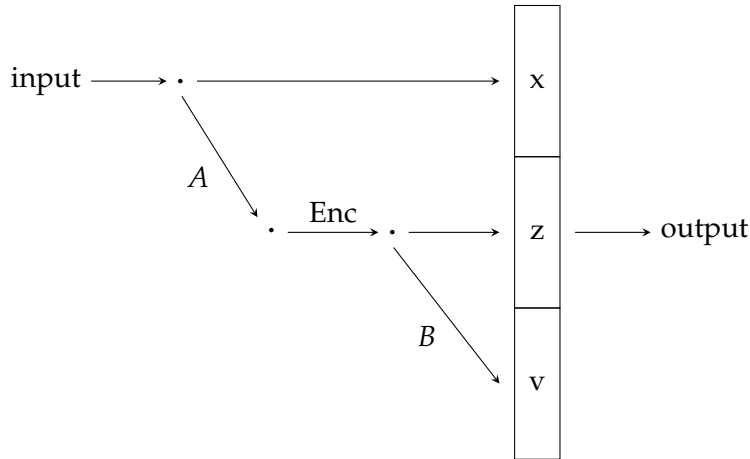


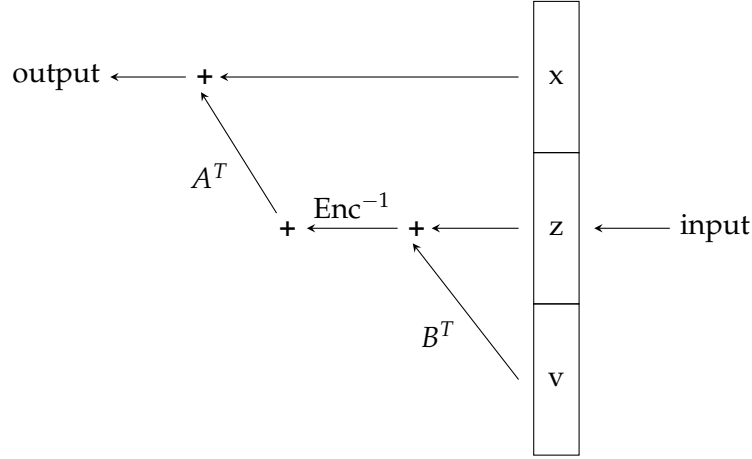
Figure 5.1: Linear Code

### 5.4 Construction

This construction transform an existing linear to another linear code. The linear code constructed will have better relative distance and is equipped with zero-knowledge property.

#### 5.4.1 Redistribution

Given the normal encoding function  $\text{Enc}()$  and message  $x$ , we first compute the codeword  $y = \text{Enc}(x) \in \mathbb{F}^n$ . Then a random expander graph  $G = (L, R, E)$  with degree  $\Delta$  satisfying lemma 5.1 will be generated. We will redistribute the symbols in  $y$  according to  $G$ . More concretely, for every



**Figure 5.2:** Reversed Linear Code

$i \in [n]$  and  $j \in [\Delta]$ , let  $\gamma(i, j)$  be the index of the  $j$ -th vertex in  $R$ . The  $(i - 1) \cdot \Delta + j$ -th entry of  $z$  is defined to be the  $y_{\gamma(i, j)}$ .


#### 5.4.2 Randomization

Given  $z \in \mathbb{F}^{n \cdot \Delta}$ , we generate a random block diagonal matrix  $H$  with  $n$  blocks each of size  $\Delta \cdot \Delta$ . We compute  $v = H \cdot z \in \mathbb{F}^{n \cdot \Delta}$ .

#### 5.4.3 Reverse Encoding

Given the reverse encoding function **Enc**  the final output is  $w = \mathbf{Enc}^{-1}(v)$ .

### 5.5 Performance

We have implemented the above construction. We measure the runtime required for the redistribution step and the randomization step, whose execution is irrelevant to the actual underlying linear code. 

According to lemma 5.2, the degree of the expander graph is very sensitive to the relative distance of underlying linear code. The larger the relative distance, the smaller the degree. And larger degree will cause the algorithm more time consuming.

Figure 5.3 presents the relation between relative distance and runtime. As relative distance approaches 0.1, the runtime increases dramatically. And even for larger relative distance, the construction is still significant slower than the original linear code, making this construction unacceptable in practice.

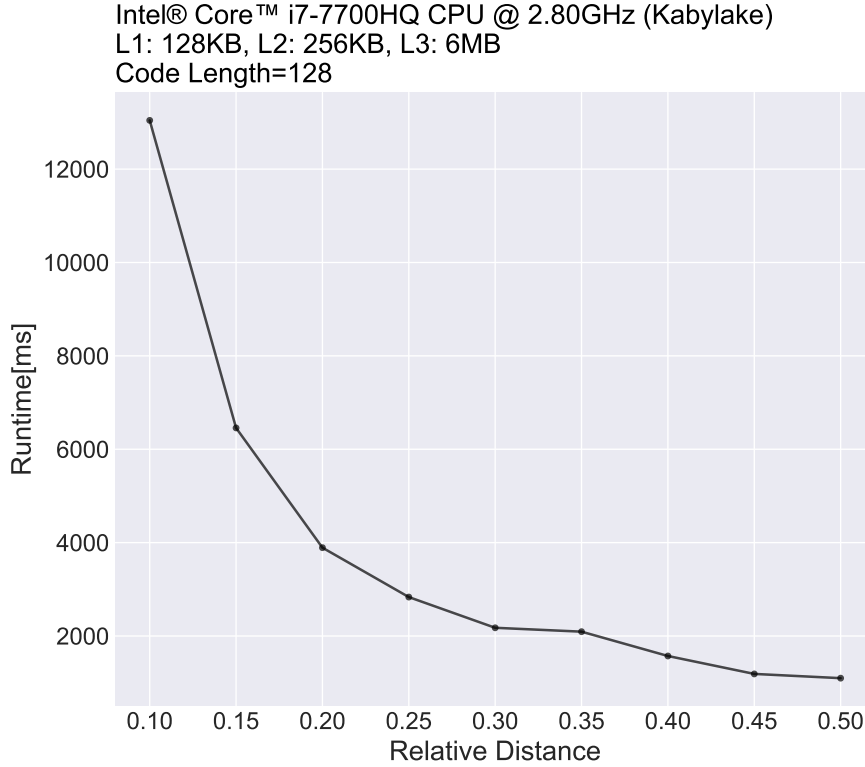


Figure 5.3: Runtime of Redistribution and Randomization Step

## 5.6 Improvement

The performance of our zero-knowledge linear code construction suffers from the degree of underlying random expander graph. Recently, we notice a idea presented in [6] that might solve this problem. They propose a new algorithm to test whether a random bipartite graph is a lossless expander graph or not based on the densest subgraph algorithm, which helps to sample lossless expanders with an overwhelming probability.

We can prove a graph is not a expander graph by providing one counter example. The densest-subgraph algorithm used by this detection algorithm can help us to find the counter example efficiently. Basically, if the graph is not a good expander graph, the detection algorithm in [6] can identify this situation with some probability  $p$  using a random input  $r$ . And if the graph is a good expander graph, the detection algorithm will not falsely identify it (no false positive). Then we can run this detection algorithm  $\lambda$  times with different random inputs and amplify the detection probability to  $1 - (1 - p)^\lambda$ . Additionally, if we find the graph is not a good expander graph, then we can simply discard it and generate a new random graph.

And if we have a efficient detection algorithm like this, we can randomly generate the expander graph with a much smaller degree  $d$  and run detection algorithm on it. Depending on the output of detection algorithm, we can either be convinced that it is a good expander graph or we can re-run the generation algorithm one more time. The redistribution step in our construction will be much more efficient with such a small degree expander graph.

**Definition 5.3** *Let  $\delta > 0$  and  $0 < \epsilon < 1$  be a constant. Let  $G = (L, R, E)$  be a  $d$ -regular bipartite graph with  $|L| = n$ . Graph  $G$  is a expander graph if the following expansion property is satisfied:*

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

However, due to the difference in expander graph definition, it is fundamentally not possible to reuse this densest-subgraph based detection algorithm in our project to improve efficiency. Definition 5.3 is the expander graph used in [6] and lemma 5.1 is the expander graph used in our project. The key difference is their expansion property. Informally speaking, definition 5.3 needs the graph has good expansion property when we choose a small subset of vertices. And lemma 5.1 needs the graph has good expansion property when we choose a large subset of vertices. The counter example found by the densest-subgraph algorithm may have a small subset of vertices. And this is enough to be a good counter example according to definition 5.3, but not enough according to lemma 5.1.

Therefore, it is remain unclear how to improve efficiency on this construction. And 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.

---

## Zero-Knowledge Proofs for LWE

---

### 6.1 Introduction

LWE (Learning with error) problem is the fundamental lattice problem upon which most of the lattice-based cryptography rests. LWE states that it is hard to distinguish a uniformly random tuple  $(A, u)$  and  $(A, u = As + e)$ , where all elements of  $s$  and  $e$  are small. In this chapter, we explore a protocol in [2] that makes use of polynomials to prove knowledge of  $s$  and  $e$  with small elements that satisfy:

$$As + e = u$$





## Appendix A

---

# Mathematical Preliminaries

---

**Lemma A.1**  $\binom{n}{m} \leq \left(\frac{en}{m}\right)^m$ , for  $n, m \in \mathbb{Z}^+$

**Proof**

$$\begin{aligned}\log m! &= \log 1 + \log 2 + \cdots + \log m \quad \text{🗨️} \\ &\geq \int_1^m \log x \, dx \\ &= [x \log x - x]_1^m \\ &= m \log m - m + 1\end{aligned}\tag{A.1}$$

$$\begin{aligned}m! &= e^{\ln m!} \\ &\geq e^{m \log m - m + 1} \quad \text{apply equation A.1} \\ &= e^{\log m^m} \cdot e^{-m} \cdot e \\ &= m^m \cdot e^{-m} \cdot e \\ &= \left(\frac{m}{e}\right)^m \cdot e \\ &\geq \left(\frac{m}{e}\right)^m\end{aligned}\tag{A.2}$$

$$\begin{aligned}\binom{n}{m} &= \frac{n \cdot (n-1) \cdot (n-2) \cdots (n-m+1)}{m!} \\ &\leq \frac{n^m}{m!} \\ &\leq \frac{n^m}{\left(\frac{m}{e}\right)^m} \quad \text{apply equation A.2} \\ &= \left(\frac{en}{m}\right)^m\end{aligned}\tag{A.3}$$

□

**Lemma A.2**  $(1 - \frac{1}{x})^x \leq \frac{1}{e}$ , for  $x \geq 1$

**Proof**

Recall that for  $x \in \mathbb{R}$

$$1 + x \leq e^x$$

Then for  $x \in \mathbb{R}$

$$1 - x \leq e^{-x}$$

Then for  $x \neq 0$

$$1 - \frac{1}{x} \leq e^{-\frac{1}{x}}$$

And, since  $t \mapsto t^x$  is increasing on  $[0, \infty]$  for  $x \geq 1$

$$(1 - \frac{1}{x})^x \leq \frac{1}{e} \quad (\text{A.4})$$

□

**Lemma A.3**  $(\frac{a}{x})^x \leq e^{\frac{a}{e}}$ , for  $x > 0$ ,  $a > 0$

**Proof**

Let  $f(x) = (\frac{a}{x})^x$

$$\ln f(x) = x \cdot \ln(\frac{a}{x}) = -x \cdot \ln \frac{x}{a}$$

Take derivative from both side

$$\frac{1}{f(x)} \frac{df(x)}{dx} = -\ln \frac{x}{a} - x \cdot \frac{a}{x} \cdot \frac{1}{a} = -\ln \frac{x}{a} - 1$$

$$\frac{df(x)}{dx} = -f(x) \cdot (\ln \frac{x}{a} + 1) = -(\frac{a}{x})^x \cdot (\ln \frac{x}{a} + 1)$$

Let  $\frac{df(x)}{dx} = 0$

$$-(\frac{a}{x})^x \cdot (\ln \frac{x}{a} + 1) = 0$$

$$(\ln \frac{x}{a} + 1) = 0$$

$$x = \frac{a}{e}$$

$\frac{df(x)}{dx} > 0$  when  $x < \frac{a}{e}$ , and  $\frac{df(x)}{dx} < 0$  when  $x > \frac{a}{e}$

Therefore,  $x = \frac{a}{e}$  is a maximum point

$$(\frac{a}{x})^x = f(x) \leq f(\frac{a}{e}) = e^{\frac{a}{e}} \quad (\text{A.5})$$

□



---

## Bibliography

---

- [1] Jonathan Bootle, Alessandro Chiesa, and Jens Groth. Linear-time arguments with sublinear verification from tensor codes. Cryptology ePrint Archive, Paper 2020/1426, 2020. <https://eprint.iacr.org/2020/1426>.
- [2] Jonathan Bootle, Vadim Lyubashevsky, Ngoc Khanh Nguyen, and Gregor Seiler. More efficient amortization of exact zero-knowledge proofs for lwe. Cryptology ePrint Archive, Paper 2020/1449, 2020. <https://eprint.iacr.org/2020/1449>.
- [3] A. Bostan, Grégoire Lecerf, and Éric Schost. Tellegen’s principle into practice. pages 37–44, 01 2003.
- [4] Erez Druk and Yuval Ishai. Linear-time encodable codes meeting the gilbert-varshamov bound and their cryptographic applications. In *Proceedings of the 5th Conference on Innovations in Theoretical Computer Science, ITCS ’14*, page 169–182, New York, NY, USA, 2014. Association for Computing Machinery.
- [5] Alexander Golovnev, Jonathan Lee, Srinath Setty, Justin Thaler, and Riad S. Wahby. Brakedown: Linear-time and post-quantum snarks for r1cs. Cryptology ePrint Archive, Report 2021/1043, 2021. <https://ia.cr/2021/1043>.
- [6] Tiancheng Xie, Yupeng Zhang, and Dawn Song. Orion: Zero knowledge proof with linear prover time. Cryptology ePrint Archive, Paper 2022/1010, 2022. <https://eprint.iacr.org/2022/1010>.



Eidgenössische Technische Hochschule Zürich  
Swiss Federal Institute of Technology Zurich

## Declaration of originality

The signed declaration of originality is a component of every semester paper, Bachelor's thesis, Master's thesis and any other degree paper undertaken during the course of studies, including the respective electronic versions.

Lecturers may also require a declaration of originality for other written papers compiled for their courses.

---

I hereby confirm that I am the sole author of the written work here enclosed and that I have compiled it in my own words. Parts excepted are corrections of form and content by the supervisor.

**Title of work** (in block letters):

**Authored by** (in block letters):

*For papers written by groups the names of all authors are required.*

**Name(s):**

**First name(s):**


With my signature I confirm that

- I have committed none of the forms of plagiarism described in the '[Citation etiquette](#)' information sheet.
- I have documented all methods, data and processes truthfully.
- I have not manipulated any data.
- I have mentioned all persons who were significant facilitators of the work.

I am aware that the work may be screened electronically for plagiarism.

**Place, date**

**Signature(s)**


*For papers written by groups the names of all authors are required. Their signatures collectively guarantee the entire content of the written paper.*