AARHUS
UNIVERSITY
DEPARTMENT OF COMPUTER SCIENCE

Lattice based Zero-knowledge proofs with VOLE in the batch setting

Master's Thesis

30 June 2026

Participant

Aakanksha Agrawal
202402594

Advisors

Peter Scholl
peter.scholl@cs.au.dk

Diego F. Aranha
dfaranha@cs.au.dk

Abstract

Post-quantum cryptography is becoming increasingly urgent as quantum computing threatens the security of widely deployed public-key crypto systems. Lattice-based constructions, and in particular those based on the hardness of the Learning With Errors problem, have emerged as strong candidates for constructing post-quantum secure primitives. Many such primitives such as digital signatures, or various privacy enhancing technologies require the prover to prove knowledge of a witness to a hard computational problem.

In this thesis, we construct efficient zero-knowledge proof for Module-LWE relations, using the parameters of the ML-KEM post-quantum standard. This requires an additional complexity- proving the witness is small. Our proofs are built on the VOLE-based zero-knowledge paradigm, which offers a favorable trade-off between prover efficiency and communication complexity. We give a commitment scheme for ring elements in R_q^l, which sidesteps the difficulties arising from zero divisors in ring arithmetic.

We investigate sublinear communication for both the sampling random VOLEs and de-randomizing them on the provers witness. For sampling, we draw on pseudorandom correlation generator (PCG) techniques from silent secure computation. For de-randomization, we exploit the central binomial distribution used in the secret and error sampling of ML-KEM, via a commit-and-sample approach that reduces proof size by roughly one third in the batch setting.

Contents

i. Abstract 2
1. Introduction 5
1.1. The Problem: Post-Quantum Primitives 5
1.2. Lattice-Based Primitives 5
1.3. Zero-Knowledge Proofs: Background 5
1.4. This Thesis 6
1.5. Project Organisation and How to Read the Thesis 7
2. Definitions and Notation 9
2.1. Mathematical background: 9
2.2. Cryptographic Primitives 15
3. Vector Oblivious Linear Evaluations(VOLE) and properties 21
3.1. Oblivious Linear Evaluation (OLE): 21
3.2. Random-OLE: 21
3.3. OLE as (2,2)-secret sharing of multiplication product $u \cdot \Delta$: 22
3.4. Vector Oblivious Linear Evaluation (VOLE) 22
3.5. s-VOLE 23
3.6. VOLE Correlations are Linear 24
3.7. VOLE as a Commitment Scheme 25
4. VOLE based zk POK 27
4.1. Quicksilver ZK 27
4.2. Translating this idea to d-degree polynomial constraints. 31
4.3. VOLE-ZK sub-protocols in $\mathbb{F}_{q^r}$ 35
5. Range Proof 37
5.1. Naive range proof 37
5.2. Range Proof using n-ary bit decomposition: 38
5.3. Interval Proof for proving that x is of short norm 38
6. ZKP for Module LWE 40
6.1. The Protocol Sketch 40
6.2. Concrete Commitments Schemes and ZKP 42
6.3. Other ZK proofs and statements 44
6.4. Communication costs 46
6.5. Towards sublinear communication 47
7. Secure Computation with Silent Preprocessing from PCGs 49
7.1. Generic MPC with correlated pseudo randomness 49
7.2. MPC template with PCG 50
7.3. Pseudoramdom Corellation Generator, definitions 51
8. Compressing random-VOLE 54
8.1. Sparse VOLE -> Pseudorandom VOLE 54
8.2. Secret sharing $\boldsymbol{e} \cdot x$ succinctly 56
8.3. Batching 62
8.4. PCG for VOLE from LPN 63
9. Commit and Sample 65
9.1. Central Binomial Distribution 65

9.2. Commit-then-Sample vs Sample-then-Commit 66
9.3. Pseudorandom Bits are Sufficient for CBD 67
9.4. Mixed Low-Degree Pseudorandom Generator 67
9.5. Commiting to sparse e_{seed} 70
9.6. LPN Parameter selection 72
9.7. ZKP protocol for batch verifying MLWE relation 73
10. Conclusion 77
10.1. Future Work 77
References 79

Chapter 1.

Introduction

1.1. The Problem: Post-Quantum Primitives

As quantum computing continues to develop, it is becoming increasingly clear that many of today's public-key cryptographic schemes will not remain secure in the long term. Quantum algorithms such as Shor's algorithm [1] show that problems like integer factorization and discrete logarithms, which are the basis of systems such as RSA, Diffie–Hellman, and elliptic-curve cryptography, can be solved efficiently on a sufficiently powerful quantum computer. This makes it necessary to explore new cryptographic constructions that can provide security even in the presence of quantum adversaries.

There is a real time pressure around the development of practical post-quantum primitves. Some projections suggest that quantum computers capable of breaking today's public-key cryptography could appear within the next one to two decades [2], [3]. This leaves a limited window to design, standardize, and deploy cryptographic alternatives that can be used at scale before current systems become obsolete. This challenge is often referred to as the "Y2Q" (Years to Quantum) problem. The idea is simple: if the transition to post-quantum cryptography is delayed for too long, the eventual arrival of quantum-capable adversaries could lead to widespread and sudden security failures across existing digital infrastructure.

1.2. Lattice-Based Primitives

The fundamental hardness assumption upon which post-quantum primitives rest is that it is computationally difficult to find a low-norm vector $\boldsymbol{s}$ satisfying

$$A \cdot \boldsymbol{s} = \boldsymbol{t} \bmod q$$

Based on different variants of this problem, such as Regev-based schemes, LWE, and MLWE, one can construct various primitives such as digital signatures, encryption schemes, and key exchange protocols.

Privacy-preserving protocols built on lattice hardness assumptions typically require proving knowledge of a vector $\boldsymbol{s}$ satisfying some linear relation like $A\boldsymbol{s} = \boldsymbol{t}$. In the discrete logarithm setting, an analogous statement, namely knowledge of a secret s with $g^s = t$, admits a clean and efficient solution via Schnorr's protocol [4]. The lattice case is considerably more involved: the norm constraint $\|\boldsymbol{s}\|$ must be small for the statement to be meaningful, and enforcing this smallness in zero-knowledge turns out to be one of the central practical challenges in lattice-based cryptography.

1.3. Zero-Knowledge Proofs: Background

Zero-knowledge proofs allow a machine, often called the prover, to convince another machine, often called the verifier, about the truth of a logical statement.

There are many different approaches to constructing zero-knowledge proofs, including schemes based on zk-SNARKs, interactive oracle proofs (IOPs), MPC-in-the-head techniques, and garbled circuits. Each of these paradigms comes with different trade-offs in terms of efficiency, interaction, communication complexity, and memory usage.

Broadly speaking, these approaches can be categorized into two classes:

- Low communication complexity with high prover memory requirements, such as zk-SNARKs, IOP-based constructions, and some MPC-in-the-head schemes. These protocols often achieve proof sizes that are sublinear in the circuit size, or even sublinear in the witness length, but typically require the prover to hold the entire circuit or witness in memory.
- Higher communication complexity with low prover memory overhead, as in the case of zero-knowledge garbled-circuit-based (ZKGC) schemes. In ZKGC constructions, the circuit is garbled gate by gate, meaning the prover never needs to store more than a constant amount of state at a time, keeping memory usage low at the cost of larger overall communication.

As illustrated in Figure 1 (adapted from [5]), many existing protocols require prover memory proportional to the number of gates t in the circuit being proven. This imposes practical limits on the size of circuits that can be handled efficiently.

	Protocol Type	Spartan [Set20] zk-SNARK	Virgo [ZXZS20] IOP-based	Ligero [AHIV17] MPC-in-the-head	[HK20] ZKGC	Wolverine sVOLE-based
Merkle tree (boolean circuit)	Prover time	55 s	53 s	400 s	7.3 s	11 s
	Verifier time	< 0.1 s	< 0.1 s	< 0.1 s	7.3 s	11 s
	Overall time	55 s	53 s	400 s	7.3 s	11 s
	Communication	$\leq$ 100 KB	253 KB	1.5 MB	182.2 MB	12.4 MB
	Prover memory	$\approx$ 7 GB	$\approx$ 1 GB	$\approx$ 5 GB	$\leq$ 400 MB	$\leq$ 400 MB
Matrix mult. (arithmetic circuit)	Prover time	677 s	64 s	–	–	320 s
	Verifier time	< 0.1 s	< 0.1 s	–	–	320 s
	Overall time	677 s	64 s	–	–	320 s
	Communication	$\leq$ 100 KB	$\approx$ 200 KB	–	–	4.2 GB
	Prover memory	$\approx$ 86 GB	$\approx$ 18 GB	–	–	$\leq$ 400 MB

Figure 1: Comparisons of ZK protocols

This is where Vector Oblivious Linear Evaluation, or VOLE-based zero-knowledge proofs, become particularly attractive. These constructions strike a more balanced trade-off: they allow for efficient provers in both time and memory, while achieving significantly lower communication than ZKGC-based approaches with comparable prover efficiency. As a result, they scale better to large circuits.

1.4. This Thesis

In this thesis, we construct zero-knowledge range proofs in the setting of the Learning With Errors (LWE) problem. We prove that an LWE ciphertext C, formed using a public matrix A, a short secret $\boldsymbol{s}$, and a small error $\boldsymbol{e}$, is valid and well-formed, without revealing $\boldsymbol{s}$ or $\boldsymbol{e}$. We work in the module-LWE (MLWE) setting, adopting the parameters and distributions from the post-quantum cryptography standard CRYSTALS-Kyber, ensuring that our constructions are grounded in a practically relevant and well-studied instantiation. Techniques presented in this thesis can be used for constructing proofs for lattice-based primitives, threshold encryption, FHE schemes, and more.

The standard VOLE-based zero-knowledge proof paradigm works as follows: the prover commits to their inputs and executes a proof protocol over those commitments, opening them according to the verifier's challenge. The soundness error of the protocol is directly tied to the binding security of the commitment scheme, which is the probability that a cheating prover can open a commitment to a different value than the one originally committed.

VOLE-based proofs work naturally over fields, where they yield a soundness error of $1/|\mathbb{F}|$. In Kyber, however, we work over rings. Unlike fields, rings may have zero divisors and lack the clean algebraic structure that standard VOLE analysis relies on, necessitating new techniques. We ask whether we can construct commitment schemes for a witness in R_q. We answer this in the positive, leveraging the fact that coefficients of the elements lie in a finite field. We use a variant called subfield-VOLE (sVOLE) to achieve the required security. We give an efficient VOLE-based commitment scheme for committing to ring elements in R_q^l.

Most VOLE commitment schemes first sample random VOLEs, which are analogous to random OTs, and then derandomize them to their inputs. We ask whether these random VOLEs can be sampled with communication sublinear in the number of VOLEs required. Boyle et al. [6] answered this in the positive. We take a detour to explore techniques from silent secure computation and the construction of a VOLE PCG, presenting them in an accessible format.

We next ask whether we can derandomize VOLE commitments to a vector of size u with communication sublinear in u. We answer this in the positive by exploiting the structure of the distribution from which u is sampled. Concretely, we show an optimization when u is sampled from the central binomial distribution. We employ a trick called commit-and-sample: we commit first and then apply CBD sampling to the committed values. In the batch setting, we are able to reduce our proof size by roughly one third.

1.5. Project Organisation and How to Read the Thesis

The report is divided as follows. Chapter 2 contains preliminaries and definitions; this should be skipped if the reader has a background in cryptography. Chapter 3 introduces VOLE correlations and some of their properties. Chapter 4 demonstrates how to build proofs given VOLEs. Chapter 5 introduces range proofs. Chapter 6 introduces the commitment scheme for elements in R_q and the zero-knowledge proof for an MLWE relation, along with the proof for the unoptimized CPA version of ML-KEM. Chapter 7 elaborates the ideas from silent secure computation and the VOLE PCG construction. Chapter 8 gives optimizations for the proofs from Chapter 6 using the techniques introduced in Chapter 7.

How to read this thesis if you have a PhD in secure computation or related field: Chapter 3 (Intro to VOLE correlation) → Skip to the end of Chapter 4 for the VOLE-ZK.BatchVer functionality → Chapter 6 → Chapter 8 → Chapter 9.

Remark: p and q are sometimes used interchangeably.

Gen AI declaration I used Gen AI such as Claude and Chat GPT in two ways: first to understand the literature. Since a lto of the literature was in papers, I often used Gen

AI to explain me a protocol with an example, since that is how I learn best. Secondly, I used it to write better. I would write as section and i need, use Gen AI to synthesize and write typst.it

Chapter 2.

Definitions and Notation

This chapter introduces some preliminaries required to read this thesis by a general computer scientist or a masters student. This section can be skipped by a more advanced reader or someone familiar with cryptology. The best way to read this chapter would be to refer back to these definitions as and when they come up.

2.1. Mathematical background:

2.1.1. Groups $(G, \cdot)$

A *group* is a set G together with a binary operation "$\cdot$" that combines any two elements to form a third, satisfying the following four axioms:

Closure For all $a, b \in G$, we have $a \cdot b \in G$.
Associativity For all $a, b, c \in G$, we have $(a \cdot b) \cdot c = a \cdot (b \cdot c)$.
Identity There exists an element $e \in G$ such that $a \cdot e = e \cdot a = a$ for all $a \in G$.
Inverse For every $a \in G$ there exists an element $a^{-1} \in G$ such that $a \cdot a^{-1} = a^{-1} \cdot a = e$.

If additionally $a \cdot b = b \cdot a$ for all $a, b \in G$, the group is called *abelian* (or commutative). The *order* of a group is the number of elements $|G|$.

Examples.

- $(\mathbb{Z}, +)$: the integers under addition. Identity is 0, inverse of a is $-a$.
- $(\mathbb{Z}_N, +)$ where $\mathbb{Z}_n := \{0, ..., N-1\}$ and $a + b := a + b \bmod N$
- $(\mathbb{Z}_N^*, \times)$: where $\mathbb{Z}_N^* := \{a \in \mathbb{Z}_N \mid \text{GCD}(a, N) = 1\}$ Identity is 1. If N is not a prime, the inverse can be found by Extended Euclidean method and the order of the group is given by $\varphi(n)$. If N is a prime, the group has a nice structure $\{1, ..., N-1\}$ since each number is coprime to N, and inverse of a is $a^{q-2} \bmod q$ by Fermat's little theorem, the order of group $\varphi(n) = n - 1$.
- $(\{0,1\}^n, \oplus)$: bit-strings of length n under XOR. Identity is the all-zero string, and every element is its own inverse.

2.1.2. Rings

A *ring* is a set R together with two binary operations, (often denoted as) addition $+$ and multiplication $\cdot$, satisfying the following axioms:

1. $(R, +)$ is an abelian group with identity 0
2. $(R, \cdot)$
   2.1 Closure under $\cdot$: For all $a, b \in R$, we have $a \cdot b \in R$.
   2.2 Associativity of $\cdot$: For all $a, b, c \in R$, we have $(a \cdot b) \cdot c = a \cdot (b \cdot c)$.
   2.3 Multiplicative identity: There exists $1 \in R$ such that $a \cdot 1 = 1 \cdot a = a$ for all $a \in R$.
   Note: Multiplication is not necessarily commutative for all elements, and multiplicative inverses do not necessarily exist

Distributivity For all $a, b, c \in R$: $a \cdot (b + c) = a \cdot b + a \cdot c$ and $(a + b) \cdot c = a \cdot c + b \cdot c$.

If additionally $a \cdot b = b \cdot a$ for all $a, b \in R$, the ring is called *commutative*. Note that unlike a group, a ring does not require multiplicative inverses to exist — elements other than 0 may not be invertible.

Examples.
- $(\mathbb{Z}_N, +, \cdot)$: the integers modulo n under multiplication and addition.
- $(\mathbb{Z}_N, +, \cdot)$ where $N = 2^{32}, 2^{64}$ etc

2.1.3. Fields

A *field* is a set $\mathbb{F}$ together with two binary operations, addition $+$ and multiplication $\cdot$, such that:

Abelian group under + $(\mathbb{F}, +)$ is an abelian group with identity 0.
Abelian group under · $(\mathbb{F} \setminus \{0\}, \cdot)$ is an abelian group with identity 1.
Distributivity For all $a, b, c \in \mathbb{F}$: $a \cdot (b + c) = a \cdot b + a \cdot c$.

In other words, a field is a commutative ring in which every nonzero element has a multiplicative inverse. The key distinction from a ring is that division is always possible (except by 0).

Examples.

- $\mathbb{Z}_q$ for a prime q: the integers modulo q. This is a finite field of order q, often written $\mathbb{F}_q$. Every nonzero element a has inverse $a^{q-2} \bmod q$ by Fermat's little theorem.

2.1.4. Finite Field $\mathbb{F}_p$

A finite field is a field that is a finite set; this means that it has a finite number of elements on which multiplication, addition, subtraction and division (excluding division by zero) are defined and satisfy the field axioms.

The elements of the finite field $\mathbb{F}_p$ are $\{0, 1, ...p - 1\}$

2.1.5. Polynomials over $\mathbb{F}_p$

A polynomial f whose coefficients lie in the field $\mathbb{F}_p$

- $f(t) = 3t^2 + 5t_{11} \in \mathbb{F}_{17}[t]$
- $f(t) = 12t^9 - 3t^2 + 5t + 11 \in F_{\{17\}}[t]$

2.1.6. Polynomial Rings over $\mathbb{F}_p[X]$

Let q be a prime modulus. The polynomial ring $\mathbb{Z}_q[x]$ is the set of all polynomials in x with coefficients in $\mathbb{Z}_q$. When adding, subtracting, multiplying, or dividing polynomials in $\mathbb{Z}_q[x]$, all coefficient arithmetic is performed modulo q.

Example. Let $q = 11$, and consider $f(x) = 3 + 2x + 5x^2 + x^3 \in \mathbb{Z}_{11}[x]$ and $g(x) = 4 + 7x + 3x^2 \in \mathbb{Z}_{11}[x]$. Then:

- $f(x) + g(x) = 7 + 9x + 8x^2 + x^3$.
- $f(x) - g(x) = 10 + 6x + 2x^2 + x^3$.
- $f(x) \times g(x) = 12 + 29x + 34x^2 + 26x^3 + 19x^4 + 3x^5 = 1 + 7x + 1x^2 + 4x^3 + 8x^4 + 3x^5 \pmod{11}$.

Question: what is the difference between a polynomial over $\mathbb{F}_p$ and polynomial ring over $\mathbb{F}_p$

A polynomial with coefficients in F_p is a polynomial. On the otherr hand, the set of all such polynomials is the polynomial ring. This can be infinite as degree of the indeterminent x can be anything.

2.1.7. Extension Field $GF_{p^r} = \mathbb{F}_p[X]/f(x)$

If we take the above polynomial ring and make it finite, and also a field, by reducing each element by an irreducible polynomial $f(x)$ we call it an extension field

For example: $GF(2^8) = \mathbb{F}_2[x]/(f(x) = x^8 + x^4 + x^3 + x + 1)$

This looks like a polynomial with co-efficients in the base field $\mathbb{F}_p$ (note: coefficients are *not* in F_p^r), but r number of such coefficients.

$$f = a_0 + a_1(X) + ... + a^{r-1}(X^{r-1})$$

The size of the field thus is $\underbrace{p \cdot p \cdot ... \cdot p}_{r} = p^r$

2.1.8. Quotient Ring $\mathcal{R} = \mathbb{F}_p[x]/f(x)$

We can make the polynomial ring finite by reducing every element modulo some fixed polynomial $f(x)$, called the *reduction polynomial*. Formally, we define the quotient ring

$$\mathcal{R} = \mathbb{F}_p[x]/(f(x))$$

where $(f(x))$ denotes the ideal generated by $f(x)$, that is, the set of all polynomial multiples of $f(x)$. Two polynomials $a(x), b(x) \in \mathbb{F}_p[x]$ are considered equivalent if their difference is divisible by $f(x)$:

$$a(x) \equiv b(x) \bmod (f(x))$$

The elements of $\mathcal{R}$ are the remainder classes — polynomials of degree strictly less than $\deg(f)$. Every element of $\mathbb{F}_p[x]$ has a unique representative of degree less than $\deg(f)$, obtained by dividing by $f(x)$ and taking the remainder. Arithmetic in $\mathcal{R}$ is then just arithmetic in $\mathbb{F}_p[x]$ followed by reduction modulo $f(x)$.

The reduction polynomial $f(x)$ does not need to be irreducible. However, the choice of $f(x)$ determines the algebraic structure of $\mathcal{R}$:

- If $f(x)$ is *irreducible* over $\mathbb{F}_p$, then $\mathcal{R} = \mathbb{F}_p[x]/(f(x))$ is an extension field $\mathbb{F}_{p^n}$ where $n = \deg(f)$. Every nonzero element has a multiplicative inverse.
- If $f(x)$ is *reducible*, then $\mathcal{R}$ is a ring but not a field. It will contain zero divisors — nonzero elements whose product is zero. This is the case for $\mathcal{R}_q = \mathbb{Z}_q[x]/(x^n + 1)$ used in Kyber, where $x^n + 1$ factors over $\mathbb{Z}_q$.

Example: effect of the choice of $f(x)$. Let $p = 5$ and consider the polynomial ring $\mathbb{F}_5[x]$. We will quotient by two different degree-2 polynomials and compare the resulting structures.

Case 1: reducible $f(x)$. Let $f(x) = x^2 - 1 = (x - 1)(x + 1)$, which factors over $\mathbb{F}_5$. The quotient ring is

$$\mathcal{R} = \mathbb{F}_5[x]/(x^2 - 1).$$

Elements are polynomials of the form $a + bx$ with $a, b \in \mathbb{F}_5$, so $|\mathcal{R}| = 25$. The relation $x^2 \equiv 1$ holds in $\mathcal{R}$. Now consider the two nonzero elements $u = x - 1$ and $v = x + 1$. Their product is

$$u \cdot v = (x - 1)(x + 1) = x^2 - 1 \equiv 0 \bmod (x^2 - 1).$$

So $u \cdot v = 0$ in $\mathcal{R}$ despite both u and v being nonzero. These are zero divisors, confirming that $\mathcal{R}$ is a ring but *not* a field — neither u nor v can have a multiplicative inverse.

Case 2: irreducible $f(x)$. Let $f(x) = x^2 + 2$, which is irreducible over $\mathbb{F}_5$ since it has no roots ($0^2 + 2 = 2$, $1^2 + 2 = 3$, $2^2 + 2 = 1$, $3^2 + 2 = 1$, $4^2 + 2 = 3$, none equal 0). The quotient ring is

$$\mathcal{R} = \mathbb{F}_5[x]/(x^2 + 2) = \mathbb{F}_{5^2} = \mathbb{F}_{25}.$$

Elements are again polynomials $a + bx$ with $a, b \in \mathbb{F}_5$, so $|\mathcal{R}| = 25$ as before. However, now every nonzero element has a multiplicative inverse. For example, take $u = 1 + x$. To find u^{-1}, we need $a + bx$ such that

$$(1 + x)(a + bx) \equiv 1 \bmod (x^2 + 2).$$

Expanding: $(a + bx + ax + bx^2) = a + (a + b)x + bx^2$. Using $x^2 \equiv -2 \equiv 3 \bmod (5)$, this becomes $(a + 3b) + (a + b)x$. Setting equal to 1:

$$a + 3b = 1 \quad \text{and} \quad a + b = 0.$$

Solving: $b = -a$, so $a - 3a = 1$, giving $-2a = 1$, so $a = -1/2 \equiv 2p \bmod 5$ and $b = -2 \equiv 3p \bmod 5$. Hence $(1 + x)^{-1} = 2 + 3x$ in $\mathbb{F}_{25}$.

Summary.

	$f(x)$ reducible	**$f(x)$ irreducible**
Structure	Ring	Field
Zero divisors	Yes	No
Every nonzero element invertible	No	Yes
Example	$\mathbb{F}_5[x]/(x^2 - 1)$	$\mathbb{F}_5[x]/(x^2 + 2) = \mathbb{F}_{25}$

2.1.9. Kyber Polynomial (quotient) Ring $\mathcal{R}_q = \mathbb{Z}_q[x]/(x^n + 1)$

The polynomial ring $\mathcal{R}_q = \mathbb{Z}_q[x]/(x^n + 1)$ is defined as follows. Let q be a prime modulus and n a positive integer. The ring $\mathcal{R}_q$ consists of all polynomials in $\mathbb{Z}_q[x]$ of degree less than n, where multiplication is performed modulo the reduction polynomial $x^n + 1$.

To multiply two polynomials $f(x), g(x) \in \mathcal{R}_q$:

1. Multiply $f(x)$ and $g(x)$ in $\mathbb{Z}_q[x]$, obtaining a polynomial $h(x)$ of degree at most $2n - 2$.
2. Divide $h(x)$ by $x^n + 1$ to get a remainder polynomial $r(x)$ of degree at most $n - 1$.
3. Then $f(x) \times g(x) = r(x)$ in $\mathcal{R}_q$.

Note that the size of $\mathcal{R}_q$ is q^n.

Example. Let $q = 17$ and $n = 4$, so $\mathcal{R}_q = \mathbb{Z}_{17}[x]/(x^4 + 1)$. Then $\mathcal{R}_q$ consists of all polynomials in $\mathbb{Z}_{17}[x]$ of degree at most 3. Let

$$f(x) = 3 + 5x + x^2 + 2x^3 \in \mathcal{R}_q \quad \text{and} \quad g(x) = 1 + 4x + 6x^2 + 3x^3 \in \mathcal{R}_q.$$

Computing $h(x) = f(x) \times g(x)$ in $\mathbb{Z}_{17}[x]$ gives

$$h(x) = 3 + 17x + 34x^2 + 36x^3 + 37x^4 + 20x^5 + 13x^6 + 6x^7.$$

Reducing modulo $x^4 + 1$ is accomplished by replacing x^4 by -1, x^5 by $-x$, x^6 by $-x^2$, and x^7 by $-x^3$, and then reducing coefficients modulo 17. We obtain

$$\begin{aligned} r(x) &= 3 + 17x + 34x^2 + 36x^3 - 37 - 20x - 13x^2 - 6x^3 \\ &= (3 - 37) + (17 - 20)x + (34 - 13)x^2 + (36 - 6)x^3 \\ &= 0 + 14x + 21x^2 + 30x^3 \\ &= 14x + 4x^2 + 13x^3 \pmod{17}. \end{aligned}$$

So $f(x) \times g(x) = 14x + 4x^2 + 13x^3$ in $\mathcal{R}_q$.

2.1.10. Representing Polynomials as Vectors

A polynomial $f(x) = a_0 + a_1 x + ... + a_{n-1}x^{n-1} \in \mathcal{R}_q$ can be represented by its vector of coefficients $\boldsymbol{f} = (a_0, a_1, ..., a_{n-1})$ of length exactly n. This representation is useful as it makes the coefficient arithmetic explicit and connects polynomial operations to familiar linear algebra.

Example. Consider $\mathcal{R}_q = \mathbb{Z}_{41}[x]/(x^4 + 1)$. The polynomials $f(x) = 23 + 11x^2 + 7x^3$ and $g(x) = 40 + 5x + 16x^2$ are represented by $\boldsymbol{f} = (23, 0, 11, 7)$ and $\boldsymbol{g} = (40, 5, 16, 0)$. In $\mathcal{R}_q$:

- $\boldsymbol{f} + \boldsymbol{g} = (22, 5, 27, 7)$,
- $\boldsymbol{f} - \boldsymbol{g} = (24, 36, 36, 7)$,
- $\boldsymbol{f} \times \boldsymbol{g} = (12, 3, 29, 7)$.

2.1.11. The Module $\mathcal{R}_q^k$

Let k be a positive integer. The module $\mathcal{R}_q^k$ consists of length-k column vectors of polynomials in $\mathcal{R}_q$. Addition and subtraction are performed component-wise, so the result is again an element of $\mathcal{R}_q^k$. The inner product of two vectors $\boldsymbol{a}, \boldsymbol{b} \in \mathcal{R}_q^k$ is defined as

$$\boldsymbol{a}^\top \boldsymbol{b} = \sum_{i=1}^{k} \boldsymbol{a}[i] \cdot \boldsymbol{b}[i] \in \mathcal{R}_q,$$

which yields a single polynomial in $\mathcal{R}_q$. All vectors in $\mathcal{R}_q^k$ are written as column vectors.

Example. Let $q = 137$, $n = 4$, $\mathcal{R}_q = \mathbb{Z}_{137}[x]/(x^4 + 1)$, and $k = 3$. Consider

$$\boldsymbol{a} = \begin{pmatrix} 93 + 51x + 34x^2 + 54x^3 \\ 27 + 87x + 81x^2 + 6x^3 \\ 112 + 15x + 46x^2 + 122x^3 \end{pmatrix} \quad \text{and} \quad \boldsymbol{b} = \begin{pmatrix} 40 + 78x + x^2 + 119x^3 \\ 11 + 31x + 57x^2 + 90x^3 \\ 108 + 72x + 47x^2 + 14x^3 \end{pmatrix} \in \mathcal{R}_q^3.$$

Then:

$$\boldsymbol{a}+\boldsymbol{b}=\begin{pmatrix}133+129x+35x^2+36x^3\\38+118x+x^2+96x^3\\83+87x+93x^2+136x^3\end{pmatrix},$$

$$\boldsymbol{a}-\boldsymbol{b}=\begin{pmatrix}53+110x+33x^2+72x^3\\16+56x+24x^2+53x^3\\4+80x+136x^2+108x^3\end{pmatrix},$$

and the inner product is

$$\boldsymbol{a}^\top\boldsymbol{b}=\boldsymbol{a}[1]\cdot\boldsymbol{b}[1]+\boldsymbol{a}[2]\cdot\boldsymbol{b}[2]+\boldsymbol{a}[3]\cdot\boldsymbol{b}[3]=93+59x+44x^2+132x^3.$$

2.1.12. Size of Polynomials

To reason about the hardness of problems like MLWE, we need a notion of "size" for the algebraic objects we work with — concretely, for:

- integers in $\mathbb{Z}_q$,
- polynomials in $\mathcal{R}_q=\mathbb{Z}_q[x]/(x^n+1)$, and
- vectors of polynomials in $\mathcal{R}_q^k$.

The size measure we use is the *infinity norm*, denoted $\|\cdot\|_\infty$, which informally captures the largest coefficient in absolute value. To define this precisely we first need the notion of *symmetric mod*, which centres the representatives of $\mathbb{Z}_q$ around zero rather than taking values in $\{0,...,q-1\}$.

Symmetric mod. For $r\in\mathbb{Z}_q$ we define the symmetric representative as the unique integer $r'\in(-(q-1)/2,(q-1)/2]$ such that $r'\equiv r \bmod q$. We write this as $r'=r \bmod^{\pm} q$. For example with $q=19$, the element $18\in\mathbb{Z}_{19}$ has symmetric representative -1, and 7 stays as 7.

We define the infinity norm $\|\cdot\|_\infty$ for elements at each level of the module hierarchy.

- **Integers modulo q.** Let $r\in\mathbb{Z}_q$. Then $\|r\|_\infty=|r \bmod q|$, where we represent r in the range $\left(-\frac{q-1}{2},\frac{q-1}{2}\right]$. In particular, $0\le\|r\|_\infty\le\frac{q-1}{2}$ if q is odd, and $0\le\|r\|_\infty\le\frac{q}{2}$ if q is even.

  Example. Let $q=19$. Then $\|7\|_\infty=7$ and $\|18\|_\infty=1$.

- **Ring elements.** Let $f(x)=f_0+f_1x+...+f_{n-1}x^{n-1}\in\mathcal{R}_q$. Then $\|f\|_\infty=\max_i\|f_i\|_\infty$.

  Example. Let $f(x)=1+12x+3x^3+18x^5\in\mathcal{R}_{19}$. Then $\|f\|_\infty=7$, since the centred mod q representation of f is $1-7x+3x^3-x^5$.

- **Module elements.** Let $\boldsymbol{a}=[a_1,a_2,...,a_k]^\top\in\mathcal{R}_q^k$. Then $\|\boldsymbol{a}\|_\infty=\max_i\|a_i\|_\infty$.

2.1.13. "Small" Polynomials

A polynomial $f\in\mathcal{R}_q$ is said to be *small* if $\|f\|_\infty$ is small relative to q. Formally, let η be a positive integer with $\eta\ll q/2$, and define

$$S_\eta=\left\{f\in\mathcal{R}_q\mid\|f\|_\infty\le\eta\right\}$$

to be the set of polynomials in $\mathcal{R}_q$ all of whose coefficients, when reduced modulo q, have absolute value at most η. We refer to S_η as the set of *small polynomials* with bound η.

Example. Let $q = 31$. Then $1 + 30x + 29x^2 + x^4 + 2x^5 \in S_2$, since the centred mod q representation of the coefficients gives values in $\{-2, -1, 0, 1, 2\}$.

Example. S_1 is the set of polynomials in $\mathcal{R}_q$ all of whose coefficients, when reduced modulo q, lie in $\{-1, 0, 1\}$.

2.1.14. Module Learning with Errors (MLWE)

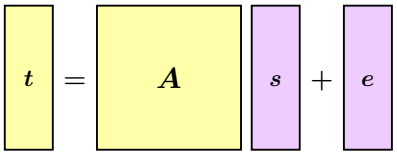
The MLWE problem is parametrised by a prime q, integers n, k, ℓ with $k \geq \ell$, and integers $\eta_1, \eta_2 \ll q/2$. An MLWE instance is generated as follows: sample a uniformly random matrix $\boldsymbol{A} \xleftarrow{R} \mathcal{R}_q^{k \times \ell}$ where $\mathcal{R}_q = \mathbb{Z}_q[x]/(x^n + 1)$, a secret vector $\boldsymbol{s} \xleftarrow{R} S_{\eta_1}^{\ell}$, and an error vector $\boldsymbol{e} \xleftarrow{R} S_{\eta_2}^{k}$. Compute

$$\boldsymbol{t} = \boldsymbol{A}\boldsymbol{s} + \boldsymbol{e} \in \mathcal{R}_q^k.$$

Search MLWE. Given $(\boldsymbol{A}, \boldsymbol{t})$, find $\boldsymbol{s}$.

Decision MLWE. Given $(\boldsymbol{A}, \boldsymbol{t})$, distinguish it from $(\boldsymbol{A}, \boldsymbol{u})$ where $\boldsymbol{u} \xleftarrow{R} \mathcal{R}_q^k$:

$$(\boldsymbol{A}, \boldsymbol{A}\boldsymbol{s} + \boldsymbol{e}) \underset{c}{\approx} (\boldsymbol{A}, \boldsymbol{u}).$$



Example. Let $q = 541$, $n = 4$, $k = 3$, $\ell = 2$, $\eta_1 = 3$, $\eta_2 = 2$, so $\mathcal{R}_q = \mathbb{Z}_{541}[x]/(x^4 + 1)$. Sample:

$$\boldsymbol{A} = \begin{pmatrix} 442 + 502x + 513x^2 + 15x^3 & 368 + 166x + 37x^2 + 135x^3 \\ 479 + 532x + 116x^2 + 41x^3 & 12 + 139x + 385x^2 + 409x^3 \\ 29 + 394x + 503x^2 + 389x^3 & 9 + 499x + 92x^2 + 254x^3 \end{pmatrix} \in \mathcal{R}_q^{3 \times 2},$$

$$\boldsymbol{s} = \begin{pmatrix} 2 - 2x + x^3 \\ 3 - 2x - 2x^2 - 2x^3 \end{pmatrix} \in S_3^2, \quad \boldsymbol{e} = \begin{pmatrix} 2 - 2x - x^2 \\ 1 + 2x + 2x^2 + x^3 \\ -2 - x^2 - 2x^3 \end{pmatrix} \in S_2^3.$$

Computing $\boldsymbol{t} = \boldsymbol{A}\boldsymbol{s} + \boldsymbol{e}$:

$$\boldsymbol{t} = \begin{pmatrix} 30 + 252x + 401x^2 + 332x^3 \\ 247 + 350x + 259x^2 + 485x^3 \\ 534 + 234x + 137x^2 + 443x^3 \end{pmatrix}, \quad \|\boldsymbol{t}\|_\infty = 259.$$

The MLWE problem is: given $(\boldsymbol{A}, \boldsymbol{t})$, recover $\boldsymbol{s}$.

2.2. Cryptographic Primitives

2.2.1. Interactive Proof System

A machine called the Prover claims that a certain logical statement is true, namely that $x \in L$, and wants to convince another machine, called the verifier. They both engage in an interactive protocol, which ends with the Verifier outputting either accept or reject and then halt.

Definition *The pair (P, V) is an interactive proof system for L if it satisfies the following two conditions:*

Completeness*: If $x \in L$, then the probability that (P, V) rejects x is negligible in the length of x.*

Soundness*: If $x \notin L$ then for any prover P^*, the probability that (P^*, V) accepts x is negligible in the length of x.*

Note: Definitions used from material provided in the course *Cryptologic Protocol theory* at Aarhus university in 2025

2.2.2. Proof Of Knowledge

A Proof of Knowledge is a stronger variant of an Interactive Proof System, where the prover's claim has a different flavor: he not only claims $x \in L$ but also knows some piece of information w (such as a secret key corresponding to a given public one). For example x is an instance of the discrete log problem, and w is the solution to it. The Prover claims that it actually knows an x such that $(x, w) \in R$.

Definition: *Let $\kappa()$ be a function from bit strings to the interval $[0..1]$. The protocol (P, V) is said to be a proof of knowledge for the relation $R \subset \{0,1\}^* X \{0,1\}^*$ with knowledge error κ, if the following are satisfied:*

Completeness: *On common input x, if the honest prover P gets as private input w such that $(x, w) \in R$, then the verifier V always accepts.*

Knowledge soundness: *There exists a probabilistic algorithm M called the knowledge extractor. This M gets input x and rewindable black-box access to the prover and attempts to compute w such that $(x, w) \in R$. We require that the following holds: For any prover P^*, let $\varepsilon(x)$ be the probability that V accepts on input x. There exists a constant c such that whenever $\varepsilon(x) > \kappa(x)$, M will output a correct w in expected time at most*

$$\frac{|x| \ c}{\varepsilon(x) - \kappa(x)}$$

where access to P^* counts as one step only.

Note: Definitions used from material provided in the course *Cryptologic Protocol theory* at Aarhus university in 2025

2.2.3. Zero Knowledge

We want to formalize the fact that by executing an interactive protocol, the prover does not reveal anything beyond the truth of the statement to the verifier. A verifer can try whatever strategy, but learns nothing more except the truth of the prover's claim. We use the simulation paradigm. So if there exists a simulator that when given the input $x \in L$, but not the witness w can produce a "fake" transcript (i.e everything that the verifier sees) that is indistinguishable from the real transcript, then the real transcript leaks nothing about the w really.

A bit more formally, *an interactive proof system or argument (P, V) for language L is zero-knowledge if for every probabilistic polynomial time verifier V^*, there is a simulator*

S_V^* running in expected probabilistic polynomial time on inputs $x \in L$, such that we have $S_V^* \underset{c}{\sim} < P, V^* >$ on input $x \in L$ and arbitrary δ (as input to V^* only).

Note: Definitions used from material provided in the course *Cryptologic Protocol theory* at Aarhus university in 2025

2.2.4. Commitment Schemes

Suppose Alice and Bob had a divorce and want to decide who gets the dog. They are talking on the phone and agree on a coin flip protocol where Alice tosses a coin, and Bob calls heads or tails. If the coin matches Bob's call, he gets the dog, otherwise not. But what is preventing Alice from cheating? If there was a way for Alice to commit to her coin toss and open it later after Bob made his choice we could prevent cheating. Perhaps Alice could put it in a locked box and send it to Bob. Bob makes the choice and then Alice sends him her key.

A commitment scheme is a cryptographic primitive that allows one to commit to a chosen value (or chosen statement) while keeping it hidden to others, with the ability to reveal the committed value later.[7] Commitment schemes are designed so that a party cannot change the value or statement after they have committed to it: that is, commitment schemes are binding. On the other hand, the other party cannot look inside the commitment to figure out what the first party committed to before getting the opening string: that is; commitment schemes are binding.

We will think of a commitment scheme (Gen, Commit, CheckParams) as being defined by a pair of polynomial time algorithms.

$\mathbf{Gen}(\mathbf{1}^{\lambda}) \rightarrow \boldsymbol{pp}$: **Gen**, which takes as input 1^{λ} where λ is a security parameter and corresponds to e.g. the length of the modulus we want. It outputs a string pp, the *public parameters* of the commitment scheme.

$\mathbf{CheckParams}(\boldsymbol{pp}) \rightarrow \{\mathbf{accept}, \mathbf{reject}\}$: **CheckParams** checks that the public parameters are valid and well-formed.

$\mathbf{Commit}_{\boldsymbol{pp}}(\boldsymbol{m}; \boldsymbol{r}) \rightarrow \boldsymbol{c}$: $\mathbf{Commit}_{pp}$ commits to a message m using randomness r.

To open, the party who generated the commitment sends m, r to the other party who can then verify the commitment

2.2.5. Pseudo-random Number Generator (PRG)

We need randomness everywhere in cryptology, but sources of true randomness are hard to come by. This necessitates a function that can stretch a few "true random bits" (seed) to generate more "random looking bits", i.e. pseudorandom bit strings that "look random" to an adversary.

Definition (fixed security parameter version): A (t, ε)-PRNG is a function $G : \{0,1\}^n \rightarrow \{0,1\}^m$ $(m \gg n)$ such that

- G is efficiently computable by a deterministic algorithm.

- G "ε-fools" all t-time statistical tests, that is, for all t-time algorithms A

$$\Pr[A(G(S)) \text{ accepts} \mid S \leftarrow \{0,1\}^m] - \Pr[A(R) \mid R \leftarrow \{0,1\}^m] < \varepsilon$$

  We say that the uniform distribution on $\{0,1\}^m$ is (t,ε)-indistinguishable from distributions $\{G(S) \mid S \leftarrow \{0,1\}^n\}$.

2.2.6. Pseudorandom Function

Suppose Alice wishes to authenticate herself to Bob, by proving she knows a secret that they share. With a PRG they could proceed as follows. They both seed a PRG with the shared secret. Bob picks some random number i and sends it to Alice, and Alice proves she knows the share secret by responding with the ith random number generated by the PRG.

But this solution requires state, and they both have to compute i random numbers. Instead, we would like "random access" to the sequence. This is the intuition behind pseudo-random functions: Bob gives Alice some random i, and Alice returns $F_K(i)$, where $F_K(i)$ is indistinguishable from a random function, that is, given any $x_1, ..., x_m, F_K(x_1), ..., F_K(x_m)$, no adversary can predict $F_K(x_{m+1})$ for any x_{m+1}

Definition: a function $f : \{0,1\}^n \times \{0,1\}^s \to \{0,1\}^m$ is a (t,ε,q)-PRF if

- Given a key $K \in \{0,1\}^s$ and an input $X \in \{0,1\}^n$ there is an "efficient" algorithm to compute $F_K(X) = F(X,K)$.
- For any t-time oracle algorithm A, we have

$$|\Pr_{K \leftarrow \{0,1\}^s}[A^{f_K}] - \Pr_{f \in \mathcal{F}}[A^f]| < \varepsilon$$

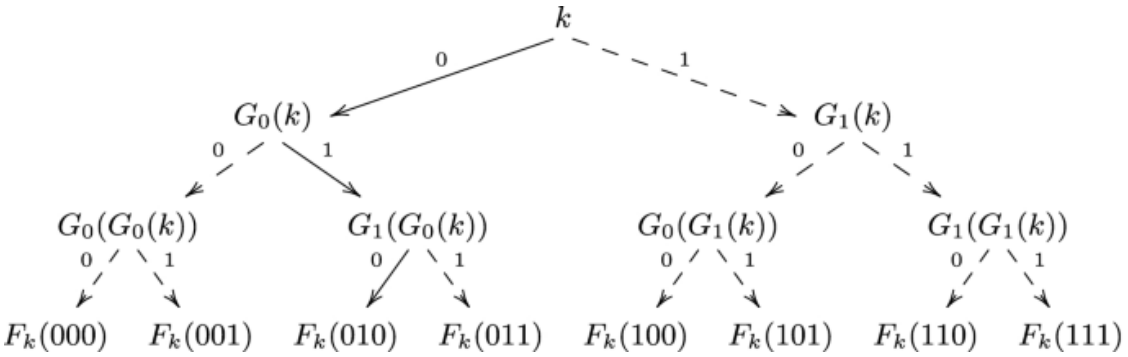
where $\mathcal{F} = \{f : \{0,1\}^n \to \{0,1\}^m\}$ and A makes at most q queries to the oracle.

2.2.7. GGM Tree

A classical result by Goldreich, Goldwasser and Micali [8] shows that a PRF can be built from any length-doubling pseudorandom generator, i.e., one with expansion rate 2. The idea is as follows. Let G be a length-doubling PRG and write $G(k) = G_0(k) \parallel G_1(k)$ where $|G_0(k)| = |G_1(k)| = \ell$, denoting the first and second halves of G's output respectively. A PRF $F_k : \{0,1\}^\ell \to \{0,1\}^\ell$ with key k and ℓ-bit input x is then defined by:

$$F_k(x_1 x_2 ... x_\ell) = G_{x_\ell}\Big(...\Big(G_{x_2}\Big(G_{x_1}(k)\Big)\Big)...\Big).$$

The structure of this construction is naturally captured by a binary tree of depth ℓ. The root takes the key k as input to the PRG, and at every internal node the left child receives G_0 of its parent's value while the right child receives G_1. To evaluate F_k on an input x, one traverses the tree from the root to a leaf by following the bits of x in order: go left at depth i if $x_i = 0$, and right if $x_i = 1$. Crucially, one does not need to materialise the entire tree — only the values along the single root-to-leaf path need to be computed, making evaluation efficient. Figure 1 illustrates the construction for $\ell = 3$, with solid lines marking the path taken to evaluate $F_k(010)$.



Theorem (GGM construction, adapted from [8]. If G is a length-doubling ε-indistinguishable pseudorandom generator, then the construction outlined above is a $qd\varepsilon$ -indistinguishable pseudorandom function, where q denotes the number of queries and d denotes the tree's depth.

2.2.8. Point Function

A point function P_{xy} is a function which evaluates to 0 at all points/indices except at some point x where it evaluates to y

2.2.9. Distributed Point function

The goal is of a Distributed Point Function(DPF) [9] is to additively share a secret point function using a succinct representation. For example, the shares are F_0 and F_1 s.t $P_{xy} = F_0 \oplus F_1$

More formally, a Distributed Point Function (DPF) is defined by two PPT algorithms:

- $\text{Gen}(x, y) = (k_0, k_1)$
- $\text{Eval}(k, x')$

such that the following properties hold:

- **Correctness.** $P_{xy}(x') = \text{Eval}(k_0, x') \oplus \text{Eval}(k_1, x')$
- **Secrecy.** k_0 (or separately k_1) can be simulated given only $|x|$ and $|y|$.

2.2.10. Private Puncturable PRFs

A puncturable pseudorandom function (PRF) [10] has a master key k that enables one to evaluate the PRF at all points of the domain, and has a punctured key k_x that enables one to evaluate the PRF at all points but one. The punctured key k_x reveals no information about the value of the PRF at the punctured point x.

A private puncturable PRF is one where k_x reveals nothing about index x. A privately puncturable PRF is also a DPF: set k_0 to be the master PRF key k, and set the key k_1 to be the punctured key k_{x^*} , punctured at x^*

2.2.11. Proving Circuits with Linear Commitments

This is a general framework showing that linearly homomorphic commitments can be used to construct zero-knowledge proofs [11]. The core idea is to prove the computation of a circuit gate by gate.

Goal. Given a circuit C, the prover wishes to convince the verifier that it knows a witness w such that $C(w) = z$, without revealing w.

Commit phase. The prover commits to the extended witness w, consisting of the circuit inputs together with the output wire of every multiplication gate.

Invariant. The protocol maintains the invariant that when the computation of a gate is proved, the commitments to its input wires have already been established and verified.

Evaluation. The prover and verifier evaluate the circuit gate by gate. The prover evaluates on the plain witness, while the verifier evaluates on the committed witness. Correctness of each gate evaluation is proved in turn.

Opening. The prover sends openings of the output wire commitments. The verifier checks that these open correctly to z. Correctness of z then follows from two observations:

1. Linear gates are handled for free by the homomorphism property of the commitment scheme: the verifier can compute commitments to linear combinations of wires directly, without any interaction.
2. Multiplication gates require an additional argument, since homomorphism alone does not suffice. This check is protocol-dependent, and different constructions have developed their own techniques to make it efficient.

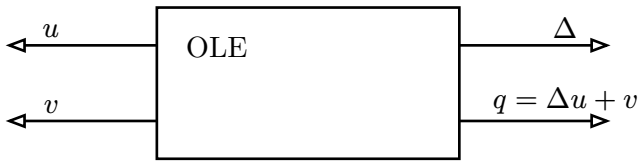
Chapter 3.

Vector Oblivious Linear Evaluations(VOLE) and properties

There exist several established approaches for constructing zero-knowledge proof systems, including techniques based on garbled circuits, MPC-in-the-head, and succinct non-interactive arguments of knowledge (SNARKs). In recent years, an alternative paradigm based on Vector Oblivious Linear Evaluations (VOLEs) has attracted significant attention and led to a number of efficient zero-knowledge constructions. This section introduces the notion of Vector Oblivious Linear Evaluations (VOLEs) and discusses their key properties. Throughout this thesis, we sometimes use the terms *sender* and *prover* interchangeably with *receiver* and *verifier*. This is because, in many VOLE-based proof systems, the sender of the VOLE protocol naturally assumes the role of the prover, while the receiver corresponds to the verifier.

3.1. Oblivious Linear Evaluation (OLE):

An Oblivious Linear Evaluation (OLE) can be viewed as a correlated randomness primitive between two parties. One party, commonly referred to as the sender, obtains random values u, v, while the other party, the receiver, obtains a random value Δ together with the value $q = u \cdot \Delta + v$. Throughout this section, we model OLE as an ideal functionality providing this correlation.



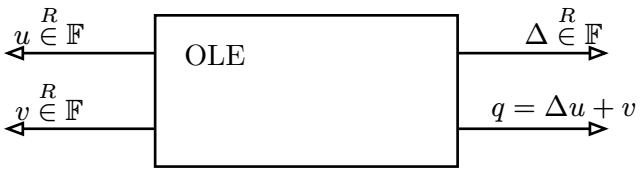
Although the term *linear* is standard in the literature, it can be somewhat misleading. A more accurate interpretation is to view OLE as an instance of an *oblivious affine evaluation*. Under this perspective, the sender holds a description of an affine function

$f(\delta) = \alpha \cdot \delta + \beta,$

while the receiver holds an input δ and learns only the evaluation $f(\delta)$, without gaining any additional information about the coefficients α and β. Note that we purposely do not say what domain the inputs and the parameters lie, as based on different domains we can get different variants.

3.2. Random-OLE:

This is a variant of OLE, where the prover and the verifier are given OLEs on a randomly chosen u, v, Δ by the Ideal Functionality.



Random OLE implies OLE: The prover and the verifier can de-randomize a random-OLE to get a valid OLE co-relation on their chosen u, v, Δ by just sending 3 field elements.

3.3. OLE as (2,2)-secret sharing of multiplication product $u \cdot \Delta$:

An OLE correlation can be interpreted as an additive (2,2)-secret sharing of the product $u \cdot \Delta$. In this view, the sender(Alice) holds the share

$$s_A = -v$$

While the receiver holds the share

$$s_B = q$$

.

Individually, neither party learns $u \cdot \Delta$; however, by combining their shares, they can reconstruct the product:

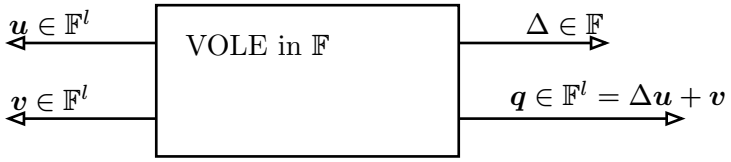
$$\begin{aligned} s_A + s_B &= -v + q \\ &= -v + (u \cdot \Delta + v) \\ &= u \cdot \Delta \end{aligned}$$

3.4. Vector Oblivious Linear Evaluation (VOLE)

A Vector Oblivious Linear Evaluation(VOLE) generalizes OLE by extending the coefficients of the affine functions as vectors instead of scalars, while the input, δ stays a scalar.

$$\overrightarrow{f(\delta)} = \vec{\alpha}\delta + \vec{\beta}$$

In a VOLE instance, the values u, v and q are vectors of some length l and chosen from some field $\mathbb{F}$, while Δ remains a scalar.





3.5. s-VOLE

This is a variant of random VOLE where the witness u has only λ' bits, much less then the security parameter λ. When we will introduce VOLE as a commitment scheme we would see that we require the mask to have λ bits. To achieve the required security the mask v and Δ is sampled from an extension field $\mathbb{F}_{p^r}$ of size λ while $\boldsymbol{u}$ is from the base field $\mathbb{F}_p$ and lifted to $\mathbb{F}_{p^r}$. This is given by the $\mathcal{F}^r_{\text{spVOLE}}$ functionality.

Functionality: $\mathcal{F}^r_{\text{sVOLE}}$

Public parameters: Base field characteristic p, extension parameter r.

Initialize: Upon receiving (Init) from $\mathcal{P}$ and $\mathcal{V}$, sample $\Delta \xleftarrow{R} \mathbb{F}_{p^r}$. Store global key Δ and send it to $\mathcal{V}$. Ignore all subsequent (Init) commands.

sVOLE: This procedure can be run multiple times. Upon receiving (svole, ℓ) from $\mathcal{P}$ and $\mathcal{V}$, where ℓ indicates a length parameter:

1. Sample $\boldsymbol{u} \xleftarrow{R} \mathbb{F}_p^\ell$.
2. Execute $\text{Auth}(\boldsymbol{u}, p, r, \Delta)$ so that the parties obtain $[\![\boldsymbol{u}]\!]$.

Macro: $\text{Auth}(\boldsymbol{u}, p, r, \Delta)$

1. Sample $\boldsymbol{v} \xleftarrow{R} \mathbb{F}_{p^r}^\ell$.
2. Compute $\boldsymbol{q} := \boldsymbol{v} + \Delta \cdot \boldsymbol{u} \in \mathbb{F}_{p^r}^\ell$.
3. Send $[\![\boldsymbol{u}]\!]$ to the parties, i.e. send $(\boldsymbol{u}, \boldsymbol{v})$ to $\mathcal{P}$ and $\boldsymbol{v}$ to $\mathcal{V}$.

After the Init phase, $\mathcal{V}$ holds a fixed global key $\Delta \in \mathbb{F}_{p^r}$. Each call to sVOLE produces a tuple $(\boldsymbol{u}, \boldsymbol{v}, \boldsymbol{q})$ satisfying

$$\boldsymbol{q} = \Delta \cdot \boldsymbol{u} + \boldsymbol{v},$$

where $\boldsymbol{u}, \boldsymbol{v} \in \mathbb{F}_p^\ell$ are held by $\mathcal{P}$ and $\boldsymbol{q} \in \mathbb{F}_{p^r}^\ell$ is held by $\mathcal{V}$. We write this as $[\![\boldsymbol{u}]\!]$ and refer to it as a *subfield VOLE commitment* to $\boldsymbol{u}$.

Remark: All protocols in this thesis will use $\mathcal{F}^r_{\text{sVOLE}}$ as a helper functionality. When quantifying the costs of such protocols, we will measure the number of sVOLE correlations consumed (which translates to the sum of lengths ℓ used in the sVOLE instruction within $\mathcal{F}^r_{\text{sVOLE}}$ across multiple invocations).

Realizing $\mathcal{F}^r_{\text{spVOLE}}$: It can be realized by multiple protocols such as OT, LPN, MPC-In-the head. We will later give an efficient protocol to realize it with sublinear communication

De-randomization: $\mathcal{P}$ and $\mathcal{V}$ can later de-randomize the random VOLES received from a protocol realizing $\mathcal{F}^r_{\text{sVOLE}}$ by sending correction values to get the correct VOLE for the provers witness w, we will refer to this as $[\![w]\!]$.

Protocol: VOLE.Derand (Derandomising a random sVOLE)

Public parameters: Base field $\mathbb{F}_p$, extension field $\mathbb{F}_{p^r}$, functionality $\mathcal{F}_{\text{sVOLE}}^r$ with global key $\Delta \in \mathbb{F}_{p^r}$ held by $\mathcal{V}$, length parameter ℓ.

$\mathcal{P}$'s input: A vector $\boldsymbol{w} \in \mathbb{F}_p^\ell$ and a random sVOLE $(\boldsymbol{u}, \boldsymbol{v})$ from $\mathcal{F}_{\text{sVOLE}}^r$.

$\mathcal{V}$'s input: Global key $\Delta \in \mathbb{F}_{p^r}$ and the corresponding $\boldsymbol{q} = \Delta \cdot \boldsymbol{u} + \boldsymbol{v}$ from $\mathcal{F}_{\text{sVOLE}}^r$.

Output: A VOLE $[\![\boldsymbol{w}]\!]$ where $\mathcal{P}$ holds $[\![\boldsymbol{w}]\!]_{\mathcal{P}} = (\boldsymbol{w}, \boldsymbol{v})$ and $\mathcal{V}$ holds $[\![\boldsymbol{w}]\!]_{\mathcal{V}} = (\boldsymbol{q}', \Delta)$, satisfying $\boldsymbol{q}' = \Delta \cdot \boldsymbol{w} + \boldsymbol{v}$.

Step 1 Compute correction. $\mathcal{P}$ computes the correction vector

$$\boldsymbol{d} \coloneqq \boldsymbol{w} - \boldsymbol{u} \in \mathbb{F}_p^\ell$$

and sends $\boldsymbol{d}$ to $\mathcal{V}$.

Step 2 Adjust. $\mathcal{V}$ updates its value as

$$\boldsymbol{q}' \coloneqq \boldsymbol{q} + \Delta \cdot \boldsymbol{d} = \Delta \cdot (\boldsymbol{u} + \boldsymbol{d}) + \boldsymbol{v} = \Delta \cdot \boldsymbol{w} + \boldsymbol{v}.$$

Output.

The parties now hold $[\![\boldsymbol{w}]\!]$ where
- $\mathcal{P}$ holds $[\![\boldsymbol{w}]\!]_{\mathcal{P}} \coloneqq (\boldsymbol{w}, \boldsymbol{v})$;
- $\mathcal{V}$ holds $[\![\boldsymbol{w}]\!]_{\mathcal{V}} \coloneqq (\boldsymbol{q}', \Delta)$.

Proposition *After running* VOLE.Derand *on* $\boldsymbol{w} \in \mathbb{F}_p^\ell$*, the output* $[\![\boldsymbol{w}]\!]$ *is a valid sVOLE, i.e.*

$$\boldsymbol{q}' = \Delta \cdot \boldsymbol{w} + \boldsymbol{v}.$$

Proof. By construction, $\mathcal{F}_{\text{sVOLE}}^r$ produces $(\boldsymbol{u}, \boldsymbol{v}, \boldsymbol{q})$ with $\boldsymbol{u}, \boldsymbol{v} \in \mathbb{F}_p^\ell$ satisfying $\boldsymbol{q} = \Delta \cdot \boldsymbol{u} + \boldsymbol{v}$. Since $\mathcal{P}$ is honest, it computes $\boldsymbol{d} = \boldsymbol{w} - \boldsymbol{u}$ correctly and sends this to $\mathcal{V}$. Since $\mathcal{V}$ is honest, it updates $\boldsymbol{q}' = \boldsymbol{q} + \Delta \cdot \boldsymbol{d}$. Substituting:

$$\begin{aligned} \boldsymbol{q}' &= \boldsymbol{q} + d\Delta \\ &= \Delta \cdot \boldsymbol{u} + \boldsymbol{v} + \Delta \cdot (\boldsymbol{w} - \boldsymbol{u}) \\ &= \Delta \cdot \boldsymbol{w} + \boldsymbol{v}. \end{aligned}$$

Hence $\boldsymbol{q}' = \Delta \cdot \boldsymbol{w} + \boldsymbol{v}$, which is a valid sVOLE on $\boldsymbol{w}$ with mask $\boldsymbol{v}$ and global key Δ. □

Note: If $\mathcal{P}$ is malicious, then it is a case of input substitution which we allow.

3.6. VOLE Correlations are Linear

VOLE correlations are linear, meaning that $\mathcal{P}$ and $\mathcal{V}$ can perform homomorphic operations on committed values *locally*, without any further interaction. This is a key property that will allow us to build the commit-and-prove style ZKP.

To see why, suppose $\mathcal{P}$ holds (u_1, v_1) and (u_2, v_2), and $\mathcal{V}$ holds Δ together with

$$q_1 = u_1\Delta + v_1 \quad \text{and} \quad q_2 = u_2\Delta + v_2.$$

Given public values $\alpha, \beta \in \mathbb{F}_p$, both parties can locally compute a new VOLE on $u' = \alpha u_1 + u_2 + \beta$ without any interaction. The prover computes

$$u' = \alpha u_1 + u_2 + \beta, \quad v' = \alpha v_1 + v_2,$$

and the verifier computes $q' = \alpha q_1 + q_2 + \beta\Delta$. To see that this is a valid VOLE, observe:

$$\begin{aligned} q' &= \alpha q_1 + q_2 + \beta\Delta \\ &= \alpha(u_1\Delta + v_1) + (u_2\Delta + v_2) + \beta\Delta \\ &= (\alpha u_1 + u_2 + \beta)\Delta + (\alpha v_1 + v_2) \\ &= u'\Delta + v'. \end{aligned}$$

These operations can be chained. Using the bracket notation $[\![\cdot]\!]$, the homomorphic operations are summarised as follows.

- **Add.** $[\![\boldsymbol{x} + \boldsymbol{y}]\!] = [\![\boldsymbol{x}]\!] + [\![\boldsymbol{y}]\!]$:
  - $\mathcal{P}$ computes $\boldsymbol{v}_z = \boldsymbol{v}_x + \boldsymbol{v}_y$
  - $\mathcal{V}$ computes $\boldsymbol{q}_z = \boldsymbol{q}_x + \boldsymbol{q}_y$
- **Add constant.** $[\![\boldsymbol{x} + \boldsymbol{c}]\!] = [\![\boldsymbol{x}]\!] + \boldsymbol{c}$ for a public constant $\boldsymbol{c} \in \mathbb{F}_p^\ell$:
  - $\mathcal{P}$ computes $\boldsymbol{v}_z = \boldsymbol{v}_x$
  - $\mathcal{V}$ computes $\boldsymbol{q}_z = \boldsymbol{q}_x + \Delta \cdot \boldsymbol{c}$
- **Multiply by constant.** $[\![c\boldsymbol{x}]\!] = c[\![\boldsymbol{x}]\!]$ for a public scalar $c \in \mathbb{F}_p$. Each party locally multiplies their value by c:
  - $\mathcal{P}$ computes $\boldsymbol{v}_z = c \cdot \boldsymbol{v}_x$
  - $\mathcal{V}$ computes $\boldsymbol{q}_z = c \cdot \boldsymbol{q}_x$

3.7. VOLE as a Commitment Scheme

With a protocol for $\mathcal{F}^r_{\text{spVOLE}}$ and VOLE.Derand, VOLE can be used to make a commitment scheme. The resulting commitment scheme has two sub-protocols: Commit and Open.

Commitment Scheme: `VOLE.Commit` **and** `VOLE.Open`

Public parameters: Base field $\mathbb{F}_p$, extension field $\mathbb{F}_{p^r}$, functionality $\mathcal{F}_{\text{sVOLE}}^r$ with global key $\Delta \in \mathbb{F}_{p^r}$ held by $\mathcal{V}$, length parameter ℓ.

Commit($\boldsymbol{w}$): To commit to a chosen $\boldsymbol{w} \in \mathbb{F}_p^\ell$ or $\boldsymbol{w} \in \mathbb{F}_{p^r}^\ell$:

1. $\mathcal{P}$ and $\mathcal{V}$ call $\mathcal{F}_{\text{sVOLE}}^r$ to obtain a random VOLE $(\boldsymbol{u}, \boldsymbol{v}, \boldsymbol{q})$ satisfying $\boldsymbol{q} = \Delta \cdot \boldsymbol{u} + \boldsymbol{v}$.
2. $\mathcal{P}$ and $\mathcal{V}$ run VOLE.Derand on $\boldsymbol{w}$, obtaining $[\![\boldsymbol{w}]\!]$ where:
   - $\mathcal{P}$ holds $[\![\boldsymbol{w}]\!]_{\mathcal{P}} := (\boldsymbol{w}, \boldsymbol{v})$;
   - $\mathcal{V}$ holds $[\![\boldsymbol{w}]\!]_{\mathcal{V}} := (\boldsymbol{q}', \Delta)$, with $\boldsymbol{q}' = \Delta \cdot \boldsymbol{w} + \boldsymbol{v}$.

We write this as $[\![\boldsymbol{w}]\!] \leftarrow \text{Commit}(\boldsymbol{w})$.

Open ($[\![\boldsymbol{w}]\!]$): To open a commitment $[\![\boldsymbol{w}]\!]$:

1. $\mathcal{P}$ sends $(\boldsymbol{w}, \boldsymbol{v})$ to $\mathcal{V}$.
2. $\mathcal{V}$ checks that $\boldsymbol{q}' = \Delta \cdot \boldsymbol{w} + \boldsymbol{v}$.
3. $\mathcal{V}$ outputs $b = 1$ if the check passes, and $b = 0$ otherwise.

We write this as $b \leftarrow \text{Open}([\![\boldsymbol{w}]\!])$.

Proposition (Perfectly hiding and statistically binding). *The commitment scheme* $(\text{Commit}, \text{Open})$ *is perfectly hiding and statistically binding given access to an ideal* $\mathcal{F}_{\text{spVOLE}}^r$ *functionality.*

Proof (Hiding). Since $\boldsymbol{v}$ is sampled uniformly at random from $\mathbb{F}_{p^r}^\ell$ by $\mathcal{F}_{\text{sVOLE}}^r$, the value $\boldsymbol{q}' = \Delta \cdot \boldsymbol{w} + \boldsymbol{v}$ is uniformly distributed over $\mathbb{F}_{p^r}^\ell$ regardless of $\boldsymbol{w}$. Hence $\mathcal{V}$'s view reveals no information about $\boldsymbol{w}$, and the scheme is perfectly hiding.

Proof (Binding). To break binding, $\mathcal{P}^*$ would need to open $[\![\boldsymbol{w}]\!]$ to a different value $\boldsymbol{w}' \neq \boldsymbol{w}$, i.e. find $(\boldsymbol{w}', \boldsymbol{v}')$ such that

$$\begin{aligned} \boldsymbol{q}' &= \Delta \boldsymbol{w} + \boldsymbol{v} = \Delta \cdot \boldsymbol{w}' + \boldsymbol{v}' \\ \Rightarrow \Delta &= (\boldsymbol{v}' - \boldsymbol{v}) \cdot (\boldsymbol{w} - \boldsymbol{w}')^{-1} \end{aligned}$$

This implied prover must have guessed Δ by coming up with $(\boldsymbol{w}', \boldsymbol{v}')$ that pass the check. This relies on $(\boldsymbol{w} - \boldsymbol{w}')^{-1}$ being invertible. In the case of a field the inverse exists and is unique. So exactly only one value of Δ satisfies the above relation, with probability at most $1/|\mathbb{F}_{p^r}|$. Hence the scheme is statistically binding with error $1/|\mathbb{F}_{p^r}|$.

Since we assume an ideal functionality $\mathcal{F}_{\text{sVOLE}}^{p,r}$, we obtain perfect hiding and statistical binding. In practice, the VOLE correlation can be generated in different ways such as from OT, from LPN, or via the VOLE-in-the-head paradigm. Therefore the concrete security guarantees depend on the chosen instantiation.

Chapter 4.

VOLE based zk POK

4.1. Quicksilver ZK

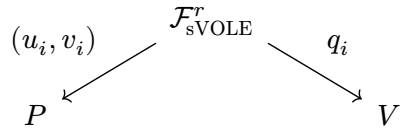
A line of works including Wolverine, Mac n cheese, Quicksilver - show methods for a Prover to demonstrate a Proof of knowledge for a witness w loosely based on techniques borrowed from Garbled Circuits and computing in a GMW style fashion using authenticated triplets(the authenticated triplets being VOLE relations)

In this section we elaborate VOLE based zero knowledge proof techniques along with the Quicksilver multiplication check with details and examples for a more general audience and in a more accessible way.

Quicksilver assumes access to a secure $\mathcal{F}^r_{\text{sVOLE}}$ functionality.

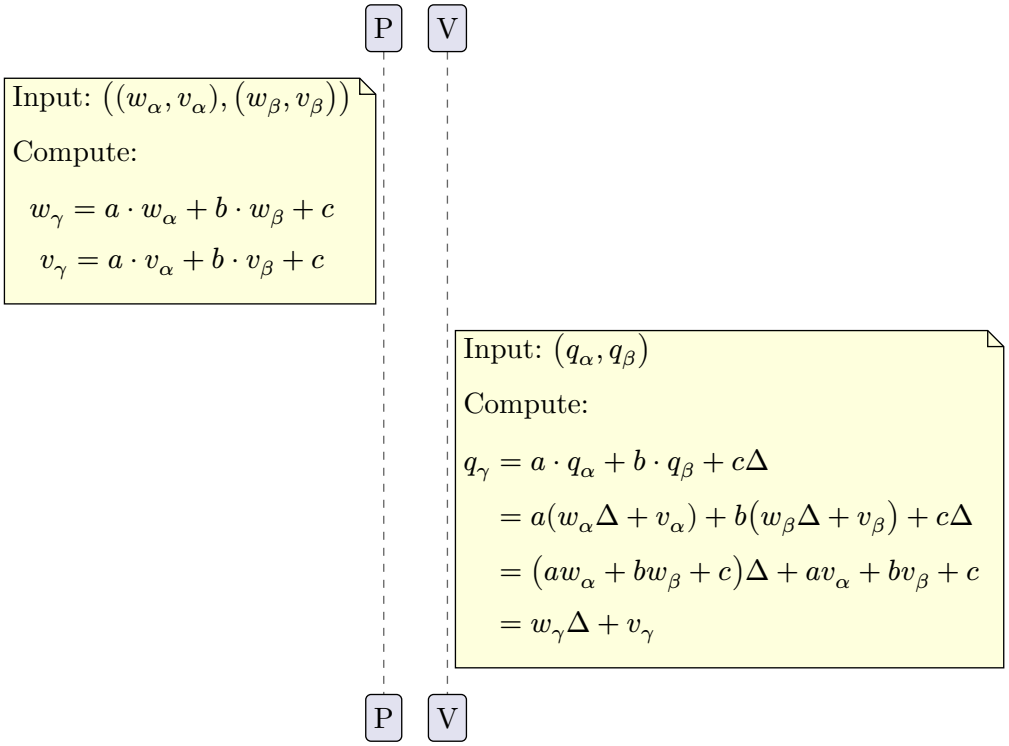
Let C be a circuit, containing t multiplication gates, for which prover knows a witness $w \in \mathbb{F}_2^n$ of length n such that $C(w) = 1$

Receiving pre-computed corellated randomness Prover and the verifier jointly request for $n + t$ random VOLEs from $\mathcal{F}^r_{\text{sVOLE}}$. The parties now have:



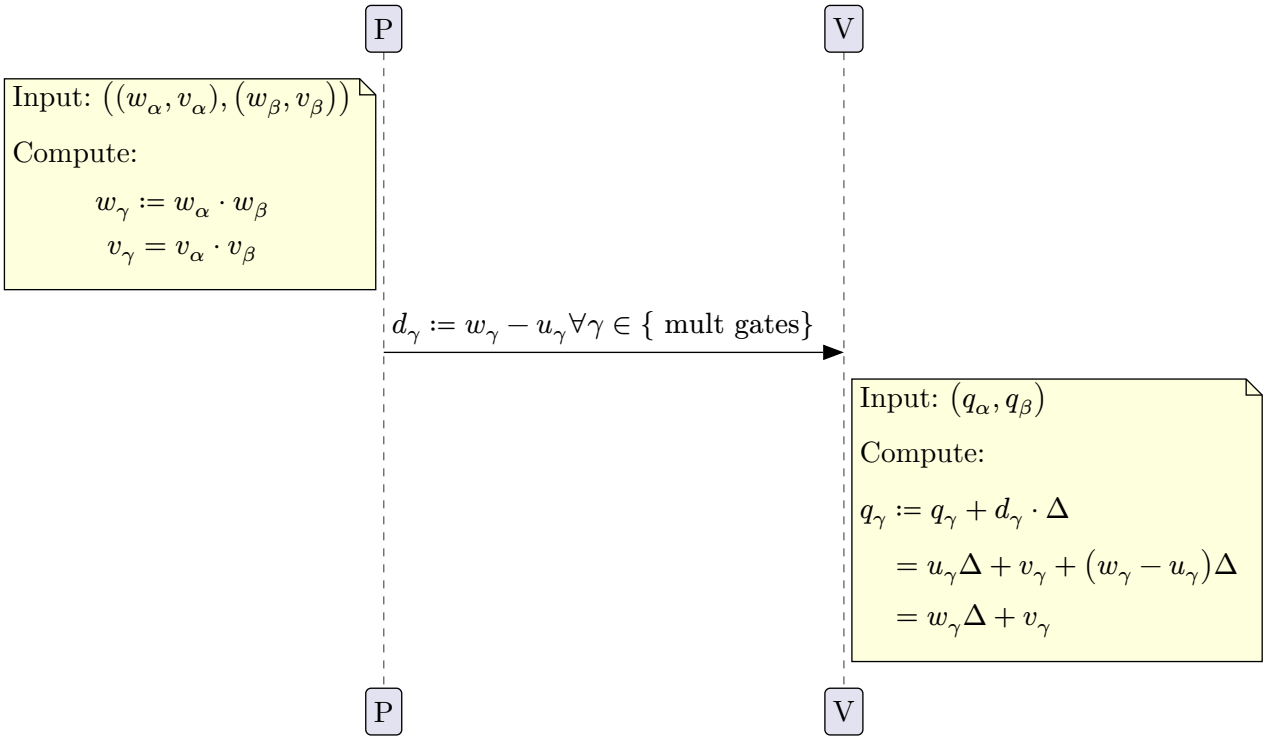
Commit Phase: In this step the parties convert their random VOLEs to valid VOLEs on the witness w. For every input element w_i, for $i \in [1..n]$, the Prover computes a correction value $d_i = w_i - u_u$. The Verifier computes $q_i = q_i + d_i\Delta$ for $i \in [1..n]$. By the end of this, parties now hold valid VOLE commitments (w_i, v_i, q_i) for $i \in [i...n]$ for the witness.

Computing the gates: The parties now compute the circuit in a gate by gate fashion, while always maintaining the invariant that they hold valid VOLEs authenticated on the witness to the input wires. For every gate in the circuit, with input wires w_α, and w_β, they proceed as follows:
1. Linear gates:



This does not require any interaction, and after this they hold valid VOLE commitments to the output wire w_γ as $q_\gamma = w_\gamma \cdot \Delta + v_\gamma$

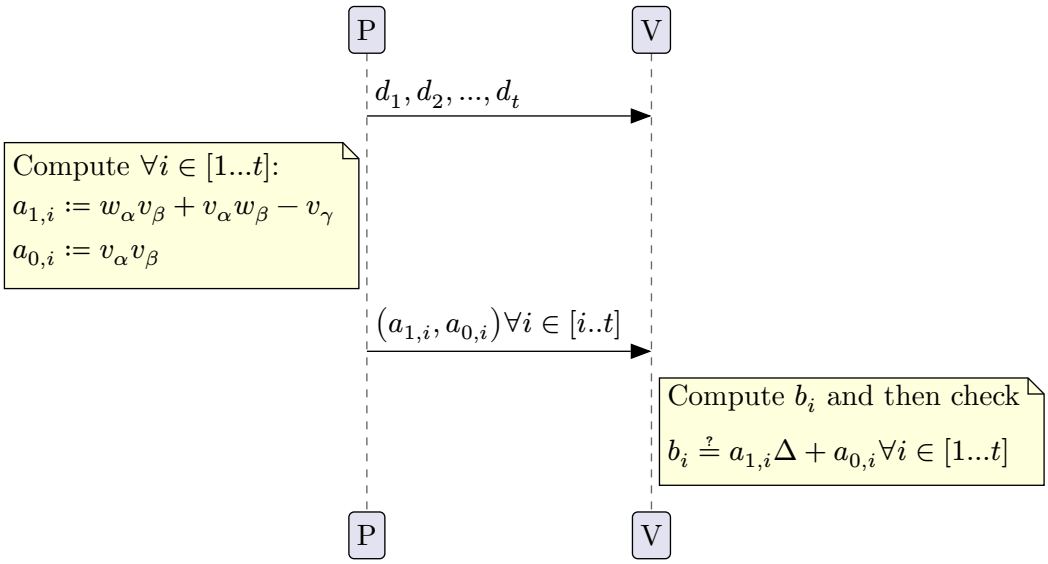
2. Multiplication gate: for the ith multiplication gate for $i \in [1..t]$, the prover computes $w_\gamma := w_\alpha \cdot w_\beta$ and sends $d_{n+i} := w_\gamma - u_{n+i} \in \mathbb{F}_2$ to the verifier. The verifier then computes $q_{n+i} := q_{n+i} + d_{n+i} \cdot \Delta$ which means that the tuple $(w_\gamma, v_{n+i}, q_{n+i})$ is now a valid VOLE MAC for the output of the i'th multiplication gate.



Checking Multiplications: Verifier needs to check if the prover sent the correct d_i for $i \in [i..t]$ values for each multiplication gate. V can compute:

$$\overbrace{q_\alpha q_\beta - q_\gamma \Delta}^{\text{known to V, denoted as } b_i} = (w_\alpha \Delta + v_\alpha)(w_\beta \Delta + v_\beta) - (w_\gamma \Delta + v_\gamma)\Delta$$
$$= (w_\alpha w_\beta - w_\gamma)\Delta^2 + w_\alpha v_\beta + v_\alpha w_\beta - v_\gamma + v_\alpha v_\beta$$
$$= 0\Delta^2 + \underbrace{(w_\alpha v_\beta + v_\alpha w_\beta - v_\gamma)}_{\substack{\text{known to P,} \\ \text{denoted as : } a_{1,i}}} \underbrace{\Delta}_{\substack{\text{global key} \\ \text{, known to V}}} + \underbrace{v_\alpha v_\beta}_{\text{known to P, denoted as : } a_{0,i}}$$

Note, if the prover was honest in their computation then the higher degree term : $w_\alpha w_\beta - w_\gamma = 0$, the above relationship becomes linear. This implies that the verifier only needs to check $b_i = a_{1,i}\Delta + a_{0,i}$ for $i \in [i..t]$, and output *accept* if the relation holds for all $i \in [1..t]$. If the underlying wire values(i.w $w_\alpha, w_\beta, w_\gamma$) are not computed correctly, then a cheating prover can convince the verifier into accepting with a probability of $\frac{2}{p^r}$: now it becomes a quadratic equation if Δ, where at most two values of Δ satisfy the equation.



4.1.1. Optimizing communication of the multiplication check:

Since each multiplication check is linear, Quicksilver optimized checking t multiplication gates at once by using a Linear Universal Hash.

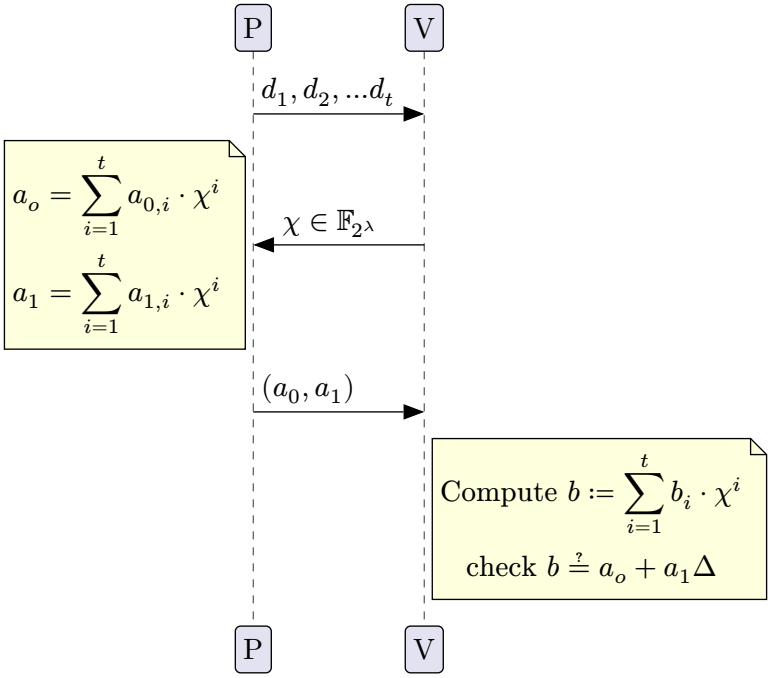
$$\sum b_i \cdot \chi^i = \underbrace{\sum (q_{\alpha,i} q_{\beta,i} - q_{\gamma,i}\Delta)\chi^i}_{\text{known to V, denoted as B}} = \underbrace{\sum a_{0,i} \cdot \chi^i}_{\substack{\text{known to P,} \\ \text{denoted as } A_0}} + \underbrace{\sum a_{1,i}\chi^i}_{\text{known to P, denoted as } A_1} \underbrace{\Delta}_{\text{known to V, global key}}$$

After receiving all the d_i correction values for the multiplication gates, the verifier samples a uniform element $\chi \in \mathbb{F}_{2^\lambda}$ and sends it to the prover. The prover computes the following linear combination for all the t gates, and sends it to the verifier:

$$a_o = \sum_{i=1}^{t} a_{0,i} \cdot \chi^i \qquad\qquad a_1 = \sum_{i=1}^{t} a_{1,i} \cdot \chi^i$$

This way the prover only sends a single pair (a_0, a_1) instead of to t pairs. On getting this, verifier computes $b = \sum_{i=[t]} b_i \cdot \mathrm{X}^i$ and checks if:

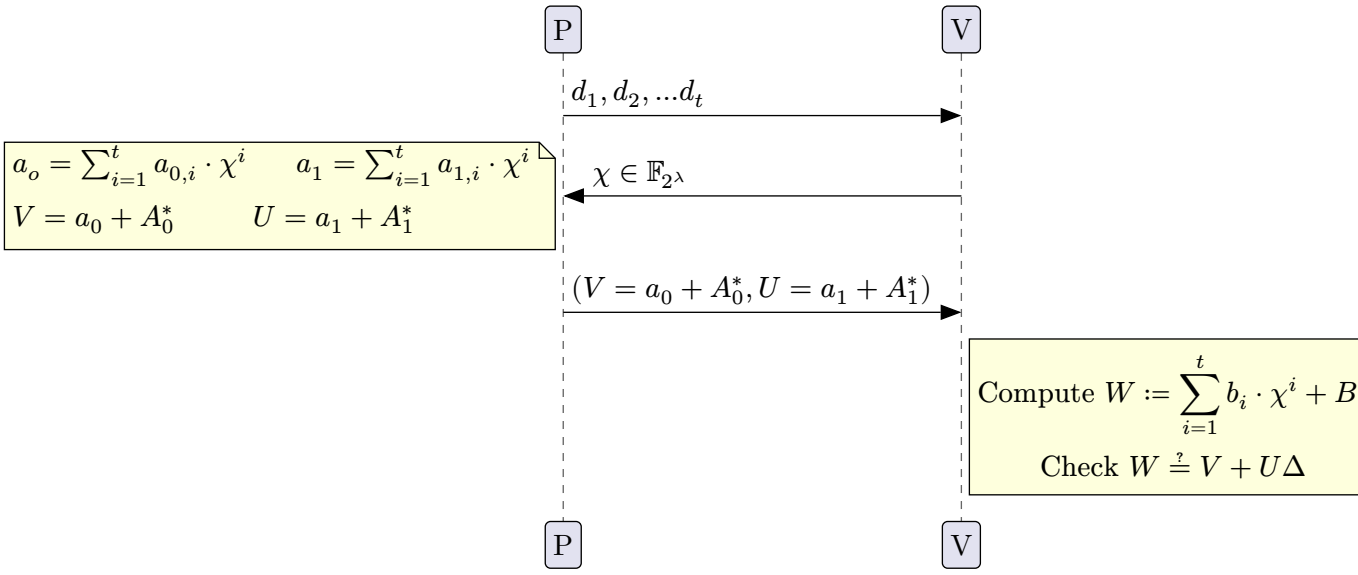
$$b \stackrel{?}{=} a_o + a_1\Delta$$



4.1.2. Adding zero knowledge

The above protocol is not yet zero knowledge as sending a_0, a_1 leaks information about the witness. We can still check $b \stackrel{?}{=} a_o + a_1\Delta$. P and V sample random elements $A_0^*, A_1^*, B^* \in \mathbb{F}_{p^r}$ such that $B^* = A_0^* + A_1^*\Delta$. Then,

$$b + B^* = a_o + A_0^* + a_1 + A_1^*\Delta$$
$$W = V + U\Delta$$



Let's prove this is a zero-knowledge proof.

4.1.2.1. Completeness:

This is evident by construction.

4.1.2.2. Soundness:

The case of malicious prover:

A prover might try and cheat while computing the multiplication value $w_\alpha \cdot w_\beta = w_\gamma$ and instead compute $w_\alpha \cdot w_\beta + e_i = w_\gamma$ for some $e_i \neq 0$.

This means if the prover is honest and $e_i = 0$, b_i is a linear equation in Δ

$$b_i = a_{0,i} + a_{1,i}\Delta$$

If the prover cheats and $e_i \neq 0$ then:

$$b_i = a_{0,i} + a_{1,i}\Delta + e_i\Delta^2$$

A quadratic equation over $\mathbb{F}_{p^r}$ has almost 2 roots i.e there are at most 2 values of Δ that satisfy the quadratic equation $b_i = a_2\Delta^2 + a_1\Delta + a_0$. So for a fixed bad gate, the relationship on b_i can only hold with a probability $\frac{2}{\text{size of the set Delta is sampled from}} = \frac{2}{p^r}$

Q: why is soundness sidderent in Quicksilver??

Note: that the online phase of the ZK protocol where the circuit and witness are known, can be made non-interactive by computing χ using a random oracle to hash the transcript up to that point. We can use the Fiat-Shamir heuristic to make the online phase non-interactive at the cost of that the information-theoretic security is degraded to the computation security. Specifically, both parties can compute $\chi \in \mathbb{F}_{p^r}$ as $H(d_1, ..., d_t)$, where $H : \{0,1\}^* \to \mathbb{F}_{p^r}$ is a cryptographic hash function modeled as a random oracle and $p^r \geq 2^\kappa$. In this case, the soundness error for the batch check of multiplication gates together with the correctness of the single output is now bounded by $(q_H + t + 3)/p^r \leq (q_H + t + 3)/2^\kappa$, where q_H is an upper bound of the number of H queries made by the adversary.

4.2. Translating this idea to d-degree polynomial constraints.

FAEST 2 and PERK both translate this idea to checking $d-$degree polynomials which is also described in the next section of the Quicksilver paper. This process is described with an example here. This should be read with the Quicksilver writeup of FAEST 2.0. Some things therefore have been copied directly from FAEST v2 specification, and later elaborated in the example.

The parties want to evaluate various, low-degree constraints on the committed witness. Let the constraints be determined by functions $(f_1, ..., f_C)$, where each function can be represented as a degree-d circuit, and the goal of the proof is to show that for all $i \in [1..C]$,

$$f_i(w_0, ..., w_{l-1}) = 0$$

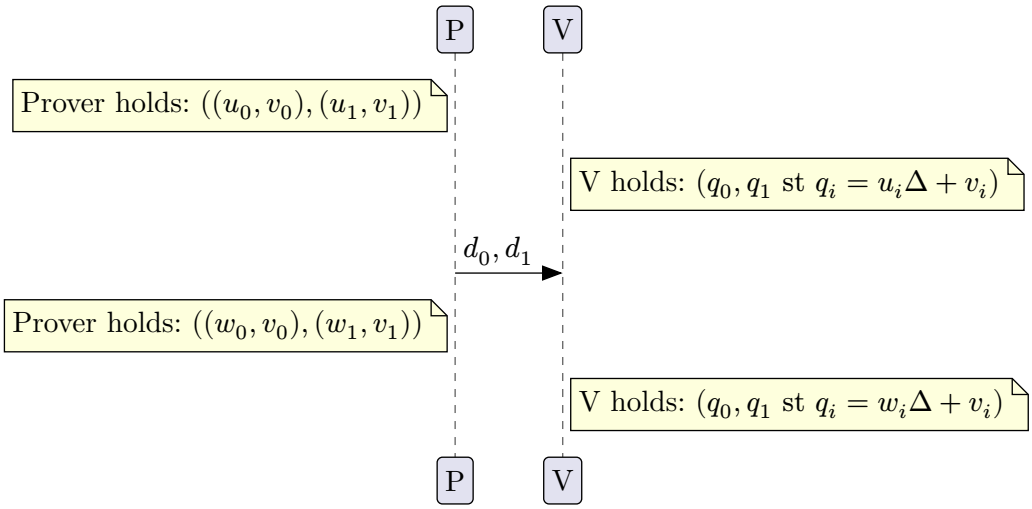
For our example, let the prover has some witness $w = (w_0, w_1)$ and wants to prove $C = 2$ constraints. $f_i(w_0, w_1) = 0 \forall i \in \{1..C\}$

Let the constraints be as follows for our example:

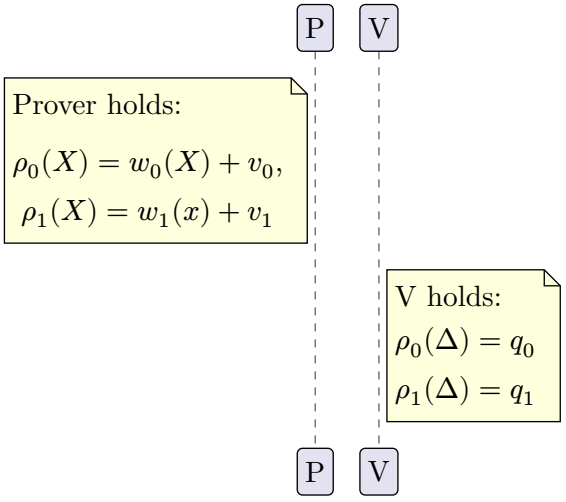
$$f_1(w_0, w_1) = w_0 + w_1 = 0$$

$$f_2(w_0, w_1) = w_0 \cdot w_1 = 0$$

The prover and the verifier first request for 2 random VOLES, then embed the witness w in them :



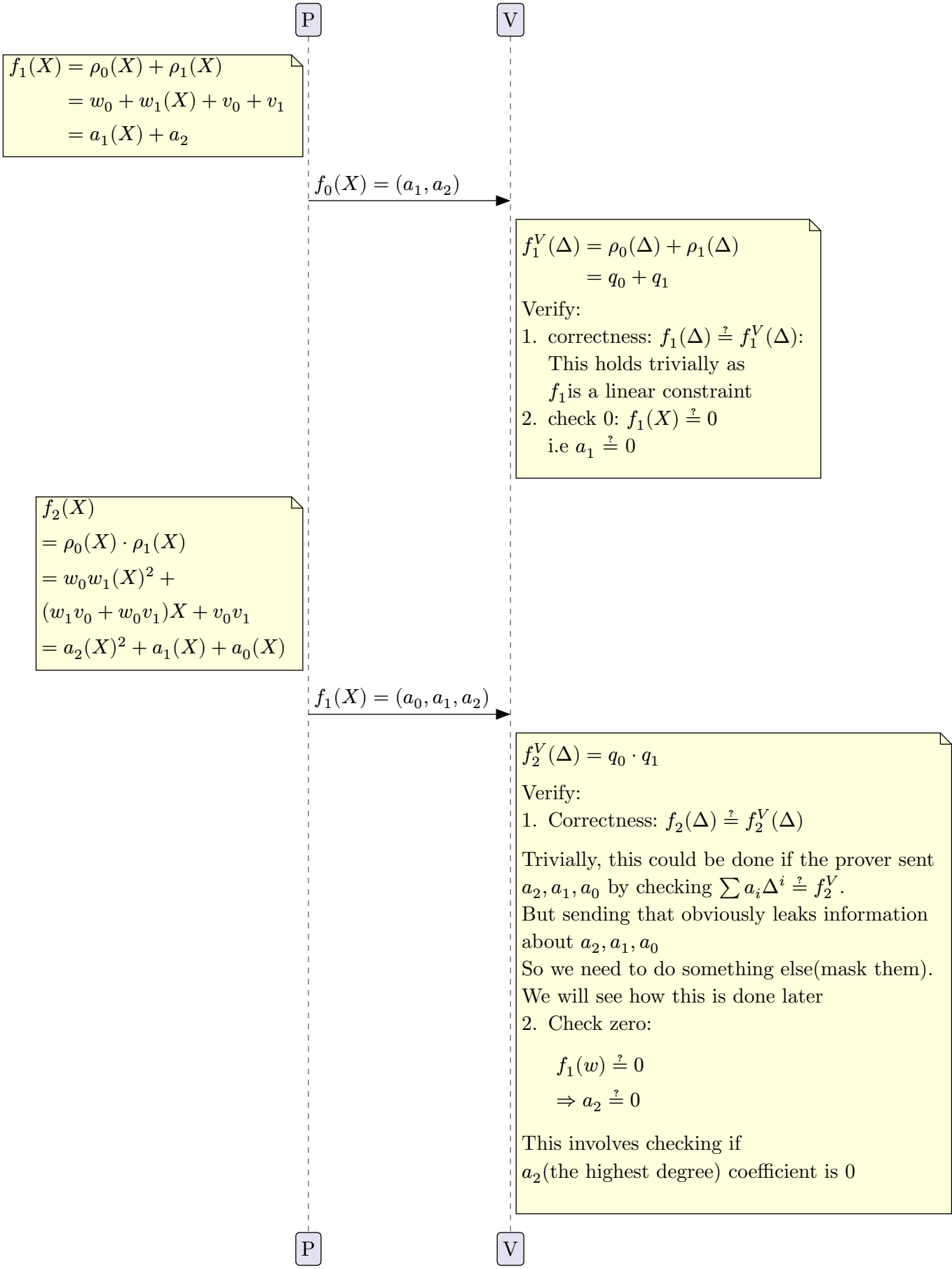
The parties now start looking at VOLE commitments as polynomial VOLE commitments:



Evaluation: The parties now evaluate the polynomial constraints: The prover evaluates on the VOLE correlation polynomial, and the verifier evaluates the constraints on the evaluations of them.

Then the prover and verifier communicate so the Prover can prove that the evaluation is correct and the constraint is satisfied.

Let's see how it looks:


P
V
$f_1(X) = \rho_0(X) + \rho_1(X)$
$= w_0 + w_1(X) + v_0 + v_1$
$= a_1(X) + a_2$
$f_0(X) = (a_1, a_2)$
$f_1^V(\Delta) = \rho_0(\Delta) + \rho_1(\Delta)$
$= q_0 + q_1$
Verify:
1. correctness: $f_1(\Delta) \stackrel{?}{=} f_1^V(\Delta)$:
This holds trivially as
$f_1$is a linear constraint
2. check 0: $f_1(X) \stackrel{?}{=} 0$
i.e $a_1 \stackrel{?}{=} 0$
$f_2(X)$
$= \rho_0(X) \cdot \rho_1(X)$
$= w_0 w_1(X)^2 +$
$(w_1 v_0 + w_0 v_1)X + v_0 v_1$
$= a_2(X)^2 + a_1(X) + a_0(X)$
$f_1(X) = (a_0, a_1, a_2)$
$f_2^V(\Delta) = q_0 \cdot q_1$
Verify:
1. Correctness: $f_2(\Delta) \stackrel{?}{=} f_2^V(\Delta)$
Trivially, this could be done if the prover sent a_2, a_1, a_0 by checking $\sum a_i \Delta^i \stackrel{?}{=} f_2^V$.
But sending that obviously leaks information about a_2, a_1, a_0
So we need to do something else(mask them).
We will see how this is done later
2. Check zero:
$f_1(w) \stackrel{?}{=} 0$
$\Rightarrow a_2 \stackrel{?}{=} 0$
This involves checking if
a_2(the highest degree) coefficient is 0
P
V

As we can see that checking correctness of constraints requires the verifier checking evaluating the polynomial co-efficients that the Prover sends at Δ and checking if it matches, and checking if the higher order co-efficient is 0, since that is the term that encodes the witness, while lower order terms have the masked values. Doing this naively such as in the protocol shown above for every constraint has 2 problems:

1. High communication complexity: The first is that it has a high communication complexity which is d field elements per polynomial constraint to be proven. This is not a problem right now as we only have $c = 2$ constraints, but this is a problem if C is large.
2. No zero knowledge: The second is that this reveals information about the witness a_2 and prover's masks that are encoded in a_1, a_0, and hence is not zero knowledge

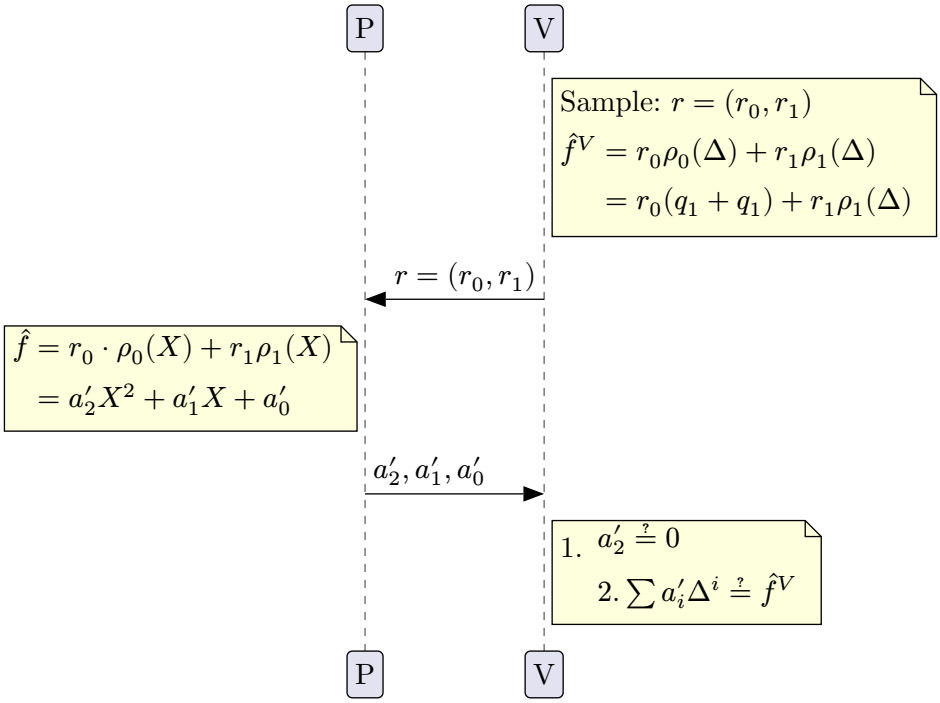
Let's solve the first problem:

1. The verifier first sends a random challenge, which can be thought of as a random vector $r \in F_{2^\lambda}^C$.
2. This is used to define a single, combined constraint which the prover calculates, and then sends he co-efficients $a_0, ..., a_d$

$$\begin{aligned}\hat{f} &= \sum r_i \rho_i(X) \\ &= a_d X^d + ...a_0\end{aligned}$$

3. The verifier evaluates checks for correctness and $0(a_d \stackrel{?}{=} 0)$ by checking if the higher order co-efficient is 0 and $\sum a_i \Delta^i \stackrel{?}{=} \sum r_i \rho_{i(\Delta)}$

For our example it looks as follows:



Soundness: Probability that P can cheat if the constraint is incorrect is $\frac{d}{2^\lambda}$ since there are atmost d roots of a d-degree $\hat{f}$, and total possible values of Δ are 2^λ.

The above communication can further be reduced by using a Linear Universal hash instead of $r \in \mathbb{F}_{2^\lambda}^C$ bringing communication complexity down further. The Verifier now sends only 1 field element instead of C And the soundness error to $2^{-\lambda} + \frac{d}{2} - \lambda$.

Zero-knowledge: P and V sample a random VOLE, and mask f with that.

4.3. VOLE-ZK sub-protocols in $\mathbb{F}_{q^r}$

In this thesis, we use a variant of the QuickSilver ZK proof system [12], which was recently optimised in [5]. This taken directly from [13]. In this protocol, the prover and verifier have established VOLE commitments using the functionality $\mathcal{F}_{\mathsf{sVOLE}}^{p,r}$ to secret inputs x_i only known to $\mathcal{P}$. Then, the VOLE-ZK protocol allows for proving degree-d constraints on the committed values without leaking anything except that the constraints are fulfilled. Here, we allow each constraint to be an arbitrary arithmetic circuit $C : \mathbb{F}_p^n \to \mathbb{F}_p$, and define the *degree* of C to be the maximal degree of the univariate polynomial $C(a_1X + b_1, ..., a_nX + b_n)$, where $a_i, b_i \in \mathbb{F}_{p^r}$ are arbitrary coefficients.

The relevant sub-protocols are described below. Note that the ("BatchVer") sub-protocol is essentially performing a batch opening of the committed z_i values and checking each $z_i = 0$, which is equivalent to checking that the underlying polynomials have degree $d_i - 1$.

VOLE-ZK sub-protocols in $\mathbb{F}_{q^r}$ from [13]

VOLE-ZK.Eval$(C, [\![x_1]\!], ..., [\![x_n]\!])$: The parties evaluate the arithmetic circuit C gate-by-gate on the commitments $[\![x_1]\!], ..., [\![x_n]\!]$, to obtain an output $[\![y]\!]_d$. Here, if the input commitments had degree-1, then d is the degree of C.

VOLE-ZK.BatchVer$\left([\![z_1]\!]_{d_1}, ..., [\![z_m]\!]_{d_m}\right)$:

1. Let $d = \max\{d_1, ..., d_m\}$.
2. Invoke the **sVOLE** instruction from $\mathcal{F}_{\mathsf{sVOLE}}^{p,r}$, by sending (svole, r), to generate $\left([\![s_{j,0}]\!]_1, ..., [\![s_{j,r-1}]\!]_1\right)$, for $j \in \{1, ..., d-1\}$. Note that $s_{j,k} \in \mathbb{F}_q$ for each $j \in \{1, ..., d-1\}$ and $k \in \{0, ..., r-1\}$.
3. For $j \in \{1, ..., d-1\}$, define $[\![s_j]\!] = \sum_{k=0}^{r-1} [\![s_{j,k}]\!] \alpha^k$ (where α is such that $(1, \alpha, ..., \alpha^{r-1})$ forms a basis of $\mathbb{F}_{q^r}$).
4. Let $\rho_{z_i}(t)$ be the degree $\leq d_i$ polynomial, with leading coefficient z_i, associated with $[\![z_i]\!]_{d_i}$. Note that if $z_i = 0$ then ρ_{z_i} has degree $d_i - 1$.
5. Similarly, let $\sigma_{j(t)}$ be the polynomial associated with $[\![s_j]\!]_1$, and $\nu_j = \sigma_j(\Delta)$ the corresponding value held by $\mathcal{V}$.
6. $\mathcal{V}$ sends a challenge $(\chi_1, ..., \chi_m) \leftarrow \mathbb{F}_{q^r}^m$.
7. $\mathcal{P}$ computes and sends to $\mathcal{V}$ the degree $d-1$ polynomial:

$$\pi(t) = \sum_{i=1}^{m} \chi_i t^{d-d_i} \rho_{z_i}(t) + \sum_{j=1}^{d-1} t^{j-1} \sigma_{s_j}(t)$$

8. $\mathcal{V}$ computes

$$q = \sum_{i=1}^{m} \chi_i \Delta^{d-d_i} \gamma_i + \sum_{j=1}^{d} \Delta^{j-1} \nu_j$$

and checks that $\pi(\Delta) = q$ and that π has degree at most $d-1$.

The VOLE-ZK.BatchVer protocol takes as input m VOLE commitments $[\![z_1]\!]_{d_1}, ..., [\![z_m]\!]_{d_m}$ over $\mathbb{F}_q$, where $d = \max\{d_1, ..., d_m\}$ is the maximum degree among all commitments, and r is the extension parameter of $\mathbb{F}_q$. The protocol consumes $(d-1) \cdot r$ correlations from $\mathcal{F}_{\text{sVOLE}}^r$, which are treated as a resource and not counted towards communication. The remaining communication costs are as follows:

Step	**Direction**	**Cost (bits)**
V sends challenges $\chi_1, ..., \chi_m \overset{R}{\leftarrow} \mathbb{F}_{q^r}^m$	$V \to P$	$m \cdot r \cdot \log q$
P sends polynomial $\pi(t)$ of degree $\leq d-1$ over $\mathbb{F}_{q^r}$	$P \to V$	$d \cdot r \cdot \log q$

The total communication is thus $(m+d) \cdot r \cdot \log q$ bits, where the $V \to P$ direction contributes $m \cdot r \cdot \log q$ bits and the $P \to V$ direction contributes $d \cdot r \cdot \log q$ bits. In settings where m is large, the verifier-to-prover communication dominates. This cost can be reduced to λ bits byt having the verifier send a PRG seed. Alternatively, in the random oracle model, we can completely eliminate this communication using the Fiat-Shamir heuristic and additionally get security against a malicious V for free. While computing costs we will use the latter.

Chapter 5.

Range Proof

This section introduces the notion and motivation behind zero knowledge range proofs, and some some efficient constructions using VOLE based commitment schemes.

Zero-knowledge range proofs (ZKRPs) allow a prover to convince a verifier that a secret value lies in a given interval.

Some applications of a zero knowledge range proof can be:

1. Suppose that Alice is a blockchain user and she wants to participate in a sealed bid auction, where the amount she bids is kept private. In order to make sure that Alice has enough money, she's required to prove that her bid does not exceed her account balance. How can she do this while keeping the amount that she bids private?
2. Imagine you're at a bar, and the bouncer asks for proof that you're 18 or older. Instead of showing your ID and revealing your exact age, what if you could prove you're old enough without disclosing any other personal details?

More concretely, for a commitment $[\![x]\!]$ where $x \in \mathbb{F}_p$ a zero knowledge range proof is an algorithm that allows a prover to prove that $x \in [0, 2^m)$ where $2^m - 1 < (p-1)/2$

5.1. Naive range proof

A naive way to design such an algorithm would be to have $\mathcal{P}$ commit to x as $[\![x]\!] \leftarrow \text{Commit}(x)$

Defining a circuit:

$$f(X) = \prod_{i=0}^{2^{m-1}} (X - i)$$

Proving inside VOLE-ZK:

$$f(X) = 0$$

The problem with this approach when using VOLE based commitments is that the degree of the circuit is exponential in bit length, $\deg(f) = 2^m$ For example, if $m = 32$, degree is $2^{32} \approx 4$ billion. If $m = 64$. degree is astronomically large. Quicksilver style ZK proofs have soundness error roughly:

$$\text{error} \approx \frac{\deg}{|F|} = \frac{2^m}{p}$$

This makes a circuit of degree 2^m impractical when m is large.

5.2. Range Proof using n-ary bit decomposition:

A classic way to get around this would be to break the witness up using bit decomposition and proving lower degree constraints on that. The prover commits to m field elements $x_0, ..., x_{m-1} \in \mathbb{F}_p$ where each x_i is the the bit decomposition of x. Now the prover proves in zero knowledge that:

1. $x_i \in \{0, 1\}$ i.e each chunk is legal
2. $x = \sum_{i=0}^{m-1} 2^i x_i$ i.e the bit decomposition is correct

This can be generalized by using base-N decomposition. The prover divides the element x into $\frac{m}{B}$ chunks, each of B-bits. The prover then commits to these $\frac{m}{B}$ bits (instead of m). The parties then check $x = \sum_{i=0}^{\frac{m}{B}} 2^{i \cdot B} \overline{x}_i$. The protocol for this is given below from [13]

Subprotocol: Pow2Range$\Big([\![x_1]\!]_{d_1}, \cdots, [\![x_n]\!]_{d_n}, m \Big)$

Public parameters: $B \mid m$ and $\ell = \frac{m}{B}$. C_{Π} is a degree-2^B circuit which maps $x \mapsto \prod_{i=0}^{2^B-1}(x - i)$. C_{lsb} is a degree-1 circuit which maps $(x, x_1, ..., x_{\ell-1}) \mapsto x - \sum_{i=1}^{\ell-1} 2^{i \cdot B} x_i$.

---

1. For all $i \in [n]$:
   (a) $\mathcal{P}$ computes the unique values $\overline{x}_{i,0}, ..., \overline{x}_{i,\ell-1} \in [0..2^B - 1]$ such that
   $$x_i = \sum_{j=0}^{\ell-1} \overline{x}_{i,j} 2^{B \cdot j}.$$
   (b) For all $j \in [0, \ell - 1]$, $\mathcal{P}$ sets $[\![\overline{x}_{i,j}]\!] \leftarrow \text{Commit}(\overline{x}_{i,j})$.
   (c) The parties let
   $$[\![z_i]\!]_{d_i} := \text{VOLE-ZK.Eval}\Big(C_{\text{lsb}}, [\![x_i]\!]_{d_i}, [\![\overline{x}_{i,1}]\!], ..., [\![\overline{x}_{i,\ell-1}]\!]\Big).$$
   (d) For all $j \in [0, \ell - 1]$, let
   $$[\![y_{i,j}]\!]_{2^B} := \text{VOLE-ZK.Eval}(C_{\Pi}, [\![\overline{x}_{i,j}]\!]).$$
2. $\mathcal{P}$ and $\mathcal{V}$ together run
   $$\text{VOLE-ZK.BatchVer}\left(\left\{\left\{[\![y_{i,j}]\!]_{2^B}\right\}_{j \in [0,\ell-1]}, [\![z_i]\!]_{d_i}\right\}_{i \in [n]}\right).$$
   $\mathcal{V}$ accepts iff the check in BatchVer holds.

5.3. Interval Proof for proving that x is of short norm

In our case, we wish to prove that the $s_i \in \mathbb{F}$ and $s_i \in \mathbb{F}$ have coefficient $x \in \{-2, -1, 0, 1, 2\}$ if $\eta = 2$ or $x \in \{-3, -2, -1, 0, 1, 2, 3\}$ if $\eta = 3$ The naive range proof suffices since the degree is very small. We define the circuit as $C_1(x) = (X + 2)(X + 1)X(X - 1)(X - 2)$ or $C_2(x) = (X + 3)(X + 2)(X + 1)X(X - 1)(X - 2)(X - 3)$ The parties commit to x, call VOLE-ZK.Eval$(C_i, [\![x]\!])$, the result would be a degree-d VOLE commitment. The parties then call VOLE-ZK.BatchVer on the outputs.

Subprotocol: `BatchedRangeProof` Range proof for $S = \{-2, -1, 0, 1, 2\}$

Public parameters:
Set $S = \{-2, -1, 0, 1, 2\}$ and vanishing polynomial

$$f(X) = (X + 2)(X + 1)X(X - 1)(X - 2),$$

satisfying $f(x) = 0 \leftrightarrow x \in S$.

Input: Commitments $[\![x_1]\!], ..., [\![x_n]\!]$ held jointly by $\mathcal{P}$ and $\mathcal{V}$, with $\mathcal{P}$ holding the private values $x_1, ..., x_n$.

Step 1 Evaluate. For each $i \in [n]$, the parties run VOLE-ZK.Eval$(f, [\![x_i]\!])$ to obtain

$$[\![y_i]\!] := f([\![x_i]\!]),$$

Step 2 Batch verify. The parties invoke VOLE-ZK.BatchVer$([\![y_1]\!], ..., [\![y_n]\!])$.

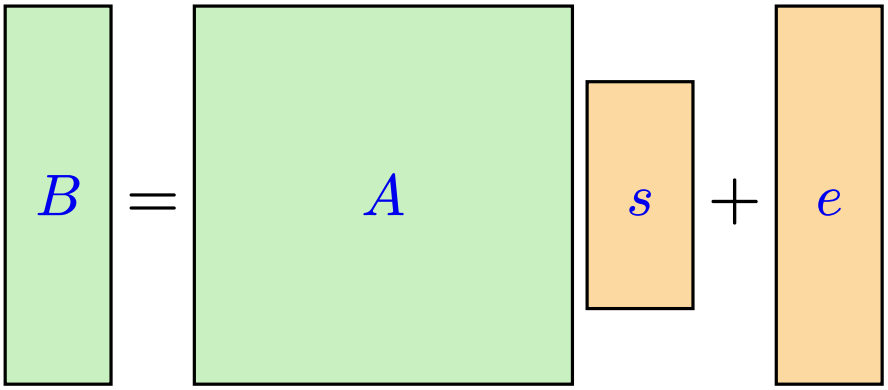
Communication cost for 1 range proof:
1. Committing to x uses one call to $\mathcal{F}^r_{\text{sVOLE}}$ and exchanging one correction value
2. ZK.BatchVer: The parties make $(r \cdot d)$ calls to $\mathcal{F}^r_{\text{sVOLE}}$ to receive voles for masking. $\mathcal{P}$ batches and sends to $\mathcal{V}$ a polynomial which has d(coefficients) in $\mathbb{F}_{p^r}$

Chapter 6.

ZKP for Module LWE

With the background from the previous chapters, we now come to the main goal of this thesis - which is to build a zero knowledge proof for the M-LWE problem using VOLE commitments. For the LWE problem, we choose the MLWE variant from the Kyber-PKE. Kyber is a quantum-safe Key Encapsulation Mechanism (KEM). It is standardized by NIST in DIPS 203, where it is called ML-KEM (Module-Lattice-based KEM). Kyber-KEM was designed by applying the Fujisaki-Okamoto transform to a public-key encryption scheme (Kyber-PKE). Details of the MLWE problem are given:

- **Parameters:**
  - prime q,
  - integers n, k, ℓ with $k \geq \ell$, and
  - integers $\eta_1, \eta_2 \ll \frac{q}{2}$.
- **Module Learning With Errors (MLWE) instance:**
  - $A \underset{R}{\in} R_q^{k \times \ell}$ (where $R_q = \mathbb{Z}_q[x]/(x^n + 1)$).
  - $B = As + e$, where $s \underset{R}{\in} S_{\eta_1}^{\ell}$ and $e \underset{R}{\in} S_{\eta_2}^{k}$ (so $t \in R_q^k$).
- **Find:** s.



6.1. The Protocol Sketch

We now describe the high-level protocol for proving the MLWE relation. We assume access to a perfectly hiding and statistically binding commitment scheme $[\![\cdot]\!]$ for elements in $\mathcal{R}_q$. With the commit-and -prove style approach this gives a zero knowledge proof with a negligible soundness error of $1/|\mathbb{F}|$ for an appropriate field $\mathbb{F}$. The concrete instantiation of this commitment scheme is the subject of section and will be given later.

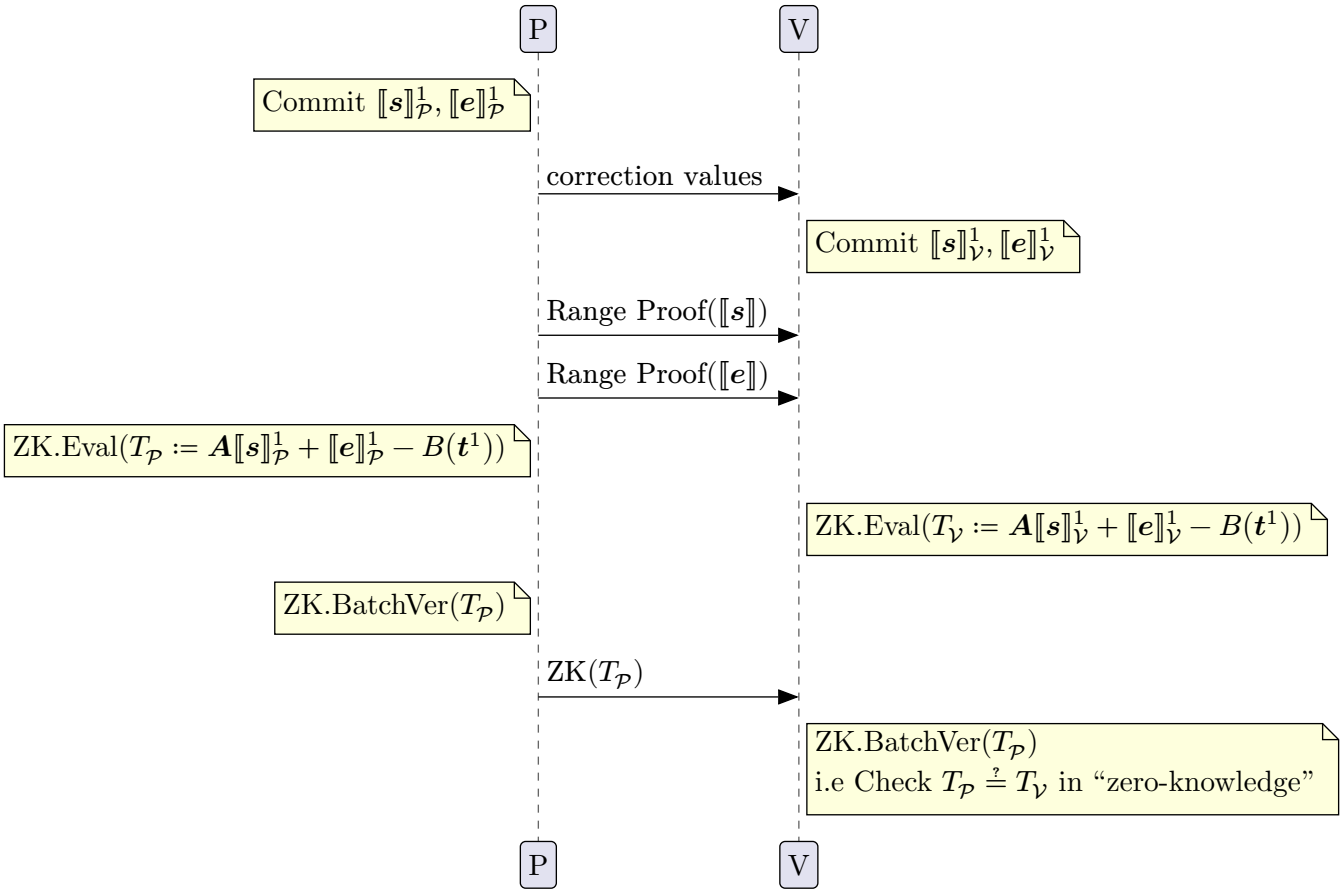
Goal. Given public $(\boldsymbol{A}, \boldsymbol{B})$ and private $(\boldsymbol{s}, \boldsymbol{e})$, the prover $\mathcal{P}$ wishes to convince the verifier $\mathcal{V}$ that:

1. $\boldsymbol{B} = \boldsymbol{A}\boldsymbol{s} + \boldsymbol{e}$, and
2. $\boldsymbol{s}$ and $\boldsymbol{e}$ have a small infinity norm, i.e. $\boldsymbol{s} \in S_{\eta_1}^k$ and $\boldsymbol{e} \in S_{\eta_1}^k$.

The two conditions are proved separately: condition 2 via a range proof on the commitments (see Chapter 5), and condition 1 via a linear consistency check using the homomorphic property of the commitment scheme

1. The parties get random degree-d VOLE commitments and then derandomise them to get $[\![s]\!]^d$, $[\![e]\!]^d$
2. $\mathcal{P}$ does the range proof for $[\![s]\!]^d$,$[\![e]\!]^d$ to prove they are small
3. The parties evaluate the circuit with VOLE-ZK.Eval$(C, [\![s_0]\!], ...[\![s_i]\!], [\![e_i]\!], ..., [\![e_j]\!]$. $\mathcal{P}$ proves to $\mathcal{V}$ that she computes the circuit C correctly, but in zero knowledge with VOLE-ZK.BatchVer$\left([\![z_1]\!]_1, ..., [\![z_m]\!]_1\right)$ with soundness error of $1/\ |\mathbb{F}|$

This protocol is informally presented via the following diagram:



The commitment problem. The soundness argument for any such ZKP protocol requires that for 2 distinct commitments $x, x' \in \mathbb{F}$, $(x' - x)^{-1}$ exists and is unique. This holds when commitment scheme is over a field $\mathbb{F}$, since every nonzero element is invertible. However, $\boldsymbol{s}$ and $\boldsymbol{e}$ are elements of $\mathcal{R}_q$, which is a *ring* , and in a ring, $(x' - x)$ may be a zero divisor with no inverse, breaking the soundness bound. This motivates the commitment scheme we construct in the next section. So out goal becomes:

Can we construct a commitment scheme for $\mathcal{R}_q$ where $\mathcal{P}^*$ can cheat at most with a probability of $1/|\mathbb{F}|$ for some appropriate large field $\mathbb{F}$ which is negligible?

6.2. Concrete Commitments Schemes and ZKP

In this subsection we will make a concrete commitment scheme and a ZKP fulfilling the above hiding and binding properties. We assume access to an ideal VOLE functionality such as $\mathcal{F}^r_{\text{sVOLE}}$

The key observation is that even though the prover's witness s_i and e_i belong to R_q, their coefficients belong to $\mathbb{F}_q$ i.e. for $s_i(X) = \sum_{j=0}^{n-1} s_{i,j}X^j \in \mathcal{R}_q$, $s_{i,j} \in \mathbb{F}_p$. We can instantiate a commitment scheme by having the parties generate de-randomized VOLES for the coefficients $s_{i,j}$ and later arranging them back in the ring R_q

Even though $s_i, e_i \in R_q$
their coefficients belong to $\mathbb{F}_q$ i.e.
for $s_i(X) = \sum_{j=0}^{n-1} s_{i,j}X^j \in \mathcal{R}_q$,
$(i,j) \in \mathbb{F}_q$.

Protocol : VOLE.CommitRing — sVOLE-based commitment to a ring element

Parameters: Prime q, degree n, extension degree r. Base ring $\mathcal{R}_q = \mathbb{F}_q[X]/(X^n + 1)$. Functionality $\mathcal{F}^r_{\text{sVOLE}}$ with global key $\Delta \in \mathbb{F}_{q^r}$ held by $\mathcal{V}$

$\mathcal{P}$'s input: A ring element $S(X) = \sum_{j=0}^{n-1} s_j X^j \in \mathcal{R}_q$.

$\mathcal{V}$'s input: Global key $\Delta \in \mathbb{F}_{q^r}$, held privately.

Output: A commitment $[\![S]\!] = ([\![S]\!]_{\mathcal{P}}, [\![S]\!]_{\mathcal{V}})$ held jointly, satisfying $[\![S]\!]_{\mathcal{V}} = [\![S]\!]_{\mathcal{P}}|_{T=\Delta}$.

Step 1 Commit to each coefficient.
For each $j \in \{0, ..., n-1\}$, $\mathcal{P}$ and $\mathcal{V}$ call `commit` on the coefficient $[\![s_j]\!] \leftarrow$ `commit` $\left(s_j \in \mathbb{F}_q\right)$. The functionality returns:

- $\mathcal{P}$ receives $s_j \cdot T + v_j$, where $v_j \xleftarrow{R} \mathbb{F}_{q^r} \forall j \in [0, n]$;
- $\mathcal{V}$ receives $s_j \cdot \Delta + v_j \in \mathbb{F}_{q^r} \forall j \in [0, n]$.

Step 2 Assemble.
Both parties arrange the n scalar outputs as coefficients of X^j in $\mathcal{R}_q$:

$$[\![S]\!]_{\mathcal{P}} := \sum_{j=0}^{n-1} \left(s_j \cdot T + v_j\right) X^j,$$

$$[\![S]\!]_{\mathcal{V}} := \sum_{j=0}^{n-1} \left(s_j \cdot \Delta + v_j\right) X^j.$$

Output.
The commitment is $[\![S]\!] = ([\![S]\!]_{\mathcal{P}}, [\![S]\!]_{\mathcal{V}})$.

Protocol : VOLE.ZKPRing — Zero-knowledge proof for MLWE over a ring R_q

Parameters and building block: Ring parameters for R_q Field parameters for $\mathbb{F}_{q^r}$, public matrix $\boldsymbol{A} \in \mathcal{R}_q^{k \times k}$, public vector $\boldsymbol{t} \in \mathcal{R}_q^k$. Access to a $\mathcal{F}_{\text{sVOLE}}^r$. Parties initiate $\mathcal{F}_{\text{sVOLE}}^r$ and $\mathcal{V}$ has Δ

$\mathcal{P}$'s input: Private vectors $\boldsymbol{s}, \boldsymbol{e} \overset{R}{\in} S_{\eta_1}^k$ such that $\boldsymbol{t} = \boldsymbol{A}\boldsymbol{s} + \boldsymbol{e}$

$\mathcal{V}$'s input: Public $\left(\boldsymbol{A} \in \mathcal{R}_q^{k \times k}, \boldsymbol{t} \in \mathcal{R}_q^k\right)$ and global key $\Delta \in \mathbb{F}_{q^r}$.

Output: $\mathcal{V}$ accepts or rejects.

Step 1 Commit. For all $s_i \in S_q^k$ The parties run VOLE.FieldCommit on each element of $\boldsymbol{s}$ and $\boldsymbol{e}$, obtaining commitments $\left(\llbracket \boldsymbol{s_i} \rrbracket_{\mathcal{P}}^1, \llbracket \boldsymbol{e_i} \rrbracket_{\mathcal{P}}^1\right)$ and $\left(\llbracket \boldsymbol{s_i} \rrbracket_{\mathcal{V}}^1, \llbracket \boldsymbol{e_i} \rrbracket_{\mathcal{V}}^1\right)$.

Step 2 Range proof. $\mathcal{P}$ executes range proofs RangeProof($\llbracket s_i \rrbracket), \llbracket e_i \rrbracket$) demonstrate that all committed values lie within the required norm bound (see Chapter 5).

Step 3 ZK.Eval The parties evaluate the circuit with VOLE-ZK.Eval($C, \llbracket s_0 \rrbracket, ... \llbracket s_{k-1} \rrbracket, \llbracket e_0 \rrbracket, ..., \llbracket e_{k-1} \rrbracket$ where $C := (A\llbracket \boldsymbol{s} \rrbracket + \llbracket \boldsymbol{e} \rrbracket) - B\left(T^d\right)$ i.e both parties non interactively compute the circuit on their shares of the VOLE co-relation. We denote $\mathcal{P}$'s computation circuit by $T_{\mathcal{P}} := \boldsymbol{A}\llbracket \boldsymbol{s} \rrbracket_{\mathcal{P}}^d + \llbracket \boldsymbol{e} \rrbracket_{\mathcal{P}}^d - B\left(\boldsymbol{t}^d\right)$. Similarly, we denote $\mathcal{V}$'s computation of the circuit by $T_{\mathcal{V}} := \boldsymbol{A}\llbracket \boldsymbol{s} \rrbracket_{\mathcal{V}}^d + \llbracket \boldsymbol{e} \rrbracket_{\mathcal{V}}^d - B\left(\boldsymbol{t}^d\right)$ where $d = 1$.

Step 4 ZK.BatchVer.
$\mathcal{P}$ proves to $\mathcal{V}$ that she computes the circuit T_p correctly i.e $T_p = T_V$, but in zero knowledge with VOLE-ZK.BatchVer$\left(\llbracket z_1 \rrbracket_1, ..., \llbracket z_m \rrbracket_1\right)$

6.2.1. Pict oral view

In this subsection we will show the circuit and the commitments in a more verbose form, this is to build intuition ands see how VOLE-Z works. For concreteness, we'll use ML-KEM-768 domain parameters: $q = 3329, n = 256, k = 3, \eta_1 = 2, \eta_2 = 2$.

We define the public matrix $A^{3\times 3} \in \left(\mathcal{R}_q = \mathbb{F}_q[X]/(X^{256}+1)\right)$ and secret vectors $S, E \in \mathcal{S}_2^3$:

$$A = \begin{pmatrix} A_{00} & A_{01} & A_{02} \\ A_{10} & A_{11} & A_{12} \\ A_{20} & A_{21} & A_{22} \end{pmatrix}$$

$$S = \begin{pmatrix} S_0 = s_{0,0} + s_{0,1}X + \cdots + s_{0,255}X^{255} \\ S_1 = s_{1,0} + s_{1,1}X + \cdots + s_{1,255}X^{255} \\ S_2 = s_{2,0} + s_{2,1}X + \cdots + s_{2,255}X^{255} \end{pmatrix}$$

$$E = \begin{pmatrix} E_0 = e_{0,0} + e_{0,1}X + \cdots + e_{0,255}X^{255} \\ E_1 = e_{1,0} + e_{1,1}X + \cdots + e_{1,255}X^{255} \\ E_2 = e_{2,0} + e_{2,1}X + \cdots + e_{2,255}X^{255} \end{pmatrix}$$

The product $B = AS + E \in \mathcal{R}_q^3$ is then:

$$B = \begin{pmatrix} A_{00} & A_{01} & A_{02} \\ A_{10} & A_{11} & A_{12} \\ A_{20} & A_{21} & A_{22} \end{pmatrix} \begin{pmatrix} S_0 \\ S_1 \\ S_2 \end{pmatrix} + \begin{pmatrix} E_0 \\ E_1 \\ E_2 \end{pmatrix} = \begin{pmatrix} A_{00}S_0 + A_{01}S_1 + A_{02}S_2 + E_0 \\ A_{10}S_0 + A_{11}S_1 + A_{12}S_2 + E_1 \\ A_{20}S_0 + A_{21}S_1 + A_{22}S_2 + E_2 \end{pmatrix}$$

How does a commitment to $S_0(X) \in \mathbb{F}_{p^r}$ look?

$$\begin{aligned} [\![S_0]\!]_p &= \rho_{S_0}(T) \\ &= s_{0,0} \cdot T + v_{s_{0,1}} + s_{0,1}(X) \cdot T + v_{S_{0,1}} + ... \\ &= \big(s_{0,0} + s_{0,1}X + \cdots + s_{0,255}X^{255}\big)T + \big(v_{0,0} + v_{0,1}X + \cdots + v_{0,255}X^{255}\big) \end{aligned}$$

where each $s_{0,j}(X) \in \mathbb{F}_q$ and $c_{S_{0,j}} \in \mathbb{F}_{q^r}$ is the VOLE mask.

$$\begin{aligned} [\![S_0]\!]_{\mathcal{V}} &= \gamma_{S_0}(\Delta) \\ &= S_0(\Delta) + V_{S_0} \end{aligned}$$

Similarly we define $[\![S_1]\!], [\![S_2]\!]$ and $[\![E_0]\!], [\![E_1]\!], [\![E_2]\!]$.

Circuit evaluation. We can now construct a circuit $C = A[\![S]\!] + [\![E]\!] - B(T)$ where T is the indeterminate. If $\mathcal{P}$ was honest computing B then the resulting commitments of this circuits would be commitments to 0 (i.e highest degree term is 0) and lower degree terms will be the some combination of VOLE masks(v)'s

$$\begin{aligned} & A \cdot [\![S]\!] + [\![E]\!] - B \cdot T \\ = & \begin{pmatrix} A_{00}[\![S_0]\!] + A_{01}[\![S_1]\!] + A_{02}[\![S_2]\!] \\ A_{10}[\![S_0]\!] + A_{11}[\![S_1]\!] + A_{12}[\![S_2]\!] \\ A_{20}[\![S_0]\!] + A_{21}[\![S_1]\!] + A_{22}[\![S_2]\!] \end{pmatrix} + \begin{pmatrix} [\![E_0]\!] \\ [\![E_1]\!] \\ [\![E_2]\!] \end{pmatrix} - \begin{pmatrix} B_0 \cdot T \\ B_1 \cdot T \\ B_2 \cdot T \end{pmatrix} \\ = & \begin{pmatrix} (A_{00}S_0 + A_{01}S_1 + A_{02}S_2 + E_0)T + V_0' \\ (A_{10}S_0 + A_{11}S_1 + A_{12}S_2 + E_1)T + V_1' \\ (A_{20}S_0 + A_{21}S_1 + A_{22}S_2 + E_2)T + V_2' \end{pmatrix} - \begin{pmatrix} B_0 \cdot T \\ B_1 \cdot T \\ B_2 \cdot T \end{pmatrix} \\ = & \begin{pmatrix} (A_{00}S_0 + A_{01}S_1 + A_{02}S_2 + E_0 - B_0)T + V_0' \\ (A_{10}S_0 + A_{11}S_1 + A_{12}S_2 + E_1 - B_1)T + V_1' \\ (A_{20}S_0 + A_{21}S_1 + A_{22}S_2 + E_2 - B_2)T + V_2' \end{pmatrix} \\ = & \begin{pmatrix} 0 \cdot T + V_0' \\ 0 \cdot T + V_1' \\ 0 \cdot T + V_2' \end{pmatrix} \end{aligned}$$

where $V_i' = V_{S_0} + V_{S_1} + V_{S_2} + V_{E_i}$ collects the VOLE masks for row i. The degree-1 terms vanish exactly because $B = AS + E$.

6.3. Other ZK proofs and statements

We give other related statements and proof circuits which can use the techniques from above, and can also be optimized using the techniques that will be showed in the following chapters. However while measuring concrete costs, we use the proof for the MLWE relation, but they can be easily extrapolated for related statements.

6.3.1. Simplified Kyber Key Gen

We now give the algorithm for the Simplified ML-KEM key generation

Kyber.Simplified.KeyGen():

1 Sample $A \xleftarrow{R} \mathcal{R}_q^{k \times k}$, $s \xleftarrow{R} \mathcal{S}_{\eta_1}^k$, **and** $e \xleftarrow{R} \mathcal{S}_{\eta_1}^k$.

2 $t = As + e$.

3 **return** $pk = (A, t), sk = s$

Proof for CPA Kyber Key Gen: $\mathcal{P}$ wants to prove in zero knowledge that his $pk = (A, t)$ is valid and well formed using his witness which is $sk = s$ and the error e

This is equivalent to proving the MLWE relation

6.3.2. CPA Kyber Encryption Scheme(unoptimised)

We now give the CPA-Kyber unoptimised encryption scheme:

Kyber.Simplified.Enc($(pk = (A, t), m \in \{0, 1\}$():

1 Select $r \xleftarrow{R} \mathcal{S}_{\eta_1}^k$, $e_1 \xleftarrow{R} \mathcal{S}_{\eta_2}^k$, **and** $e_2 \xleftarrow{R} \mathcal{S}_{\eta_2}$.

2 $u = A^T r + e_1$, $v = t^T r + e_2 + \lceil \frac{q}{2} \rceil m$.

3 **return** $c = (u, v) \in \mathcal{R}_q^k \times \mathcal{R}_q$.

Kybe.Simplified.Dec($sk = s, c = (u, v)$)():

1 $m = \text{Round}_q(v - s^T u)$.

2 **return** m

Proof of CPA Kyber unoptimised ciphertext:

$\mathcal{P}$ wants to prove that the ciphertext $c = (u, v)$ is a valid ciphertext under the $pk = (A, t)$ for the witness $= (m, r, e_1, e_2)$

1. c is well formed i.e $u = A^T[\![r]\!] + [\![e_1]\!]$ and $v = t[\![r]\!] + [\![e_2]\!] + \lceil \frac{q}{2} \rceil [\![m]\!]$
2. $[\![r]\!], [\![e]\!], [\![e_1]\!], [\![e_2]\!]$ is of short infinity norm in their respective ranges and $[\![m]\!] \in \{0, 1\}$

Procedure:

1. The parties VOLE-commit to $[\![r]\!], [\![e]\!], [\![e_1]\!], [\![e_2]\!]$
2. The parties execute range proofs
3. The parties evaluate in VOLE-Z : ZK.Eval(C_i, w) where $C_A = u - A^T[\![r]\!] + [\![e_1]\!]$ and $C_b = v - t[\![r]\!] + [\![e_2]\!] + \lceil \frac{q}{2} \rceil [\![m]\!]$
4. $\mathcal{P}$ proves to $\mathcal{V}$ that he computed the circuit correctly and the initial commits match up, but in zero knowledge. The parties run VOLE.ZK-BatchVer on the circuit outputs

6.3.3. CPA Kyber Encryption Scheme (with compression)

We now give the CPA-Kyber encryption scheme with ciphertext compression, where d_u, d_v are the compression parameters and $\text{Compress}_q(x, d) = \left\lceil \left(\frac{2^d}{q}\right) x \right\rfloor \bmod 2^d$, $\text{Decompress}_q(y, d) = \left\lceil \left(\frac{q}{2^d}\right) y \right\rfloor$.

Kyber.Simplified.Enc$(pk = (A, t), m \in \{0,1\})()$:

1 Select $r \underset{R}{\in} \mathcal{S}_{\eta_1}^k$, $e_1 \underset{R}{\in} \mathcal{S}_{\eta_2}^k$, **and** $e_2 \underset{R}{\in} \mathcal{S}_{\eta_2}$.
2 $u = \text{Compress}_q(A^T r + e_1, d_u)$,
3 $v = \text{Compress}_q(t^T r + e_2 + \lceil \frac{q}{2} \rceil m, d_v)$.
4 **return** $c = (u, v) \in \mathcal{R}_{2^{d_u}}^k \times \mathcal{R}_{2^{d_v}}$.

Kyber.Simplified.Dec $(sk = s, c = (u, v))()$:

1 $u' = \text{Decompress}_q(u, d_u)$, $v' = \text{Decompress}_q(v, d_v)$.
2 $m = \text{Round}_q(v' - s^T u')$.
3 **return** m

Proof of compressed CPA Kyber ciphertext:

$\mathcal{P}$ wants to prove the ciphertext $c = (u, v)$ is valid under $pk = (A, t)$ for the witness (m, r, e_1, e_2). Since u, v are public, the parties first compute the public decompression targets

$$\hat{u} = \text{Decompress}_q(u, d_u) \in \mathcal{R}_q^k, \quad \hat{v} = \text{Decompress}_q(v, d_v) \in \mathcal{R}_q.$$

Compression is then proven backward:

1. c is well formed inside the rounding windows, i.e. do range proofs for the following:

$$|A^T [\![r]\!] + [\![e_1]\!] - \hat{u}| \leq \delta_u, \quad |t^T [\![r]\!] + [\![e_2]\!] + \left\lceil \frac{q}{2} \right\rceil [\![m]\!] - \hat{v}| =\leq \delta_v,$$

   where δ_u and δ_v are the known rounding errors. This proves the linear relation along with the compression

2. $[\![r]\!], [\![e_1]\!], [\![e_2]\!]$ are of short infinity norm in their respective ranges, $[\![m]\!] \in \{0,1\}$

Procedure:

1. The parties VOLE-commit to $[\![r]\!], [\![e_1]\!], [\![e_2]\!], [\![m]\!]$.
2. The parties compute the public targets $\hat{u}, \hat{v}$ from the ciphertext.
3. The parties execute range proofs:
   - secret-range proofs on $[\![r]\!], [\![e_1]\!], [\![e_2]\!]$,
   - the binary check $[\![m]\!]([\![m]\!] - 1) = 0$,
   - window range proofs directly on the linear combinations $A^T [\![r]\!] + [\![e_1]\!] - \hat{u}$ and $t^T [\![r]\!] + [\![e_2]\!] + \lceil \frac{q}{2} \rceil [\![m]\!] - \hat{v}$, bounding each by B_{d_u} resp. B_{d_v}.
4. The parties run VOLE.ZK-BatchVer to verify all range-proof and binary constraints in zero knowledge.

6.4. Communication costs

Communication cost for proving 1 MLWE relation depends on the total number of elements in $\mathbb{F}_{q^r}$ the parties have to commit to. They commit to all coefficients in $s \in S_{\eta_1}^k$ and $e \in S_{\eta_1}^k$. One element in s has $S_0 \in S_{\eta_1} \subset \mathbb{Z}_{3329}[X]/(x^n + 1)$ has n coefficients in $\mathbb{F}_q$. The entire S vector has k elements i.e $n \cdot k$ coefficients. For both S and E it is twice this i.e $2(n \cdot k)$

So communication cost to prove 1 MLWE relation becomes:

1. Preprocessing: $2(n \cdot k)$ calls to $\mathcal{F}_{\text{sVOLE}}^{r}$ to receive random VOLEs.
2. Derandomisation: $2(n \cdot k)$ calls to VOLE.Derand to communicate correction values i.e $2(n \cdot k) \log q$ bits.
3. Range Proof: Each range proof also has a preprocessing call to $\mathcal{F}_{\text{sVOLE}}^{r}$ and derandomization. We dont need to include costs for preprocessing and randomization as it's the same as for proving both conditions. We need to count the cost to run Zk.BatchVer for a degree-d. This can be batched with step 4.
4. ZK.BatchVer: The protocol makes $(d-1) \cdot r$ calls to $\mathcal{F}_{\text{sVOLE}}^{r}$ to receive sVOLES in for masking. $\mathcal{P}$ batches and sends to $\mathcal{V}$ a polynomial which has d(coefficients) in $\mathbb{F}_{p^r}$ $((d-1) \cdot r \cdot \log p$ bits)

Table below writes the costs for proving 1 MLWE relation (or 1 CPA ML-KEM Public Key)i, based on different security levels:

(For presentation:)
Communication Costs:
1. Preprocessing: $O(|\text{co-efficients}|)$ calls to $\mathcal{F}_{\text{sVOLE}}^{r}$ where $|\text{co-efficients}| = 2(n \cdot k)$
2. Derandomisation $\mathcal{P} \rightarrow \mathcal{V}$: $O\big(|\text{co-efficients} \in \mathbb{F}_p|\big)$
3. Range Proofs
4. VOLE-ZK.BatchVer:
   4.1 $O(d \cdot r)$ calls to $\mathcal{F}_{\text{sVOLE}}^{r}$
   4.2 $\mathcal{P} \rightarrow \mathcal{V}$: polynomial with d co-effs in $\mathbb{F}_{p^r}$

	$\lambda = 128$ $k = 2$ $\eta_1 = 3$ $r = 11$	$\lambda = 192$ $k = 3$ $\eta_1 = 2$ $r = 17$	$\lambda = 256$ $k = 4$ $\eta_1 = 2$ $r = 22$
1. # calls to $\mathcal{F}_{\text{sVOLE}}^{r}$ (during Preprocessing+VOLE.BatchVer)	1090	1621	2158
2. # KB in VOLE.Derand	1.49	2.24	3
3. # bits in VOLE.BatchVer ($\mathcal{P} \rightarrow \mathcal{V}$)	773	995	1287
4. # KB for proof of 1 MLWE relation (excluding costs of $\mathcal{F}_{\text{sVOLE}}^{r}$)	1.59	2.37	3.15

6.5. Towards sublinear communication

1: **Realizing $\mathcal{F}_{\text{sVOLE}}^{r}$ with sublinear communication**: We ask if $\mathcal{F}_{\text{sVOLE}}^{r}$ can be realized with sublinear communication?

$\mathcal{F}_{\text{sVOLE}}^{r}$ can be realised by various protocols such as OT, MPC-in-the head etc, but it can also be realized with sublinear communication by using silent secure computation i.e a VOLE PCG. The ideas are elaborated in Chapter 7 and concrete instantiation for VOLE is given in Chapter 8

2. ***O*$(n \cdot k)$ communication per VOLE** We ask if we can achieve sublinear communication for derandomization?

We will use ideas for Chapter 7 and Chapter 8 along with the fact that the witness is sampled via central binomial distribution. Instead of doing binomial sampling and then committing, we could instead do this opposite: i.e commit to some 0s and 1s in $\mathbb{F}_p$ and then apply binomial sampling. This poses an additional challenges- We need to commit to more elements and not less. To solve this we consider committing to a *small amount* of 0′s and 1s in $\mathbb{F}_p$, expanding them to get a *large number* of low-degree commitments to 0s and 1s in $\mathbb{F}_p$ using a low degree PRG and then doing binomial sampling on the committed values. This is described in Chapter 10

Q 1: *Can we realize* $\mathcal{F}^r_{\text{sVOLE}}$ *efficiently with sublinear communication?*

Q 2. *Can we achieve sublinear communication for derandomization, which is currently* $O(|\text{co-efficients}|)$ *communication per VOLE ? where* $|\text{co-efficients}| = 2(n \cdot k)$

Chapter 7.

Secure Computation with Silent Preprocessing from PCGs

This chapter introduces the ideas behind compressing the VOLE correlation, while the actual constructions are given in the next chapter. Compressing or reducing the communication complexity of the offline phase of MPC protocols (where the parties recieve precomputed correlated randomness) comes from the field of silent secure computation.

> **Remark 7.1**
>
> In order to understand these concepts I extensively referred to Geoffroy Couteau's HDR [14] which synthetises these ideas from different papers and presents them in a very natural progression. As a result, parts of this chapter, especially the formal definitions come from the HDR. I also try to highlight what the formal definition says in more colloquial terms after each one is presented. This chapter can be skipped by someone who is already familiar with these ideas. I do not claim these ideas as mine.

7.1. Generic MPC with correlated pseudo randomness

The generic template for secure computation with preprocessing is as follows:

1. In a preprocessing step, the parties generate correlated randomness such as OT Beaver Triples, random OLEs
2. In the online phase, after the parties receive their respective inputs to the protocol such as GMW or ZK Proofs, they this correlated randomness thus making the online phase much cheaper

The communication of the preprocessing phase dominates the total communication by a large factor. In an effort to optimize this, we can try replacing the expensive preprocessing phase with novel techniques. Geoffrey et al made a key observation :

The long correlated random strings do not have to be truly random: they only have to look random i.e pseudo randomness is enough

This is analogous to the case of secure communication. In secure computation for example the one time pad, in an offline expensive preprocessing step the parties generate long randomness. In the shorter online phase, they consume it to encrypt and decrypt the message. In this if we are okay with computational security then the long expensive randomness from the offline can be replaced by a cheaper two step process:

1. Generate short shared key k
2. In an offline phase, locally expand this short key to get a longer key y using a PRG $y = G(k)$

Definition 7.1.1 (PRG)

A pseudorandom generator (PRG) is a polynomial-time algorithm $G : \{0,1\}^\lambda \to \{0,1\}^m$, with $m \gg \lambda$, such that for any polynomial-time distinguisher A, it holds that

$$|\Pr[A(y) = 0 \mid x \leftarrow \{0,1\}^\lambda, y \leftarrow G(x)] - \Pr[A(y) = 0 \mid y \leftarrow \{0,1\}m]| \leq \mu(\lambda)$$

where μ is a negligible function.

The definition means that no polynomial-time algorithm A can behave differently (output something different – i.e. notice a difference) in the two following scenarios:
1. It receives the value $y = G(x)$ on a uniformly random short input $x \leftarrow \{0,1\}^\lambda$, or
2. It receives a uniformly random long value $y \leftarrow \{0,1\}^m$.

Note that since x is λ-bit long, $y = G(x)$ cannot have more than $\lambda \ll m$ bits of entropy: thus, y in the first scenario is very far from random (it was deterministically computed from a short input), yet all polytime algorithms will still fail to distinguish them from uniformly random inputs. We have now replaced the expensive m bits of communication to just λ bits

The goal of the silent preprocessing model for secure computation is more or less the same: replace the expensive preprocessing with cheaper 2 steps: a PCG (PCG.Gen, PCG.Expand), PCG.Gen generated two keys which are given to the respective parties ($(k_1, k_2) \leftarrow$ PCG.Gen) and evaluating the keys with PCG.Expand(k_i) results in long random looking strings (R_0, R_1) but with the caveat- *they are no longer the exact same, rather mathematically related.* This is a much harder task because this requires preserving this delicate balance- achieving pseudo-randomness while preserving some mathematical relation.

7.2. MPC template with PCG

Equipped with the above ideas, Geoffreu puts forth the notion of secure computation in the silent preprocessing model:

1. In a one time short interaction phase the parties distributively evaluate `Gen` to generate short keys for their chosen corelation
2. A "silent" computation phase, where the parties go offline and locally expand their short keys into long pseudorandom strings, which are indistinguishable from long random correlated strings, using the `PCG.Expand` algorithm.
3. A "non-cryptographic" online phase, where the parties consume the corelated strings using GMW or ZKPs, etc.

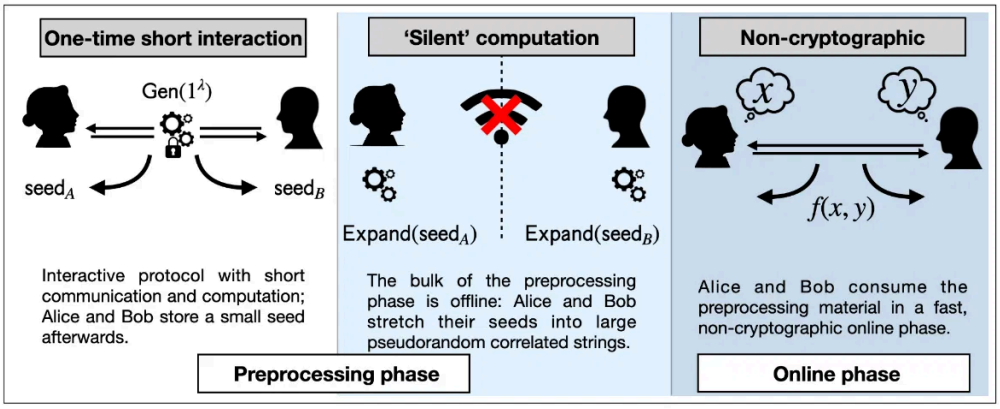


Figure 2: The three steps of an MPC protocol in the silent preprocessing model. Figure taken from Couteau [14].

7.3. Pseudoramdom Corellation Generator, definitions

Let's try to motivate the definition of a PCG. We would want to make it in relation to an "ideal" long target two-party correlation(R_0, R_1). We would like parties to generate a pair of correlates short seeds s_1 and s_2 and expand them locally, using a deterministic procedure to output (z_1, z_2). For correctness, we would want that this should be done such that the joint distribution is indistinguishable from (R_0, R_1). This also protects protects from an outside world adversary. For security, we would also want some protection from an inside adversary who knows her own seed s_b and tries to learn something about the other parties output z_{1-b}

To formalize these notions, we introduce some definitions from [14]

Definition 7.3.1 (Correlation Generator)

A PPT algorithm C is called a correlation generator, if C on input 1^n outputs a pair of strings in $\{0,1\}^n \times \{0,1\}^n$.

Since this is defined in relation to a correlated string, this definition basically says that C is a randomized algorithm that outputs two correlated strings $(R_0, R_1) \leftarrow C(1^n)$

Definition 7.3.2 (Reverse-sampleable Correlation Generator)

Let C be a correlation generator. We say C is reverse sampleable if there exists a PPT algorithm RSample such that for $\sigma \in \{0,1\}$ the following are indistinguishable:

$$\begin{gathered}(R_\sigma, R_{1-\sigma}) \mid R_{1-\sigma} \leftarrow \text{RSample}(\sigma, R_\sigma)\} \\ \overset{c}{\approx} \\ (Z_0, Z_1) \leftarrow C(1^n)\end{gathered}$$

What the above definition says is that if I fix one side R_σ, I can sample the other size $R_{1-\sigma}$ in a way that looks correct:

$$R_\sigma, R_{1-\sigma} \approx (R_0, R_1) \leftarrow C(1^n)$$

It's important because it says that given one party's output, I can fake the other party's output so that the pair still looks like it came from the real distribution. For example Random OT correlation:

The correlated outputs are:
- Party 0 holds (s_0, s_1)
- Party 1 holds (b, s_b)

Case 1: Given Party 0's view (s_0, s_1). We can simulate Party 1's view by:
- Sampling $b \leftarrow \{0,1\}$
- Outputting (b, s_b)

This perfectly matches the real distribution.

Case 2: Given Party 1's view (b, s).
We need to reconstruct (s_0, s_1):
- One of them must equal s
- The other is uniformly random

So we sample $s' \leftarrow \{0,1\}^\lambda$ and construct:

$$(s_0, s_1) = \begin{cases} (s, s') \text{ if } b = 0 \\ (s', s) \text{ if } b = 1 \end{cases}$$

Equipped with the above we can now give the definition of a PCG. Remember the main objective is that the PCG replaces "sampling full correlations" with **short seeds + local expansion**

> **Definition 7.3.3 (Pseudorandom Correlation Generator (PCG) adapted from [BCG+19b]))**
>
> A PCG for a reverse samplable correlation C is a pair of PPT algorithms $(\texttt{PCG.Gen}, \texttt{PCG.Expand})$:
> - $\texttt{PCG.Gen}(1^\lambda) \to (k_0, k_1)$;
> - $\texttt{PCG.Expand}(\sigma, k_\sigma) \to R_\sigma$ for $\sigma \in \{0,1\}$,

The generation algorithm outputs 2 short seeds, one for each party. Then each party locally expands it's short seeds to longer seeds without any communication.

> **Definition 7.3.4 (Pseudorandom Correlation Generator (PCG) Requirements adapted from [BCG+19b])**
>
> $\texttt{PCG.Gen}, \texttt{PCG.Expand}$ should satisfy the following
>
> - Pseudorandomness.
>
> $$\Big\{(R_0, R_1) \overset{C}{\approx} (Z_0, Z_1) \leftarrow C(1^\lambda) \mid$$
> $$R_\sigma \leftarrow \texttt{PCG.Expand}(\sigma, k_\sigma)_{\sigma=0,1}, (k_0, k_1) \leftarrow \texttt{PCG.Gen}(1^\lambda)\Big\}$$

- Security. For any $\sigma \in \{0,1\}$ and $(k_0, k_1) \leftarrow \texttt{PCG.Gen}(1^\lambda)$, the following two distributions are computationally indistinguishable(We write it here for $\sigma = 1$ for readability):

$$\{(k_0, R_1) \mid R_1 \leftarrow \texttt{PCG.Expand}(\sigma = 1, k_1)\}$$

$$\overset{c}{\approx}$$

$$\{(k_0, R_1) \mid R_1 \leftarrow \text{RSample}(0, R_0\},$$
$$R_0 \leftarrow \texttt{PCG.Expand}(\sigma = 0, k_0)$$

where RSample is the reverse sampling algorithm for correlation C.

The security definition highlights the security notion from an inside adversary and says that an (R_0, R_1) generated through PCG.Genb and PCG.Expand should be indistinguishable even if out inside adversary reverse sample $R_{1-\sigma}$ from her R_σ. i.e knowledge of R_σ does not help her in any way to gain information about $R_{1-\sigma}$

Chapter 8.

Compressing random-VOLE

This chapter goes over the concrete ways to realise the $\mathcal{F}^r_{\text{sVOLE}}$ functionality, with sublinear communication. We begin with the question:

Can we compress random VOLE?

Boyle et al [6] noticed that if we can settle for computational security then *pseudorandomness is enough*. So instead of using a truly random VOLE we can instead use a pseudorandom VOLE. So our question reduces to:

Can we compress pseudorandom VOLE?

We return back to the view of seeing VOLE correlation as a "shared vector-scalar product". The sender knows a random vector $u \in \mathbb{F}^n$, the receiver knows a random scalar $x \in \mathbb{F}$ and they both hold additive shares of ux.

$$C : (u, x) \to (u \cdot x)$$

Boyle et all notice that compressing an additive relation C is easy if C is of low enough degree and we have some notion of sparsity i.e a special case where u is sparse, or it has a few non zero entries and use **FSS** to secret share the product ux. However it is not enough that u is sparse, it also needs to be pseudorandom to the receiver, which is not the case for sparse-VOLE. Therefore, in an offline phase then we convert the sparse VOLE to a pseudorandom VOLE using the LPN assumption. Bole et al showed [6] that pseudorandom-VOLE implies VOLE.

So our question now reduces to:

Does sparse vole → pseudorandom VOLE? If yes, can we compress sparse VOLE

8.1. Sparse VOLE -> Pseudorandom VOLE

The learning parity with noise assumption (LPN) is one of the most fundamental assumptions of cryptography. It states that given a random secret vector s over $\mathbb{F}_2$, and given access to (an arbitrary polynomial number of) samples of the form $(a, \langle a, s\rangle + e)$, where a is a random vector and e is a random Bernoulli noise (i.e., e is 1 with some probability p, and 0 otherwise), it is infeasible to recover s. In other terms: while linear systems of equations are easy to solve (using Gaussian elimination), it becomes infeasible to solve them as soon as you add a bit of noise to the equations. To convert "sparse" to "pseudorandom" we rely on the LPN assumption. Based on different variants of the LPN assumption we use, we can achieve different stretch and security parameters. For this explanation we use the primal LPN

We now show how to get pseudorandom VOLE from sparse VOLE:

Pseudorandom VOLE using primal LPN: The idea proposed by Boyle et al [6] is to start with short vectors a, b, c such that $c = ax + b$, which the 2 parties can generate efficiently using MPC:

$$c = ax + b$$

Then they can locally expand them to pseudorandom vectors such that they hold long VOLE correlation $w = ux + v$. A first approach is that the prover and the verifier can expand their respective values a, b, c using a code generating matrix M.

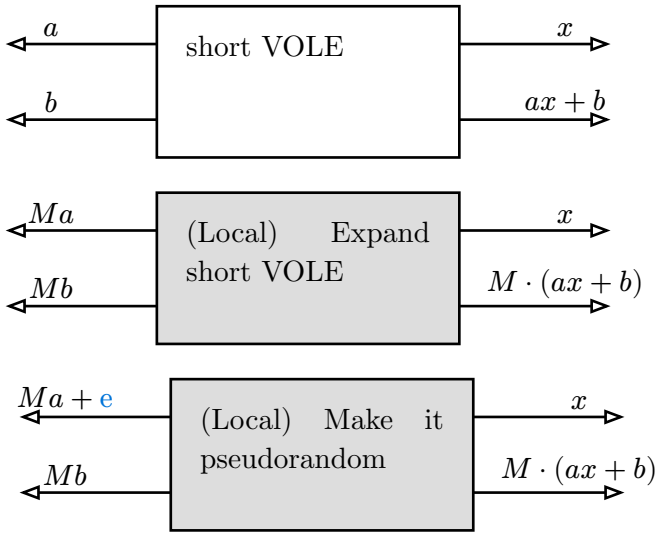
$$c = ax + b$$
$$Mc = Max + Mb$$

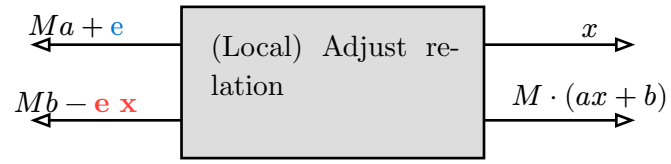
However for (Ma) to be pseudorandom, prover needs to add some error e to it otherwise the verifier can trivially guess a by gaussian elimination. So the prover has $u = aC + e$. For correctness to hold in the expanded relation the prover need to add ex or the verifier needs to subtract ex

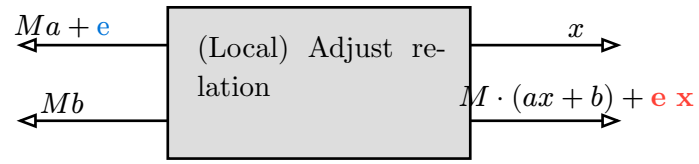
$$c = ax + b$$
$$Mc = Max + Mb$$
$$Mc = (Ma + \mathrm{e})x + Mb - \mathrm{e\ x}$$
$$Mc + \mathrm{ex} = (Ma + e)x + Mb$$

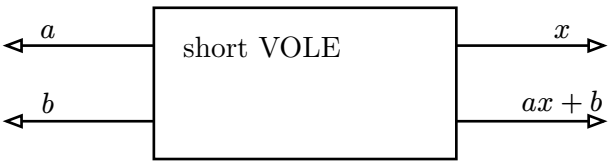
However we cannot share e in the clear with the verifier as it would be then trivial for her to guess prover's a, neither can we share x in the clear with the prover as that would violate the entire VOLE correlation. What we can instead do, is secret share the product $e \cdot x = [e \cdot x]_p + [e \cdot x]_v$ which the prover and the verifier can use to adjust the corelation.

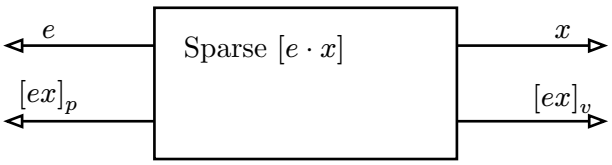
$$c = ax + b$$
$$Mc = Max + Mb$$
$$\underbrace{Mc + [ex]_v}_{w} = \underbrace{(Ma + e)}_{u}x + \underbrace{Mb - [ex]_p}_{v}$$

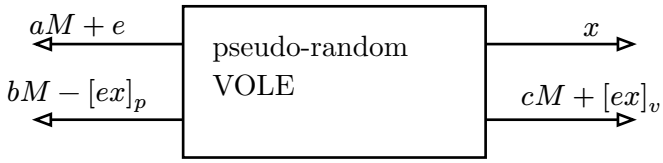


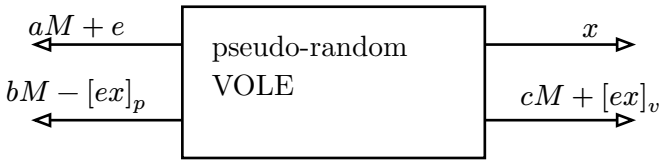












This answers the first part of our question in the positive that sparse VOLE $\rightarrow$ pseudo-random VOLE

So now, all that is left is compressing sparse VOLE which reduces to sharing a short $[\boldsymbol{e} \cdot x] \rightarrow [\boldsymbol{ex}]_{\mathcal{P}} + [\boldsymbol{ex}]_{\mathcal{V}}$, where $\boldsymbol{e}$ is some sparse error vector for the LPN assumption, and x is $\mathcal{V}$'s input to VOLE

Can we secret share ex succinctly??

8.2. Secret sharing $e \cdot x$ succinctly

All left to do now, is to make a PCG to secret share additive shares of a sparse vector e and a scalar x. We note that if we can make an FSS scheme for 1-sparse $e \cdot x$ i.e e has all zeroes except in one position, then we can use batching to make an FSS scheme for a t −sparse$e \cdot x$

So lets start by constructing an FSS for secret sharing additive shares of $[ex]$ and e is 1-sparse i.e it has only one non zero entry. This kind of a function which outputs zero at all places except one is called a point function.

Definition 8.2.1

A **point function** $f_{\alpha,\beta}$ is a function $f_{\alpha,\beta} : \{0,1\}^\ell \to G$ parameterized by a **position** $\alpha \in \{0,1\}^\ell$ and a **value** $\beta \in G$, defined as:

$$f_{\alpha,\beta}(x) = \begin{cases} \beta \text{ if } x = \alpha \\ 0 \text{ otherwise} \end{cases}$$

A function secret sharing scheme for a point function is also called a Distributed Point Function. Schoppmann et al [15] noticed that FSS is overrkill for compressing VOLE, we can infact use a relaxed variant. They observe that the non zero indices of the shared sparse noise vector needed for LPN are part of the output to one of the parties i.e one party(the party who gets u and v) knows where f is non zero in the clear. This means that a privately punctured PRF suffices instead of a full FSS scheme

We will now build a privately punctured PRF. We will now give a construction of an $n-1$ out of n random OT construction

8.2.1. $(n-1)$-OUT-OF-n RANDOM OT

The key ingredient we need is a way for P_2 to learn $n-1$ out of n pseudorandom values while P_1 holds all n, with P_1 not learning *which* value P_2 is missing. This is the $(n-1)$ -out-of-n Random OT functionality: P_1 outputs a pseudorandom vector $\boldsymbol{u} \in \mathbb{F}^n$, and P_2 outputs all entries of $\boldsymbol{u}$ except at some index i of P_2's choosing, where i remains hidden from P_1.

A naive approach would run n standard OTs, costing $O(\lambda n)$ communication. Schoppmann et al.[15] propose that based on the fact that all values are *pseudorandom*, we can do much better- $O(\lambda \log n)$ in communication by having P_1 generate all n values from a single PRG seed using a GGM tree.

The GGM tree. P_1 samples a single root seed s_0 and builds a complete k-ary tree of depth $\log_k(n)$, with n leaves. Each node's k children are derived by applying a PRG $G : \{0,1\}^\lambda \to \{0,1\}^{k\lambda}$ to the node's seed and splitting the output into k equal parts:

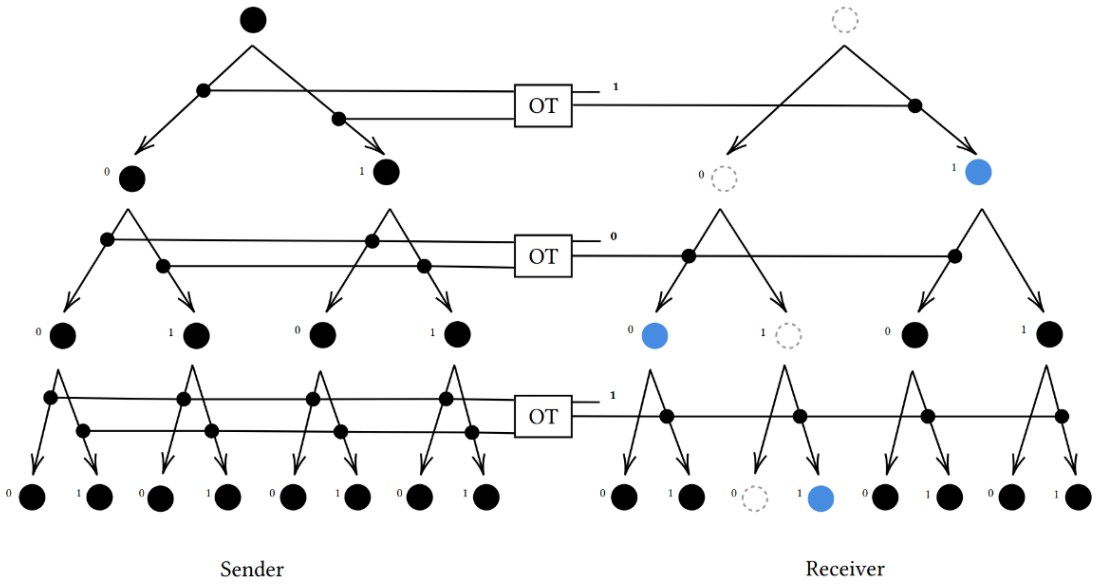
$$(s_1 \mid s_2 \mid \cdots \mid s_k) := G(s_0).$$

The n leaves are the n pseudorandom output values. P_1's key is just $K_1 = s_0$, a single seed of length λ.

Giving P_2 all leaves except the i-th. P_2 wants to reconstruct the whole tree except for the path to the i-th leaf, which has length $\log_k(n)$. At each level l, the parties run a single $(k-1)$-out-of-k OT:

- P_1 computes k messages, where the j-th message is the XOR of all seeds of the j-th children across every subtree at that level. This construction hides which subtree P_2 is missing.
- P_2, knowing the l-th digit of i's k-ary encoding, requests all messages *except* the one corresponding to that digit. Using those XOR values together with previously reconstructed subtrees, P_2 recovers all children of the unknown subtree except the one on the path to i.

After $\log_k(n)$ such OTs, which can all be run in parallel, P_2 has reconstructed all n leaves except the i-th. P_2's key K_2 consists of the $\log_k(n)$ seeds received from the OTs.



Example 8.2.2

$k = 2$, $n = 8$, $i = 3$. The path to leaf 3 in binary is 010. At each level, P_2 runs a 1-out-of-2 OT to obtain the XOR of either all left or all right children at that level — specifically the children *not* on the path to i. Using the subtrees already reconstructed, P_2 recovers all leaves except leaf 3, and P_1 learns nothing about which leaf was omitted.

The protocol (from [15]) is given below:

Protocol: $(n-1)$-out-of-n Random OT

Public parameters: PRG G of stretch $k > 1$, security parameter λ, integer $n = k^c$ for some $c > 0$.

Inputs: P_1: $\perp$; P_2: index $i \in [n]$.

Outputs: P_1: n random values $\left(r_j\right)_{j\in[n]}$; P_2: $n-1$ random values $\left(r_j\right)_{j\in[n],j\neq i}$.

Key Generation $\mathrm{ROT.Gen}\left(1^\lambda, n, i\right)$:

1. P_1 samples a PRG seed $s_0 \xleftarrow{R} \{0,1\}^\lambda$.
2. P_1 computes a k-ary GGM tree $T = T(s_0, \alpha)$ of depth $\alpha = \log_{k(n)}$, expanding each node s as
$$(s_1 \mid s_2 \mid \cdots \mid s_k) := G(s).$$
3. P_2 computes $(b_1, ..., b_\alpha)$, the k-ary encoding of $i-1$.
4. The parties execute α instances of $(k-1)$-out-of-k OT in parallel. For the l-th OT:
   - P_1 acts as sender. The j-th message is
   $$m_j := \bigoplus_{s\in\{p_x : x\equiv j \bmod k\}} s$$
   where $(p_1, ..., p_{k^l})$ are the seeds at level l of T.
   - P_2 acts as chooser with selection set $\{0, ..., k-1\} \setminus \{b_l\}$, obtaining seeds $q_{l,j}$ for $j \in [k] \setminus \{b_l\}$.
5. P_1 outputs $K_1 \leftarrow s_0$.
6. P_2 outputs $K_2 \leftarrow \left(q_{l,j}\right)_{l\in[\alpha],j\in[k]\setminus\{b_l\}}$.

Expansion $\mathrm{ROT.Expand}(b, K_b)$:

- If $b = 1$: P_1 returns the list of leaves of $T(K_1, \alpha)$.
- If $b = 2$: P_2 uses the seeds $\left(q_{l,j}\right)$ to reconstruct T level by level, recovering all leaves except the one at position i.

Connection to privately punctured PRFs. This protocol gives exactly a privately punctured PRF: P_1 holds the full PRF key (the GGM root s_0) and can evaluate the PRF at any input. P_2 holds a punctured key and can evaluate everywhere except position i, which remains hidden from P_1. This differs from classical punctured PRFs, where the party holding the full key generates the punctured key and therefore knows the punctured point.

Communication cost. P_1's key is a λ-bit seed. P_2's key is $\log_k(n)$ PRG seeds. Expansion costs $O(n)$ PRG evaluations per party. Total communication is $O(\lambda \log n)$, compared to $O(\lambda n)$ for a naive approach.

This construction implies a privately punctured PRF key where one party(P1) has the PRF key(GGM root) and can evaluate the PRF at any input. The other party, P2 has

a punctured key which allows it to evaluate the PRF on all but one inputs. OT helps P2 to get the punctured key without revealing the punctured point to P1.

8.2.2. Known-Index SPFSS

This section shows how the $(n-1)$-out-of-n Random OT protocol can be used to jointly generate FSS keys for a point function. The setup assumes that one party, P_2, already knows the non-zero evaluation point i (but not the evaluation β at that point). Schoppmann et al. [15] call this *Known-Index SPFSS* to emphasise that one party knows the non-zero index.

From Random OT to shares of a point function. After executing the $(n-1)$-out-of-n ROT and expanding, P_1 holds all n leaves of the GGM tree as the vector $\boldsymbol{r}^1$. P_2 holds all leaves except at index i, giving a vector $\boldsymbol{r}^2$ that has the same values at all positions as $\boldsymbol{r}^2$ except at position i. Since $r^1_j = r^2_j$ for all $j \neq i$, if P_2 negates their vector then $\boldsymbol{r}^1 + (-\boldsymbol{r}^2) = \boldsymbol{0}$ everywhere except at position i. The parties therefore already hold additive shares of the zero vector- with the i-th slot "unfinished".

$r^1_j = r^2_j$ for all $j \neq i$

$\boldsymbol{r}^1 + (-\boldsymbol{r}^2) = \boldsymbol{0}$ everywhere except at position i.

The correction step. What we actually want is for the i-th position to sum to $\beta = \beta_1 + \beta_2$ rather than some random value. The difficulty is that P_1 knows β_1 but not i, and P_2 knows i and β_2 but not β_1 or r^1_i.

P_1 computes the sum of all their leaves $R = \sum_{j\in[n]} r^1_j$ and sends P_2 the value:

$$R_\beta = R - \beta_1.$$

P_2 then computes the correction:

$$\tilde{r} = \beta_2 - R_\beta + \sum_{j\in[n]\setminus\{i\}} r^2_j.$$

Since $r^2_j = r^1_j$ for all $j \neq i$, this simplifies to:

$$\begin{aligned}\tilde{r} &= \beta_2 - (R - \beta_1) + \sum_{j\neq i} r_j \\ &= (\beta_2 + \beta_1) - \sum_{j\in[n]} r_j + \sum_{j\neq i} r_j \\ &= \beta - r^1_i.\end{aligned}$$

So $r^1_i + \tilde{r} = \beta$ at position i, giving a correct additive sharing of the point function. P_2 stores $\tilde{r}$ as part of their key, and applies the correction during expansion.

This does not leak i to P_1. P_2 receives $R_\beta = \sum_j r^1_j - \beta_1$. Since r^1_i is the one leaf P_2 did not receive from the ROT, it acts as a one-time pad masking β_1. P_2 cannot extract any information about β_1 from R_β. P_1 sends only R_β and receives nothing, so P_1 learns nothing about i.

Keys and evaluation. The final keys are:

- $K_1 = K_1^{\mathrm{ROT}}$: the GGM root; P_1 evaluates the PRF at any point to get r^1_x.

- $K_2 = \left(K_2^{\mathrm{ROT}}, \tilde{r}\right)$: the punctured PRF key plus the correction value.

Evaluation at input x:
- P_1 outputs r_x^1.
- P_2: if $x = i$, output $\tilde{r}$; otherwise output $-r_x^2$.

Together they sum to β at position i and 0 everywhere else.

(for slides, below:) **The correction step.**
Goal: $[i]$: $\beta = \beta_1 + \beta_2$

$P_2(i, \beta_2)$ $\quad$ $P_1(\beta_1)$

The difficulty is that P_1 knows β_1 but not i, and P_2 knows i and β_2 but not β_1 or r_i^1.

$$R = \sum_{j \in [n]} r_j^1, R_\beta = R - \beta_1.$$

$$\tilde{r} = \beta_2 - R_\beta + \sum_{j \in [n] \setminus \{i\}} r_j^2.$$

Since $r_j^2 = r_j^1$ for all $j \neq i$, this simplifies to:
$\tilde{r} = \beta_2 - (R - \beta_1) + \sum_{j \neq i} r_j^1$
$= \beta - \sum_j r_j^1 + \sum_{j \neq i} r_j^1$
$= \beta - r_i^1$.

So $r_i^1 + \tilde{r} = \beta$ at position i, giving a correct additive sharing of the point function. P_2 stores $\tilde{r}$ as part of their key, and applies the correction during expansion.

P_1 does not learn i

P_2 does not learn β_1:

P_2 receives $R_\beta = \sum_j r_j^1 - \beta_1$.
r_i^1 acts as a one-time pad masking β_1.

P_2 cannot extract any information about β_1 from R_β. P_1 sends only R_β and receives nothing, so P_1 learns nothing about i.

Protocol: Distributed Known-Index SPFSS

Building blocks: $(n-1)$-out-of-n ROT. Point function $f : [n] \to \mathbb{G}$ with $f(i) = \beta$, $f(j) = 0$ for all $j \neq i$. Random shares β_1, β_2 with $\beta_1 + \beta_2 = \beta$.

Parties: P_1, P_2

Inputs: P_1: β_1; P_2: β_2, i.

Key Generation $\text{SPFSS.Gen}\big(1^\lambda, f_{i,\beta}\big)$:

1. The parties run $\text{ROT.Gen}\big(1^\lambda, n, i\big)$ to obtain keys K_1^{ROT} and K_2^{ROT}.
2. The parties locally run ROT.Expand: P_1 gets $\big\{r_j\big\}_{j\in[n]}$, P_2 gets $\big\{r_j\big\}_{j\in[n],j\neq i}$.
3. Let $R = \sum_{j\in[n]} r_j$. P_1 sends $R_\beta = R - \beta_1$ to P_2.
4. P_2 computes $\tilde{r} = \beta_2 - R_\beta + \sum_{j\in[n]\setminus\{i\}} r_j$.
5. P_1 outputs $K_1 \leftarrow K_1^{\text{ROT}}$.
6. P_2 outputs $K_2 \leftarrow \big(K_2^{\text{ROT}}, \tilde{r}\big)$.

Expansion $\text{SPFSS.Eval}(b, K_b, x)$:

- If $b = 1$: compute $\boldsymbol{v}^1 \leftarrow \text{ROT.Expand}(1, K_1)$ and output v_x^1.
- If $b = 2$: parse K_2 as $\big(K_2^{\text{ROT}}, \tilde{r}\big)$. If $x = i$, output $\tilde{r}$. Otherwise compute $\boldsymbol{v}^2 \leftarrow \text{ROT.Expand}\big(2, K_2^{\text{ROT}}\big)$ and output $-v_x^2$.

8.3. Batching

To handle a t-sparse noise vector $\boldsymbol{e}$ i.e. one with t non-zero entries at positions $i_1, ..., i_t$, we need to batch t instances of Known-Index SPFSS into a single multi-point FSS.

The simplest approach is to run t independent Known-Index SPFSS instances, one per non-zero point. Instance l generates keys $\big(K_1^l, K_2^l\big)$ for the point function f_{i_l,β_l}. Evaluation at any input x is then the sum of all t evaluations:

$$\sum_{l=1}^{t} \text{SPFSS.Eval}\big(b, K_b^l, x\big).$$

Since we are working in an additive group, this correctly evaluates to β_l at each i_l and 0 everywhere else, which a valid secret sharing of the t-point function.

8.3.1. Multi-Point FSS via Naive Batching

With naive batching, a t-point FSS is simply t independent Known-Index SPFSS instances whose evaluations are summed.

Protocol: Known-Indices MPFSS (Naive Batching)

Parameters: Domain $[n]$, number of points t. Point function $f_{\boldsymbol{i},\boldsymbol{\beta}} : [n] \to G$ with $f(i_l) = \beta_l$ for $l \in [t]$ and $f(j) = 0$ otherwise.

Inputs: P_1: $(\beta_1^1, ..., \beta_1^t)$; P_2: $(i_1, ..., i_t)$, $(\beta_2^1, ..., \beta_2^t)$ where $\beta_1^l + \beta_2^l = \beta_l$ for each l.

Key Generation $\text{MPFSS.Gen}\big(1^\lambda, f_{\boldsymbol{i},\boldsymbol{\beta}}\big)$:

1. For each $l \in [t]$, run $\text{SPFSS.Gen}\Big(1^\lambda, f_{i_l,\beta_l}\Big)$ to obtain keys $\big(K_1^l, K_2^l\big)$.
2. P_1 outputs $K_1 \leftarrow (K_1^1, ..., K_1^t)$.
3. P_2 outputs $K_2 \leftarrow (K_2^1, ..., K_2^t)$.

Expansion $\text{MPFSS.Eval}(b, K_b, x)$:

Output $\sum_{l=1}^{t} \text{SPFSS.Eval}\big(b, K_b^l, x\big)$.

Communication cost: t calls to SPFSS.Gen, each costing $O(\lambda \log n)$, for a total of $O(\lambda t \log n)$.

Evaluation cost: over the full domain costs $O(tn)$.

8.4. PCG for VOLE from LPN

We now have all the ingredients to assemble the full distributed VOLE protocol. Recall the goal: P_1 obtains vectors $\boldsymbol{u}, \boldsymbol{v} \in \mathbb{F}^n$ and P_2 obtains $x \in \mathbb{F}$ and $\boldsymbol{w} \in \mathbb{F}^n$ such that $\boldsymbol{u}x + \boldsymbol{v} = \boldsymbol{w}$.

The idea, due to Boyle et al. [6], is to start from short vectors $\boldsymbol{a}, \boldsymbol{b}, \boldsymbol{c} \in \mathbb{F}^k$ with $\boldsymbol{c} = \boldsymbol{a}x + \boldsymbol{b}$, generated using a small MPC, and expand them to long pseudorandom vectors using the LPN assumption via a code generating matrix $C \in \mathbb{F}^{k \times n}$. Adding a sparse noise vector $\boldsymbol{e}$ and its secret-shared product $\boldsymbol{e}x = \boldsymbol{e}\boldsymbol{x}_{\mathcal{P}} + \boldsymbol{e}\boldsymbol{x}_{\mathtt{cal}}(V)$, which generated using our MPFSS gives the full VOLE correlation.

Protocol 6: Distributed Pseudorandom VOLE

Parameters: Vector length n, LPN parameters t, k, code generating matrix $C \in \mathbb{F}^{k \times n}$.

Inputs: None.

Outputs: P_2: $\boldsymbol{u}, \boldsymbol{v} \in \mathbb{F}^n$; P_1: $\boldsymbol{w} \in \mathbb{F}^n$, $x \in \mathbb{F}$, such that $\boldsymbol{u}x + \boldsymbol{v} = \boldsymbol{w}$.

Share Generation $\text{VOLE.Setup}\big(1^\lambda, \mathbb{F}, n\big)$:

1. P_2 chooses random positions $S = \{s_1, ..., s_t\} \subset [n]$, random values $\boldsymbol{y} = (y_1, ..., y_t) \in \mathbb{F}^t$, and random vectors $(\boldsymbol{a}, \boldsymbol{b}) \leftarrow \mathbb{F}^k$. P_1 chooses random $x \xleftarrow{R} \mathbb{F}$.
2. For each $l \in [t]$, the parties run a secure multiplication to obtain additive shares $\beta_1^l + \beta_2^l = xy_l$, with P_1 holding β_1^l and P_2 holding β_2^l.
3. The parties run $\text{MPFSS.Gen}\big(1^\lambda, f_{S,\boldsymbol{xy}}\big)$ to obtain keys (K_1, K_2).
4. The parties run an MPC on inputs $\boldsymbol{a}, \boldsymbol{b}$ (from P_2) and x (from P_1) to give P_1 the vector $\boldsymbol{c} = \boldsymbol{a}x + \boldsymbol{b} \in \mathbb{F}^k$.
5. P_2 outputs $\text{seed}_2 \leftarrow (K_2, S, \boldsymbol{y}, \boldsymbol{a}, \boldsymbol{b})$.
6. P_1 outputs $\text{seed}_1 \leftarrow (K_1, x, \boldsymbol{c})$.

Expansion $\text{VOLE.Expand}(b, \text{seed}_b)$:

- If $b = 2$: P_2 computes $[\boldsymbol{ex}]_{\mathcal{P}}[i] \leftarrow \text{MPFSS.Eval}(2, K_2, i)$ for all $i \in [n]$, and constructs the sparse vector $\boldsymbol{e} \in \mathbb{F}^n$ with $e[s_l] = y_l$ for $l \in [t]$ and $e[j] = 0$ for $j \notin S$. P_2 outputs
$$\boldsymbol{u} = \boldsymbol{a}C + \boldsymbol{e}, \quad \boldsymbol{v} = \boldsymbol{b}C - [\boldsymbol{ex}]_{\mathcal{P}}.$$
- If $b = 1$: P_1 computes $[\boldsymbol{ex}]_{\mathcal{V}}[i] \leftarrow \text{MPFSS.Eval}(1, K_1, i)$ for all $i \in [n]$. P_1 outputs
$$\boldsymbol{w} = \boldsymbol{c}C + [\boldsymbol{ex}]_{\mathcal{V}}.$$

This answers our original question in the positive: sparse VOLE, combined with the LPN assumption and our MPFSS construction, gives a full pseudorandom VOLE with sublinear communication.

Chapter 9.

Commit and Sample

The main communication bottleneck remaining in our interactive proof system for proving the MLWE relation is that $\mathcal{P}$ and $\mathcal{V}$ must communicate $2(n \cdot k)$ field elements to de-randomize their VOLE commitments to $\boldsymbol{s}$ and $\boldsymbol{e}$. We now ask whether this can be reduced.

Can $\mathcal{P}$ and $\mathcal{V}$ communicate sublinearly in $2(n \cdot k)$ to obtain de-randomized VOLE commitments to $\boldsymbol{s}$ and $\boldsymbol{e}$?

The key observation is that in the optimized CPA Kyber parameters, $\boldsymbol{s}$ and $\boldsymbol{e}$ are not arbitrary elements of S_η^k, they are sampled from a binomial distribution i.e $(s, e) \xleftarrow[\text{CBD}]{} S_\eta^k$. We exploit this structure below.

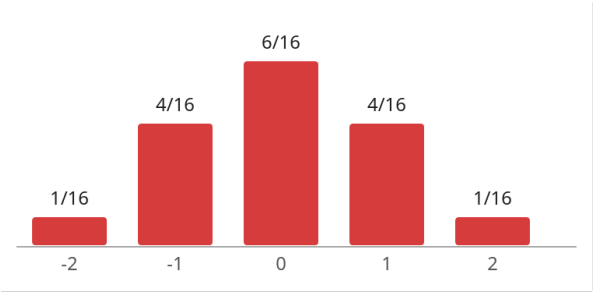
9.1. Central Binomial Distribution

A polynomial can be selected from S_η by drawing each coefficient according to the *central binomial distribution* (CBD). To sample one coefficient c, draw η pairs of bits (a_i, b_i) uniformly at random and compute:

$$c = \sum_{i=1}^{\eta} (a_i - b_i).$$

Note that $c \in [-\eta, \eta]$, and for each $j \in [-\eta, \eta]$:

$$\Pr(c = j) = \binom{2\eta}{\eta + j} / 2^{2\eta}.$$



Sampling cost. To get one coefficient in $\mathbb{F}_q$ via CBD requires first sampling 2η random bits in $\mathbb{F}_q$. To sample all $2(n \cdot k)$ coefficients of $\boldsymbol{s}$ and $\boldsymbol{e}$ for one MLWE relation requires $2\eta \cdot 2(n \cdot k)$ random bits in $\mathbb{F}_p$ in total. (if $\eta_1 = \eta_2$)
On the other hand, if $\eta_1 \neq \eta_2$, to sample $(n \cdot k)$ coefficients for s, we need $2\eta_1(n \cdot k)$ bits in $\mathbb{F}_p$. To sample $(n \cdot k)$ coefficients for e, we need $2\eta_2(n \cdot k)$ random bits in $\mathbb{F}_q$

9.2. Commit-then-Sample vs Sample-then-Commit

There are two natural orderings for generating VOLE commitments to $\boldsymbol{s}$ and $\boldsymbol{e}$:

- **Sample-then-commit:** sample $\boldsymbol{s} \in S_{\eta_1}^k$ and $\boldsymbol{e} \in S_{\eta_2}^k$ via CBD first, then commit to the resulting $2(n \cdot k)$ coefficients via $\mathcal{F}_{\text{sVOLE}}^r$.
- **Commit-then-sample:** commit to $2\eta \cdot 2(n+k)$ (if $\eta_{\neq}\eta_2$ then $2\eta_1(n \cdot k) + 2\eta_2(n \cdot k)$) *random bits* in $\mathbb{F}_p$ first, then have both parties apply the deterministic CBD transformation locally to the committed bits to derive the committed s and e.

Let $n_{\text{e_coeffs}}$:= number of co-effs in e commited in 1 MLWE relation Let $n_{\text{s_coeffs}}$:= "number of co-effs in s commited in 1 MLWE relation"

- **Sample-then-commit:**
  1. $\boldsymbol{s} \overset{\text{CBD}}{\leftarrow} S_{\eta_1}^k$, $\boldsymbol{e} \overset{\text{CBD}}{\leftarrow} S_{\eta_1}^k$
  2. $[\![s]\!] \leftarrow \text{commit}(s), [\![e]\!] \leftarrow \text{commit}(e)$

  Calls to VOLE.Derand : $2 \cdot (n \cdot k)$ for 1 MLWE relation
  Calls to $\mathcal{F}_{\text{sVOLE}}^r$: $2 \cdot (n \cdot k)$ for 1 MLWE relation
- **Attempt 1: Commit-then-sample:**
  1. $b_i \overset{R}{\leftarrow} \{0,1\} \subset \mathbb{F}_p \quad \forall i \in [2\eta_1 \cdot 2(n \cdot k)]$
  2. $[\![b_i]\!] \leftarrow \text{commit}(b_i) s.t [\![b_i]\!] = [\![b_e]\!] | \, [\![b_s]\!]$
  3. $s = \text{CBD}_{\eta_1}([\![b_s]\!]), e = \text{CBD}_{\eta_2}([\![b_e]\!])$

  Calls to VOLE.Derand : $2\eta_1 \cdot 2(n \cdot k)$ for 1 MLWE relation
  Calls to $\mathcal{F}_{\text{sVOLE}}^r$: $2\eta_1 \cdot 2(n \cdot k)$ for 1 MLWE relation
- **Attempt 2: Commit-then-sample with PRG:**
  1. short seed $\overset{R}{\leftarrow} \{0,1\} \subset \mathbb{F}_p$
  2. $[\![\text{short seed}]\!] \leftarrow \text{commit(short seed)}$
  3. $\text{PRG}([\![\text{short seed}]\!]) \rightarrow [\![b_i]\!] \quad s.t. [\![b_i]\!] = [\![b_e]\!] | \, [\![b_s]\!]$
  4. $s = \text{CBD}_{\eta_1}([\![b_s]\!]), e = \text{CBD}_{\eta_2}([\![b_e]\!])$

  Calls to VOLE.Derand : $O(|\text{short seed}|)$ for 1 MLWE relation
  Calls to $\mathcal{F}_{\text{sVOLE}}^r$: $O(|\text{short seed}|)$ for 1 MLWE relation
- **Attempt 3: Commit-then-sample with PRG in the batch setting:**
  1. short seed $\overset{R}{\leftarrow} \{0,1\} \subset \mathbb{F}_p$
  2. $[\![\text{short seed}]\!] \leftarrow \text{commit(short seed)}$
  3. $\text{PRG}([\![\text{short seed}]\!]) \rightarrow [\![b_i]\!] \quad s.t. [\![b_i]\!] = z \cdot ([\![b_e]\!] | \, [\![b_s]\!])$
  4. $s_j = \text{CBD}_{\eta_1}([\![b_s]\!]), e_j = \text{CBD}_{\eta_2}([\![b_e]\!]) \quad \forall i \in [z]$

  Calls to VOLE.Derand : $O(|\text{short seed}|)$ for z MLWE relation
  Calls to $\mathcal{F}_{\text{spVOLE}}^r$: $O(|\text{short seed}|)$ for z MLWE relation

At first glance the second approach appears more expensive, not less: it requires $2\eta \cdot 2(n \cdot k)$ calls to $\mathcal{F}_{\text{sVOLE}}^r$ and $2\eta \cdot 2(n \cdot k)$ de-randomization calls, compared to only $2(n \cdot k)$ of each in the first approach. However, there is a crucial difference: the first approach requires committing to structured values drawn from a CBD, whereas the second requires

committing only to *random bits* in $\mathbb{F}_p$. This opens the door to a classical trick: pseudorandomness is enough.

9.3. Pseudorandom Bits are Sufficient for CBD

If the bits (a_i, b_i) fed into the CBD are pseudorandom rather than truly random, the resulting coefficients are computationally indistinguishable from genuine CBD samples, provided the underlying PRG is secure. This follows directly from the security of the PRG: any PPT adversary distinguishing pseudorandom CBD samples from true CBD samples could be used to distinguish the PRG output from uniform, contradicting its security.

This suggests the following optimization: instead of committing to $2\eta \cdot 2(n \cdot k)$ random bits, $\mathcal{P}$ and $\mathcal{V}$ commit to a *much smaller* seed of λ bits and expand it locally using a PRG to obtain pseudorandom commitments to the required $2\eta \cdot 2(n \cdot k)$ bits, and then apply CBD. If expanding the seed is cheap relative to committing directly, and the PRG has a low enough multiplicative depth over $\mathbb{F}_q$ this yields a net reduction in communication.

Goal: find a PRG that is cheap to evaluate on committed inputs and the output shave a low-degree, i.e. one that is compatible with the VOLE commitment scheme.

9.4. Mixed Low-Degree Pseudorandom Generator

A mixed low-degree PRG [13] differs from a standard PRG in two ways: the seed may be drawn from a structured distribution rather than being uniformly random, and instead of a single evaluation algorithm the PRG is specified by circuits that may depend on the modulus q.

Definition 9.4.1 (Mixed low-degree PRG)

A mixed low-degree pseudorandom generator PRG is defined by the following pair of efficient algorithms:

`PRG.Setup`(1^λ) A probabilistic algorithm that takes the security parameter λ and outputs parameters `pars`, containing the seed length ℓ, output length m, and a seed distribution $\mathcal{D}$ over $\{0,1\}^\ell$.

`PRG.CircGen(pars,`q, i`)` A deterministic algorithm that takes `pars`, a prime q, and an index $i \in [m]$, and outputs a circuit C over $\mathbb{F}_q$ with addition and multiplication gates. The circuit has ℓ input wires and computes the i-th output bit of the PRG.

For a prime p, we say PRG has *degrees* (d_2, d_p) if the circuits output by `CircGen(pars` $, 2, \cdot)$ and `CircGen(pars,`$p, \cdot)$ have degree at most d_2 and d_p respectively.

For correctness, the circuits generated by `CircGen` must produce the same output vector across all fields. For security, the outputs must be pseudorandom. We are interested in constructions where both degrees (d_2, d_p) are as small as possible.

9.4.1. Instantiation from Sparse LPN

The sparse LPN assumption states that $(\boldsymbol{A}, \boldsymbol{s}\boldsymbol{A} + \boldsymbol{e})$, where $\boldsymbol{A} \xleftarrow{R} \mathbb{F}_2^{k \times n}$, $\boldsymbol{s} \xleftarrow{R} \mathbb{F}_2^k$, and $\boldsymbol{e}$ is a low Hamming weight noise vector, is indistinguishable from a uniformly random pair $(\boldsymbol{A}, \boldsymbol{u})$.

We use a *sparse* variant due to Alekhnovich, where the matrix $\boldsymbol{A}$ is sampled from an algorithm $G(k, n, t)$ such that each column contains exactly d ones at random positions. We further assume a fixed-weight *regular error distribution* $N(n, t)$, where the noise vector $\boldsymbol{e}$ consists of n/t consecutive blocks each of Hamming weight 1. This gives a mixed low-degree PRG as follows:

- `Setup` generates `pars` with seed length $\ell = k + n$ and output length n. The seed distribution $\mathcal{D}$ outputs $(\boldsymbol{s}, \boldsymbol{e})$ where $\boldsymbol{s} \underset{R}{\in} \{0,1\}^k$ and $\boldsymbol{e} \leftarrow N(n, t)$.
- `CircGen` for $q = 2$ outputs the Boolean circuit computing $\boldsymbol{sA} + \boldsymbol{e} \bmod 2$. For $q > 2$, the circuit emulates the same Boolean operations over $\mathbb{F}_q$.

Since each column of $\boldsymbol{A}$ is d-sparse, computing each output entry of $\boldsymbol{sA} + \boldsymbol{e}$ requires *only* d XOR operations, rather than k ($d - 1$ XOR operations for $A \cdot s$, and 1 XOR operation to add e), which translates to a degree-$d + 1$ polynomial over $\mathbb{F}_p$. This gives a $\left(d_2, d_p\right)$ degree PRG with $d_2 = 1$ and $d_p = d + 1$.

9.4.2. Instantiation from Sparse LPN

The sparse LPN assumption states that $(\boldsymbol{A}, \boldsymbol{sA} + \boldsymbol{e})$, where $\boldsymbol{A} \overset{R}{\leftarrow} \mathbb{F}_2^{k \times n}$, $\boldsymbol{s} \overset{R}{\leftarrow} \mathbb{F}_2^k$, and $\boldsymbol{e}$ is a low Hamming weight noise vector, is indistinguishable from a uniformly random pair $(\boldsymbol{A}, \boldsymbol{u})$.

Sparse LPN assumption:

$$(\boldsymbol{A}, \boldsymbol{sA} + \boldsymbol{e}) \approx (\boldsymbol{A}, \boldsymbol{u})$$

- $\boldsymbol{A} \overset{R}{\in} \mathbb{F}_2^{k \times n} \leftarrow$ algorithm $G(k, n, t)$ s.t column contains exactly d ones at random positions.
- $s \overset{R}{\leftarrow} \mathbb{F}_2^{1 \times k}$
- $e \in \mathbb{F}_2^{1 \times n} \leftarrow N(n, t)$ the noise vector $\boldsymbol{e}$ consists of n/t consecutive blocks each of Hamming weight 1.

Since each column of $\boldsymbol{A}$ is d-sparse, computing each output entry of $\boldsymbol{sA} + \boldsymbol{e}$ requires *only* d XOR operations($d - 1$ XOR operations for $A \cdot s$, and 1 XOR operation to add e), rather than k , which translates to a degree-$(d + 1)$ polynomial over $\mathbb{F}_p$. This gives a $\left(d_2, d_p\right)$ degree PRG with $d_2 = 1$ and $d_p = d + 1$.

- seed length $\ell = k + n$ and output length n. The seed distribution $\mathcal{D}$ outputs $(\boldsymbol{s}, \boldsymbol{e})$ where $\boldsymbol{s} \underset{R}{\in} \{0,1\}^k$ and $\boldsymbol{e} \leftarrow N(n, t)$.
- **Attempt 4: Commit-then-sample with PRG**
  in the batch setting:
  1. $[\![s_{\text{seed}}]\!] \leftarrow \text{commit}(s_{\text{seed}})$, $[\![e_{\text{seed}}]\!] \leftarrow \text{commit}(e_{\text{seed}})$
  2. $\text{PRG}([\![e_{\text{seed}}]\!] | [\![s_{\text{seed}}]\!]) \rightarrow [\![b_i]\!] \quad s.t. [\![b_i]\!] = z \cdot ([\![b_e]\!] \mid [\![b_s]\!])$
  3. $s_j = \text{CBD}_{\eta_1}([\![b_s]\!]), e_j = \text{CBD}_{\eta_2}([\![b_e]\!]) \quad \forall i \in [z]$

  Calls to VOLE.Derand : $|s_{\text{seed}}| = k_{\text{LPN}}$ for z MLWE relation
  Calls to $\mathcal{F}^r_{\text{sVOLE}}$: $|s_{\text{seed}}| = k_{\text{LPN}}$ for z MLWE relation
  Calls to $\mathcal{F}^r_{\text{spVOLE}}$: d, each costs $(\log n_{\text{LPN}} \cdot \lambda + r \cdot \log q)$ bits in communication

9.4.3. Worked Example

To make the construction concrete, we trace through a small instance. The parties hold VOLE commitments to a seed $(\llbracket \boldsymbol{s} \rrbracket_1, \llbracket \boldsymbol{e} \rrbracket)_1$, where both vectors are committed as elements of $\mathbb{F}_q$ despite their values being bits. The goal is to expand this short committed seed into a longer committed binary string $\boldsymbol{y} \in \{0,1\}^n$, without any further interaction.

We use parameters $k = 3$, $n = 6$, $d = 2$(hamming weight of each column of A), and $t = 2$(hamming weight of e), giving each block of size $n/t = 3$.

The vector $\boldsymbol{e}$ is drawn from $N(6, 3)$: each block of size $n/t = 3$ contains exactly one 1. The seed has total length $\ell = k + n = 9$ bits, all committed over $\mathbb{F}_p$. To commit to the dense s, the parties first get random voles, and then de-randomize them to 0 and 1s costing $O(|s|) = O(k)$. To commit to the sparse e naively costs $O(|e|) = O(n)$ (and n is very big), so we can use a GGM tree optimization costing $t \cdot O(\log n)$

Further, committing to n bits of $\boldsymbol{e}$ is optimized (by committing to $\frac{n}{t}$ GGM trees)

Seed. The parties hold the commitments:

$$\llbracket \boldsymbol{s} \rrbracket = (\llbracket 1 \rrbracket, \llbracket 0 \rrbracket, \llbracket 1 \rrbracket), \llbracket \boldsymbol{e} \rrbracket = \left(\underbrace{\llbracket 0 \rrbracket, \llbracket 1 \rrbracket, \llbracket 0 \rrbracket}_{\text{block 1}}, \underbrace{\llbracket 0 \rrbracket, \llbracket 0 \rrbracket, \llbracket 1 \rrbracket}_{\text{block 2}} \right).$$

Public matrix. The matrix $\boldsymbol{A} \in \mathbb{F}_2^{3\times 6}$ is generated by $G(k, n, t)$ with exactly $d = 2$ ones per column:

$$\boldsymbol{A} = \begin{pmatrix} 1 & 1 & 1 & 0 & 1 & 0 \\ 1 & 0 & 1 & 1 & 0 & 1 \\ 0 & 1 & 0 & 1 & 1 & 1 \end{pmatrix}.$$

Computing the output. The parties locally evaluate the circuit $C = \llbracket s \rrbracket A + \llbracket e \rrbracket$ using `VOLE-ZK.Eval`$(C, (\llbracket \boldsymbol{s} \rrbracket, \llbracket \boldsymbol{e} \rrbracket))$. Some details are illustrated below:

The j-th output bit is

$$y_j = \left(\bigoplus_{i : A_{ij} = 1} s_i \right) \oplus e_j.$$

Because each column of $\boldsymbol{A}$ has exactly $d = 2$ ones, each output bit depends on only 2 entries of $\boldsymbol{s}$, regardless of how large k is. This is the key efficiency gain from sparsity: instead of touching all k entries of $\boldsymbol{s}$, only d are read per output bit. Evaluating all six outputs:

$$\begin{aligned} y_1 &= s_1 \oplus s_2 \oplus e_1 = 1 \oplus 0 \oplus 0 = 1 \\ y_2 &= s_1 \oplus s_3 \oplus e_2 = 1 \oplus 1 \oplus 1 = 1 \\ y_3 &= s_1 \oplus s_2 \oplus e_3 = 1 \oplus 0 \oplus 0 = 1 \\ y_4 &= s_2 \oplus s_3 \oplus e_4 = 0 \oplus 1 \oplus 0 = 1 \\ y_5 &= s_1 \oplus s_3 \oplus e_5 = 1 \oplus 1 \oplus 0 = 0 \\ y_6 &= s_2 \oplus s_3 \oplus e_6 = 0 \oplus 1 \oplus 1 = 0 \end{aligned}$$

So $\boldsymbol{y} = (1, 1, 1, 1, 0, 0)$. The parties now hold VOLE commitments to these 6 output bits, produced from a seed of 9 committed bits.

Emulating XOR over $\mathbb{F}_p$. The VOLE commitments live over $\mathbb{F}_p$, so the parties evaluate the circuit using $\mathbb{F}_p$ arithmetic. Over $\mathbb{F}_2$, XOR is just addition. Over $\mathbb{F}_p$, however, ordinary addition gives the wrong answer when both inputs are 1: $1 + 1 = 2 \neq 0$. For two committed bits $a, b \in \{0, 1\}$, XOR is instead emulated as

$$a \oplus b = a + b - 2ab \pmod{p}.$$

The correction term $-2ab$ equals -2 precisely when $a = b = 1$, reducing 2 back to 0; in all other cases $ab = 0$ and the formula reduces to plain addition. Each such emulation requires one multiplication on committed values which in turn increases the VOLE degree of the output by one.

To illustrate, take $y_2 = s_1 \oplus s_3 \oplus e_2$. The parties evaluate over $\mathbb{F}_p$ as

$$\begin{aligned} [\![r]\!]_2 &= [\![s_1]\!]_1 + [\![s_3]\!]_1 - 2[\![s_1]\!]_1[\![s_3]\!]_1 \quad \text{(degree 2 over } \mathbb{F}_p\text{)} \\ [\![y_2]\!]_3 &= [\![r]\!]_2 + [\![e_2]\!]_1 - 2[\![r]\!]_2[\![e_2]\!]_1 \end{aligned}$$

Each output bit therefore costs d multiplications on committed values, which is a small constant independent of k or n. This gives $d_p = d + 1$ (or d+1 if error term in 1-confirm with peter), and with $d = 10$ in the concrete parameters the homomorphic evaluation remains entirely practical inside the zero-knowledge proof.

9.5. Commiting to sparse e_{seed}

Functionality $\mathcal{F}^{p,r}_{\text{spsVOLE}}$

Parameters: Prime p, extension degree r, vector length $n = 2^h$.

Roles: P_A (prover), P_B (verifier) with global key $\Delta \in \mathbb{F}_{p^r}$.

Initialize (run once):

Upon receiving init from P_A and P_B:
1. Sample $\Delta \xleftarrow{R} \mathbb{F}_{p^r}$ and send Δ to P_B.

Extend (may be run multiple times):

Upon receiving $(\mathsf{sp\text{-}extend}, n)$ from P_A and P_B:
1. Sample $\boldsymbol{v} \xleftarrow{R} \mathbb{F}_{p^r}^n$.
2. Sample $\boldsymbol{u} \in \mathbb{F}_p^n$ uniformly subject to exactly one nonzero entry at index $\alpha \xleftarrow{R} [0, n)$.
3. Compute $\boldsymbol{w} := \boldsymbol{v} + \Delta \cdot \boldsymbol{u} \in \mathbb{F}_{p^r}^n$.
4. Send $(\boldsymbol{u}, \boldsymbol{w})$ to P_A and $\boldsymbol{v}$ to P_B.

Realizing $\mathcal{F}^{p,r}_{\text{spsVOLE}}$ with sublinear communication: We can realize $\mathcal{F}^{p,r}_{\text{spsVOLE}}$ with logarithmic communication and one round using the $(n-1)$-out-of-n random OT protocol. The formal protocol with malicious security and proof is given in [5]. But idea is the same

as from Section 8.2.1. For clarity we fix $\beta = 1$, so $\boldsymbol{u}$ has a single entry equal to 1 at the secret index α. The protocol proceeds as follows.

$\mathcal{V}$ samples a root seed $s \xleftarrow{R} \{0,1\}^\lambda$ and expands it into a binary tree of depth $h = \log_2 n$ using a PRG $G : \{0,1\}^\lambda \to \{0,1\}^{2\lambda}$, producing two child seeds from each parent by writing $\big(s_{2j}, s_{2j+1}\big) \coloneqq G\big(s_j\big)$ at each internal node. At the leaves, $\mathcal{V}$ applies a second PRG $G' : \{0,1\}^\lambda \to \mathbb{F}_{p^r}^2$ to each pair of adjacent leaf seeds, obtaining the full leaf vector $\boldsymbol{v} = (v[0], ..., v[n-1]) \in \mathbb{F}_{p^r}^n$. $\mathcal{V}$ retains all leaf values.

$\mathcal{P}$ holds the secret index $\alpha \in [0, n)$. At each level $i \in [h]$ of the tree, let $|(\alpha)_i$ denote the complement of the i-th bit of the binary representation of α, and let K_0^i, K_1^i be the XOR of all even-indexed, respectively odd-indexed, node seeds at level i. $\mathcal{P}$ and $\mathcal{V}$ run an OT at each level: $\mathcal{P}$ inputs $|(\alpha)_i$ and $\mathcal{V}$ inputs (K_0^i, K_1^i), so $\mathcal{P}$ receives $K^i_{|(\alpha)_i}$. From these h values, $\mathcal{P}$ runs $\mathrm{GGM}'\left(\alpha, \left\{K^i_{|(\alpha)_i}\right\}_{i\in[h]}\right)$ to reconstruct $\{v[j]\}_{j\neq\alpha}$, while $v[\alpha]$ remains hidden. The OT phase requires h transfers of λ bits each, for a total of $h \cdot \lambda$ bits.

Having obtained all leaf values except $v[\alpha]$, $\mathcal{P}$ can now construct its share. The spsVOLE relation requires $\boldsymbol{w} = \boldsymbol{v} + \Delta \cdot \boldsymbol{u}$, where $\boldsymbol{w}$ is $\mathcal{P}$'s output vector. Setting $w[i] = v[i]$ for $i \neq \alpha$ satisfies the relation at those positions since $u[i] = 0$. At position α, the relation requires $w[\alpha] = v[\alpha] + \Delta$, but $\mathcal{P}$ knows neither $v[\alpha]$ nor Δ.

To resolve this, $\mathcal{V}$ computes and sends to $\mathcal{P}$ the correction value

$$d = \sum_{i=0}^{n-1} v[i] + \Delta.$$

Note that $\mathcal{V}$ computes d purely from quantities it already holds, all leaf values together with Δ, and in particular does not need to know α. Since $\mathcal{P}$ knows d and every $v[j]$ with $j \neq \alpha$, it recovers

$$\begin{aligned} w[\alpha] &= d - \sum_{i\neq\alpha} v[i] \\ &= \left(\sum_{i=0}^{n-1} v[i] + \Delta\right) - \sum_{i\neq\alpha} v[i] \\ &= v[\alpha] + \Delta, \end{aligned}$$

$$d = \sum_{j\in[n]} r_j + x$$

$$\begin{aligned} r_j^2 &= d - \sum_{j\in[n]\setminus\{i\}} \\ &= \sum_{j\in[n]} r_j + x - \sum_{j\in[n]\setminus\{i\}} \\ &= r_j^1 + x \end{aligned}$$

which is exactly the required value.

For presentation: (below)

To resolve this, $\mathcal{V}$ computes and sends to $\mathcal{P}$ the correction value

$$d = \sum_{j \in [n]} r_j^1 + x.$$

Note that $\mathcal{V}$ computes d purely from quantities it already holds, all leaf values together with Δ, and in particular does not need to know α. Since $\mathcal{P}$ knows d and every $v[j]$ with $j \neq \alpha$, it recovers

$$\begin{aligned} r_i^2 &= d - \sum_{j \neq i} r_j \\ &= \left(\sum_{j \in [n]} r_j^1 + x \right) - \sum_{j \neq i} r_j \\ &= r_i^1 + x \end{aligned}$$

which is exactly the required value.

Old:
To resolve this, $\mathcal{V}$ computes and sends to $\mathcal{P}$ the correction value

$$d = \sum_{i=0}^{n-1} v[i] - \Delta.$$

Since $\mathcal{P}$ knows d and all $v[j]$ for $j \neq \alpha$, it recovers

$$w[\alpha] = \Delta - d + \sum_{i \neq \alpha} v[i] = \Delta - \left(\sum_{i=0}^{n-1} v[i] - \Delta \right) + \sum_{i \neq \alpha} v[i].$$

Rearranging:

$$w[\alpha] = v[\alpha] + \Delta,$$

which is exactly the required value. Note that d does not reveal α to $\mathcal{P}$, since it is a global sum over all leaf values. Symmetrically, $\mathcal{V}$ cannot determine α from the protocol transcript, as OT security ensures α remains hidden.

The total additional communication beyond the OT phase is a single field element $d \in \mathbb{F}_{p^r}$, giving an overall communication complexity of $O(\lambda \log n)$, logarithmic in n rather than linear. We can insatiate oblivious transfer via the OT extension getting amortized costs per OT.

9.6. LPN Parameter selection

As suggested in prior work [5], we choose the public LPN matrix A as a generator of a 10-local linear code, which means that each column of A contains exactly 10 (uniform) nonzero entries. We use values from prior work as $k = 589,760, n = 10805248, t = 1319$ Using these parameters we can actually prove not one but z MLWE relations in a batch setting.

9.7. ZKP protocol for batch verifying MLWE relation

With all this background, in this section we now give a method for the prover to sample z MLWE relations according to the parameters from ML-KEM where s_i and e_i are sampled from a central binomial distribution. The binomial sampling is applied not on random bits, rather on pseudorandom bits generated using a low degree PRG based on binary sparse LPN. After this, we give the concrete joint ZKP statement that the prover wishes to prove on both the pseudorandom bits, binomial sampling on those pseudorandom bits, and the proof of MLWE relation. After that we give a protocol based on [13] and [12] that achieves it in zero knowledge with soundness error of $1/|\mathbb{F}_{q^r}|$. We calculate concrete communication costs after. Note that becuause we sample the pseudorandom bits using a PRG, we are able to prove not one but many MLWE relations in a batch setting. This also gives us nicer amortized costs.

The prover generates z MLWE relations and sets up the proof as follows:

1. **Generate LPN based PRG parameters and short random seeds:** $\mathcal{P}$ runs `Setup` and `CircGen` to get $s_{\text{seed}} \underset{R}{\in} \{0,1\}^k, e_{\text{seed}} \leftarrow N(n,t)$, $A \in \mathbb{F}_2^{n \times k}$
2. **Evaluate PRG to get a long pseudorandom bits:** $\mathcal{P}$ evaluates the circuit $s_{\text{seed}} A + e_{\text{seed}} \bmod 2$ and gets n output bits
3. **Setup MLWE:**

   3.1 $\mathcal{P}$ samples the MLWE public matrix $A' \leftarrow R_q^{k' \times k'}$ where k' is the ML-KEM parameter. $\mathcal{R}_q = \mathbb{Z}_q[x]/\left(x^{n'}+1\right)$.

   3.2 $\mathcal{P}$ and interprets these n bits as elements of $\mathbb{F}_q$. It then partitions the n bits into z blocks, each of size $2\eta(2(n' \cdot k'))$. So there are at least $\left\lfloor \frac{n}{2\eta(2\cdot(n'\cdot k'))} \right\rfloor$ blocks

   3.3. For each block in $[0, z-1]$:
   a. interpret $2\eta(2(n' \cdot k'))$ bits as elements in $\mathbb{F}_q$.
   b. $\mathcal{P}$ applies CBD sampling to obtain $2(n' \cdot k')$ CBD samples.
   c. Pack them into a ring such that $(s_i, e_i) \underset{\text{CBD}}{\in} \left(S_\eta^k \times S_\eta^k\right)$

   3.4. $\mathcal{P}$ computes $t_i = A' s_i + e_i$ for each $i \in [0, z)$.
   3.4 return MLWE challenge $t_i = A' s_i + e_i$ where public $= (A', t_i)$, secret $= (s_i, e_i)$
4. **Setup Proof:** $\mathcal{P}$ publishes $\left(A \in \mathbb{F}_2^{n \times k}, A' \in \mathcal{R}_q, t_i \in \mathcal{R}_q\right)$ and retains $(s_i, e_i)_{i \in [0,z)}, (s_{\text{seed}}, e_{\text{seed}})$ as the private witness.

   ZKP statement: Given public $\left(A \in \mathbb{F}_2^{n \times k}, A' \in \mathcal{R}_q^{k' \times k'}, (t_i)_{i \in [0,z)}\right)$, $\mathcal{P}$ wishes to prove in zero knowledge the knowledge of PRG seeds $s_{\text{seed}} \in \{0,1\}^k$, $e_{\text{seed}} \in N(n,t)$ and MLWE secrets $(s_i, e_i)_{i \in [0,z)} \in \left(S_\eta^{k'}\right)^2$ such that:

   1. **PRG correctness:** the pairs (s_i, e_i) are obtained by evaluating the low-degree PRG circuit $C : s_{\text{seed}} A + e_{\text{seed}} \bmod 2$ to produce n bits, interpreting them in $\mathbb{F}_q$, partitioning into z blocks, and applying CBD sampling to each block.
   2. **Norm bound:** s_i and e_i have small infinity norm, i.e. $(s_i, e_i) \in S_\eta^{k'} \times S_\eta^{k'}$, for all $i \in [0, z)$.
   3. **MLWE relation:** $t_i = A' s_i + e_i$ holds for all $i \in [0, z)$.

We give the protocol `VOLE.BatchMLWE` to prove this statement below:

Protocol : VOLE.BatchMLWE — Batch zero-knowledge proof for z MLWE relations

Parameters: LPN parameters k, n, t. Public LPN matrix $A \in \mathbb{F}_2^{k \times n}$. Public MLWE matrix $A' \in \mathcal{R}_q^{k' \times k'}$. Ring parameters for $\mathcal{R}_q = \mathbb{F}_{q[X]}/(X^n + 1)$. Extension degree r. Number of relations z. Access to functionality $\mathcal{F}_{\text{sVOLE}}^r$ with global key $\Delta \in \mathbb{F}_{q^r}$ held by $\mathcal{V}$.

$\mathcal{P}$'s input: Binary LWE secrets -Dense seed $s_{\text{seed}} \in \mathbb{F}_q^k$, sparse seed e_{seed}. MLWE secrets $(s_i, e_i)_{i \in [0,z)} \underset{\text{CBD}}{\in} \left(S_\eta^{k'} \times S_\eta^{k'}\right)$ such that $t_i = As_i + e_i$.

$\mathcal{V}$'s input: global key $\Delta \in \mathbb{F}_{q^r}$.

Output: $\mathcal{V}$ accepts or rejects.

Step 1: Commit to dense seed s_{seed}.
$\mathcal{P}$ and $\mathcal{V}$ call $\mathcal{F}_{\text{sVOLE}}^r(\text{svole}, k)$ to obtain k random VOLEs, then call VOLE.Derandomise to derandomize them to s_{seed}. Cost: $O(k)$, concretely $k \log q$ bits.

Step 2: Commit to sparse seed e_{seed}.
$\mathcal{P}$ commits to each block using the protocol for $\mathcal{F}_{\text{spsVOLE}}$. Cost: $t \cdot O(\log n)$, concretely $t \cdot (\log n \cdot \lambda + r \cdot \log q)$ bits.

Step 3: Range proof on dense seed.
$\mathcal{P}$ proves each committed element $[\![s_i]\!]$ of $\boldsymbol{s}_{\text{seed}}$ lies in $\{0, 1\}$ by running VOLE-ZK.Eval on $C = s_i(s_i - 1)$ for each $i \in \{1, ..., k\}$. No range proof is needed for $\boldsymbol{e}_{\text{seed}}$, since $\mathcal{F}_{\text{spsVOLE}}$ enforces that each block has exactly one nonzero entry.

Step 4: Expand seed using PRG.
Both parties locally expand their committed seed shares with A to obtain n degree-d VOLE commitments to pseudorandom bits in $\mathbb{F}_q$.

Step 5: CBD sampling.
Both parties divide the n expanded committed bits into z blocks and locally apply deterministic CBD algorithm to the degree-d VOLE commitments, obtaining committed pairs $([\![s_i]\!], [\![e_i]\!]) \underset{\text{CBD}}{\in} \left(S_\eta^k \times S_\eta^k\right)$ for each $i \in [0, z)$.

Step 6: ZK.Eval.
The parties run VOLE-ZK.Eval on the circuit $C := \left(A'[\![\boldsymbol{s_i}]\!]^d + [\![\boldsymbol{e_i}]\!]^d\right) - B\left(T^d\right)$ for all $i \in [0, z)$. $\mathcal{P}$'s circuit share is $T_{\mathcal{P}} := \boldsymbol{A'}[\![\boldsymbol{s}]\!]_{\mathcal{P}}^d + [\![\boldsymbol{e}]\!]_{\mathcal{P}}^d - B\left(\boldsymbol{t}^d\right)$ and $\mathcal{V}$'s is $T_{\mathcal{V}} := \boldsymbol{A'}[\![\boldsymbol{s}]\!]_{\mathcal{V}}^d + [\![\boldsymbol{e}]\!]_{\mathcal{V}}^d - B\left(\boldsymbol{t}^d\right)$, where $d = 11$.

Step 7: ZK.BatchVer.
$\mathcal{P}$ proves $T_{\mathcal{P}} = T_{\mathcal{V}}$ in zero knowledge via VOLE-ZK.BatchVer$\left([\![z_i]\!]^d\right)$ where $d = 11$. Cost: $O(d)$, independent of the number of VOLE correlations.

This gives a zero knowledge protocol for proving the above mentioned statement.

Figure 3 reports the communication costs across all three ML-KEM security levels. Rows 1 and 2 give the unoptimized baseline for a single MLWE relation: the difference between the two rows reflects the one-time preprocessing cost of $\mathcal{F}^r_{\text{spVOLE}}$, which Wolverine [5] reports as 1.32 bits (without setup) amortized over many correlations. For a single relation this adds between 1,438 and 2,849 bits depending on the security level, a modest overhead. The PRG approach (rows 3–6) changes the picture significantly. By batching $z \approx 1,727$ relations together for the 128-bit and 192-bit security levels and $z \approx 1,295$ for the 256-bit level, the total cost for the entire batch stays around relatively security levels, and the amortized per-relation cost drops to roughly 6,374–8,633 bits. Crucially, $\mathcal{F}^r_{\text{sVOLE}}$ preprocessing cost, which appeared as a noticeable overhead in the single-relation baseline, is almost entirely amortized away in the batch setting: rows 5 and 6 differ by only 1 bit across all security levels. This confirms that the PRG approach achieves close to a $2-3$ reduction reduction in per-relation communication compared to the unoptimized baseline, and with that the cost for $\mathcal{F}^r_{\text{spVOLE}}$ becomes negligible at batch scale.

	$\lambda = 128$ $k = 2$ $\eta_1 = 3$ $r = 11$	$\lambda = 192$ $k = 3$ $\eta_1 = 2$ $r = 17$	$\lambda = 256$ $k = 4$ $\eta_1 = 2$ $r = 22$
1. # KB for ZKP for 1 MLWE relation (unoptimized, excluding costs for $\mathcal{F}^r_{\text{sVOLE}}$)	1.59	2.37	3.15
2. # KB for ZKP for 1 MLWE relation (unoptimized, including $\mathcal{F}^r_{\text{sVOLE}}$ costs)	1.77	2.63	3.51
3. Batch size z (PRG approach)	1727	1727	1295
4. # bits for ZKP of z MLWE relations (PRG approach)	11007831	11100782	11178242
5. Amortized # bits per MLWE relation (PRG approach, excluding costs of $\mathcal{F}^r_{\text{sVOLE}}$)	6374	6428	8633
6. Amortized # bits per MLWE relation (PRG approach, including costs of $\mathcal{F}^r_{\text{sVOLE}}$)	6375	6429	8634

Figure 3: Concrete communication costs for the MLWE ZKP across all three ML-KEM security levels. Rows 1 and 2 give the unoptimized baseline cost for a single MLWE relation, excluding and including the cost of $\mathcal{F}_{\text{sVOLE}}$ respectively. Row 3 gives the batch size z used in the PRG approach. Row 4 gives the total cost for proving z MLWE relations in batch. Rows 5 and 6 give the amortized per-relation cost under the PRG approach, again excluding and including $\mathcal{F}_{\text{sVOLE}}$ costs respectively.

	$\lambda = 128$ $k = 2$ $\eta_1 = 3$ $r = 11$	$\lambda = 192$ $k = 3$ $\eta_1 = 2$ $r = 17$	$\lambda = 256$ $k = 4$ $\eta_1 = 2$ $r = 22$
1. # KB for ZKP for 1 MLWE relation (unoptimized, excluding costs for $\mathcal{F}^r_{\text{sVOLE}}$)	1.59	2.37	3.15
2. # KB for ZKP for 1 MLWE relation (unoptimized, including $\mathcal{F}^r_{\text{sVOLE}}$ costs)	1.77	2.63	3.51
3. No of batches z (PRG approach)	1758	1758	1319
4. # KB for ZKP of z MLWE relations without costs to $\mathcal{F}^r_{\text{sVOLE}}$ (PRG approach)	1377.02	1388.65	1398.35
5. Amortized # KB per MLWE relation (PRG approach, excluding costs of $\mathcal{F}^r_{\text{sVOLE}}$)	0.78	0.78	1.06
6. Amortized # KB per MLWE relation (PRG approach, including costs of $\mathcal{F}^r_{\text{sVOLE}}$)	0.78	0.79	1.06

Figure 4: Concrete communication costs for the MLWE ZKP across all three ML-KEM security levels. Rows 1 and 2 give the unoptimized baseline cost for a single MLWE relation, excluding and including the cost of $\mathcal{F}_{\text{sVOLE}}$ respectively. Row 3 gives the no of batches z proved simultaneously with the PRG approach. Row 4 gives the total cost for proving z MLWE relations in a batch. Rows 5 and 6 give the amortized per-relation cost under the PRG approach, again excluding and including $\mathcal{F}_{\text{sVOLE}}$ costs respectively.

Chapter 10.

Conclusion

This thesis constructed a VOLE-based zero-knowledge proof for the MLWE relation underlying ML-KEM (Kyber). Starting from an ideal subfield VOLE functionality, we built a homomorphic commitment scheme that is perfectly hiding and statistically binding, and showed how to lift ring elements of $\mathcal{R}_q$ into an extension field $\mathbb{F}_{q^r}$ to recover the field-invertibility needed for soundness. Using this commitment scheme together with a QuickSilver-style multiplication check and range proofs, we obtained a complete proof system for the MLWE relation.

The main bottleneck of the naive protocol is its linear communication cost: committing to the secret and error vectors requires $2(n \cdot k)$ field elements. We addressed this by exploiting the binomial structure of Kyber's secrets, committing instead to a short pseudorandom seed and expanding it locally using a sparse-LPN-based low-degree PRG, then applying central binomial sampling on the committed bits. Our concrete cost analysis shows that this commit-and-sample approach roughly halves the per-relation communication when proofs are batched, and that the one-time preprocessing cost of VOLE generation is almost entirely amortized away at batch scale.

Several directions remain open. Our analysis targets the MLWE relation rather than the full Kyber KEM, and extending the proof to cover the complete encapsulation and decapsulation procedures is natural future work. The LPN parameters were chosen following the methodology of Wolverine, but a dedicated security analysis over the small Kyber base field $\mathbb{F}_{3329}$ would strengthen the concrete guarantees. Finally, instantiating the protocol with a concrete VOLE PCG and benchmarking an implementation would confirm the practical efficiency suggested by our cost estimates.

10.1. Future Work

1. Proof for optimized CPA ML-KEM:

The CPA version of ML-KEM has 3 optimizations to reduce the ciphertext and public key size. One is to sample the public key via a seed, another is central binomial sampling, and the last one is ciphertext compression and decompression. The compression optimization gets rid of the lower order bits of the ciphertext since it's the higher order bits that have the most affect in the scaling and rounding and therefore in the correctness of decryption. The compression function is $\text{compress}_{q(x,d)} = \text{round}\Big(\Big(\frac{2^d}{q}\Big) \cdot x\Big) \bmod 2^d$. Since the witness x is scaled by a rational, and does not stay in the field, it breaks the linear homomorphism of our VOLE commitments. This would require finding an appropriate linear circuit.

2. Proof of CCA ML-KEM:

The CCA Kyber-KEM is derived by applying the Fujisaki Okamoto(FO) transform to

the CPA version. The FO transform is a generic method for converting a public- key encryption scheme that is secure against chosen-plaintext attacks to one that is secure against chosen-ciphertext attacks. To prove this, we need to prove 3 hash function in our VOLE based ZKP.

Chapter 10.1

References

[1] P. Shor, "Algorithms for quantum computation: discrete logarithms and factoring," in *Proceedings 35th Annual Symposium on Foundations of Computer Science*, 1994, pp. 124–134. doi: 10.1109/SFCS.1994.365700.

[2] M. Mosca, "Cybersecurity in an Era with Quantum Computers: Will We Be Ready?," *IEEE Security & Privacy*, vol. 16, no. 5, pp. 38–41, 2018.

[3] National Institute of Standards and Technology, "Status Report on the Third Round of the NIST Post-Quantum Cryptography Standardization Process," technical report, 2022.

[4] C. P. Schnorr, "Efficient Identification and Signatures for Smart Cards," in *Advances in Cryptology — CRYPTO' 89 Proceedings*, G. Brassard, Ed., New York, NY: Springer New York, 1990, pp. 239–252.

[5] C. Weng, K. Yang, J. Katz, and X. Wang, "Wolverine: Fast, Scalable, and Communication-Efficient Zero-Knowledge Proofs for Boolean and Arithmetic Circuits." [Online]. Available: https://eprint.iacr.org/2020/925

[6] E. Boyle, G. Couteau, N. Gilboa, and Y. Ishai, "Compressing Vector OLE." [Online]. Available: https://eprint.iacr.org/2019/273

[7] O. Goldreich, *Foundations of Cryptography*. Cambridge University Press, 2001.

[8] O. Goldreich, S. Goldwasser, and S. Micali, "How to construct random functions," *J. ACM*, vol. 33, no. 4, pp. 792–807, Aug. 1986, doi: 10.1145/6490.6503.

[9] N. Gilboa and Y. Ishai, "Distributed Point Functions and Their Applications," in *Advances in Cryptology – EUROCRYPT 2014*, P. Q. Nguyen and E. Oswald, Eds., Berlin, Heidelberg: Springer Berlin Heidelberg, 2014, pp. 640–658.

[10] D. Boneh, S. Kim, and H. Montgomery, "Private Puncturable PRFs From Standard Lattice Assumptions." [Online]. Available: https://eprint.iacr.org/2017/100

[11] R. Cramer and I. Damgård, "Zero-knowledge proofs for finite field arithmetic, or: Can zero-knowledge be for free?," in *Advances in Cryptology — CRYPTO '98*, H. Krawczyk, Ed., Berlin, Heidelberg: Springer Berlin Heidelberg, 1998, pp. 424–441.

[12] K. Yang, P. Sarkar, C. Weng, and X. Wang, "QuickSilver: Efficient and Affordable Zero-Knowledge Proofs for Circuits and Polynomials over Any Field." [Online]. Available: https://eprint.iacr.org/2021/076

[13] A. Agarwal, C. Baum, L. Braun, and P. Scholl, "Low-Bandwidth Mixed Arithmetic in VOLE-Based ZK from Low-Degree PRGs," in *Advances in Cryptology – EURO-*

CRYPT 2025, S. Fehr and P.-A. Fouque, Eds., Cham: Springer Nature Switzerland, 2025, pp. 396–426.

[14] G. Couteau, "Correlated Pseudorandomness in Secure Computation," Doctoral dissertation, 2024.

[15] P. Schoppmann, A. Gascón, L. Reichert, and M. Raykova, "Distributed Vector-OLE: Improved Constructions and Implementation." [Online]. Available: https://eprint.iacr.org/2019/1084