From zero to HEro: zkSNARKs proof construction with HE

Pritam Pal

KU LEUVEN

From zero to HEro: zkSNARKs proof construction with HE

DISSERTATION SUBMITTED IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE DEGREE OF

Master of Technology
in
Cryptology and Security

by

Pritam Pal
[Roll No: CrS2208]

under the guidance of

Prof. Dr. Nigel Smart
Prof. Dr. Bimal Kumar Roy
Emad Heydari Beni
Mariana Gama
Jiayi Kang

July 2024

CERTIFICATE

This is to certify that the dissertation entitled **"From zero to HEro: zkSNARKs proof generation with HE"** submitted by **Pritam Pal** to Indian Statistical Institute, Kolkata, in partial fulfilment for the award of the degree of **Master of Technology in Cryptology and Security** is a bonafide record of work carried out by him under my supervision and guidance. The dissertation has fulfilled all the requirements as per the regulations of this institute and, in my opinion, has reached the standard needed for submission.

Nigel Smart
Professor,
COSIC
Katholieke Universiteit Leuven,
Leuven, BELGIUM.

Bimal Kumar Roy
Professor,
Cryptology and Security Research Unit,
Indian Statistical Institute,
Kolkata-700108, INDIA.

Acknowledgments

I want to show my highest gratitude to Prof. Dr Bimal Kumar Roy, Prof. Dr. Bart Preneel and Prof. Dr. Nigel Smart for giving me the opportunity to work at COSIC, KU Leuven. I am deeply grateful for their guidance, continuous support and encouragement throughout the journey.

I would like to thank my daily supervisors, Emad Heydari Beni, Mariana Gama, and Jiayi Kang for their invaluable suggestions and discussions. They have taught me how to approach problems, think critically about different solutions, and motivated me with excellent insights and innovative ideas. Last but not least, extend my heartfelt thanks to Pela Noe for all her help during my stay in Leuven.

My deepest thanks to all the teachers of the Indian Statistical Institute, for their valuable suggestions and discussions throughout the course which added an important dimension to my research work.

Finally, I am very thankful to my parents and family for their everlasting support. Last but not least, I would like to thank all of my friends for their help and support. I thank all those I have missed from the above list.

Pritam Pal
Indian Statistical Institute
Kolkata-700108, India

Abstract

In recent times, the development of the zkSNARKs protocols opens up many applications to prove the authenticity of the data, computations and also the sender without revealing the secret data with very little communication and verification cost. However, resource-constrained devices such as security cameras, mobile phones, and sensors, do not have enough memory and computation power to generate the proof. Now, outsourcing zkSNARK-proof construction leads to privacy concerns as cloud providers may learn secret information. Different from the collaborative proof generation over distributed servers [28, 23], we discuss an approach using fully homomorphic encryption to delegate the proof construction securely to the cloud server.

Generating the proof of a circuit, we need to commit the polynomials which represent the constraints of the circuit. If the circuit contains n constraints, we apply the commitment scheme $O(n)$ times. Therefore we have focused on the KZG polynomial commitment scheme which is common in most zkSNARK protocols. Now, the approach to delegate computation of the commitment generation to the cloud server contains the precomputation of elliptic curve points which results client's high memory usage. We have presented the idea of using PIR protocols such as Vectorized BatchPIR and SimplePIR, to retrieve the precomputed points from the cloud server which reduces the user's memory usage. We have marked some difficulties we faced with the implementation and future possibilities for improvement.

Keywords. Fully Homomorphic Encryption, Pairing-friendly Elliptic Curves, zkSNARK, Private Information Retrieval.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Problem Statement

zkSNARK schemes are a zero-knowledge proof system (the prover can prove to the verifier the knowledge of a witness without revealing the witness), where the proof size and the verification time are sublinear in the circuit size. zkSNARKs are typically used in privacy-focused cryptocurrencies, dapp development, decentralized finance, and identity verification.

For the zkSNARKs schemes like Groth16 [17], Marlin [11], Plonk [13], etc. the proof size and the verification time are very feasible. So, if the verifier is a resource-constrained computer or mobile phone, it can still verify the proof. However, the computational cost and the memory overhead for proof generation are still very high for complex circuits. If one wants to run a full-fledged zkProver[4], it takes at least 1TB of RAM which is huge. Outsourcing the proof construction to the cloud server can be a good option.

Cloud computing refers to data storage, computing services, and software services. There are cloud service providers who manage all these services. Therefore we can delegate all the complex computations required for the proof generation of the zk-SNARKs with some communication cost. However, uploading all this information may cause an outbreak of sensitive information, particularly the witness of the zk-SNARK protocols, to the untrusted cloud providers. Now, let us discuss the computation on encrypted data (COED).

Fully homomorphic encryption, the "holy grail" of cryptography, enables arbitrary function computation on encrypted data. The secure delegation of computationally complex operations to the cloud server is one of the most intuitive applications of FHE. Now using FHE the client can delegate the complex operations to the server to compute over encrypted data.

Our vision is to find a sustained and feasible way for secure delegation of zkSNARKs proof construction to the server.

1.2 Motivation

The motivation for this thesis stems from the critical importance and extensive application of zkSNARK proofs. There are many emerging applications of zkSNARK proofs for IoT devices.

- In recent years the use of AI-generated images has increased for misinformation and harm. This issue can be bypassed by using zkSNARK to attest the images, taken by a specific camera [20].
- The authenticity of unauthorized presence detected by the security cameras, smart door locks etc. can be verified by sending zkSNARK proofs.
- In distributed IoT networks, it is possible to prove identity using zkSNARK protocols.

However, these are the devices with very little computation and memory resources. The main challenge using zkSNARKs proof systems is the computation cost and memory usage of proof construction. It is not feasible for these devices with minimal resources. If we can have a viable and secure delegating method then the usage of zkSNARK proof systems will be much wider.

1.3 Previous Work

Most of the zkSNARKs protocols are based on elliptic curves. Now, this is primarily because the elliptic curve-based snarks offer very small proof sizes and faster verification with the pairing property of the elliptic curves. The points in elliptic curve cryptography belong to the cyclic group and therefore every point can be constructed by scalar multiplication with a generator. However, the scalar multiplication of an elliptic curve point is one of the most costly operations for the elliptic curve-based zkSNARKs. In the paper [5], an idea to delegate the scalar multiplication of an elliptic curve point to the cloud server was presented, which uses fully homomorphic encryption and windowing techniques. Some precomputation is needed on the client's side which needs a large memory for the client.

1.3.1 Windowing Technique

Let, $k < n$ be the scalar and G be the generator of the elliptic curve group. For each doubling or addition to compute $k \cdot G$ using double and add algorithm in an

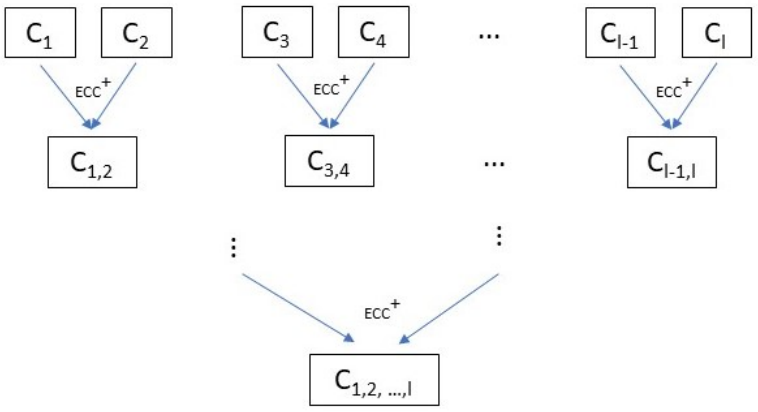

Figure 1.1: Server side computation [5]

untrusted cloud (UC) server, the multiplication depth increases. As k increases the depth is also increased for UC computation. So, a classic windowing technique was used to reduce the homomorphic multiplication depth with some precomputation on the client side.

Let, the window size ω i.e. each segment of the scalar contains ω bits and the maximum bit size of the scalar is $l_{2,n}$. Now, the client will precompute $2^\omega \cdot l_{2,n}/\omega$ elliptic curve points. The client will store the elliptic curve points of the form $P_{i,j} = \left\{\{i \cdot 2^{\omega(j-1)}\}G\right\}$ for each $j \in \{1, 2, ..., l_{2,n}/\omega\}$ and $i \in \{1, 2, ..., 2^\omega - 1\}$. Now, every scalar can be written as $k = \sum_{j=1}^{l_{2,n}/\omega} k^{(j)} \cdot 2^{\omega(j-1)}$ where $\{k^{(1)}, k^{(2)}, ..., k^{(\ell)}\}$ is the decomposition of k in base 2^ω. Then we have that

$$\begin{aligned} k \cdot G &= \left\{ \sum_{j=1}^{l_{2,n}/\omega} k^{(j)} \cdot 2^{\omega(j-1)} \right\} G \\ &= \sum_{j=1}^{l_{2,n}/\omega} \left\{ k^{(j)} \cdot 2^{\omega(j-1)} \right\} G \\ &= \sum_{j=1}^{l_{2,n}/\omega} P_{k^{(j)},j} \end{aligned}$$

So, the scalar computation only costs $l_{2,n}/\omega$ point additions. Now, the client uses fully homomorphic encryption $\mathcal{E}$ to encrypt the points and send them to UC. For the

batch size b, the client sends (CX_j, CY_j) to UC where,

$$CX_j = Enc_{\mathcal{E}}((Proj_X(P_{k_1^{(j)},j}, ..., Proj_X(P_{k_b^{(j)},j})) \tag{1.1}$$

$$CY_j = Enc_{\mathcal{E}}((Proj_Y(P_{k_1^{(j)},j}, ..., Proj_X(P_{k_b^{(j)},j})) \tag{1.2}$$

Then UC does the addition on encrypted elliptic curve points, CtCtAdd(C_{j_1}, C_{j_2}) where $C_j = (CX_j, CY_j, CZ_j), CZ_j = Enc_{\mathcal{E}}((1, ..., 1))$ as in figure 1.1 and sends back the encrypted point. Then the client will decrypt the point $C_{1,...,l}$ and get the scalar multiplication.

1.4 Our Contribution

We will be using the previous technique [5] to delegate the zkSNARKs proof generation to the UC using FHE. The construction of the zkSNARK scheme consists of a polynomial commitment scheme and a polynomial interactive oracle proof. We are focusing on delegating the KZG polynomial commitment scheme. The KZG commitment of a polynomial includes the sum of the multi-scalar multiplication(MSM) of the elliptic curve group. The MSM will be converted to the addition of several elliptic curve points. Then we will delegate the addition encrypting the elliptic curve points. The number of elliptic curve points depends on the polynomial degree and the window size.

We have faced different challenges,

1. We need to store the precomputed elliptic curve points on the client side which needs a huge memory for the client. It increases with the size e of the polynomials. Our primary challenge was to determine how to reduce the client-side memory usage for resource-constrained devices.

2. we need to choose pairing-friendly elliptic curves like BN254 curves, and BLS12-381 curves for the verification of KZG polynomial commitment which leads to a large plaintext modulus p. The Secondary challenge was to find the best parameter set that satisfies the multiplication depth and security properties we need.

The cloud storage can be a solution for storing the precomputed points. However, considering the semi-honest setup, the server can learn the secret from the client's queries. Therefore the use of PIR protocols allows the client to securely retrieve the points without disclosing the points. We have explored two PIR protocols vectorized BatchPIR and SimplePIR. We have analyzed which fits our cases the best. In our case, the database size is huge but the batch size is small.

1.5 Thesis Outline

Chapter 2 introduces the necessary background to understand and critically assess the contributions. Chapter 3 presents our proposed approach to the problem we stated. Chapter 4 introduces some PIR protocols we have explored for the protocol. Chapter 5 marks the parameter selections and our efforts in the implementations. Finally, chapter 6 summarises the contribution of the thesis and marks future possibilities to improve.

Chapter 2

Preliminaries

In this chapter, we have explained the cryptographic primitive and algorithms we need to know beforehand to go to the main approach.

2.1 Zero Knowledge Proof

Zero-knowledge proof systems are methods by which a prover can generate the proof of a statement and the verifier can check its authenticity without knowing anything about the statement. We can summarise the zero-knowledge proof in three properties,

- **Completeness.**If the prover is honest i.e. it knows the statement, then the verifier must accept the proof with high probability.
- **Knowledge Soundness.** If the prover is dishonest i.e. the claimed statement is false then the verifier must reject the proof.
- **Zero-KNowledge.** The prover convinced the verifier without revealing the statement.

The next question one might consider is what kind of statement it is. It can be the knowledge of the private key corresponds to the public key or the knowledge of 'w' where SHA256(w) is public or more generally knowledge of a witness w for which (x, w) holds an NP-hard relation or the value of a circuit for a given input.

Let us say $(x, w) \in R$ be a hard binary relation i.e. one cannot guess w with knowledge of x. Now, in a zero-knowledge proof system, there is a Setup, a Proof generation algorithm and a Verification algorithm. The goal for prover(P) is to convince verifier(V) about the knowledge of w such that (x, w) holds and on the other hand the verifier can accept or reject the prover's claim without any knowledge of w.

2.2 zkSNARK

In recent times, Zero-knowledge Succinct Non-interactive Argument of Knowledge, in short zkSNARK, took the attention of cryptographers all over the world because of its unique properties. It offers zero-knowledge proofs generation with the properties - **Succint** means the proof size (communication time) and the verification time are sublinear in the size of the circuit, **Non-Interactive** - No online interaction between the prover and the verifier to generate the proof.

One can build a zkSNARK proof scheme by mixing up 1)**Polynomial Commitment Scheme** allows generating commitment for any given polynomial $\phi(x)$ and opens at any points to prove the claimed value, 2) **Interactive Oracle Proofs(IOP)** combines the interactive proofs(IP) and probabilistically checkable proofs(PCP) i.e. the prover and the verifier engaged in multiple rounds of interaction as in IPs, in each round the prover sends an entire proof and the verifier queries to check as in PCPs.

2.3 KZG commitment Scheme

KZG [21] polynomial commitment scheme introduced by Kate, Zaverucha, and Goldberg, widely used in many zkSNARK proof systems such as Groth16, Marlin, and Plonk. For any given polynomial $\phi(x) \in \mathbb{F}_r[x]$ of any order one can create a constant size commitment of the polynomial i.e. the size of the commitment does not depend on the degree of the polynomial. Let us discuss the KZG commitment scheme.

- **Public parameters:** Let G_1, G_2, G_T be the prime r order groups and g_1, g_2, g_T be the generators of the groups. $e : G_1 \times G_2 \longrightarrow G_T$ be the bilinear pairing map which follows the properties :
  - **Bilinearity:** For $x, y \in G_1$, $z, w \in G_2$ then pairing satisfies
    $$e(xy, z) = e(x, z) \cdot e(y, z) \quad e(x, zw) = e(x, z) \cdot e(x, w)$$
  - For any $x \in G_1$ and $y \in G_2$, one can efficiently computes $e(x, y)$.
  - There exist $x \in G_1$ and $y \in G_2$ such that $e(x, y) \neq O_{G_T}$, where O_{G_T} is the identity element of G_T.
  - For $x \in G_1$, $y \in G_2$ and $a, b \in \mathbb{Z}$, then pairing satisfies
    $$e(ax, by) = ab \cdot e(x, y)$$
- **Setup:** There is one trusted server that chooses a random secret $\tau \in \mathbb{Z}_r$ and generates the global parameters(gp). gp contains $\{H_0 = g_1, H_1 = \tau \cdot g_1, H_2 = \tau^2 \cdot g_1, ..., H_M = \tau^M \cdot g_1, H'_0 = g_2, H'_1 = \tau \cdot g_2\}$. After that, the server deletes the secret τ.

- **Commit:** Let us say we want to commit the polynomial $\phi(x) \in \mathbb{F}_r[X]$ of degree t or less. For $\phi(x) = \sum_{i=0}^{\deg(\phi)} \phi_i \cdot x^i$, the commitment for the polynomial,

$$\mathcal{C}_\phi = \phi(\tau) \cdot g_1 = \left(\sum_{k=0}^{\deg(\phi)} \phi_k \tau^k \right) \cdot g_1 = \sum_{k=0}^{\deg(\phi)} \phi_k (\tau^k \cdot g_1) = \sum_{k=0}^{\deg(\phi)} \phi_k \cdot H_k \tag{2.1}$$

- **CreateWitness:** Let us say we open the value of the polynomial $\phi(x)$ at point a, $\phi(a) = b$. Also we compute the polynomial $q(x) = \frac{\phi(x)-\phi(a)}{(x-a)}$ and the commitment $\mathcal{C}_q = q(\tau) \cdot g_1$ in the same way we got for the polynomial $\phi(x)$.

- **VerifyEval:** One can verify the evaluation $\phi(a) = b$ by only knowing the commitments $\mathcal{C}_\phi, \mathcal{C}_q$ and the global parameters. If

$$e(\mathcal{C}_\phi, g_2) \stackrel{?}{=} e(\mathcal{C}_q, (H_1' - a \cdot g_2)) + \phi(a) \cdot e(g_1, g_2) \tag{2.2}$$

  satisfies then the verifier accepts, otherwise, it rejects the evaluation.

- **Correctness: VerifyEval** is correct because

$$\begin{aligned} q(x) &= \frac{\phi(x) - \phi(a)}{(x-a)} \\ q(x)(x-a) &= \phi(x) - \phi(a) \\ \phi(\tau) &= q(\tau)(\tau - a) + \phi(a) \end{aligned}$$

  Now, e is a bilinear pairing map so,

$$\begin{aligned} e(\mathcal{C}_\phi, g_2) &= \phi(\tau) \cdot e(g_1, g_2) \\ &= \{q(\tau)(\tau - a) + \phi(a)\} \cdot e(g_1, g_2) \\ &= e(q(\tau) \cdot g_1, (\tau - a) \cdot g_2) + \phi(a) \cdot e(g_1, g_2) \\ &= e(\mathcal{C}_q, (H_1' - a \cdot g_2)) + \phi(a) e(g_1, g_2) \end{aligned}$$

2.4 Pairing Friendly Curves

The pairing-friendly elliptic curves, $E(\mathbb{F}_p)$ are the one which allows efficient computation of a bilinear pairing function i.e. a bilinear map $e : G_1 \times G_2 \longrightarrow G_T$ is defined where G_1, G_2 are the subgroups of order r where $r | p^k - 1$ of the group of elliptic curve points and G_T is a group resides in $\mathbb{F}_{p^k}^\times$, k is the embedding degree of the curve. Pairing-friendly elliptic curves are used in identity-based cryptography, zkSNARK protocols, blockchain and cryptocurrencies etc.

BN254 curve

The BN254 curve [31] is defined by the equation,

$$Y^2 = X^3 + 3$$

over the field $\mathbb{F}_p$ with prime p of size 254 bits. The constants p, r, t are defined as

$$\begin{aligned} p &= 36x^4 + 36x^3 + 24x^2 + 6x + 1 \\ r &= 36x^4 + 36x^3 + 18x^2 + 6x + 1 \\ t &= 6x^2 + 1 \end{aligned}$$

where x is chosen as per the implementation so that r is a prime and t is the trace of the Frobenius of the curve. The prime r is also of size 254 bits. The parameter x is set to $x = 4965661367192848881$ for BN254 [29]. The embedding degree of the BN254 curve is 12 which implies that the smallest value of k is 12 so that $E(\mathbb{F}_{p^k})[r] = E(\bar{\mathbb{F}_p})[r]$. An optimal ate pairing $e : G_1 \times G_2 \longrightarrow G_T$ is defined over the BN254 curve where $G_1 = E(\mathbb{F}_p)$ and $G_T \subset \mathbb{F}_{p^{12}}^{\times}$ the rth root unity. BN254 curve provides security of 128 bits (though it has dropped to 100 bits with the recent attacks) and supports the efficient implementation of ate pairing, making it a practical choice for most of the zkSNARK protocols.

BLS12-381 curve

The equation defines the BLS12-381 [12] curve,

$$E : Y^2 = X^3 + 4$$

The 381 is the number of bits required to represent the coordinates of the BLS curve and the embedding degree of the curve is 12. Assume another prime r such that $r|p^{12} - 1$, r is of size 255 bits. The two groups, used to define the bilinear map, are

$$\begin{aligned} &G_1 \subset E(\mathbb{F}_p), \text{ where } E : Y^2 = X^3 + 4 \\ &G_2 \subset E'(\mathbb{F}_{p^2}), \text{ where } E : Y^2 = X^3 + (4 + i) \end{aligned}$$

where G_1 and G_2 both are groups of order r. The pairing, $e : G_1 \times G_2 \longrightarrow G_T$ defined over the curve simply takes a point $P \in G_1$, a point $Q \in G_2$ and outputs $e(P, Q) \in G_T$ where G_T resides in $\mathbb{F}_{p^{12}}$. BLS12-381 curve offers security of 128 bits. Therefore, its popularity has increased amongst cryptographers for use in BLS signature, Zcash, Ethereum etc.

2.5 Homogeneous Projective embedding

Let E be an elliptic curve defined by the equation $y^2 = x^3 + ax + b$. Now, introducing one extra variable Z we get $Y^2Z = X^3 + aXZ^2 + bZ^3$ which is the equation of the

elliptic curve in the homogeneous projective plane of the elliptic curve E.

$$Y^2Z = X^3 + aXZ^2 + bZ^3 \implies \left(\frac{Y}{Z}\right)^2 = \left(\frac{X}{Z}\right)^3 + a\left(\frac{X}{Z}\right) + b \implies Y'^2 = X'^3 + aX' + b$$

where (X, Y, Z) is the projective coordinates of the elliptic curve on the projective plane. Therefore $\left(\frac{X}{Z}, \frac{Y}{Z}\right)$ is the corresponding Cartesian coordinate of (X, Y, Z). Similarly, for the Cartesian coordinate (x, y), the corresponding coordinate of the elliptic curve in the homogeneous projective plane is $(x, y, 1)$.

2.6 Addition Law of elliptic curve points

Let $E(\mathbb{F}_p)$ be an elliptic curve and $P_1(X_1, Y_1, Z_1)$, $P_2(X_2, Y_2, Z_2)$ be two points in projective embedding of $E(\mathbb{F}_p) : y^2 = x^3 + b$. The addition of P_1 and P_2 can be computed in multiplication depth 2 by *Renes et al.* addition law [30].

Algorithm 1 (Addition Law)

Input: $P_1(X_1, Y_1, Z_1)$, $P_2(X_2, Y_2, Z_2)$
Output: $\mathrm{ADD}(P_1, P_2) = P_3(X_3, Y_3, Z_3)$

$1.t_0 \longleftarrow X_1 \cdot X_2$	$2.t_1 \longleftarrow Y_1 \cdot Y_2$	$3.t_2 \longleftarrow Z_1 \cdot Z_2$
$4.t_3 \longleftarrow X_1 + Y_1$	$5.t_4 \longleftarrow X_2 + Y_2$	$6.t_3 \longleftarrow t_3 \cdot t_4$
$7.t_4 \longleftarrow t_0 + t_1$	$8.t_3 \longleftarrow t_3 - t_4$	$9.t_4 \longleftarrow Y_1 + Z_1$
$10.X_3 \longleftarrow Y_2 + Z_2$	$11.t_4 \longleftarrow t_4 \cdot X_3$	$12.X_3 \longleftarrow t_1 + t_2$
$13.t_4 \longleftarrow t_4 - X_3$	$14.X_3 \longleftarrow X_1 + Z_1$	$15.Y_3 \longleftarrow X_2 + Z_2$
$16.X_3 \longleftarrow X_3 \cdot Y_3$	$17.Y_3 \longleftarrow t_0 + t_2$	$18.Y_3 \longleftarrow X_3 - Y_3$
$19.X_3 \longleftarrow t_0 + t_0$	$20.t_0 \longleftarrow X_3 + t_0$	$21.t_2 \longleftarrow b_3 \cdot t_2$
$22.Z_3 \longleftarrow t_1 + t_2$	$23.t_1 \longleftarrow t_1 - t_2$	$24.Y_3 \longleftarrow b_3 \cdot Y_3$
$25.X_3 \longleftarrow t_4 \cdot Y_3$	$26.t_2 \longleftarrow t_3 \cdot t_1$	$27.X_3 \longleftarrow t_2 - X_3$
$28.Y_3 \longleftarrow Y_3 \cdot t_0$	$29.t_1 \longleftarrow t_1 \cdot Z_3$	$30.Y_3 \longleftarrow t_1 + Y_3$
$31.t_0 \longleftarrow t_0 \cdot t_3$	$32.Z_3 \longleftarrow Z_3 \cdot t_4$	$33.Z_3 \longleftarrow Z_3 + t_0$
return X_3, Y_3, Z_3		

The cost for adding two elliptic curve points, $12\mathbf{M} + 2\mathbf{m}_b + 29\mathbf{a}$ where, $\mathbf{M}$ is the cost of the multiplication of two field elements, $\mathbf{m}_b$ is the cost of the multiplication with constant b, $\mathbf{a}$ is the cost of the addition of field elements.

2.7 Doubling of elliptic curve points

Again, $P(X, Y, Z)$ be a point in projective embedding of $E(\mathbb{F}_p) : y^2 = x^3 + b$. Now, doubling algorithm of the elliptic curve points [30],

Algorithm 2 (Doubling)

Input: $P(X, Y, Z)$
Output: DBL(P) $= Q(X', Y', Z') = 2P(X, Y, Z)$

$1.t_0 \longleftarrow Y \cdot Y$	$2.Z' \longleftarrow t_0 + t_0$	$3.Z' \longleftarrow Z' + Z'$
$4.Z' \longleftarrow Z' + Z'$	$5.t_1 \longleftarrow Y \cdot Z$	$6.t_2 \longleftarrow Z \cdot Z$
$7.t_2 \longleftarrow b_3 \cdot t_2$	$8.X' \longleftarrow t_2 \cdot Z'$	$9.Y' \longleftarrow t_0 + t_2$
$10.Z' \longleftarrow t_1 \cdot Z'$	$11.t_1 \longleftarrow t_2 \cdot t_2$	$12.t_2 \longleftarrow t_1 + t_2$
$13.t_0 \longleftarrow t_0 - X_2$	$14.Y' \longleftarrow t_0 \cdot Y'$	$15.Y' \longleftarrow X' + Y'$
$16.t_1 \longleftarrow X \cdot Y$	$17.X' \longleftarrow t_0 \cdot t_1$	$18.X' \longleftarrow X' + X'$

return X', Y', Z'

The cost of doubling an elliptic curve points $6\mathbf{M} + 2\mathbf{S} + \mathbf{m}_b + 9\mathbf{a}^*$.

2.8 Double and Add algorithm

Algorithm 3 (Double and Add)

Input: The binary decomposition of ϕ_k where, bits$(\phi_k) = l_{2,\phi_k}$, The elliptic curve point H_k
Output: $\phi_k \cdot H_k$

```
result ⟵ 𝒪,
P ⟵ H_k,
for each bit in φ_k do
    if bit == 1 then
        result = ADD(result, P),
    end if
    P = DBL(P)
end for
return result
```

*S is the cost of squaring an field element

So, in total $O(\ell_{2,\phi_k})$ many Addition and ℓ_{2,ϕ_k} many doubling of elliptic curve points consists in Double and Add algorithm.

2.9 Fully Homomorphic Encryption

What will be more fascinating than doing whatever processing we need over the secret data without decrypting it? Fully Homomorphic Encryption, introduced by C. Gentry [15], opens up a whole new world of technologies. FHE ensures data privacy and security throughout the whole computation process. Consider that $\mathcal{E}$ be the FHE scheme and $\{C_1, C_2, ..., C_k\}$ be the encryption of the data $\{m_1, m_2, ..., m_k\}$. Most of the data breaches happen when the data is decrypted for processing. Let $f(x_1, x_2, ..., x_k)$ be the circuit of evaluation, and **evk** be the evaluation key.

$$\mathbf{Dec}_{\mathcal{E}}\left(\mathbf{Eval}_{\mathcal{E}}(C_1, C_2, ..., C_k, f, \mathbf{evk})\right) = \mathbf{Dec}_{\mathcal{E}}\left(f(C_1, C_2, ..., C_k)\right) = f(m_1, m_2, ..., m_k)$$

FHE promises a new and efficient direction to how secret data is handled in industries like cloud computing, secure data analysis, and privacy-preserving machine learning.

2.10 BFV

The following set of algorithms describes a variant of the BFV scheme [22, 10].

Parameters. Let $R = \frac{\mathbb{Z}[X]}{(X^n+1)}$ be a cyclotomic ring of integers of degree n. The degree of the polynomial n and the ciphertext modulus q are chosen according to the required security level. The ciphertext space is R_q^2 and the plaintext space is R_t where $R_q = \mathbb{Z}[X]/(q, X^n + 1), R_t = \mathbb{Z}[X]/(t, X^n + 1)$. t is chosen much smaller than q. $\Delta = \lfloor q/t \rfloor$ is the scaling parameter. Let χ be the discrete Gaussian distribution with the standard deviation σ.

Key Generation. The secret $s \longleftarrow \chi$ as a polynomial with ternary coefficient is used to create the public key, $(\mathbf{pk}_0, \mathbf{pk}_1) = ([-a \cdot s + e]_q, a)$ where $a \xleftarrow{\$} R_q, e \longleftarrow \chi$. Therefore the key generation outputs the secret key and corresponding public key.

Evaluation Key Generation. For sample $i = 0, ..., \ell$ where $\ell = \lceil \log(q, w) \rceil$ generates $a_i \xleftarrow{\$} R_q, e_i \longleftarrow \chi$. The evaluation keys

$$\mathbf{evk} = \left([-a_i \cdot s + e_i + w^i s^2]_q, a_i\right) \in R_q^2$$

Encryption. Let the message be $\mathbf{pt} \in R_t$. We generate

$$b' = [\mathbf{pk}_0 \cdot u + e_1]_q, a' = [\mathbf{pk}_1 \cdot u + e_2]_q$$

where, $u \longleftarrow \chi, e_1, e_2 \longleftarrow \chi$ are used to the re-randomise the encryption. We will use $(b', a') \in R_q^2$ to encrypt the plaintext $\mathbf{pt}$,

$$\mathbf{ct} = ([\Delta \cdot \mathbf{pt} + b']_q, a')$$

Decryption. Consider the ciphertext $\mathbf{ct} = (c_0, c_1)$ and the secret s. Compute

$$[c_0 + c_1 \cdot s]_q = [\Delta \cdot \mathbf{pt} + b' + a's]_q \equiv [\Delta \cdot \mathbf{pt} + e']_q$$

where $e' = eu + e_1 + e_2 s$ is the noise of the ciphertext. Now, the size of the noise must be small enough to have the right decryption. The decryption of $\mathbf{ct}$ is

$$\mathbf{pt} = \left[\left\lfloor \frac{t}{q}[c_0 + c_1 \cdot s]_q \right\rceil\right]_t \in R_t$$

CtCtAdd. Let we aim to add the ciphertexts $\mathbf{ct}_0$ and $\mathbf{ct}_1$. The addition is straightforward,

$$\mathbf{ct_{Add}} = (\mathbf{ct}_0[0] + \mathbf{ct}_1[0], \mathbf{ct}_0[1] + \mathbf{ct}_1[1])$$

The noise growth for CtCtAdd is linear i.e. twice the noise of the ciphertexts.

CtCtMul. The multiplication is not as straightforward as the addition. Let us define the multiplication of the ciphertexts $\mathbf{ct}_0 = (c_0, c_1)$ and $\mathbf{ct}_1 = (d_0, d_1)$,

$$c_0' = \left[\left\lfloor \frac{t}{q} c_0 d_0 \right\rceil\right]_q, c_1' = \left[\left\lfloor \frac{t}{q}(c_0 d_1 + c_1 d_0) \right\rceil\right]_q, c_2' = \left[\left\lfloor \frac{t}{q} c_1 d_1 \right\rceil\right]_q$$

Now, the ciphertext dimension increases to 3. We will relinearize the ciphertext using the evaluation key. Expressing c_2' in base w we get, $c_2' = \sum_{i=0}^{\ell} c_2'^{(i)} w^i$. Now, compute

$$c_0 = c_0' + \sum_{i=0}^{\ell} evk[i][0] c_2'^{(i)}, c_1 = c_1' + \sum_{i=0}^{\ell} evk[i][1] c_2'^{(i)}$$

and outputs $(c_0, c_1) \in R_q^2$.

2.11 SIMD batching

Single Instruction Multiple Data, in short SIMD, allows encrypting a vector of messages and applying homomorphic operations on them in parallel. The main idea is to factorize $x^n + 1$ modulo p and express the plaintext space as the product of smaller

fields, then encrypt a vector of elements and operate them with homomorphic operations in parallel. Let consider

$$X^n + 1 = f_1(X) \cdot f_2(X) \cdots f_k(X) \pmod{p}$$

Then the plaintext space is isomorphic to the product of k smaller fields,

$$\mathbb{Z}_p[X]/(X^n + 1) = \prod_{i=1}^{k} \mathbb{Z}_p[X]/f_i(X) \pmod{p}$$

i.e. instead of encrypting one large polynomial from the plaintext space $\mathbb{Z}_p[X]/(X^n + 1)$, using SIMD, we can encrypt and operate over a vector of elements in parallel. Therefore we can compute a circuit over a batch of k encrypted elements.

2.12 Parameter Selection for BFV

Consider the computation circuit containing A levels of additions and one multiplication level, iterated D times. Then the multiplication depth of the circuit is D. The choice for the standard deviation of χ is $\sigma = 3.2$ [9]. The heuristic estimation of noise after computation of the circuit [10],

$$(14tn2^A)^D \frac{42\sigma tn}{q} \tag{2.3}$$

For the correctness of the decryption after circuit evaluation, the noise must be less than $1/2$. Then we will use the heuristic depth estimate,

$$D \leq \left\lfloor \frac{\log q - \log(84\sigma tn)}{\log(14tn) + A} \right\rfloor \tag{2.4}$$

The plaintexts are chosen with $|\mathbf{pt}|_\infty \leq t/2$. As in equation 2.3 and equation 2.4, for the applications with large plaintext modulus t, the noise grows faster and supports less multiplication depth. Then we must select a large ciphertext modulus. We can choose the plaintext polynomial degree n and the ciphertext modulus q by employing lattice-estimator [†].

[†]Lattice-Estimator [7], https://github.com/malb/lattice-estimator, is a software tool which provides security measurements of given LWE instances

Chapter 3

Delegation of KZG commitment

The chapter contains our approach for delegating the KZG commitment generation to the cloud server by FHE. We have conducted a comparative analysis of the number of elliptic curve additions required by the client in our proposed approach versus the traditional approach.

3.1 Method

Parameters: Let consider the pairing-friendly elliptic curve group $E(\mathbb{F}_p)$ for the KZG commitment scheme defined over the field $\mathbb{F}_p$. Let us assume another prime r satisfying $r|p^k - 1$ where k is the embedding degree of the curve E. Now, G is the generator of the field $\mathbb{F}_r$ and $\phi(x) \in \mathbb{F}_r[X]$ is the polynomial with the degree t or less. In the section 2.4 we have discussed the BN254 curve and BLS12-381 curve. Let us consider the BN254 curve for the rest of the thesis. For the BN254 curve, the prime p and r are of size 254 bits i.e. $\ell_{2,p} = \ell_{2,r}$.

Setup: Similarly, there is one trusted server that chooses a random secret $\tau \in \mathbb{Z}_r$. Then it generates the global parameters from the generator G of the elliptic curve group. So, in this case, the global parameters are a set of elliptic curve points from the base curve. Then the trusted server deletes the secret τ which is not needed for the later computations.

Trusted Server	$\tau \in \mathbb{Z}_{\mathrm{p}}$ $\downarrow$ $gp \longrightarrow \{H_0 = G, H_1 = \tau \cdot G, H_2 = \tau^2 \cdot G, ..., H_t = \tau^t \cdot G\}$ $\downarrow$ Deletes τ

Commitment: The prover generates the commitment for any given polynomial $\phi(x) \in \mathbb{F}_r[X]$ using the global parameters. The commitment for $\mathcal{C}_\phi = \sum_{k=0}^{\deg(\phi)} \phi_k \cdot H_k$.

As we can see the commitment is nothing but the addition of some scalar(ϕ_k) multiple of elliptic curve points, H_k. Here we will be using the windowing to reduce the computation of the scalar multiplications, $\phi_k \cdot H_k$ in UC.

Windowing: As we have explained the windowing technique 1.3.1, let decompose each coefficients ϕ_k in ω bits of windows. Since, $\phi_k \in \mathbb{F}_p$ so, $\ell_{2,\phi_k} \leq \ell_{2,p}$. Let $\{\phi_k^{(1)}, ..., \phi_k^{(\ell)}\}$ be the decomposition of ϕ_k with base 2^ω where $\ell = \ell_{2,p}/\omega$ i.e. $\phi_k = \sum_{j=1}^{\ell} \phi_k^{(j)} \cdot 2^{\omega(j-1)}$. Then we have that $\phi_k \cdot H_k$ is the addition of ℓ many elliptic curve points.

$$\begin{aligned}\phi_k \cdot H_k &= \left\{ \sum_{j=1}^{\ell_{2,p}/\omega} \phi_k^{(j)} \cdot 2^{\omega(j-1)} \right\} H_k \\ &= \sum_{j=1}^{\ell_{2,p}/\omega} \{\phi_k^{(j)} \cdot 2^{\omega(j-1)}\} \cdot H_k\end{aligned}$$

So, the KZG commitment $\mathcal{C}_\phi = \sum_{k=0}^{\deg(\phi)} \phi_k \cdot H_k$, is the sum of $\ell \cdot (t+1)$ many elliptic curve points,

$$\begin{array}{cccc} \phi_0^{(1)} H_0 & \phi_1^{(1)} H_1 & \cdots & \phi_t^{(1)} H_t \\ \phi_0^{(2)} 2^\omega H_0 & \phi_1^{(2)} 2^\omega H_1 & \cdots & \phi_t^{(2)} 2^\omega H_t \\ \vdots & \vdots & \cdots & \vdots \\ \phi_0^{(\ell)} 2^{\omega(\ell-1)} H_0 & \phi_1^{(\ell)} 2^{\omega(\ell-1)} H_1 & \cdots & \phi_t^{(\ell)} 2^{\omega(\ell-1)} H_t \end{array}$$

Precomputation: Consider that the client has the precomputed elliptic curve points $P_{i,j}^k = \left\{\{i \cdot 2^{\omega(j-1)}\} H_k\right\}$ for each $j \in \{1, 2, ..., \ell_{2,p}/\omega\}$, $i \in \{1, 2, ..., 2^\omega - 1\}$ and $k \in \{0, 1, ..., t\}$. Then the commitment of the polynomial $\phi(x)$ is the sum of $\ell \cdot \deg(\phi)$ many elliptic curve points. But the client needs to store $(2^\omega \cdot \ell_{2,p}/\omega \cdot t)$ elliptic curve points.

Delegation: Let $\mathcal{E}$ be the FHE encryption scheme and b be the SIMD batch size. Since, the KZG commitment is the sum of the elliptic curve points, $P^k_{\phi_k^{(j)},j}$ for $j \in \{1, 2, ..., \ell_{2,p}/\omega\}$ and $k \in \{0, 1, ..., t\}$, one can encrypt the points using $\mathcal{E}$ scheme and

sends the server $C_j = (CX_j, CY_j)$ where

$$CX_j = Enc_{\mathcal{E}}((Proj_X(P^0_{\phi_0^{(j)},j}, ..., Proj_X(P^b_{\phi_b^{(j)},j})) \tag{3.1}$$

$$CY_j = Enc_{\mathcal{E}}((Proj_Y(P^0_{\phi_0^{(j)},j}, ..., Proj_X(P^b_{\phi_b^{(j)},j})) \tag{3.2}$$

Now the server does further additions by the addition law 2.6 over the encrypted points as shown in Figure 1.1.

3.2 Memory trade-off and Mult. depth

Let the degree of the polynomial $\phi(x)$ be $t = 1024$ or less. The maximum number of scalar multiplication of the elliptic curve group is $t = 1024$. Again for the BN254 curve the $\phi(x) \in \mathbb{F}_r[X]$ where r is of 254-bit size. As mentioned in section 2.6, the multiplication depth of addition of two elliptic curve points is $d_{\text{add}} = 2$. Now, The server will compute the addition over the encrypted elliptic curve points (C_{j_1}, C_{j_2}) as shown in figure 1.1. The server receives $\ell_{2,p}/\omega$ number of elliptic curve points to add. Therefore, the multiplication depth of the circuit 1.1 is $d = \log_2(\ell_{2,p}/\omega)$ to calculate $\phi_k \cdot H_k$ i.e. the FHE scheme should support d times the depth d_{add}. The server will return the scalar multiplications to the client. The client still has to perform $t = 1024$ additions. If the client wants to delegate all the computation, then the depth of the circuit is $\log_2(t \cdot \ell_{2,p}/\omega)$ i.e. the FHE scheme should support $\log_2(t \cdot \ell_{2,p}/\omega) = d + \log_2(t)$ times the depth d_{add}.

The following table shows the impact of using the windowing technique over the multiplication depth and the client's memory trade-off.

Window Size	Required depth for the scalar mult.	Required depth for total computation	Memory needed for $\phi_k H_k$	Total outsourced Mem. for client
$\omega = 8$	$5 \cdot d_{add}$	$(d + \log_2(t))d_{add} = 15 \cdot d_{add}$	512 KB	512KB $\times(t+1)$
$\omega = 16$	$4 \cdot d_{add}$	$14 \cdot d_{add}$	64MB	64MB $\times(t+1)$
$\omega = 32$	$3 \cdot d_{add}$	$13 \cdot d_{add}$	2TB	2 TB $\times(t+1)$

Table 3.1: Memory Trade-off and Mult. Depth

3.3 Difficulties

We can have some observations from table 3.1.

1. As we increase the window size the multiplication depth decreases by only 2. The multiplication depth of the circuit 1.1 for computing $\phi_k \cdot H_k$ is achievable. However, it is still very high to achieve for the FHE schemes when the server adds all the elliptic curve points.

2. For large window size the total outsourced memory trade-off for the client is huge. Now this huge memory trade-off is not realistic for the resource-constrained devices. Therefore, we must first find a solution for the client's memory usage.

3. The base fields for the polynomials are pairing-friendly elliptic curves. In this thesis, we have considered the BN254 curve. Therefore the plaintext modulus will be of size $\ell_{2,p} = 254$ bits. Achieving 128-bit security and sufficient computational depth forces a large ciphertext modulus and high plaintext dimension, which is impractical. Since, the client performs almost $\ell_{2,p}/\omega$ encryptions, the computation overhead for the client will be high. Again the client sends $\ell_{2,p}/\omega$ ciphertexts to the server i.e. for large ciphertext modulus the communication overhead will also be very high. We will describe the parameters in chapter 5.

3.4 Outsourcing Precomputation

At the pre-computation stage, the client needs to compute $2^\omega \cdot \ell_{2,p}/\omega \cdot t$ number of elliptic curve points and store the points. A simple solution for clients would be to outsource the computation of the points to the cloud server and use the cloud storage to store the points. Then whenever and whatever points the client needs, it can be retrieved from the cloud but in exchange for some communication costs.

Again there arrives the question what if the cloud server is not trustworthy? The client must send the index or the value (i, j, k)* to retrieve the required points. However, it leads to the cloud learning the coefficients of the polynomial i.e. security breach.

Let us discuss private information retrieval. PIR allows the client to send the indices or the values (i, j, k) to the server and the server gives back the corresponding elliptic curve points but does not learn anything about the indices or the values (i, j, k). Now, more than one client can retrieve the required elliptic curve points from the cloud storage using PIR.

Let the window size be $\omega = 32$, plaintext polynomial degree for the FHE scheme $n = 2^{16}$, the plaintext modulus p is of size 254 bits, the maximum degree of the

*elliptic curve points, $P^k_{i,j} = \{i \cdot 2^{\omega(j-1)}\}H_k : i \in \{1, 2, ..., 2^\omega - 1\}, j \in \{1, 2, ..., \ell\}, k \in \{0, 1, ..., t\}$

polynomial $t = 1024$. Then the decomposition of each ϕ_k, $\{\phi_k^{(1)}, \phi_k^{(2)}, ..., \phi_k^{(\ell)}\}$ consists of 32 bit entries. The client has

$$t \cdot \ell = t \cdot \ell_{2,p}/\omega = 2^{10} \cdot \frac{254}{32} \approx 2^{13}$$

indices to query the server. The following table 3.2 shows how the number of queries changes with the window size and degree of the polynomials.

	$t = 1024$	$t = 8192$
$\omega = 8$	2^{15}	2^{18}
$\omega = 16$	2^{14}	2^{17}
$\omega = 32$	2^{13}	2^{16}

Table 3.2: Number of queries

Server stores $2^{\omega} \cdot \ell_{2,p}/\omega \cdot t$ number of elliptic curve points of the form $\big\{\{i \cdot 2^{\omega(j-1)}\}H_k\big\}$. So, each database entry will be of size $254 \times 2 = 508$ bits. The following table 3.3 shows the size of the PIR database. We have looked into Vectorized BatchPIR [26] and

	$t = 1024$	$t = 8192$
$\omega = 8$	2^{23}	2^{26}
$\omega = 16$	2^{30}	2^{33}
$\omega = 32$	2^{45}	2^{48}

Table 3.3: Database Size

SimplePIR [19] to find which will work better for our setup, explained in chapter 4.

3.5 Comparative Analysis

The KZG commitment generation can be thought of as Multi Scalar Multiplication(MSM) of the elements of the elliptic curve group, $E(\mathbb{F}_p)$. There are different existing methods for optimizing the multi-scalar multiplication like the Straus method, and the Pippenger's bucket method, see section 2 of [24]. Our approach to commitment construction is more similar to the Straus method. Here is a comparison of the number of elliptic curve additions needed for multi-scalar multiplication using Pippenger's method and our approach.

As in 2.1, the commitment generation is the addition of t scalar multiplication of elliptic curve points. Now, in Pippenger's method, the large scalars are divided into small segments of length ω. There are many variants of Pippenger's bucket method. Here we have described one such variant. In this variant, the precomputed points

$\{2^{\omega(j-1)}H_k | j = 1, 2, ..., \ell_{2,p}/\omega, k \leq t\}$ i.e. $(t+1) \cdot \ell_{2,p}/\omega$ elliptic points are stored in outsourced memory. Again we want to compute the multi-scalar multiplication

$$\sum_{j=1}^{\ell_{2,p}/\omega} \left\{ \{\phi_0^{(j)} 2^{\omega(j-1)}\} H_0 + \{\phi_1^{(j)} 2^{\omega(j-1)}\} H_1 + \cdots + \{\phi_t^{(j)} 2^{\omega(j-1)}\} H_t \right\} \tag{3.3}$$

where $2^{\omega(j-1)}H_k, k \leq t$ points are precomputed. At first, these points $\{2^{\omega(j-1)}H_k | j = 1, 2, ..., \ell_{2,p}/\omega, k \leq t\}$ are sorted into $2^\omega - 1$ buckets concerning the scalars $\phi_k^{(j)}$ and the intermediate sums $S_i, i = 1, 2, ..., 2^\omega - 1$ are computed. The total number of elliptic curve additions required for this is $(t+1)\ell_{2,p}/\omega - (2^\omega - 1) = (t+1) \cdot \ell - (2^\omega - 1)$. Now, the summation 3.3 becomes $\sum_{i=1}^{2^\omega - 1} i \cdot S_i$ which computes by the algorithm 4 with $2(2^\omega - 2)$ elliptic curve additions. Therefore total elliptic curve additions are $(t+1) \cdot \ell - (2^\omega - 1) + 2(2^\omega - 2) \approx ((t+1)\ell + 2^\omega)$. ω can be chosen smaller accordingly to reduce the computation.

Algorithm 4 (Subsum accumulation algorithm)

Input: $S_1, S_2, S_3, ..., S_m$
Output: $1S_1 + 2S_2 + \cdots + mS_m$

```
tmp = 0
tmp1 = 0
for i = m to 1 do
    tmp = tmp + S_i
    tmp1 = tmp1 + tmp
end for
return tmp1
```

The following table 3.4 shows the difference between the two approaches.

	Outsourced Storage	No. of local elliptic curve addition
Pippenger's Method	$\ell_{2,p}/\omega \cdot (t+1)$	$((t+1)\ell_{2,p}/\omega + 2^\omega)$
Our Approach	$2^\omega \cdot \ell_{2,p}/\omega \cdot (t+1)$	$(t+1) \cdot \frac{\ell_{2,p}}{\omega} \Big/ 2^{d/2}$

Table 3.4: Comparison of two approaches

In our approach, the precomputation is significantly extensive. The precomputed elliptic curve points are

$$\left\{ \{i \cdot 2^{\omega(j-1)}\} \cdot H_k | i = 1, ..., 2^\omega - 1, j = 1, 2, ..., \ell_{2,p}/\omega, k \leq t \right\}$$

i.e. $2^{\omega} \cdot \ell_{2,p}/\omega \cdot (t+1)$ elliptic curve points. The way we see commitment generation involves

$$\left\{\sum_{j=1}^{\ell} \phi_0^{(j)} 2^{\omega(j-1)} H_0\right\} + \left\{\sum_{j=1}^{\ell} \phi_1^{(j)} 2^{\omega(j-1)} H_1\right\} \cdots + \left\{\sum_{j=1}^{\ell} \phi_t^{(j)} 2^{\omega(j-1)} H_t\right\} \tag{3.4}$$

In our protocol, the client delegates the computation of the scalar multiplication, $\phi_k \cdot H_k = \left\{\sum_{j=1}^{\ell} \phi_k^{(j)} 2^{\omega(j-1)} H_k\right\}$ to the server. Therefore, the server does $(t+1) \cdot \ell_{2,p}/\omega$ elliptic curve additions and returns the ciphertext containing the scalar multiplications $\phi_k \cdot H_k$ for $k \leq t$. Now, the client must do t elliptic curve additions locally. According to the depth supported by the FHE scheme $\mathcal{E}$ the client can delegate more computation to the server. Let us consider $\mathcal{E}$ is capable of supporting multiplicative depth d. Then the client can send $2^{d/2}$ ciphertexts to the server. Now, the client needs to do $(t+1) \cdot \frac{\ell_{2,p}}{\omega} \Big/ 2^{d/2}$ additions locally.

On the other hand in Pippenger's bucket method, because of the structure of the algorithm, we cannot delegate the computation. But again we can use PIR to retrieve the precomputed elliptic curve points.

Chapter 4

Private Information Retrieval

This chapter will discuss the PIR protocols we have explored for our setups. The primary protocols we have analysed are the Vectorized BatchPIR [26] and SimplePIR [19]. Let N be the total number of elements in the database(**DB**). We have provided the different values of N in table 3.3 and the number of queries the client needs to do in table 3.2. We will briefly introduce the protocols and their communication and computation costs.

4.1 Private Information Retrieval

Given an index i and a database with N entries, the client wants to retrieve the i-th entry DB_i. A PIR protocol should satisfy the following,

1. **Privacy of Client:** The server learns nothing about the index the client is requesting.

2. **Correctness:** The client and server follow the protocol correctly, and the client can download the requested entry successfully.

4.2 BatchPIR

In a BatchPIR, instead of a single index, the client wants to download a batch of entries corresponding to the indices $\{i_1, i_2, ..., i_n\}$ i.e. the output of the BatchPIR looks like $\{DB_1, DB_2, ..., DB_n\}$. Like PIR schemes, BatchPIRs should satisfy,

1. **Privacy of client:** The server learns nothing about the batch indices, the client is requesting.

2. **Correctness:** The client and server follow the protocol correctly, and the client can download the requested batch successfully.

4.3 Vectorized BatchPIR

The Vectorized BatchPIR [26] is more efficient from both communication and computation perspective. Especially for small entry sizes($\leq$ 512B), it outperforms the previous BatchPIRs. In our case, the entries are the elliptic curve points (x, y). For the BN254 curve, the entries are nearly 64 bytes in size. However, the database size is very large which affects the communication and computation overhead. It uses RLWE homomorphic encryption, $\mathcal{E}$.

4.3.1 Protocol Overview

Public Parameters

- b, the batch size
- B, Number of buckets, is set to $1.5b$
- $h_1, h_2, ..., h_w$ the hash functions, usually $w = 3$
- d, the dimension of the encoded database, is set to 3
- N_B, the size of the large bucket
- N'_B, size of the dimension, $N'_B = \sqrt[d]{N_B}$
- n, the plaintext dimension for the RLWE instances and q is the ciphertext modulus
- $g_B = n/N'_B$

Setup In the one-time setup stage the server hashes the database entries using $h_1, h_2, ..., h_w$ hash functions and divides into B buckets. Now, in each bucket B, the entries are encoded in d dimensions of size N'_B. Generally, we will have more plaintext slots than the dimension size, g_B buckets will be merged into one $\tilde{\mathbf{DB}}_j$ i.e. server prepares the databases $\tilde{\mathbf{DB}}_0, \tilde{\mathbf{DB}}_1, ..., \tilde{\mathbf{DB}}_{\lceil B/g_B \rceil - 1}$.

QueryGen Let $\{i_1, i_2, ..., i_b\}$ be the batch of indices to query. The client will use the same encoding as the server to generate $\lceil B/g_B \rceil$ many queries and then uses FHE encryption $\mathcal{E}$ to get encrypted queries $\tilde{Q}_0, \tilde{Q}_1, ..., \tilde{Q}_{\lceil B/g_B \rceil - 1}$.

Response For each query $i = 0, ..., \lceil B/g_B \rceil - 1$, the server creates the response R_i and merges the responses to get $\lceil B/n \rceil$ number of ciphertexts and sends it. The client decrypts and gets corresponding entries to $\{i_1, i_2, ..., i_b\}$.

4.3.2 Communication Cost

To calculate communication costs, we need to calculate the number of ciphertext exchanges between the client and the server. The communication between them consists of packed vectorized PIR and response ciphertext.

1. To begin with n is the plaintext dimension, so let $g_B = n/N'_B$ be the number of buckets to be merged in a plaintext vector. Now, the entries are represented in a d-dimensional hypercube so the client's query consists of $d \cdot \lceil B/g_B \rceil$ number of ciphertexts.

2. Due to the merging algorithm of the responses, the server's response consists of $\lceil B/n \rceil$ number of ciphertexts.

3. Total number of ciphertext communicated between the client and the server is $d \cdot \lceil B/g_B \rceil + \lceil B/n \rceil$.

4. Total communication of $(d \cdot \lceil B/g_B \rceil + \lceil B/n \rceil) \cdot \log q$ bits.

For our case the database size, to store all the precomputed elliptic curve points $(t+1) \cdot 2^{\omega} \cdot \ell_{2,p}/\omega$, is $N = 2^{45}$ for window size $\omega = 32$ and $N = 2^{30}$ for window size $\omega = 16$, See table 3.3. If $\omega = 16$ then for each $k = 0, 1, 2, ..., t$ we can have $(t+1)$ separate databases to store the points $\{i \cdot 2^{\omega(j-1)}\}H_k$. Therefore the size for each database is 2^{20} and the batch size for each database is 16. Now consider the parameters for the homomorphic encryption as follows: the plaintext modulus p of size 28 bits, the ciphertext modulus q be of size 218 bits, and the plaintext slot $n = 8192$. We have marked the benchmark for these parameters, window size $\omega = 16$ and $N = 2^{20}$ using the implementation of the Vectorized BatchPIR [27].

- The client takes **6** ms to generate queries for a batch of 16 elements
- The response of one batch query the server takes **3249** ms.
- Total communication cost per batch query is **707** KB.

Similarly, for $\omega = 32$, the size of each $(t+1)$ database is 2^{35} and batch size is 8. Since the batch size is very small in both cases, using separate databases we cannot use BatchPIR with optimal communication. Rather we can store all the points in one database, then we can have a much larger batch.

4.3.3 Computation Cost

The computational overhead at the server end is proportional to the total number of elements across all the buckets. In the setup stage, the server hashed the entries into the buckets which results in the database size being $3N$. Therefore the computational

overhead is $O(N)$. All computation overheads for the server come from the CtRotate and CtCtMul of the entries. The server calls the response algorithm B/g_B times. In each call, there is N_B/N'_B number of CtPtMul in the first dimension, $N_B/(N'_B)^2$ number of CtCtMul in the second dimension, only one CtCtMul in the third dimension (say, $d = 3$). Now, there is N'_B number of CtRotate in the first dimension, $N_B/(N'_B)^2$ number of CtRotate in the second dimension and $(\log N'_B - 1)$ number of CtRotate for merging the response ciphertext.

1. Total number of CtPtMul $= B/g_B \cdot N_B/N'_B = B \cdot N_B/n$.
2. Total CtCtMul $= B/g_B \cdot (N_B/(N_B)^2 + 1) = B/n \cdot (N_B/N'_B + N'_B)$.
3. Total number of CtRotate

$$\begin{aligned} &= B/g_B \cdot \big(N_B/N'_B + N_B/(N'_B)^2 + (\log N'_B - 1)\big) \\ &= B/n(N_B + N_B/N'_B + N'_B(\log N'_B - 1) \end{aligned}$$

4. Number of multiplication depth is the number of dimensions.

Therefore the response time for Vectorized BatchPIR is more for the expensive operations CtCtMul and CtRotate.

4.4 SimplePIR [19]

The security of SimplePIR lies in the assumption of the learning-with-error. Regev's LWE encryption scheme is employed in this context, consequently reducing the computational overhead. However, the clients need to download a hint even before making a query to the server.

4.4.1 Protocol Preview

Setup The server encodes the database of size N to a $\sqrt{N} \times \sqrt{N}$ order matrix and outputs two hints, one for the server and one for the client. $\mathbf{hint}_c$ is a matrix of order $\sqrt{N} \times n$.

$$\mathbf{Setup}(\mathbf{db} \in \sqrt{N} \times \sqrt{N}) \longrightarrow (\mathbf{hint}_s, \mathbf{hint}_c)$$

QueryGen Let i be the index to query. The client encodes $i \longrightarrow (i_{\text{row}}, i_{\text{col}})$ and generates $(\mathbf{st}, \mathbf{qu})$. The parameters for Regev's LWE encryption are as follows: the LWE samples $A \in \mathbb{Z}_{\|}^{\sqrt{N} \times n}$, the error matrix $e \xleftarrow{\mathbf{R}} \chi^{\sqrt{N}}$, the secret key $s \xleftarrow{R} \mathbb{Z}_q^n$, the scaling parameter $\Delta = \lfloor q/p \rfloor \in \mathbb{Z}$, the plaintext vector $u_{i_{col}}$. Then the algorithm outputs $\mathbf{qu} \longleftarrow (A \cdot s + e + \Delta \cdot u_{i_{\text{col}}}), \mathbf{st} \longleftarrow (s, i_{\text{row}})$.

Answer The server generates $\mathbf{ans} \longleftarrow \mathbf{db} \cdot \mathbf{qu} \in \mathbb{Z}_q^{\sqrt{N}}$. There is $O(\sqrt{N})$ number of ciphertext-ciphertext addition i.e. Regev's LWE encryption should support these many CtCtAdds.

Recover The client uses the state $\mathbf{st} = (s, i_{row})$, $\mathbf{hint}_c$ and $\mathbf{ans}$ to recover the i-th data. $\mathbf{ans}$ is a $\sqrt{N} \times 1$ column vector whose i_{row} -th entry i.e. $\mathbf{ans}[i_{row}]$ contains the required data. Then computes $\hat{d} \longleftarrow \mathbf{ans}[i_{row}] - \mathbf{hint}_c[i_{row}, :] \cdot s$ where $\mathbf{hint}_c[i_{row}, :]$ is the i_{row}-th row of the matrix. Expressing $\hat{d}$ to the nearest multiple of Δ and dividing by Δ gives the required data.

4.4.2 Choosing Parameters and Bound on number of additions on encrypted data

The client sends the server $\mathbf{qu} \longleftarrow (A.s + e + \Delta \cdot u_{i_{\text{col}}})$ where $A \in \mathbb{Z}_q^{\sqrt{N} \times n}$, $s \in \mathbb{Z}_q^n$, $e \longleftarrow \chi^{\sqrt{N}}$ and $u_{i_{\text{col}}}$ is a $\sqrt{N}$ size vector where i_{col}-th position contains 1 and others are 0. At the server side $ans \longleftarrow \mathbf{db} \cdot \mathbf{qu}$ i.e. there are $\sqrt{N}$ number of CtCt addition needed. We have to choose the ciphertext modulus which satisfies $\sqrt{N}$ number of CtCtadd.Let $|e|_\infty \leq B$, Total error for $\sqrt{N}$ number of CtCtAdd is $\sqrt{N} \cdot p \cdot B$ i.e. q should satisfy

$$\sqrt{N} \cdot p \cdot B \leq \frac{1}{2} \cdot \lfloor q/p \rfloor \leq \frac{q}{2p} \implies q \geq 2 \cdot p^2 \cdot \sqrt{N} \cdot B$$

In paper [19], they proposed a better bound,

$$\lfloor q/p \rfloor \geq \sqrt{2} \cdot \sigma \cdot p \cdot N^{1/4} \cdot \sqrt{\ln 2/\delta} \tag{4.1}$$

The database entries are the elliptic curve points of the BN254 curve i.e. the entries are (x, y). Now, the prime is 254 bits, so the size of the entries is 508 bits. In setup step the database encoded as $\sqrt{N} \times \sqrt{N}$ matrix i.e. $\mathbf{db} \in Z_p^{\sqrt{N} \times \sqrt{N}}$. Since the entry size is big rather than using large $\mathbb{Z}_p$, we will be using entries as d-sized vectors i.e. $\mathbf{entries} \in \mathbb{Z}_p^d$ where p is much smaller. So, the total database size is dN and encoded as $\mathbf{db} \in \mathbb{Z}_p^{\sqrt{dN} \times \sqrt{dN}}$.

4.4.3 Different Parameter Sets

The LWE parameters (n, p, χ) are chosen for 128-bit security against the LWE attacks, employing the LWE estimator. q should be chosen to satisfy the required addition depth i.e. it should follow the equation 4.1. We have discussed three different parameter sets for three different choices of p. Depending on p the dimension of the database $(\sqrt{dN} \times \sqrt{dN})$ will be changed, again which changes the number of the LWE instances.

Parameter set 1:

1. Let us say $p = 2^{32}$. Now, the elliptic curve points (x, y), representing 508 bits. Now, to represent the points (x, y) in $\mathbb{Z}_p^d$. So, d should be

$$d = \frac{512}{32} = 16$$

   i.e. we can decompose the (x, y) like $\{d_1, d_2, ..., d_{16} \in \mathbb{Z}_{2^{32}}\}$.

2. Then the database $\mathbf{db} \in \mathbb{Z}_{2^{32}}^{\sqrt{dN} \times \sqrt{dN}}$.

3. So, we need to have $\sqrt{dN}$ number of LWE samples. Now, we need to see what the ciphertext modulus q and the lattice dimensions are.

4. For the number of entries $N = 2^{35}$, q should be

$$\lfloor q/p \rfloor \geq \sqrt{2} \cdot \sigma \cdot p \cdot N^{1/4} \cdot \sqrt{\ln 2/\delta} > \sqrt{2} \cdot 6.4 \cdot 2^{64} \cdot \sqrt[4]{2^{39}} \approx 2^{77} \tag{4.2}$$

5. **Hint Size.**

   - The server first sends $\mathbf{hint}_c$ to the client which is $\mathbf{hint}_c \longleftarrow \mathbf{db} \cdot A \in \mathbb{Z}_q^{\sqrt{N} \times n}$ where A is the LWE samples. The client needs to download $n \cdot \sqrt{N}$ number of elements in $\mathbb{Z}_q$.
   - Let consider the number of elliptic curve points $N = 2^{35}$ then the number of elements in server database must be $dN = 2^{39}$ i.e. $\sqrt{N} = 2^{19}\sqrt{2}$. Now, $n = 2^{12}$ and q are set to 90 bits to satisfy the 128-bit security and the equation 4.2. The client needs to download $n \cdot \sqrt{dN} = 2^{31}\sqrt{2}$ elements of 90 bits.

Parameter set 2:

1. The communication, the client sends $\sqrt{dN}$ number of elements to the server and server also sends $\sqrt{dN}$ elements as **ans**.

2. Let us say $p = 2^{16}$ and the database size $N = 2^{35}$. Then d should be $d = \frac{512}{16} = 32$ i.e. (x, y) can be decomposed like $\{d_1, d_2, ..., d_{32}\} \in \mathbb{Z}_{2^{16}}$.

3. Then the database $\mathbf{db} \in \mathbb{Z}_{2^{16}}^{\sqrt{dN} \times \sqrt{dN}}$

4. By the previous relation,

$$q > \sqrt{2} \cdot 6.4 \cdot 2^{32} \cdot \sqrt[4]{32 \cdot 2^{35}} \approx 2^{45} \tag{4.3}$$

5. Again by LWE estimator and satisfying the equation 4.3, n is set to 2^{11} and the size of q is set to 54 bits.

6. **Hint Size.** Considering $N = 2^{35}$, the total number of entries in the database is $\sqrt{dN} = 2^{20}$. Therefore the client's offline downloads are $n \cdot \sqrt{dN} = 2^{31}$ elements of 54 bits.

7. Benchmarking for LWE dimension $n = 2048$, the ciphertext modulus q of size 54 bits and the plaintext modulus p of size 16 bits i.e. $p = 36011$ and database size $N = 2^{35}$,

   Offline Download: 14591610 KB
   Online upload: 7124 KB
   Online download: 7124 KB

8. If we change the database size to $N = 2^{20}$,

   Offline Download: 80784 KB
   Online upload: 39 KB
   Online download: 39 KB

Parameter set 3:

1. Let us say $p = 2^8$. Then d should be $d = \frac{512}{8} = 64$ i.e. (x, y) can be decomposed like $\{d_1, d_2, ..., d_{64}\} \in \mathbb{Z}_{2^8}$.

2. Then the database $\mathbf{db} \in \mathbb{Z}_{2^8}^{\sqrt{dN} \times \sqrt{dN}}$

3. By the previous relation,

$$q > \sqrt{2} \cdot 6.4 \cdot 2^{16} \cdot \sqrt[4]{64 \cdot 2^{35}} \approx 2^{30} \tag{4.4}$$

4. Now, satisfying the equation 4.4 and to get 128-bit security against the LWE attacks, n is set to 2^{10} and q is chosen to be of size 32 bits.

5. **Hint Size.** Again considering $N = 2^{35}$, $\sqrt{dN}$ is set to $\sqrt{2^{41}} = 2^{20}\sqrt{2}$. Now the client will download $n \cdot \sqrt{dN} = 2^{30}\sqrt{2}$ elements of 32 bits.

6. For LWE dimension $n = 1024$, the ciphertext modulus q of size 32 bits and the plaintext modulus p of size 8 bits i.e. $p = 247$ and database size $N = 2^{35}$,

   Offline Download: 5931776 KB
   Online upload: 5792 KB
   Online download: 5792 KB

7. If we change the database size to $N = 2^{20}$,

   Offline Download: 29536 KB
   Online upload: 28 KB
   Online download: 28 KB

4.4.4 Batching in SimplePIR

To efficiently retrieve a batch of k indices from the server, it is proposed to generate batch queries. This approach minimizes the server-side computation by avoiding the execution of the SimplePIR k times independently over the whole database $\mathbf{db} \in \mathbb{Z}_p^{\sqrt{dN} \times n}$.

- The server sends the encrypted column i_{col} to the client. Suppose we want to retrieve 2 records from the same column then from i_{row^1}, i_{row^2} client can retrieve the records by the equations

$$\hat{d_1} \longleftarrow \mathbf{ans}[i_{row^1}] - \mathbf{hint}_c[i_{row^1}, :] \cdot s$$
$$\hat{d_2} \longleftarrow \mathbf{ans}[i_{row^2}] - \mathbf{hint}_c[i_{row^2}, :] \cdot s$$

  In this case, the server response time is still the same as a single query and the server will send only one ciphertext.

- Let consider that k records fall into distinct k chunks of the database. Let us divide the database $\mathbf{db}$ in $\sqrt{dN}/k \times \sqrt{dN}$ chunk of k databases. Then we can use SimplePIR k times to query the server and the server will compute over the chunks $\sqrt{dN}/k \times \sqrt{dN}$. Therefore server's computation costs remain the same and again send $\sqrt{N} \times 1$ order column matrix to the client as $\mathbf{ans}$, same as before 4.4.1. On the other hand, the client sends k ciphertexts to the server which is k times the before.

This chapter presents an overview of the BatchPIR and the SimplePIR. Since the batch size in our case is very small so, the batch query version of SimplePIR is more plausible for our approach. But again we cannot emphasize that because we don't have proper benchmarking for the BatchPIR. Also, there are some other nice PIRs we have marked in future work 6.

Chapter 5

Instantiation & Parameterization

In this chapter, we have discussed the BFV parameters selection and the different instances for the server and the client during the implementation. We also address our efforts and the difficulties we faced in the implementation.

BFV parameter selection. For verifying the KZG commitment protocol, the verifier needs to check the equality of the equation 2.2 where e is a bilinear pairing map. Therefore, we must choose pairing-friendly elliptic curve groups i.e. there should exist a bilinear pairing function $e : G_1 \times G_2 \longrightarrow G_T$. We have calculated further computations using the BN254 curve in the thesis. Therefore, the plaintext polynomial coefficients must be 254 bits in size, i.e., the BFV scheme's plaintext modulus should be 254 bits in size. The standard deviation of the distribution χ is fixed at $\sigma = 3.2$. The plaintext polynomial degree n and the ciphertext modulus q can be fixed by employing the lattice-estimator [3, 7] as in table 5.1 and see the figure 5.1.

Polynomial degree(n)	Ciphertext Modulus(q) in bits	Mult. Depth(d)	Security Level(λ)
2^{15}	1400	6	$\approx$ 81 bits
	1600	7	$\approx$ 72 bits
2^{16}	1600	7	$\approx$ 141 bits
	1783	8	$\approx$ 128 bits
2^{17}	3550	17	$\approx$ 128 bits

Table 5.1: (n, q, d) by lattice estimator

We need to choose the parameters so that it satisfies 128-bit security for modern LWE attacks [6]. Then one choice for (n, q) is $n = 2^{16}$ and the ciphertext modulus q of size 1783 bits and another one is $n = 2^{17}$ and q of size 3550 bits. Again from the table 3.1, we can conclude that for window size $\omega = 16$ the BFV parameters should satisfy the multiplication depth $d = 8$ and for $\omega = 32$ the BFV parameters

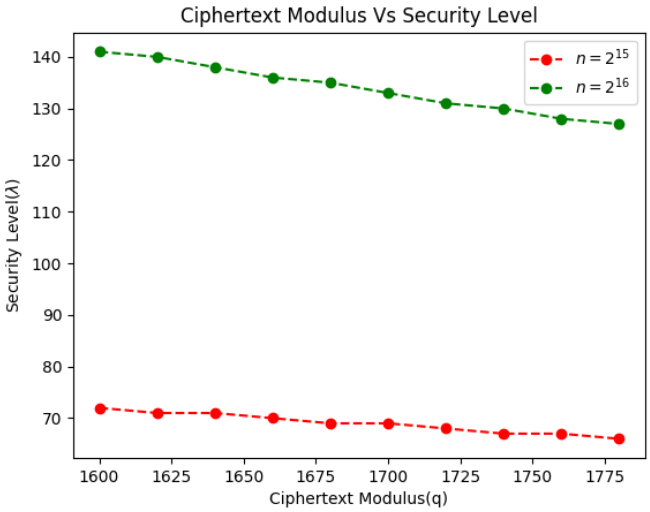

Figure 5.1: Ciphertext Modulus Vs Security Level

should satisfy the multiplication depth $d = 6$ to calculate the scalar multiplications. Large plaintext dimensions and the ciphertext modulus result in high computation and communication costs. Therefore n is set to 2^{16} and q is set to a prime of size 1783 bits which are still very large that it slows the FHE.

Server's Side. In our delegation protocol of KZG commitment generation of polynomial $\phi(x)$,

$$C_\phi = \sum_{k=0}^{t} \phi_k \cdot H_k$$

the Untrusted Cloud(UC) server computes homomorphically the multi-scalar multiplications for $k \in \{0, 1, 2, ..., t\}$,

$$\phi_k \cdot H_k = \left\{ \sum_{j=1}^{\ell} \phi_k^{(j)} 2^{\omega(j-1)} \right\} \cdot H_k = \sum_{j=1}^{\ell} P^k_{\phi_k^{(j)}, j}$$

with the received encryption of the coordinates of the elliptic curve points $P^k_{\phi_k^{(j)}, j}$ i.e. (CX_j, CY_j) as in 3.1,3.2 for $j \in \{1, 2, ..., \ell\}$. The UC uses the binary tree structure to reduce the depth of the multiplication circuit. The UC can add points up to a certain level that the BFV protocol supports i.e. the server returns the scalar multiplications $\phi_k \cdot H_k$. The later t additions i.e. $\sum_{k=0}^{t} \phi_k \cdot H_k$, must be completed over the plaintext by the client. If the BFV parameters support more multiplication depth then the client can send more encrypted points. Let us consider d as the supported depth, then the client can send $2^{d/2}$ ciphertexts. Therefore the client can delegate more additions to the server.

Client's Side. In our delegation protocol, the client's computational overhead consists of three main tasks: generating the PIR queries, performing BFV encryption of elliptic curve points, and executing elliptic curve point additions. The number of queries to make is $t \cdot \ell_{2,p}/\omega$. For both the PIRs we have discussed, the query generation overhead is not very high. The client will encrypt the coordinates of the elliptic curve points i.e. the client applies the BFV scheme $\mathcal{E}$ over the $2 \cdot \ell_{2,p}/\omega$ number of plaintexts. Now, our parameters do not give a good throughput for encryption and decryption. The last computation overhead for the client is the addition of the elliptic curve performed by the client over the plaintext after server computations. The second and third instances of the client's computation overhead depend on the multiplication depth supported by the BFV scheme. If the scheme supports more multiplication depth the client needs to compute more encryptions but fewer elliptic curve additions. The communication overhead for the client depends on the communication with the PIR server and the communication with the UC.

Our Efforts. The well-known efficient FHE libraries SEAL [9], OpenFHE [8], HElib [1, 18] don't support these large parameters. During this time we have implemented it with these parameters on our own. We have used NTT and multi-threaded programming in Python. However, the implementation is extremely slow and not feasible. We have not tried to implement the RNS-variant, which can be more feasible.

We have retrieved some benchmarkings (Sec 4.4) for different parameters employing the implementation of SimplePIR [2]. Also, we have noticed that for large entry sizes, when a large batch of indices is retrieved, it fails the verification.

We have given parameters and benchmarks for small database size 2^{20} and batch size 16 (See 4.3) by the implementation of Vectorized BatchPIR [26, 27]. However, we could not find the best parameters for fully homomorphic encryption to run the implementation of Vectorized BatchPIR with our large database 2^{35} and entry sizes of 64 Bytes.

Chapter 6

Conclusion & Future Directions

In this work, we present an approach to securely delegate the commitment generation of the KZG commitment scheme for resource-constrained devices. First, there are cloud servers where the precomputed elliptic curve points are stored in the setup stage which costs a substantial amount of storage. Now, the clients make PIR queries to retrieve the precomputed values which cost the clients a good amount of money for using the huge cloud storage. Then the clients use FHE to encrypt the points and delegate the computations to the untrusted cloud(UC) server. Therefore, the clients can reduce their memory usage and the computation overhead. Though we are not able to conclude with implementation and benchmarking for the computation and communication trade-off, we want to mark some difficulties and future directions to this work.

The problem with the implementation appears for the use of large BFV parameters. This issue can be resolved by replacing the plaintext modulus p with the polynomial $(x-b)$ which leads to the new plaintext space $\mathbb{Z}/(b^n+1)\mathbb{Z}$ where $b \geq 2$ is an integer(See [10]). A plaintext m is encoded as a polynomial $\hat{m}$ where $||\hat{m}|| \leq (b+1)/2$, satisfying $\hat{m}(b) = m \pmod{b^n+1}$. This new approach reduces noise growth. For example, it allows us to evaluate circuits of depth 9 with 32-bit integer inputs whereas the previous scheme allows only up to depth 2 under the same ciphertext parameters. But for that, it is essential to identify an elliptic curve that supports encoding and is also pairing-friendly. Then it will be possible to set a smaller ciphertext modulus which supports the required security properties and the multiplication depth. It can be one direction for future improvement.

There are several companies such as Intel, Cornami, Galois, and Optalysis aiming at accelerating FHE with hardware. Using those platforms, more operations such as bootstrapping can be performed faster than regular computing platforms. There are also algorithmic improvements of bootstrapping, making it faster and more efficient [14, 25]. Therefore with bootstrapping the server can perform sufficient elliptic

curve additions over the encrypted data. The server does not need to abort early. Hence our delegation approach can be viable to implement in the future.

There are some recent works in the field of PIR protocols which can be explored. PIANO [32] is an extremely simple single server PIR with sublinear server computation. Whereas the other PIR protocols are based on complex areas like LWE assumptions, PIANO relies on pseudo-random function (PRF). It achieves sublinear computation overhead for the server, therefore the response time is less than the state-of-art PIRs. It opens up the applications with a large database like in our approach. There is another preprocessing PIR scheme [16] without public key cryptography. It achieves online bandwidth of $\tilde{O}(n^{1/4})$ i.e. the bandwidth required to get the answer. Also, it relies on a straightforward cryptographic primitive, pseudo-random function. These are some possibilities we have not explored yet.

Bibliography

[1] Helib: an open-source (apache license v2.0) software library that implements homomorphic encryption (he), https://github.com/homenc/HElib, accessed: 2024-06-24

[2] This repository contains the code for simplepir and doublepir, two high-throughput single-server pir schemes, https://github.com/ahenzinger/simplepir/tree/main, accessed: 2024-06-25

[3] This sage module provides functions for estimating the concrete security of learning with errors instances, https://github.com/malb/lattice-estimator, accessed: 2024-06-25

[4] zkprover: a component of polygon zkevm which is responsible for proving and verification of the transaction, https://docs.polygon.technology/zkEVM/architecture/zkprover/, accessed: 2024-06-25

[5] Aguilar-Melchor, C., Deneuville, J.C., Gaborit, P., Lepoint, T., Ricosset, T.: Delegating elliptic-curve operations with homomorphic encryption. In: 2018 IEEE Conference on Communications and Network Security (CNS). pp. 1–9. IEEE (2018)

[6] Albrecht, M., Chase, M., Chen, H., Ding, J., Goldwasser, S., Gorbunov, S., Halevi, S., Hoffstein, J., Laine, K., Lauter, K., et al.: Homomorphic encryption standard. Protecting privacy through homomorphic encryption pp. 31–62 (2021)

[7] Albrecht, M.R., Player, R., Scott, S.: On the concrete hardness of learning with errors. Journal of Mathematical Cryptology 9(3), 169–203 (2015)

[8] Badawi, A.A., Bates, J., Bergamaschi, F., Cousins, D.B., Erabelli, S., Genise, N., Halevi, S., Hunt, H., Kim, A., Lee, Y., Liu, Z., Micciancio, D., Quah, I., Polyakov, Y., R.V., S., Rohloff, K., Saylor, J., Suponitsky, D., Triplett, M., Vaikuntanathan, V., Zucca, V.: Openfhe: Open-source fully homomorphic encryption library. Cryptology ePrint Archive, Paper 2022/915 (2022), https://eprint.iacr.org/2022/915, https://eprint.iacr.org/2022/915

[9] Chen, H., Laine, K., Player, R.: Simple encrypted arithmetic library-seal v2. 1. In: Financial Cryptography and Data Security: FC 2017 International Workshops, WAHC, BITCOIN, VOTING, WTSC, and TA, Sliema, Malta, April 7, 2017, Revised Selected Papers 21. pp. 3–18. Springer (2017)

[10] Chen, H., Laine, K., Player, R., Xia, Y.: High-precision arithmetic in homomorphic encryption. In: Cryptographers' Track at the RSA Conference. pp. 116–136. Springer (2018)

[11] Chiesa, A., Hu, Y., Maller, M., Mishra, P., Vesely, N., Ward, N.: Marlin: Preprocessing zksnarks with universal and updatable srs. In: Advances in Cryptology–EUROCRYPT 2020: 39th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Zagreb, Croatia, May 10–14, 2020, Proceedings, Part I 39. pp. 738–768. Springer (2020)

[12] Edgington, B.: Bls12-381 for rest of us, `https://hackmd.io/@benjaminion/bls12-381s`, accessed: 2024-06-25

[13] Gabizon, A., Williamson, Z.J., Ciobotaru, O.: Plonk: Permutations over lagrange-bases for oecumenical noninteractive arguments of knowledge. Cryptology ePrint Archive (2019)

[14] Geelen, R., Iliashenko, I., Kang, J., Vercauteren, F.: On polynomial functions modulo p^e and faster bootstrapping for homomorphic encryption. Cryptology ePrint Archive, Paper 2022/1364 (2022), `https://eprint.iacr.org/2022/1364`, `https://eprint.iacr.org/2022/1364`

[15] Gentry, C.: A fully homomorphic encryption scheme. Stanford university (2009)

[16] Ghoshal, A., Zhou, M., Shi, E.: Efficient pre-processing PIR without public-key cryptography. Cryptology ePrint Archive, Paper 2023/1574 (2023), `https://eprint.iacr.org/2023/1574`, `https://eprint.iacr.org/2023/1574`

[17] Groth, J.: On the size of pairing-based non-interactive arguments. In: Advances in Cryptology–EUROCRYPT 2016: 35th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Vienna, Austria, May 8-12, 2016, Proceedings, Part II 35. pp. 305–326. Springer (2016)

[18] Halevi, S., Shoup, V.: Design and implementation of a homomorphic-encryption library. IBM Research (Manuscript) 6(12-15), 8–36 (2013)

[19] Henzinger, A., Hong, M.M., Corrigan-Gibbs, H., Meiklejohn, S., Vaikuntanathan, V.: One server for the price of two: Simple and fast {Single-Server} private information retrieval. In: 32nd USENIX Security Symposium (USENIX Security 23). pp. 3889–3905 (2023)

[20] Kang, D., Hashimoto, T., Stoica, I., Sun, Y.: Zk-img: Attested images via zero-knowledge proofs to fight disinformation. arXiv preprint arXiv:2211.04775 (2022)

[21] Kate, A., Zaverucha, G.M., Goldberg, I.: Constant-size commitments to polynomials and their applications. In: Advances in Cryptology-ASIACRYPT 2010: 16th International Conference on the Theory and Application of Cryptology and Information Security, Singapore, December 5-9, 2010. Proceedings 16. pp. 177–194. Springer (2010)

[22] Kim, A., Polyakov, Y., Zucca, V.: Revisiting homomorphic encryption schemes for finite fields. In: Advances in Cryptology–ASIACRYPT 2021: 27th International Conference on the Theory and Application of Cryptology and Information Security, Singapore, December 6–10, 2021, Proceedings, Part III 27. pp. 608–639. Springer (2021)

[23] Liu, X., Zhou, Z., Wang, Y., Zhang, B., Yang, X.: Scalable collaborative zk-SNARK: Fully distributed proof generation and malicious security. Cryptology ePrint Archive, Paper 2024/143 (2024), https://eprint.iacr.org/2024/143, https://eprint.iacr.org/2024/143

[24] Luo, G., Fu, S., Gong, G.: Speeding up multi-scalar multiplication over fixed points towards efficient zksnarks. IACR Transactions on Cryptographic Hardware and Embedded Systems pp. 358–380 (2023)

[25] Ma, S., Huang, T., Wang, A., Wang, X.: Accelerating BGV bootstrapping for large p using null polynomials over $\mathbb{Z}_{\text{!`}}$. Cryptology ePrint Archive, Paper 2024/115 (2024), https://eprint.iacr.org/2024/115, https://eprint.iacr.org/2024/115

[26] Mughees, M.H., Ren, L.: Vectorized batch private information retrieval. In: 2023 IEEE Symposium on Security and Privacy (SP). pp. 437–452. IEEE (2023)

[27] Muhammad Haris Mughees, L.R.: contains an implementation of the vectorized batch private information retrieval (pir) protocol, https://github.com/mhmughees/vectorized_batchpir, accessed: 2024-06-25

[28] Ozdemir, A., Boneh, D.: Experimenting with collaborative zk-SNARKs: Zero-knowledge proofs for distributed secrets. Cryptology ePrint Archive, Paper 2021/1530 (2021), https://eprint.iacr.org/2021/1530, https://eprint.iacr.org/2021/1530

[29] arkworks is a Rust ecosystem for zkSNARK programming: arkworks-rs/curves: implementation of popular elliptic curves, https://github.com/arkworks-rs/curves, accessed: 2024-06-25

[30] Renes, J., Costello, C., Batina, L.: Complete addition formulas for prime order elliptic curves. In: Advances in Cryptology–EUROCRYPT 2016: 35th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Vienna, Austria, May 8-12, 2016, Proceedings, Part I 35. pp. 403–428. Springer (2016)

[31] Wang, J.: Bn254 for the rest of us, `https://hackmd.io/@jpw/bn254`, accessed: 2024-06-25

[32] Zhou, M., Park, A., Shi, E., Zheng, W.: Piano: Extremely simple, single-server PIR with sublinear server computation. Cryptology ePrint Archive, Paper 2023/452 (2023), `https://eprint.iacr.org/2023/452`, `https://eprint.iacr.org/2023/452`