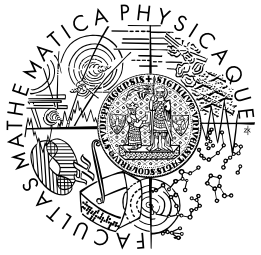

FACULTY
OF MATHEMATICS
AND PHYSICS
Charles University

MASTER THESIS

Bc. Ľubica Jančová

Effectivity and Limitations of Homomorphic Secret Sharing Schemes

Computer Science Institute of Charles University

Supervisor of the master thesis: Mgr. Pavel Hubáček, Ph.D.

Study programme: Mathematics

Study branch: Mathematics for Information Technologies

Prague 2022

I declare that I carried out this master thesis independently, and only with the cited sources, literature and other professional sources. It has not been used to obtain another or the same degree.

I understand that my work relates to the rights and obligations under the Act No. 121/2000 Sb., the Copyright Act, as amended, in particular the fact that the Charles University has the right to conclude a license agreement on the use of this work as a school work pursuant to Section 60 subsection 1 of the Copyright Act.

In ..... date ....

Author's signature

I would like to thank this thesis supervisor Pavel Hubáček and consultant Ignacio Cascudo for their time, patience, and all the valuable advice. I would like to thank my family for their infinite support and Nicolas Mazzocchi for being here for me at every moment.

Title: Effectivity and Limitations of Homomorphic Secret Sharing Schemes

Author: Bc. Ľubica Jančová

Institute: Computer Science Institute of Charles University

Supervisor: Mgr. Pavel Hubáček, Ph.D., Computer Science Institute of Charles University

Consultant: Dr. Ignacio Cascudo Pueyo, Research assistant professor, IMDEA Software Institute

Abstract: This thesis focuses on constructions of Homomorphic Secret Sharing (HSS) based on assumptions not known to imply fully homomorphic encryption. The efficiency of these constructions depends on the complexity of the Distributed Discrete Logarithm (DDLog) problem in the corresponding groups. We describe this problem in detail, focusing on the possibility of leveraging preprocessing in prime order groups, and deriving upper bounds on the success probability for the DDLog problem with preprocessing in the generic group model. Further, we present a new HSS construction. We base our construction on the Joye-Libert encryption scheme which we adapt to support an efficient distributed discrete logarithm protocol. Our modified Joye-Libert scheme requires a new set of security assumptions, which we introduce, proving the IND-CPA security of our scheme given these assumptions.

Keywords: Homomorphic Secret Sharing, Distributed Discrete Logarithm, Generic group model

Contents

Introduction

The possibility of privately outsourcing computation is a major application of cryptography. We can easily imagine a small device, for example, a mobile phone, employing a server with a significantly larger computational power to tackle its computational tasks. We want to ensure that the delegated computation stays *private*. Depending on what is required to stay private, we can distinguish between input privacy and function privacy. We want to guarantee that no information on the client's inputs is leaked to the server performing the computation in input privacy. Whereas we want to keep the function, computed over the inputs, hidden from the server in the function privacy. The function privacy has been tackled for example by the *distributed point function* in [GI14] and *function secret sharing* in [BGI15]. In this thesis, we focus on the privacy of the client's inputs.

The input privacy can be ensured by *Fully Homomorphic Encryption* (FHE), which is an encryption scheme, that supports the evaluation of an arbitrary efficiently computable function on encrypted data. For example, using FHE, the client would encrypt his inputs and send the encrypted data together with the description of the function he wants to evaluate to the server, the server would evaluate the function on the encrypted data and send an output to the client, and the client would decrypt the output provided by the server and thus obtain the desired result. All of the known FHE constructions (e.g. [Gen09]) are lattice-based encryption schemes, and they rely on assumptions as *Learning with errors* (LWE) or *Ring Learning with errors* (RLWE). The lattice-based approach bears some issues with noise which grows with every operation performed on the encrypted data. To control the noise growth, in order to be able to decrypt correctly at the end of the evaluation, it is necessary to perform costly bootstrapping operations, which allow keeping the noise small.

In search of a more efficient way to securely outsource computation, we can relax this model to its secret-sharing variant. Instead of one server performing the computation, we secret-share the data between two servers so that each of the shares computationally hides the client's data. Then each of the servers evaluates the function chosen by the client on its share and sends the evaluation output to the client. Given the outputs from both servers, the client composes the result of the computational task. A scheme that implements this model is called a *Homomorphic Secret Sharing* (HSS). We emphasize that the trust model in HSS is different from the trust model of FHE. In FHE, the client does not trust the server and keeps his data computationally hidden from it. In the HSS case, the client does not trust the servers either, and the share of data given to each individual server computationally hides the data; however, the client trusts that the two servers will not collude to uncover his data, i.e., given both shares, the client's data are not hidden.

It is possible to construct an HSS based on known constructions of FHE. Moreover, [BKS19] constructed HSS for the class of *Restricted Multiplication Straightline* (RMS) programs from LWE, which does not imply FHE. Their construction avoids the costly operations of FHE and enjoys a considerably better efficiency than the HSS constructions implying FHE. Currently, there exist several constructions of HSS for RMS programs that rely on conceptually different assumptions,

which are not known to imply FHE. These comprise constructions based on the ElGamal encryption scheme [BGI16], relying on the *Decisional Diffie-Hellman assumption* (DDH) and constructions based on the Paillier encryption scheme [OSY21], relying on the *Decisional Composite Residuosity* (DCR) assumption.

In this thesis, we focus on the constructions of HSS based on the assumptions not known to imply FHE. In Chapter 1, we review some basic cryptographic definitions, including the definition of HSS. We also recall the *Distributed Discrete Logarithm* (DDLog) problem, which is used in the constructions of HSS from [BGI16] and [OSY21]. We focus on the DDLog problem for the prime order groups, corresponding to the case of [BGI16].

[DKK20] gave lower bounds on the error probability of the DDLog problem for prime order groups in the generic group model. This error probability scales with $\Omega(T^{-2})$, where T denotes the number of performed group operations, which also implies a non-negligible correctness error in the HSS construction form [BGI16]. In Chapter 2, we analyze the DDLog problem for prime order groups in the preprocessing model. We examine the possibility of saving computation while allowing the parties to precompute an advice string in advance. We perform this analysis in the generic group model using the approach from [CDG18]. We show that for generic groups, if we allow the parties to precompute a small advice string (of size $S = O(N/W)$, where N is the size of the group, and W denotes the size of the interval in the DDLog problem), then the preprocessing does not meaningfully help. In particular, for a success probability ϵ, we prove a lower bound on the number of group operations $T^2 = \Omega(\epsilon W)$, while a DDLog algorithm with no preprocessing presented by [DKK20] reaches the complexity $T^2 = O(\frac{W}{1-\epsilon})$. For a constant probability of success these two bounds match. If we allow a big advice string ($S = \Omega(N/W)$), then the problem allows a time-space trade-off bounded by $ST^2 = \Omega(\epsilon N)$, where ϵ denotes the success probability.

In Chapter 3, we present a new construction of HSS, based on the Joye-Libert (JL) encryption scheme ([JL13]). The original Joye-Libert scheme does not seem to naturally support an efficient way of calculating DDLog. In order to get an efficient DDLog protocol, we introduce some modifications to the JL scheme. We call this new encryption scheme *modified Joye-Libert* (mJL) scheme. Due to our modifications, the assumptions on which the security of mJL relies are not the same as the ones of the original scheme. Therefore, we define a set of assumptions the security of mJL relies on, and we prove its IND-CPA security given these assumptions. Then, we present our DDLog protocol for mJL, and we employ it forwards a construction of HSS using mJL as its underlying encryption scheme.

1. Preliminaries

In this chapter, we review the basic definitional framework. We define Homomorphic Secret Sharing, as well as the class of Restricted Multiplication Straight-line programs, which we will be able to evaluate with our construction of HSS. We recall the Distributed Discrete Logarithm problem, explain its relation to the Homomorphic Secret Sharing and define it in the context of the generic group model.

1.1 Basic cryptographic definitions

In this section, we review several standard cryptographic definitions. We also introduce the notation we use throughout the text.

Notation. For $n \in \mathbb{N}$, we denote by $[n]$ the set of natural numbers from 1 to n, i.e., $[n] = \{1, \ldots, n\}$. By the symbol $\leftarrow$, e.g. $a \leftarrow [n]$, we denote a uniformly random choice of an element from a given set. Furthermore, for a probabilistic algorithm $\mathcal{A}$, $a \leftarrow \mathcal{A}$ means that a has been an output generated by the algorithm $\mathcal{A}$ with a uniformly random choice of the random coins of the algorithm.
For $a, b \in \mathbb{Z}$, we denote by $\gcd(a, b)$ the greatest common divisor of a and b.
By a boolean expression in square brackets, e.g., $[a < b]$, we denote its truth value, that is 1 if the expression is true, 0 if it is false.
By *PPT* we denote a probabilistic polynomial-time Turing machine, i.e., a probabilistic Turing machine for which there exists $k \in \mathbb{N}$, such that its running time on an input of size n is $O(n^k)$.

Definition 1 (Public-key encryption scheme)**.** *Let $\lambda \in \mathbb{N}$ be a security parameter.* A public key encryption scheme *with message space $\mathcal{M}$ is a sequence of PPT algorithms* $(\mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec})$ *such that* KeyGen *is a key generation algorithm that on input 1^λ outputs a key pair composed of a public key and a secret key* $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{KeyGen}(1^\lambda)$. Enc *is an encryption algorithm and* Dec *is a decryption algorithm such that for every $m \in \mathcal{M}$ we have* $\mathsf{Dec}(\mathsf{sk}, \mathsf{Enc}(\mathsf{pk}, m)) = m$.

Definition 2 (Negligible function)**.** *Let f be a function $f : \mathbb{N} \to \mathbb{R}$. We say f is negligible if for every $c \in \mathbb{N}$ exists $n_c \in \mathbb{N}$ such that for every $x \geq n_c$ we have $|f(x)| < \frac{1}{x^c}$. We denote this by $f = \mathrm{negl}(x)$.*

We review the standard notion of $\mathsf{IND\text{-}CPA}$ security.

Definition 3 ($\mathsf{IND\text{-}CPA}$ security)**.** *Let* $(\mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec})$ *be a public-key encryption scheme and $\lambda \in \mathbb{N}$ be a security parameter. For a PPT algorithm $\mathcal{A}$, attacker on the* $\mathsf{IND\text{-}CPA}$ *property, we define its advantage* $\mathsf{Adv}^{\mathsf{IND\text{-}CPA}}_{\mathcal{A}}(\lambda)$ *in the* $\mathsf{IND\text{-}CPA}$ *security experiment (Figure 1) as follows:*

$$\mathsf{Adv}^{\mathsf{IND\text{-}CPA}}_{\mathcal{A}}(\lambda) := \left| \Pr[1 \leftarrow \mathsf{Exp}^{\mathsf{IND\text{-}CPA}}_{\mathcal{A}}(1^\lambda)] - \frac{1}{2} \right|.$$

We say a public-key encryption scheme is $\mathsf{IND\text{-}CPA}$ secure *if for every non-uniform PPT attacker $\mathcal{A}$ is its* $\mathsf{IND\text{-}CPA}$ *advantage negligible, i.e.,* $\mathsf{Adv}^{\mathsf{IND\text{-}CPA}}_{\mathcal{A}}(\lambda) = \mathrm{negl}(\lambda)$.

$\mathsf{Exp}^{\mathsf{IND\text{-}CPA}}_{\mathcal{A}}(1^\lambda)$**:**

1. The challenger runs KeyGen with security parameter λ to generate the secret key sk and the public key pk and it sends $(1^\lambda, \mathsf{pk})$ to the adversary $\mathcal{A}$.
2. On input $(1^\lambda, \mathsf{pk})$ the adversary $\mathcal{A}$ chooses two messages m_0, m_1 from the message space of the public-key encryption scheme and sends m_0, m_1 to the challenger.
3. The challenger chooses uniformly at random a bit $b \in \{0,1\}$ and sends $c = \mathsf{Enc}(\mathsf{sk}, m_b)$ to $\mathcal{A}$.
4. $\mathcal{A}$ outputs a guess b'.
5. The output of the experiment is 1 if $b = b'$, otherwise the output of the experiment is 0.

Figure 1 − $\mathsf{IND\text{-}CPA}$ security experiment.

Definition 4. *Let D_0, D_1 be two probability distributions and $\lambda \in \mathbb{N}$ be a security parameter. Consider a PPT algorithm $\mathcal{D}$, we call it a distinguisher, and the indistinguishability experiment described in Figure 2. We say $\mathcal{D}$ wins in the indistinguishability experiment if $b = b'$ and we define the advantage of $\mathcal{D}$ in the indistinguishability experiment as follows:*

$$\mathsf{Adv}^{D_0,D_1}_{\mathcal{D}}(\lambda) \coloneqq \Big|\Pr[b' = 1 \mid b = 1] - \Pr[b' = 1 \mid b = 0]\Big|.$$

We say the distributions D_0, D_1 are computationally indistinguishable, we denote $D_0 \overset{\text{c}}{\approx} D_1$, if for every PPT $\mathcal{D}$, its advantage in the indistinguishability experiment for the distributions D_0, D_1 is negligible.

$\mathsf{Exp}^{D_0,D_1}_{\mathcal{D}}(1^\lambda)$:

1. The challenger chooses a bit $b \leftarrow \{0,1\}$ and a challenge $x \leftarrow D_b$ and sends $(1^\lambda, x)$ to $\mathcal{D}$.
2. On input $(1^\lambda, x)$ the distinguisher $\mathcal{D}$ outputs a guess $b' \in \{0,1\}$.

Figure 2 − Indistinguishability experiment for the distributions D_0, D_1 .

We review the notion of *Key-Dependent Message security* (KDM) introduced by Black, Rogaway and Shrimpton in [BRS02]. We base our definition of KDM security on the version from [OSY21, Definition 2.2].

Definition 5 (KDM security)**.** *Let λ denote the security parameter, F be some set of functions and let $(\mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec})$ be a public-key encryption scheme. We define an advantage of a PPT attacker $\mathcal{A}$ in the KDM security experiment as follows:*

$$\mathsf{Adv}^{\mathsf{KDM}}_{\mathcal{A}}(\lambda) \coloneqq \left|\Pr[1 \leftarrow \mathsf{Exp}^{\mathsf{KDM}}_{\mathcal{A}}(1^\lambda)] - \frac{1}{2}\right|.$$

$\mathsf{Exp}^{\mathsf{KDM}}_{\mathcal{A}}(1^\lambda)$:

1. The challenger runs KeyGen on input 1^λ and receives the output $(\mathsf{pk}, \mathsf{sk})$. It forwards $(1^\lambda, \mathsf{pk})$ to $\mathcal{A}$.
2. On input $(1^\lambda, \mathsf{pk})$ the attacker $\mathcal{A}$ chooses a polynomial subset $\{f_1, \ldots, f_p\} \subset F$ and send his choice to the challenger.
3. The challenger chooses $b \leftarrow \{0,1\}$. If $b = 1$ it sets $c_i = \mathsf{Enc}(\mathsf{pk}, f_i(\mathsf{sk}))$, if $b = 0$ it sets $c_i = \mathsf{Enc}(\mathsf{pk}, 0^{|f_i(\mathsf{sk})|})$ for $i \in [p]$, and it sends $c_1, \ldots, c_p$ to $\mathcal{A}$.
4. $\mathcal{A}$ outputs $b' \in \{0,1\}$, the guess for the bit b.
5. The output of the experiment is 1 if $b = b'$, otherwise the output of the experiment is 0.

Figure 3 – KDM security experiment.

We say a public-key encryption scheme is KDM*-secure (Key-Dependent Message secure) over the set of functions F if for every non-uniform PPT $\mathcal{A}$ is its advantage* $\mathsf{Adv}^{\mathsf{KDM}}_{\mathcal{A}}(\lambda)$ *negligible, i.e.,* $\mathsf{Adv}^{\mathsf{KDM}}_{\mathcal{A}}(\lambda) = \mathrm{negl}(\lambda)$.

1.2 Homomorphic Secret Sharing

We base the definition of Homomorphic secret sharing on the definitions of HSS given by [BGI16] and [BGI+18].

Definition 6 (Homomorphic secret sharing)**.** *Let $n \in \mathbb{N}$ and $\lambda \in \mathbb{N}$ be a security parameter. We define a 2-party* Homomorphic secret sharing (HSS) *for a class of programs $\mathcal{P}$ as a sequence of PPT algorithms* $(\mathsf{Share}, \mathsf{Eval}, \mathsf{Dec})$ *with the following syntax*

$\mathsf{Share}(1^\lambda, (w_1, \ldots, w_n))$**:** *Given a security parameter and inputs $(w_1, \ldots, w_n)$ the algorithm* Share *outputs* $(\mathsf{share}_0, \mathsf{share}_1)$*, the secret shares for party 0 and party 1 respectively.*

$\mathsf{Eval}(P, b, \mathsf{share}_b)$**:** *Given a program $P \in \mathcal{P}$, a party index b and b-th shares of the inputs, the algorithm* Eval *outputs* out_b*, corresponding to the b-th share of the final output* out*.*

$\mathsf{Dec}(\mathsf{out}_0, \mathsf{out}_1, \beta)$**:** *Given the output shares* $\mathsf{out}_0, \mathsf{out}_1$ *and a modulus β the algorithm returns a final output* out*.*

such that the following correctness and security requirement hold.

Correctness

Let δ be a given error bound. A HSS is said to be δ-correct if for every polynomial p there is a negligible function ϵ such that for every $\lambda \in \mathbb{N}$, for

every input $(w_1, \ldots, w_n)$*, for every program* $P \in \mathcal{P}$ *of input length* n *and size* m *such that* $m \leq p(\lambda)$ *the following holds:*

$$\Pr\left[\mathsf{Dec}(\mathsf{out}_0, \mathsf{out}_1, \beta) = P(w_1, \ldots, w_n) \middle| (\mathsf{share}_0, \mathsf{share}_1) \leftarrow \mathsf{Share}(1^\lambda, (w_1, \ldots, w_n)), \right.$$
$$\left. \mathsf{out}_b \leftarrow \mathsf{Eval}(P, b, \mathsf{share}_b), b \in \{0,1\}\right] \geq 1 - \delta - \epsilon(\lambda)$$

We say the HSS is statistically correct, if it is 0-correct.

Security

For every $b \in \{0,1\}$*, for every polynomial* p *and every* $n \leq p(\lambda)$ *and every two input sequences* $w_1, \ldots, w_n$ *and* $v_1, \ldots, v_n$ *such that* $|w_i| = |v_i|$*,* $i \in [n]$*, for every non-uniform PPT distinguisher* $\mathcal{A}$ *there exists a negligible function* ϵ *such that for every* $\lambda \in \mathbb{N}$

$$\left|\Pr[1 \leftarrow \mathcal{A}(1^\lambda, \mathsf{share}_b)] - \Pr[1 \leftarrow \mathcal{A}(1^\lambda, \mathsf{share}_b')]\right| \leq \epsilon(\lambda),$$

where $(\mathsf{share}_0, \mathsf{share}_1) \leftarrow \mathsf{Share}(1^\lambda, (w_1, \ldots, w_n))$ *and* $(\mathsf{share}_0', \mathsf{share}_1') \leftarrow \mathsf{Share}(1^\lambda, (v_1, \ldots, v_n))$.

The Homomorphic secret sharing constructions from [BGI16], [BKS19] and [OSY21] as well as our protocol described in Chapter 3 are HSS schemes for the class of Restricted Multiplication Straight-line programs. We define this class of programs below.

Definition 7 (RMS program, [BGI16, Definition 3.1])**.** *Let* $\beta, m, M \in \mathbb{N}$*. A* Restricted Multiplication Straight-line program (RMS) *with a magnitude bound* M *and size* m *is a sequence of* m *instructions of the following types:*

- *Load input value into memory* $(\mathsf{id}, y_i \leftarrow w_i)$*,*
- *Add two memory values* $(\mathsf{id}, y_u \leftarrow y_i + y_j)$*,*
- *Multiply an input value by a memory value* $(\mathsf{id}, y_u \leftarrow w_i \cdot y_j)$*,*
- *Output value from memory as an element of* $\mathbb{Z}_\beta$ $(\mathsf{id}, \beta, \mathsf{out}_j \leftarrow y_i)$*,*

where each of the instructions has its unique identifier id*. If the size of any memory value exceeds* M *at any point of the execution, the program's output on the corresponding input is defined to be* $\perp$*. Otherwise the output is defined to be a sequence of values* $\mathsf{out}_j \in \mathbb{Z}_\beta$*, outputed by the instructions of type* $(\mathsf{id}, \beta, \mathsf{out}_j \leftarrow y_i)$*, sorted by* id*.*

1.3 Distributed Discrete Logarithm problem

In this section, we review the notion of *distributed discrete logarithm* (DDLog) problem and we give a brief explanation on how the DDLog problem is connected to HSS. Then, we focus on the DDLog problem for prime order groups. We also recall the concept of the generic group model and formally define the DDLog

problem for prime order groups in the generic group model.

Let $(\mathbb{G}, \cdot)$ be a group in the multiplicative notation and $\omega \in \mathbb{G}$. The distributed discrete logarithm problem is a problem of transforming multiplicative shares of some secret m, to its additive shares. More precisely, two players, P_0 and P_1, are holding $h_0 \in \mathbb{G}$ and $h_1 \in \mathbb{G}$ respectively, such that $\frac{h_1}{h_0} = \omega^m$ for some given subgroup generator ω. The players want to convert their shares, without any communication in between them, to shares a_0, a_1, such that $a_1 - a_0 = m$.

Solving the distributed discrete logarithm problem efficiently is essential for some of the constructions of HSS, for example, the constructions from [BGI16] and [OSY21]. In these constructions, during the homomorphic evaluation, the parties hold encryptions of the secret inputs with some public-key encryption scheme (e.g., ElGamal encryption in [BGI16] and Paillier encryption in [OSY21]), as well as the additive shares of the inputs. After performing the multiplication operation *Multiply an input value by a memory value* of the RMS program, instead of additive secret shares of the result, the parties naturally obtain its multiplicative secret shares. For further evaluation of the RMS program, the additive shares are needed, and this is where the DDLog problem plays its role. By solving the distributed discrete logarithm problem, the parties convert their multiplicative shares h_0, h_1 to the shares a_0, a_1 such that $a_1 - a_0 = m$, which are the additive shares of m.

In the case of [BGI16], the underlying group is a prime order group and the element ω is its generator. We examine the DDLog problem in this case in more detail.

Distributed Discrete Logarithm problem for prime order groups. Let $(\mathbb{G}, \cdot)$ be a cyclic group of prime order N with a generator g. In the distributed discrete logarithm problem in $\mathbb{G}$ on interval of size $W \in \mathbb{N}, W \leq N$ two independent players $P^{(0)}$ and $P^{(1)}$, who cannot communicate, get as input g^b and g^{b+x} respectively, where $x \leftarrow \{x \in \mathbb{Z} \cap [-W/2, W/2]\}$ and $b \leftarrow \mathbb{Z}_N$, while the representation of the group $\mathbb{G}$ is known by both players. At the end of their execution, the players output elements $P^{(0)}(g^b), P^{(1)}(g^{b+x}) \in \mathbb{Z}_N$ respectively. We say the players $P^{(0)}, P^{(1)}$ solved the distributed discrete logarithm problem, if $P^{(1)}(g^{b+x}) - P^{(0)}(g^b) = x$. We say that the error event occurred in the solution of DDLog problem if $P^{(1)}(g^{b+x}) - P^{(0)}(g^b) \neq x$.

In order to solve the DDLog problem, [BGI16] proposed a DDLog procedure ([BGI16, Algorithm 1]) resulting in the error probability δ, with the time complexity $O(\frac{W \log(1/\delta)}{\delta})$ group operations [BGI16, Proposition 3.2]. Their procedure selects a set of special group elements (using a pseudorandom function) and each of the players searches through T points closest to his input, in sense of multiplication by the group generator g. If the player finds a special point, the output of the DDLog procedure on his input will be the number of steps (multiplication by g) he made until finding the special point. It is clear, that if both of the players find the same special point, $P^{(0)}(g^b) - P^{(1)}(g^{b+x}) = x$ will hold. To be consistent with our definition of solution of the DDLog problem, let the players output minus the number of steps made, then $P^{(1)}(g^{b+x}) - P^{(0)}(g^b) = x$. Therefore, the success probability of this DDLog procedure is exactly the probability of

the players synchronizing on the same special point. In [BGI16], the error probability introduced by the DDLog procedure propagates throughout the entire HSS scheme and subsequently affects its efficiency.

The DDLog problem in prime order groups has been studied in detail in [DKK20]. The authors proposed a more sophisticated DDLog procedure resulting in the error probability $O(W/T^2)$ ([DKK20, Theorem 2, Corollary 1]), where T denotes the number of group-operations performed. Their procedure consists of iterating random walks with carefully chosen parameters of maximal step length and number of steps in each stage, in the way that they continuously reduce the probability of the two players not synchronizing on their path.

The authors also analysed the limitations of the DDLog problem. They proved, by a reduction of *the discrete logarithm on the interval* (DLI) problem to the DDLog problem, that in specific families of groups, where the DLI problem is difficult, the error probability of DDLog is $\Omega(W/T^2)$ [DKK20, Theorem 3].

[DKK20] also analyzed the DDLog problem in the generic group model, the model, which inhibits the attacker from exploiting a particular structure and properties of the related group. In [DKK20, Theorem 5] they showed that the error probability of any generic DDLog procedure is $\Omega(W/T^2)$.

These two results gave the lower bound to the error probability in the DDLog problem, matching the upper bound of their DDLog procedure. Therefore, this procedure can be considered as optimal. The error probability $\Omega(W/T^2)$ also implies a non-negligible correctness error in the HSS construction from [BGI16].

Nevertheless, these results do not rule out DDLog procedures exploiting particular properties of some groups, where the DLI is not a difficult problem, or where the order of the underlying group is not prime. This is the case of the DDLog procedure and a subsequent HSS construction in [OSY21] from the Paillier encryption scheme, as well as our DDLog procedure and HSS construction presented in Chapter 3.

Now we describe the generic group model we consider, and we define the DDLog problem in this generic group model. The generic group model was introduced by Shoup in [Sho97], we use a version from [CDG18] with small adjustments.

The Generic Group Model. Let $\mathbb{G}$ be a cyclic group of prime order N with a generator g. In the generic group model, we assume that the structure of $\mathbb{G}$ is random, in the sense that the adversary does not have more information about the group structure than he would have if the group was given by a random injective function $\sigma : \mathbb{Z}_N \to [M]$. This means that the elements of $\mathbb{G}$ are represented by the elements from $\mathrm{Im}(\sigma)$. More precisely, for $a \in \mathbb{Z}_N$, the representation of the group element g^a is $\sigma(a)$. We call the mapping σ *an encoding function.*

Definition 8 (Encoding function)**.** *Let $N, M \in \mathbb{N}, M \geq N$ and consider $\mathbb{Z}_N$, the additive group of integers modulo N, and the set $[M] = \{1, \ldots, M\}$. An encoding function σ of $\mathbb{Z}_N$ on $[M]$ is an injective mapping from $\mathbb{Z}_N$ to $[M]$. We denote $\mathcal{I}_{N,M}$ the set of all encoding functions of $\mathbb{Z}_N$ on $[M]$ and $\mathcal{Y}_\sigma$ the range of σ.* [1]

[1] We omit the subscript σ in $\mathcal{Y}_\sigma$, if the mapping is clear from the context.

In order to allow an adversary to perform group operations, the adversary in the generic group model is granted an access to a group operation-oracle $\mathcal{O}$, which chooses a random encoding function $\sigma \leftarrow \mathcal{I}_{N,M}$ at the beginning of an experiment and then answers adversary's queries of the following types:

Forward query
A query $a \in \mathbb{Z}_N$ is answered by $\sigma(a) \in [M]$.

Group-operation query
A query (s_1, s_2), where $s_1, s_2 \in \mathcal{Y}_\sigma$ is answered by $\sigma(x_1 + x_2)$, where x_1, x_2 are elements of $\mathbb{Z}_N$ such that $\sigma(x_1) = s_1$ and $\sigma(x_2) = s_2$. If any of s_1, s_2 is not in $\mathcal{Y}_\sigma$, the query is answered by $\perp$.

Inverse query
A query s, where $s \in \mathcal{Y}_\sigma$ is answered by $\sigma(-x)$, where $x \in \mathbb{Z}_N$ is the preimage of s, i.e., $\sigma(x) = s$. If $s \notin \mathcal{Y}_\sigma$, the query is answered by $\perp$.

In the generic group model we measure the time complexity of the attacker by the number of oracle queries performed during its execution.

We define an unpredictability application [CDG18], which captures in the generic group model the problem of guessing some secret information x given a challenge from a challenge space $\mathcal{CH}_x$.

Definition 9 (Unpredictability application)**.** *Let $\lambda \in \mathbb{N}$ be a security parameter. An* unpredictability application *G in the oracle $\mathcal{O}$-model is defined by a space of secrets $\mathcal{X}$, a set of challenge spaces $\{\mathcal{CH}_x \mid x \in \mathcal{X}\}$, a PPT challenger $\mathcal{C}$ with oracle access and an oracle $\mathcal{O}$. We define the advantage of $\mathcal{A}$ on G, denoted $\mathsf{Adv}_{\mathcal{A}}^{G^{\mathcal{O}}}(\lambda)$, as the probability of success of $\mathcal{A}$ in the unpredictability experiment $\mathsf{Exp}_{\mathcal{A}}^{G^{\mathcal{O}}}(1^\lambda)$ (Figure 4), i.e.,*

$$\mathsf{Adv}_{\mathcal{A}}^{G^{\mathcal{O}}}(\lambda) = \Pr[1 \leftarrow \mathsf{Exp}_{\mathcal{A}}^{G^{\mathcal{O}}}(1^\lambda)].$$

We define the probability of error of $\mathcal{A}$ in G to be the probability of the output of the unpredictability experiment $\mathsf{Exp}_{\mathcal{A}}^{G^{\mathcal{O}}}(1^\lambda)$ being 0, i.e.,

$$\mathsf{Err}_{\mathcal{A}}^{G^{\mathcal{O}}}(\lambda) = 1 - \mathsf{Adv}_{\mathcal{A}}^{G^{\mathcal{O}}}(\lambda).$$

We say $\mathcal{A}$ runs in the time T if it performs at most T oracle queries.

The distributed discrete logarithm problem can also be seen as a problem of guessing secret information, given a challenge from some challenge space dependent on the secret. In particular, the attacker tries to guess x given the challenge of the form g^b, g^{b+x}. Nevertheless, the unpredictability application does not provide a good representation for this problem, as it carries several differences compared to it. Especially, the attacker in the distributed discrete logarithm problem is composed of two algorithms who cannot communicate and each of these algorithms only gets to see half of the challenge, moreover, to solve the DDLog problem, is to get the secret x secret-shared between these two algorithms, not known explicitly by any of them. We call this special type of attacker a *distributed attacker* and we represent similar problems by *distributed unpredictability application*.

$\mathsf{Exp}_{\mathcal{A}}^{G^{\mathcal{O}}}(1^\lambda)$:

1. The challenger $\mathcal{C}$ generates a secret x from $\mathcal{X}$ and a challenge c from the challenge space $\mathcal{CH}_x$. It forwards $(1^\lambda, c)$ to $\mathcal{A}$.
2. On input $(1^\lambda, c)$ the attacker $\mathcal{A}$ makes queries to oracle $\mathcal{O}$ and receives answers from the oracle.
3. The attacker $\mathcal{A}$ chooses a guess $x' \in \mathcal{X}$ and sends it to $\mathcal{C}$.
4. The output of the experiment is defined to be 1 if $x = x'$, otherwise the output of the experiment is 0.

Figure 4 − Unpredictability experiment.

Definition 10 (Distributed attacker)**.** *A* distributed attacker $(\mathcal{A}^{(0)}, \mathcal{A}^{(1)})$ *is a pair of independent PPT* $\mathcal{A}^{(0)}$ *and* $\mathcal{A}^{(1)}$*, who cannot communicate. We say the distributed attacker runs in the time* T *if each of* $\mathcal{A}^{(0)}$ *and* $\mathcal{A}^{(1)}$ *does not make more than* T *oracle queries during its execution.*

Definition 11 (Distributed unpredictability application)**.** *Let* $\lambda \in \mathbb{N}$ *be a security parameter. A* distributed unpredictability application G *in the oracle* $\mathcal{O}$*-model is defined by a space of secrets* $\mathcal{X}$*, a set of challenge spaces* $\{\mathcal{CH}_x \mid x \in \mathcal{X}\}$*, where the elements of* $\mathcal{CH}_x$ *are of form* (c_0, c_1)*, a PPT challenger* $\mathcal{C}$ *with oracle access and an oracle* $\mathcal{O}$*. We define the advantage of the distributed attacker* $\mathcal{A} = (\mathcal{A}^{(0)}, \mathcal{A}^{(1)})$ *on* G*, denoted* $\mathsf{Adv}_{\mathcal{A}}^{G^{\mathcal{O}}}(\lambda)$*, as the probability of success of* $\mathcal{A}$ *in the distributed unpredictability experiment* $\mathsf{Exp}_{\mathcal{A}}^{G^{\mathcal{O}}}(1^\lambda)$ *(Figure 5), i.e.,*

$$\mathsf{Adv}_{\mathcal{A}}^{G^{\mathcal{O}}}(\lambda) = \Pr[1 \leftarrow \mathsf{Exp}_{\mathcal{A}}^{G^{\mathcal{O}}}(1^\lambda)].$$

We define the probability of error of $\mathcal{A}$ *in* G *to be the probability of the output of the unpredictability experiment* $\mathsf{Exp}_{\mathcal{A}}^{G^{\mathcal{O}}}(1^\lambda)$ *being 0, i.e.,*

$$\mathsf{Err}_{\mathcal{A}}^{G^{\mathcal{O}}}(\lambda) = 1 - \mathsf{Adv}_{\mathcal{A}}^{G^{\mathcal{O}}}(\lambda).$$

Now, we define the distributed discrete logarithm problem in the generic group model.

Definition 12 (Distributed discrete logarithm application)**.** *Let* $\lambda \in \mathbb{N}$ *be a security parameter, let* $N, W \in \mathbb{N}$*,* $N > W$ *and* $\mathcal{O}$ *be a group operation oracle. The* (N, W) *- distributed discrete logarithm application* $G_{\mathsf{DDLog}}^{\mathcal{O}}(N, W)$ *is a distributed unpredictability application in* $\mathcal{O}$*-model, where the space of secrets is* $\mathcal{X} = \{x \mid x \in \mathbb{Z} \cap [-W/2, W/2]\}$*, the challenge space for* $x \in \mathcal{X}$ *is defined as* $\mathcal{CH}_x = \{(\sigma(b), \sigma(b+x)) \mid \sigma \in \mathcal{I}_{N,M}, b \in \mathbb{Z}_N\}$*, and the challenger* C^{DDLog} *is a PPT that samples* $b \leftarrow \mathbb{Z}_N$ *and* x *from the set of integers in the interval* $[-W/2, W/2]$ *uniformly at random. Then* C^{DDLog} *makes two forward queries* $b, b+x$ *to the oracle and passes* $(1^\lambda, \sigma(b))$ *as a challenge to* $\mathcal{A}^{(0)}$ *and* $(1^\lambda, \sigma(b+x))$ *as a challenge to* $\mathcal{A}^{(1)}$*.*

$\mathsf{Exp}_{\mathcal{A}}^{G^{\mathcal{O}}}(1^{\lambda})$:

1. The challenger $\mathcal{C}$ generates a secret x from $\mathcal{X}$ and a challenge (c_0, c_1) from the challenge space $\mathcal{CH}_x$. It forwards $(1^{\lambda}, c_0)$ to $\mathcal{A}^{(0)}$ and $(1^{\lambda}, c_1)$ to $\mathcal{A}^{(1)}$.
2. For $i = 0, 1$:
   (a) On input $(1^{\lambda}, c_i)$ the attacker $\mathcal{A}^{(i)}$ makes queries to oracle $\mathcal{O}$ and receives answers from the oracle.
   (b) The attacker $\mathcal{A}^{(i)}$ chooses a guess $x_i \in \mathcal{X}$ and sends it to $\mathcal{C}$.
3. The output of the experiment is defined to be 1 if $x_1 - x_0 = x$, otherwise the output of the experiment is 0.

Figure 5 – Distributed unpredictability experiment.

2. The Distributed Discrete Logarithm problem with preprocessing

In this chapter, we focus on the distributed discrete logarithm problem for prime order groups in the preprocessing model. We examine to what extent the preprocessing can help the attacker to solve the distributed discrete logarithm problem. In other words, we will allow the attacker to precompute a bitstring of length S in the *offline phase*, i.e., before receiving the challenge, without any restriction on the time of its computation. Then, the precomputed *advice* bitstring is passed as an additional input to the attacker running in the *online phase*, i.e., after receiving the challenge. Regarding the time complexity of the attacker, we are only interested in the time complexity of its online phase. We examine whether the additional *advice* computed during the offline phase can help to reduce the error probability in the DDLog problem. We study this question in the context of generic groups, that is, in the generic group model.

2.1 Preprocessing in the generic group model

In this section, we introduce the definitional framework of the generic group model for the algorithms with preprocessing. The framework is adapted from [CDG18] with some small adjustments in order to capture the distributed attacker. First, we define the preprocessing oracle.

Definition 13 (Preprocessing oracle)**.** *The* preprocessing oracle $\mathcal{O}$ *is an oracle formed by a pair of oracles* $(\mathcal{O}_{\mathsf{pre}}, \mathcal{O}_{\mathsf{main}})$*, where* $\mathcal{O}_{\mathsf{pre}}$ *can only be queried during the offline phase of an experiment, i.e., before the challenge is generated and* $\mathcal{O}_{\mathsf{main}}$ *can only be queried in the online phase of an experiment.*

The attacker in the preprocessing model is composed of two algorithms, one of them running in the offline phase and having access to $\mathcal{O}_{\mathsf{pre}}$, the other one running in the online phase and having an access to $\mathcal{O}_{\mathsf{main}}$.

Definition 14 ((S,T)-attacker, [CDG18, Definition 1])**.** *Let* $S, T \in \mathbb{N}$ *and* $\mathcal{O} = (\mathcal{O}_{\mathsf{pre}}, \mathcal{O}_{\mathsf{main}})$ *be a preprocessing oracle. An* (S,T)*-attacker* $\mathcal{A} = (\mathcal{A}_0, \mathcal{A}_1)$ *in the* $\mathcal{O}$*-model consist of two probabilistic algorithms:*

A preprocessing algorithm $\mathcal{A}_0$*, which is computationally unbounded and which interacts with* $\mathcal{O}_{\mathsf{pre}}$ *and outputs a bitstring of length at most* S *bits.*

An online algorithm $\mathcal{A}_1$*, which takes as input an* S*-bit output of* $\mathcal{A}_0$ *and a challenge from the challenger, then makes at most* T *queries to* $\mathcal{O}_{main}$*, and outputs a guess.*

Now, we define the notions of *unpredictability application* and *distributed unpredictability application* in the preprocessing model. The following definitions (Definition 15, 16, 17) correspond to Definition 9, 10 and 11 extended to the preprocessing model. The (S,T)*-attacker* and the *unpredictability application with*

preprocessing correspond to the model used by [CDG18], whereas the *distributed* (S,T)*-attacker* and the *distributed unpredictability application with preprocessing* allow us to model the DDLog problem with preprocessing in the generic group model.

The following definition formalizes the problem of guessing secret information x given a challenge from a related challenge space $\mathcal{CH}_x$, while we allow the attacker to precompute advice string before receiving the challenge.

Definition 15 (Unpredictability application with preprocessing)**.** *Let $\lambda \in \mathbb{N}$ be a security parameter. An* unpredictability application with preprocessing *G in the oracle $(\mathcal{O}_{\mathsf{pre}}, \mathcal{O}_{\mathsf{main}})$-model is defined by a space of secrets $\mathcal{X}$, a set of challenge spaces $\{\mathcal{CH}_x \mid x \in \mathcal{X}\}$, a PPT challenger $\mathcal{C}$ with oracle access to $\mathcal{O}_{\mathsf{main}}$ and a preprocessing oracle $(\mathcal{O}_{\mathsf{pre}}, \mathcal{O}_{\mathsf{main}})$. We define the advantage of $\mathcal{A}$ on G, denoted $\mathsf{Adv}_{\mathcal{A}}^{G^{(\mathcal{O}_{\mathsf{pre}},\mathcal{O}_{\mathsf{main}})}}(\lambda)$, as the probability of success of $\mathcal{A}$ in the unpredictability experiment with preprocessing $\mathsf{Exp}_{\mathcal{A}}^{G^{(\mathcal{O}_{\mathsf{pre}},\mathcal{O}_{\mathsf{main}})}}(1^\lambda)$ (Figure 6), i.e.,*

$$\mathsf{Adv}_{\mathcal{A}}^{G^{(\mathcal{O}_{\mathsf{pre}},\mathcal{O}_{\mathsf{main}})}}(\lambda) = \Pr[1 \leftarrow \mathsf{Exp}_{\mathcal{A}}^{G^{(\mathcal{O}_{\mathsf{pre}},\mathcal{O}_{\mathsf{main}})}}(1^\lambda)].$$

We define the probability of error of $\mathcal{A}$ in G to be the probability of the output of the unpredictability experiment with preprocessing $\mathsf{Exp}_{\mathcal{A}}^{G^{(\mathcal{O}_{\mathsf{pre}},\mathcal{O}_{\mathsf{main}})}}(1^\lambda)$ being 0, i.e.,

$$\mathsf{Err}_{\mathcal{A}}^{G^{(\mathcal{O}_{\mathsf{pre}},\mathcal{O}_{\mathsf{main}})}}(\lambda) = 1 - \mathsf{Adv}_{\mathcal{A}}^{G^{(\mathcal{O}_{\mathsf{pre}},\mathcal{O}_{\mathsf{main}})}}(\lambda).$$

We say an unpredictability application with preprocessing G is $((S,T),\epsilon)$-secure in the $(\mathcal{O}_{\mathsf{pre}}, \mathcal{O}_{\mathsf{main}})$-model if for every (S,T)-attacker $\mathcal{A}$ for every $\lambda \in \mathbb{N}$ it holds that $\mathsf{Adv}_{\mathcal{A}}^{G^{(\mathcal{O}_{\mathsf{pre}},\mathcal{O}_{\mathsf{main}})}}(\lambda) \leq \epsilon(\lambda)$.

As the online attacker in the DDLog problem is not a single algorithm, yet an attacker composed of two algorithms who cannot communicate, we need to introduce a definition of such attacker in the preprocessing model.

Definition 16 (Distributed (S,T)-attacker)**.** *Let $S, T \in \mathbb{N}$ and $\mathcal{O} = (\mathcal{O}_{\mathsf{pre}}, \mathcal{O}_{\mathsf{main}})$ be a preprocessing oracle. A distributed (S,T)-attacker $\mathcal{A} = (\mathcal{A}_0, \mathcal{A}_1^{(0)}, \mathcal{A}_1^{(1)})$ in the $\mathcal{O}$-model consist of three probabilistic algorithms:*

A preprocessing algorithm $\mathcal{A}_0$, which is computationally unbounded and which interacts with $\mathcal{O}_{pre}$ and outputs a bitstring of length S bits.

Two independent online algorithms $\mathcal{A}_1^{(0)}, \mathcal{A}_1^{(1)}$, which cannot communicate and each of which takes as input an S-bit output of $\mathcal{A}_0$ and a challenge, then makes at most T queries to $\mathcal{O}_{main}$, and outputs a guess.

Now, we define the *distributed unpredictability application with preprocessing*, which formalizes the problem of the distributed (S,T)-attacker's online algorithms guessing additive shares of secret information x, given a challenge from the related challenge space $\mathcal{CH}_x$.

Definition 17 (Distributed unpredictability application with preprocessing)**.** *Let $\lambda \in \mathbb{N}$ be a security parameter. A* distributed unpredictability application with preprocessing *G in the oracle $(\mathcal{O}_{\mathsf{pre}}, \mathcal{O}_{\mathsf{main}})$-model is defined by a space of secrets $\mathcal{X}$, a set of challenge spaces $\{\mathcal{CH}_x \mid x \in \mathcal{X}\}$, where the elements of $\mathcal{CH}_x$ are of the form (c_0, c_1), a PPT challenger $\mathcal{C}$ with oracle access to $\mathcal{O}_{\mathsf{main}}$ and a preprocessing*

$\mathsf{Exp}_{\mathcal{A}}^{G^{(\mathcal{O}_{\mathsf{pre}},\mathcal{O}_{\mathsf{main}})}}(1^\lambda)$:

1. The attacker $\mathcal{A}_0$ makes queries to oracle $\mathcal{O}_{\mathsf{pre}}$ and receives answers from the oracle.
2. At the end of its execution, $\mathcal{A}_0$ outputs an advice bitstring adv of maximal length S bits and forwards adv to the online phase attacker $\mathcal{A}_1$.
3. The challenger $\mathcal{C}$ generates a secret x from the space $\mathcal{X}$ and a challenge c from the challenge space $\mathcal{CH}_x$. It forwards $(1^\lambda, c)$ to $\mathcal{A}_1$.
4. On input $(1^\lambda, \mathsf{adv}, c)$, the attacker $\mathcal{A}_1$ makes queries to the oracle $\mathcal{O}_{\mathsf{main}}$ and receives answers from the oracle.
5. The attacker $\mathcal{A}_1$ chooses a guess $x' \in \mathcal{X}$ and sends it to $\mathcal{C}$.
6. The output of the experiment is defined to be 1 if $x = x'$, otherwise the output of the experiment is 0.

Figure 6 – Unpredictability experiment with preprocessing.

oracle $(\mathcal{O}_{\mathsf{pre}}, \mathcal{O}_{\mathsf{main}})$. *We define the advantage of a distributed* (S,T)*-attacker* $\mathcal{A} = (\mathcal{A}_0, \mathcal{A}_1^{(0)}, \mathcal{A}_1^{(1)})$ *on* G*, denoted* $\mathsf{Adv}_{\mathcal{A}}^{G^{(\mathcal{O}_{\mathsf{pre}},\mathcal{O}_{\mathsf{main}})}}(\lambda)$*, as the probability of success of* $\mathcal{A}$ *in the distributed unpredictability experiment with preprocessing* $\mathsf{Exp}_{\mathcal{A}}^{G^{(\mathcal{O}_{\mathsf{pre}},\mathcal{O}_{\mathsf{main}})}}(1^\lambda)$ *(Figure 7), i.e.,*

$$\mathsf{Adv}_{\mathcal{A}}^{G^{(\mathcal{O}_{\mathsf{pre}},\mathcal{O}_{\mathsf{main}})}}(\lambda) = \Pr[1 \leftarrow \mathsf{Exp}_{\mathcal{A}}^{G^{(\mathcal{O}_{\mathsf{pre}},\mathcal{O}_{\mathsf{main}})}}(1^\lambda)].$$

We define the probability of error of $\mathcal{A}$ *in* G *to be the probability of the output of the unpredictability experiment with preprocessing* $\mathsf{Exp}_{\mathcal{A}}^{G^{(\mathcal{O}_{\mathsf{pre}},\mathcal{O}_{\mathsf{main}})}}(1^\lambda)$ *being 0, i.e.,*

$$\mathsf{Err}_{\mathcal{A}}^{G^{(\mathcal{O}_{\mathsf{pre}},\mathcal{O}_{\mathsf{main}})}}(\lambda) = 1 - \mathsf{Adv}_{\mathcal{A}}^{G^{(\mathcal{O}_{\mathsf{pre}},\mathcal{O}_{\mathsf{main}})}}(\lambda).$$

We say a distributed unpredictability application with preprocessing G *is* $((S,T),\epsilon)$*-secure in the* $(\mathcal{O}_{\mathsf{pre}}, \mathcal{O}_{\mathsf{main}})$*-model if for every distributed* (S,T)*-attacker* $\mathcal{A}$*, for every* $\lambda \in \mathbb{N}$ *it holds that* $\mathsf{Adv}_{\mathcal{A}}^{G^{(\mathcal{O}_{\mathsf{pre}},\mathcal{O}_{\mathsf{main}})}}(\lambda) \leq \epsilon(\lambda)$.

Now, we define the DDLog problem with preprocessing in the generic group model.

Definition 18 (Distributed discrete logarithm application with preprocessing)**.** *Let* $\lambda \in \mathbb{N}$ *be a security parameter, let* $N, W \in \mathbb{N}$*,* $N > W$*. The* (N,W)*-distributed discrete logarithm application with preprocessing* $G_{\mathsf{DDLog}}^{(\mathcal{O}_{\mathsf{pre}},\mathcal{O}_{\mathsf{main}})}(N,W)$ *is a distributed unpredictability application with preprocessing in* $(\mathcal{O}_{\mathsf{pre}}, \mathcal{O}_{\mathsf{main}})$*-model, where* $\mathcal{O}_{\mathsf{pre}}$ *is an oracle that samples* $\sigma \leftarrow \mathcal{I}_{N,M}$ *at the beginning of the experiment and* $\mathcal{O}_{\mathsf{main}}$ *is a group operation oracle for the encoding functon* σ*, the space of secrets is* $\mathcal{X} = \{x \mid x \in \mathbb{Z} \cap [-W/2, W/2]\}$*, the challenge space for* $x \in \mathcal{X}$ *is defined as* $\mathcal{CH}_x = \{(\sigma(b), \sigma(b+x)) \mid b \in \mathbb{Z}_N\}$*. The challenger* C^{DDLog} *is a PPT*

$\mathsf{Exp}_{\mathcal{A}}^{G^{(\mathcal{O}_{\mathsf{pre}},\mathcal{O}_{\mathsf{main}})}}(1^\lambda)$:

1. The attacker $\mathcal{A}_0$ makes queries to oracle $\mathcal{O}_{\mathsf{pre}}$ and receives answers from the oracle.
2. At the end of its execution $\mathcal{A}_0$ outputs an advice bitstring adv of maximal length S bits and forwards adv to the online phase attackers $\mathcal{A}_1^{(0)}, \mathcal{A}_1^{(1)}$.
3. The challenger $\mathcal{C}$ generates a secret x from the space $\mathcal{X}$ and a challenge (c_0, c_1) from the challenge space $\mathcal{CH}_x$. It forwards $(1^\lambda, c_0)$ to $\mathcal{A}_1^{(0)}$ and $(1^\lambda, c_1)$ to $\mathcal{A}_1^{(1)}$.
4. For $i = 0, 1$:
   (a) On input $(\mathsf{adv}, 1^\lambda, c_i)$ the attacker $\mathcal{A}^{(i)}$ makes at most T queries to oracle $\mathcal{O}_{\mathsf{main}}$ and receives answers from the oracle.
   (b) The attacker $\mathcal{A}^{(i)}$ chooses a guess $x_i \in \mathcal{X}$ and sends it to $\mathcal{C}$.
5. The output of the experiment is defined to be 1 if $x_1 - x_0 = x$, otherwise, the output of the experiment is 0.

Figure 7 – Distributed unpredictability experiment with preprocessing.

that samples $x \leftarrow \mathcal{X}$ and $b \leftarrow \mathbb{Z}_N$. Then C^{DDLog} makes two forward queries $b, b+x$ to the oracle $\mathcal{O}_{\mathsf{main}}$ and passes $(1^\lambda, \sigma(b))$ as a challenge to $\mathcal{A}_1^{(0)}$ and $(1^\lambda, \sigma(b+x))$ as a challenge to $\mathcal{A}_1^{(1)}$.

2.2 Auxiliary input and Bit-fixing model

Following the technique of [CDG18], we define two different preprocessing oracles: *Auxiliary-input generic group oracle* and *Bit-fixing generic group oracle*. The auxiliary-input generic group oracle allows us to model the preprocessing experiment. Nevertheless, it seems difficult to perform an analysis of complexity directly in this model, while the bit-fixing generic group oracle offers a model that is easier to analyse. [CDG18, Theorem 1] proved a relation between an attackers' success probabilities in these two models. We state this result in Proposition 1.

The auxiliary-input generic group oracle allows modelling the preprocessing experiments for generic groups, in the sense that the interface $\mathcal{O}_{\mathsf{pre}}$ allows the offline attacker to see the entire group structure, i.e., the mapping σ. Then, $\mathcal{A}_0$ can choose a bitstring of maximal length S and pass it to the online phase attacker $\mathcal{A}_1$ as an additional input.

On the other hand, the bit-fixing generic group oracle allows the offline attacker to fix P points $(a, s) \in \mathbb{Z}_N \times \mathcal{Y}$ and the mapping σ is chosen afterwards, in the way that it respects these fixed points.

Definition 19. *We define:*

Auxiliary-input generic group oracle $\mathsf{AI\text{-}GG}(N, M)$ *as a pair* $(\mathcal{O}_{\mathsf{pre}}, \mathcal{O}_{\mathsf{main}})$*, where:*

- $\mathcal{O}_{pre}$*: Samples* $\sigma \leftarrow \mathcal{I}_{N,M}$ *and outputs* σ*.*
- $\mathcal{O}_{main}$*: Answers forward queries, group-operation queries, and inverse queries using* σ *sampled by* $\mathcal{O}_{pre}$*.*

Bit-fixing generic group oracle $\mathsf{BF\text{-}GG}(P, N, M)$ *as a pair* $(\mathcal{O}_{\mathsf{pre}}, \mathcal{O}_{\mathsf{main}})$*, where:*

- $\mathcal{O}_{pre}$*: Samples* $\mathcal{Y} \subset [M]$ *of size* N *uniformly at random, takes as input at most* $P \in \mathbb{N}$ *pairs of the form* (a, s)*,* $a \in \mathbb{Z}_N$*,* $s \in \mathcal{Y}$ *with no collisions, and samples* σ *uniformly at random from the subset of* $\mathcal{I}_{N,M}$ *containing all the injections consistent with the sampled range* $\mathcal{Y}$ *and the given fixed points.*
- $\mathcal{O}_{main}$*: Answers forward queries, group-operation queries, and inverse queries using* σ *sampled by* $\mathcal{O}_{pre}$*.*

Proposition 1 ([CDG18, Theorem 1])**.** *Let* $P, N, M \in \mathbb{N}$*,* $N \geq 16$ *and* $\gamma > 0$*. Consider a* $((S, T), \epsilon')$*-secure unpredictability application with preprocessing* G *in the* $\mathsf{BF\text{-}GG}(P, N, M)$*-model. If* $P \geq 6(S + \log \gamma^{-1}) \cdot T_G^{\mathsf{comb}}$*, then* G *is* $((S, T), \epsilon)$*-secure in the* $\mathsf{AI\text{-}GG}(N, M)$*-model for* $\epsilon \leq 2\epsilon' + \gamma$*. Where* T_G^{comb} *denotes the combined number of queries of the attacker and the challenger.*

Proof. For the proof we refer to [CDG18, Appendix A]. □

We remark that Proposition 1 was formulated in [CDG18] only for $((S, T), \epsilon)$-secure unpredictability applications with preprocessing, the distributed applications were not considered. We are interested in proving upper bounds on the success probability of an attacker in the *distributed* discrete logarithm application with preprocessing, which is a distributed unpredictability application with preprocessing. If we applied Proposition 1 to the DDLog problem directly, we would be forced to represent the distributed attacker as a non-distributed attacker in the $\mathsf{BF\text{-}GG}(P, N, M)$-model, i.e., an attacker with an online algorithm that gets both challenges and performs up to $2T$ oracle queries. Then, we would apply the theorem on this stronger attacker and we would obtain the bounds in the $\mathsf{AI\text{-}GG}(N, M)$-model. Thus, the clear disadvantage of the distributed attacker of having the challenge given to two separate algorithms that are not allowed to communicate cannot be exploited in the $\mathsf{BF\text{-}GG}(P, N, M)$-model before applying the Proposition 1. Overall, this approach leads to loose bounds on the success probability of the distributed attacker. Our central observation is that the theorem holds in the same way for a distributed attacker.

We also remark that the generic group model used in [CDG18] does not allow the attacker to perform inverse queries. This approach is justified by the fact that the authors apply Proposition 1 to derive bounds precise up to a polylogarithmic factor (polynomial in $\log N$). The inverse of an element x in a group of order N is equal to x^{N-1}. Therefore, applying the Square and Multiply algorithm, we can simulate the inverse operation using $O(\log N)$ group-operations. Thus, the analysis in a version of the generic group model without the inverse queries

translates to the result precise up to a polylogarithmic factor in a model, where these queries are allowed. However, we seek to get bounds for the DDLog problem without neglecting the logarithmic factors, and, therefore, we explicitly include the inverse query in our generic group model. We note that Proposition 1 holds also in our version of the generic group model. We explain this in more detail in the proof sketch of Theorem 2.

Theorem 2. *Let $P, N, M \in \mathbb{N}$, $N \geq 16$ and $\gamma > 0$. Consider a $((S,T),\epsilon')$-secure distributed unpredictability application with preprocessing G in the $\mathsf{BF\text{-}GG}(P,N,M)$-model. If $P \geq 6(S+\log\gamma^{-1})\cdot T_G^{\mathsf{comb}}$, then G is $((S,T),\epsilon)$-secure in the $\mathsf{AI\text{-}GG}(N,M)$-model for $\epsilon \leq 2\epsilon' + \gamma$. Where T_G^{comb} denotes the combined number of queries of the attacker and the challenger.*

Proof sketch. The proof follows from the proof of Proposition 1 stated in [CDG18, Appendix A] replacing the (S,T)-attacker with preprocessing by a distributed (S,T)-attacker with preprocessing.

In order to prove [CDG18, Theorem 1], the authors first prove closeness of two distributions of encoding functions, in the sense that they bound the probability that a distinguisher which is allowed to make T forward (query $a \in \mathbb{Z}_N$ is answered by $\sigma(a)$) and backward (query $l \in [M]$ is answered by $\sigma^{-1}(l)$) oracle queries succeeds to guess from which distribution the encoding function σ of $\mathbb{Z}_N$ on $[M]$ was chosen. This proof is general and can be applied in the same manner in the setting with distributed attackers.

Then, to prove Proposition 1, they construct an (S,T)-attacker $\mathcal{A}' = (\mathcal{A}'_0, \mathcal{A}'_1)$ for the $\mathsf{BF\text{-}GG}(P,N,M)$ oracle model from an (S,T)-attacker $\mathcal{A} = (\mathcal{A}_0, \mathcal{A}_1)$ for the $\mathsf{AI\text{-}GG}(N,M)$ oracle model. Where $\mathcal{A}'_1$ is defined as $\mathcal{A}_1$ and $\mathcal{A}'_0$ first simulates $\mathcal{A}_0$ to get the advice string and, based on it, it fixes at most P points in the encoding function and, thus, forces the oracle to choose the encoding function from a convenient distribution. Then, they let the distinguisher $\mathcal{D}$ for the encoding function, taking as an input the advice string calculated by $\mathcal{A}_0$, internally run the online algorithm $\mathcal{A}_1$ and the challenger $\mathcal{C}$ for the unpredictability experiment with preprocessing, the output of the distinguisher being defined as the output of the unpredictability experiment with preprocessing resulting from the interaction of $\mathcal{A}_1$ and $\mathcal{C}$. The probability of the distinguisher outputting 1 corresponds either to the probability of success of $\mathcal{A}$ in the $\mathsf{AI\text{-}GG}(N,M)$ oracle model or to the probability of success of $\mathcal{A}'$ in the $\mathsf{BF\text{-}GG}(N,M)$ oracle model, depending on which distribution was σ chosen from. Thanks to the bound on the distinguishing probability between the two aforementioned distributions, they get a relation between the success probabilities of $\mathcal{A}$ in the $\mathsf{AI\text{-}GG}(N,M)$-model and $\mathcal{A}'$ in the $\mathsf{BF\text{-}GG}(P,N,M)$-model stated in the theorem, which concludes their proof.

If we replace the attacker $\mathcal{A}$ by a distributed attacker and define $\mathcal{A}'$ in the same way as in the original proof, $\mathcal{A}'$ will also be a distributed attacker (as the online algorithms of $\mathcal{A}$ and $\mathcal{A}'$ are defined to be the same) and we can perform the proof in the very same fashion as in [CDG18] and get the same results for the distributed attacker.

The proof of Proposition 1 can also be adapted to the version of generic group model which allows inverse queries. When the distinguisher $\mathcal{D}$ simulates $\mathcal{A}_1$ and $\mathcal{C}$, it must provide the answers to their oracle queries. In the proof, [CDG18] only deals with the forward query, which $\mathcal{D}$ passes as a forward query to its oracle and

passes the answer back, and the group-operation query, which is a query of the form $(a_1, a_2) \in [M]^2$, answered by $\sigma(\sigma^{-1}(a_1) + \sigma^{-1}(a_2))$. The group-operation query is simulated by $\mathcal{D}$ by performing two backward queries $\sigma^{-1}(a_1), \sigma^{-1}(a_2)$ and one forward query $\sigma(\sigma^{-1}(a_1) + \sigma^{-1}(a_2))$ to its oracle. In our case, we have to deal also with the inverse query, which is a query of the form $a \in [M]$, answered by the element corresponding to the inverse of a. This query can be simulated as one backward query $\sigma^{-1}(a)$ and one forward query $\sigma(-\sigma^{-1}(a))$. As in the original proof, our distinguisher makes at most $3T^{\mathsf{comb}}$ queries, where T^{comb} is the combined number of queries of $\mathcal{A}_1$ and $\mathcal{C}$. The rest of the proof is the same, and the same results follow. □

2.3 Lower bounds for DDLog with preprocessing in the generic group model

First, we review the Schwartz-Zippel lemma, we use the version from [CDG18, Lemma 37], and the standard Boole's inequality, known also as the union bound, that we will need later.

Lemma 3 (Schwartz-Zippel)**.** *Let $\mathbb{F}$ be a field. Let $f \in \mathbb{F}[X_1, \ldots, X_k]$ be a non-zero polynomial of total degree $d \geq 0$. Let S be a finite subset of $\mathbb{F}$ and let $x_1, \ldots, x_k$ be chosen independently uniformly at random from S. Then it holds*

$$\Pr[f(x_1, \ldots, x_k) = 0] \leq \frac{d}{|S|}.$$

Lemma 4 (Union Bound)**.** *Let $n \in \mathbb{N}$ and consider events $E_1, \ldots, E_n$, then*

$$\Pr\left[\bigcup_{i=1}^{n} E_i\right] \leq \sum_{i=1}^{n} \Pr\left[E_i\right].$$

Now, we present our main theorem giving an upper bound on the success probability of a distributed attacker with preprocessing in the DDLog problem. Our theorem is based on [CDG18, Theorem 10], which examines the discrete logarithm problem in the preprocessing model. The structure of our proof is similar to the one of [CDG18, Theorem 10]. If we simply adjusted the proof of [CDG18, Theorem 10] to our case, we would get the bound for the success probability of an attacker $\epsilon = O\left(\frac{ST^2+T^2\cdot\log(W)}{W}\right)$.

Nevertheless, the distributed attacker allows us to make more a thoughtful analysis and obtain better results. As the distributed attacker is a weaker attacker than its non-distributed representation, it allows us to obtain tighter bound in the $\mathsf{BF\text{-}GG}(P, N, M)$ oracle model. Then, we translate this bound to the $\mathsf{AI\text{-}GG}(N, M)$ oracle model by Theorem 2.

Theorem 5. *Let $N, W \in \mathbb{N}$, $N > W$, N be a prime. The (N, W)-distributed discrete logarithm application with preprocessing $G_{\mathsf{DDLog}}^{\mathsf{AI\text{-}GG}(N,M)}(N, W)$ is $((S, T), \epsilon)$-secure in the $\mathsf{AI\text{-}GG}(N, M)$-model for any*

$$\epsilon = O\left(\frac{T^2}{W} + \frac{\max\{S, \log(W)\} \cdot T^2}{N}\right),$$

where W denotes the length of the interval in the DDLog problem. Furthermore, if $N \geq W \cdot \log(W)$ the theorem holds for

$$\epsilon = O\left(\frac{T^2}{W} + \frac{S \cdot T^2}{N}\right).$$

Proof. We first consider the interaction of $\mathcal{A} = (\mathcal{A}_0, \mathcal{A}_1^{(0)}, \mathcal{A}_1^{(1)})$ with C^{DDLog} in the $\mathsf{BF\text{-}GG}(P, N, M)$-model. $\mathcal{O}_{\text{pre}}$ outputs $\mathcal{Y}$, the range of σ. For the rest of the proof we condition on the choice of $\mathcal{Y}$.

We define an alternative experiment:

During the experiment, we construct a table of pairs $(v(X), s)$, where $s \in \mathcal{Y}$ and $v(X) \in \mathbb{Z}_N[X, B]$ is a formal polynomial corresponding to the preimage of s under σ. To construct the table, we proceed as follows:

1. $\mathcal{A}_0$ fixes σ in at most P points (a, s), $a \in \mathbb{Z}_N$, $s \in \mathcal{Y}$. We add each such point to the table as a pair (a, s), where a is a constant polynomial.

2. To create the challenge s_b for $\mathcal{A}_1^{(0)}$, C^{DDLog} chooses s_b from all unused values in $\mathcal{Y}$ uniformly at random. The pair (B, s_b) is stored in the table.

3. The execution of $\mathcal{A}_1^{(0)}$:

   (a) For a forward query $q \in \mathbb{Z}_N$ the table is checked for the occurrence of $q \in \mathbb{Z}_N[X, B]$ as a constant polynomial. If such occurrence is found, we respond by the corresponding $s_q \in \mathcal{Y}$ that occurs in the table in a pair with q. If not, we sample s_q uniformly at random from all the unused values in $\mathcal{Y}$ and we store the pair (q, s_q) in the table.

   (b) To a group-operation query (s_1, s_2) we respond by $\perp$ if s_1 or s_2 is not in $\mathcal{Y}$. Otherwise, if s_1 is not in the table, we sample a_1 uniformly at random from all unused values in $\mathbb{Z}_N$ and we store the pair (a_1, s_1) in the table. The same applies for s_2. Afterwards, both s_1, s_2 are already stored in the table in pairs with some polynomials $u_1, u_2 \in \mathbb{Z}_N[B]$. We check the table for an occurrence of the polynomial $u_1 + u_2$. If there is a record $(u_1 + u_2, s_3)$ for some $s_3 \in \mathcal{Y}$ in the table, we respond by s_3 to the query. Otherwise we sample s_3 uniformly at random from all unused values in $\mathcal{Y}$, we store $(u_1 + u_2, s_3)$ in the table, and we respond by s_3 to the query.

   (c) To an inverse query s_1, we respond by $\perp$ if $s_1 \notin \mathcal{Y}$. If $s_1 \in \mathcal{Y}$ and s_1 is not in the table, we sample u uniformly at random from all unused values in $\mathbb{Z}_N$ and we append the pair (u, s_1) to the table. Now s_1 is in the table in pair with some polynomial $u \in \mathbb{Z}_N[B]$. We check the table for an occurrence of $(N-1) \cdot u$, if such entry $((N-1) \cdot u, s_2)$ is found for some $s_2 \in \mathcal{Y}$, we answer the query by s_2. Otherwise we sample s_2 uniformly at random from all unused labels from $\mathcal{Y}$, we answer the query by s_2 and we add the pair $((N-1) \cdot u, s_2)$ to the table.

   (d) At the end of its execution, $\mathcal{A}_1^{(0)}$ outputs x_0.

4. To create the challenge s_{b+x} for $\mathcal{A}_1^{(1)}$, C^{DDLog} chooses s_{b+x} from all unused values in $\mathcal{Y}$ uniformly at random. The pair $(B + X, s_{b+x})$ is stored in the table.

5. The execution of $\mathcal{A}_1^{(1)}$ is handled in the same way as the execution of $\mathcal{A}_1^{(0)}$. Except if

   - $\mathcal{A}_1^{(1)}$ queries a group-operation query (s_1, s_2) such that either $(\alpha \cdot B + \beta, s_1)$, or $(\alpha \cdot B + \beta, s_2)$, for some $\alpha, \beta \in \mathbb{Z}_N$, $\alpha \neq 0$ is already in the table, or
   - $\mathcal{A}_1^{(1)}$ queries an inverse query s_1 such that $(\alpha \cdot B + \beta, s_1)$ for some $\alpha, \beta \in \mathbb{Z}_N$, $\alpha \neq 0$ is already in the table.

   We will denote both of these by an event F and if the event F occurs, we answer the query by $\perp$ and we do not append anything to the table.

   At the end of its execution, $\mathcal{A}_1^{(1)}$ outputs x_1.

6. C^{DDLog} chooses x uniformly at random from all integers in $[-W/2, W/2]$ and b uniformly at random from $\mathbb{Z}_N$. C^{DDLog} outputs 1 if and only if $x_1 - x_0 = x$.

We remark that all of the polynomials in the table at the end of the execution are distinct, as we always first check for an occurrence of a polynomial before adding it to the table. We define a collision event as an event when after a substitution of the values x, b, chosen by C^{DDLog}, for the variables X, B (respectively) in the polynomials in the table, there exist two entries (a, s) and (a, s') in the table such that $s \neq s'$. We denote this event by E, which corresponds to a discrepancy in the query responses to the attacker. In other words, there is no encoding function σ such that all of our query responses would be correct because we associated the image of a with two different elements s, s'. If this event does not occur and the event F does not occur either then there exists an encoding function σ compatible with all of our query responses. Furthermore, as we always choose the elements uniformly at random from appropriate sets, also the distribution of responses in the alternative experiment is identical to the distribution of responses in the real distributed unpredictability experiment with preprocessing.

Thus, the distribution of answers seen by $\mathcal{A}$ in the alternative experiment differs from the one in the honest experiment only if at least one of the events E, F occurs. We bound the probability that the execution differs from the honest execution by bounding the probability $\Pr[E \cup F] \leq \Pr[E \mid \neg F] + \Pr[F]$.

We introduce a lemma that characterizes the structure of the contents of the table at the end of the alternative experiment.

Lemma 6. *At most $2T + 2$ non-constant polynomials are in the table at the end of the execution of the alternative experiment. Moreover,*

1. *at most $T + 1$ of those are of the form $\alpha B + \beta$, for some $\alpha, \beta \in \mathbb{Z}_N$, $\alpha \neq 0$ (we say* polynomials of type 1*) and they were added either as the challenge for $\mathcal{A}_1^{(0)}$ in the step 2 or as a polynomial corresponding to a group-operation query response during the execution of $\mathcal{A}_1^{(0)}$ in the step 3 b, or as a polynomial in pair with a response to an inverse query in the step 3 c. Furthermore, the value $s \in \mathcal{Y}$ in pair with such polynomial in the table is never returned as an answer to a query made by $\mathcal{A}_1^{(1)}$. Also,*

2. *at most $T+1$ of non-constant polynomials in the table are of the form $\alpha(X+B)+\beta$, for some $\alpha, \beta \in \mathbb{Z}_N$, $\alpha \neq 0$ (we say* polynomials of type 2*) and they were added either as the challenge for $\mathcal{A}_1^{(1)}$ in the step 4 or as a polynomial corresponding to a group-operation query response or inverse query response during the execution of $\mathcal{A}_1^{(1)}$ in step 5. Furthermore, the value $s \in \mathcal{Y}$ in pair with such polynomial in the table is never returned as an answer to a query made by $\mathcal{A}_1^{(0)}$.*

There are no non-constant polynomials in the table of forms different than the polynomials of type 1 and 2.

Proof. First, notice that a non-constant polynomial can only be introduced in the table in pair with a challenge s_b, s_{b+x}, or in pair with a response to a group-operation or inverse query, where at least one of the queried elements s_1, s_2 or the queried element s_1 (for the inverse query) has already been in the table in pair with a non-constant polynomial. As there are 2 challenges and at most 2T queries performed by the distributed attacker, there is at most 2T+2 non-constant polynomials in the table at the end of the execution.

Next, we notice that the part 2 holds:

We notice, that the first polynomial with a non-zero coefficient next to the variable X is added to the table with the challenge s_{b+x} in the step 4. Therefore, the polynomials with a non-zero coefficient next to the variable X can be added to the table as a result of a group-operation query or an inverse query only after this moment. There are at most T queries performed after this moment, therefore, there cannot be more than $T+1$ polynomials with a non-zero coefficient next to the variable X, and thus, not more than $T+1$ polynomials of type 2. Furthermore, when a new polynomial is added to the table, we sample its pair value s from the unused values in $\mathcal{Y}$. As the polynomials of type 2 appear in the table only after the end of execution of $\mathcal{A}_1^{(0)}$, none of the values s in pairs with such polynomials could have been returned as an answer to a query made by $\mathcal{A}_1^{(0)}$.

Next, we prove that at the end of the execution there are no non-constant polynomials of forms different than polynomials of type 1 and 2 in the table:

We have already noticed, that non-constant polynomials are only being added to the table in the challenge pair, during a group-operation query as a sum of two polynomials already present in the table, where at least one of them is non-constant, or during an inverse query, as an $(N-1)$ multiple of a polynomial in pair with the queried element, if it is non-constant. Therefore, it is obvious that all polynomials in the table are at most of degree 1 in both, B and X. Now, it is enough to show there is no polynomial of the form $\alpha_1 B + \alpha_2 X + \beta$, where $\alpha_1, \alpha_2, \beta \in \mathbb{Z}_N$, $\alpha_2 \neq 0$, $\alpha_1 \neq \alpha_2$, we will say *polynomial of type 3*. We already know a polynomial of this form has not been added to the table before the step 4 of the execution, because all of the polynomials until this step are of degree 0 in the variable X. The polynomial added with the challenge in the step 4 is $X+B$, which is a polynomial of type 2. Therefore, it is enough to look at the polynomials added to the table during the step 5. We will prove by induction, there are no non-constant polynomials of type different than type 2 added to the table during the step 5. We suppose there has been no non-constant polynomials

different than type 2 added to the table during the step 5 before the i-th query in step 5. Trivially, for $i = 1$ the assumption holds. We will prove it holds for $i + 1$. From the induction hypothesis it follows that there are no polynomials different from the constant polynomials, polynomials of type 1 and polynomials of type 2 before the i-th query of step 5 in the table. In case i-th query is a forward query, either a constant polynomial or no polynomial is added to the table. In case the i-th query is a group operation query (s_1, s_2), the following cases can occur:

1. At least one of the pair s_1, s_2 is not in $\mathcal{Y}$. Then $\perp$ is returned and no polynomials are added to the table.

2. $s_1, s_2 \in \mathcal{Y}$, none of them is in the table, or one of them is in the table in pair with a constant polynomial and the other one is not in the table, or both are in the table in pair with a constant polynomial. Then only constant polynomials are added to the table.

3. $s_1, s_2 \in \mathcal{Y}$, one of them is in the table in pair with a polynomial of type 2 and the other one is not in the table, is in the table in pair with a constant polynomial, or is in the table in pair with a polynomial of type 2. Then the response will be in the table in pair with a polynomial of type 2 or a constant polynomial.

4. $s_1, s_2 \in \mathcal{Y}$, one of them is in the table in pair with a polynomial of type 1. Then the query is answered by $\perp$ and no polynomials are added to the table.

In case the i-th query is an inverse query s_1, the following cases can occur:

1. $s_1 \notin \mathcal{Y}$. Then $\perp$ is returned and no polynomials are added to the table.

2. $s_1 \in \mathcal{Y}$ and s_1 is not in the table, or is in the table in pair with a constant polynomial. Then only constant polynomials are added to the table.

3. $s_1 \in \mathcal{Y}$ and s_1 is in the table in pair with a polynomial of type 2, then the response is in the table in pair with a polynomial of type 2.

4. $s_1 \in \mathcal{Y}$ and s_1 is in the table in pair with a polynomial of type 1, then the query is answered by $\perp$ and no polynomials are added to the table.

By our analysis listing all of the possible cases, no non-constant polynomial of type different than type 2 was added during the i-th query. Therefore no non-constant polynomial of type different than type 2 was added during the step 5.

Therefore, no non-constant polynomials of forms different than the polynomials of type 1 and 2 are in the table at the end of the execution of the alternative experiment.

Part 1 follows from the fact that no non-constant polynomial of type different than type 2 are being added to the table during the execution of $\mathcal{A}_1^{(1)}$. More precisely, the polynomials of type 1 can only be introduced in the table in pair with the challenge s_b or in pair with a response to a group-operation query or an inverse query by $\mathcal{A}_1^{(0)}$. $\mathcal{A}_1^{(0)}$ makes at most T queries, therefore, at most $T + 1$ polynomials of type 1 are in the table at the end of the execution. By the analysis of the possible group-operation and inverse query responses during the step 5 an

s in pair with a polynomial of type 1 will never be returned as an answer to a query made by $\mathcal{A}_1^{(1)}$. This concludes the proof of the lemma. □

To prove Theorem 5, we can bound the collision probability $\Pr[E \mid \neg F]$ in two steps.

1. First, we estimate the probability that there exist two polynomials $p_1(X, B)$, $p_2(X, B)$ in the table such that when we substitute the value $x \in [-W/2, W/2]$ chosen by $\mathcal{C}^{\mathsf{DDLog}}$ for the variable X then $p_1(x, B) = p_2(x, B)$. We refer to this event as *collision in step 1.*

2. Second, we condition on the fact that the collision in step 1 did not happen and we consider the set of polynomials from the table after the substitution for X and estimate the probability that in this set there exist two polynomials $p_1(B), p_2(B)$ such that when we substitute the value $b \in \mathbb{Z}_N$ chosen by $\mathcal{C}^{\mathsf{DDLog}}$ for the variable B then $p_1(b) = p_2(b)$. We refer to this event as *collision in step 2.*

Regarding the first step, we have already established that all of the polynomials in the table are distinct and that there are only at most $T + 1$ polynomials dependent on variable X and that these are the polynomials of type 2 (Lemma 6). Therefore, we only examine the probability that one of the polynomials of type 2 from the table collide after the substitution for X with another polynomial from the table. Therefore, without loss of generality, we can assume p_1 is a polynomial of type 2. We look at the possible forms of p_2:

- p_2 is a constant polynomial. After the substitution for X, p_1 is still a polynomial linear in B, therefore $p_1(x, B) \neq p_2(x, B)$.

- p_2 is a polynomial of type 1. A collision in step 1 can occur in this case.

- p_2 is a polynomial of type 2. Let $p_1(X, B) = \alpha_1(X + B) + \beta_1$, $p_2(X, B) = \alpha_2(X + B) + \beta_2$, where $\alpha_1, \alpha_2 \neq 0$, as both of the polynomials are of type 2. Consider $p_1(x, B) = p_2(x, B)$, i.e., $\alpha_1 B + (\beta_1 + \alpha_1 x) = \alpha_2 B + (\beta_2 + \alpha_2 x)$, and thus, $\alpha_1 = \alpha_2$ and $\beta_1 = \beta_2$. Then $p_1(X, B) = p_2(X, B)$, but there were no identical polynomials in the table before the substitution. Therefore, $p_1(x, B) \neq p_2(x, B)$.

Therefore, we only need to estimate the probability of $p_1(x, B) = p_2(x, B)$, where p_1 is a polynomial of type 2 from the table and p_2 is a polynomial of type 1 from the table. We will consider a polynomial $p'(X, B) = p_2(X, B) - p_1(X, B)$ for each possible choice of p_1 from the polynomials of type 2 in the table and p_2 from the polynomials of type 1 in the table. As, by Lemma 6, there are at most $T + 1$ polynomials of type 1 and at most $T + 1$ polynomials of type 2, we have $(T + 1) \cdot (T + 1)$ polynomials p' to consider. The polynomial $p'(X, B)$ is a non-zero polynomial and, after the substitution of x for X, it is equal 0 if and only if $p_1(x, B) = p_2(x, B)$. Now, we apply the Schwartz-Zippel lemma (Lemma 3) over the field of fractions of $\mathbb{Z}_N[B]$ to bound the probability of $p'(x, B) = 0$ for an x chosen uniformly at random from the set of size W. We get $\Pr[p'(x, B) = 0] \leq \frac{1}{W}$. Then, by the union bound (Lemma 4), the probability of the collision in step 1

is at most $\frac{(T+1)^2}{W}$.

Regarding the second step, we condition on the event that there was no collision in step 1. After the substitution for X, we are left with at most $2T+2$ non-constant distinct polynomials from $\mathbb{Z}_N[B]$ and the total number of entries in the table is at most $P+3T+2$. We consider all the difference polynomials $p'(B) = p_1(B) - p_2(B)$, where $p_1(B)$ belongs to the set of non-constant polynomials from the table after the substitution for X and $p_2[B]$ belongs to the set of all polynomials (including constant polynomials) from the table after the substitution for X, different from $p_1(B)$. Using the same argumentation as in the step 1, the probability of the collision in step 2 is equal to the probability of any of the difference polynomials $p'(B)$, after the substitution of $b \leftarrow \mathbb{Z}_N$, being 0. We apply the Schwartz-Zippel lemma over the field $\mathbb{Z}_N$ and the union bound to get that the probability of collision in step 2 is at most $\frac{(2T+2)\cdot(P+3T+1)}{N}$.

Together, we have

$$\Pr[E \mid \neg F] \leq \frac{(T+1)^2}{W} + \frac{(2T+2)\cdot(P+3T+1)}{N}.$$

Now, we bound $\Pr[F]$. Let us denote $\mathcal{T}$ the set of elements $s \in \mathcal{Y}$ occurring in the table in pair with a polynomial of type 1 at the end of the alternative experiment, i.e.,

$$\mathcal{T} = \{s \in \mathcal{Y} \mid \exists \alpha, \beta \in \mathbb{Z}_N,\ \alpha \neq 0 : (\alpha B + \beta, s) \in \text{Table}\}.$$

The probability $\Pr[F]$ is the probability that, for some $s \in \mathcal{T}$, $\mathcal{A}_1^{(1)}$ makes a group-operation query $(s, \cdot)$, $(\cdot, s)$, or an inverse query s during its execution. We call such queries *unexpected* and we bound the probability of the first occurrence of an unexpected query.

Let $S[i]$ denote the set of elements from $\mathcal{Y}$ that has been revealed to $\mathcal{A}_1^{(1)}$ before its i-th query.

$$\begin{aligned} S[i] = P \cup \{s_{b+x}\} \cup \\ \{s \in \mathcal{Y} \mid s \text{ was a response to the } j\text{-th query for some } j < i\} \cup \\ \{s \in \mathcal{Y} \mid s \text{ or } (s,\cdot) \text{ or } (\cdot, s) \text{ was the } j\text{-th query for some } j < i\}. \end{aligned}$$

We examine the probability that an unexpected query appears as the i-th query of $\mathcal{A}_1^{(1)}$ for the first time during the execution.

In case the i-th query of $\mathcal{A}_1^{(1)}$ is a group-operation query, we are interested in the probability

$$\Pr[\{s_1, s_2\} \cap \mathcal{T} \neq \emptyset \mid S[i] \cap \mathcal{T} = \emptyset],$$

where (s_1, s_2) is the i-the query made by $\mathcal{A}_1^{(1)}$ and the probability is taken over the randomness of the experiment.

In case the i-th query of $\mathcal{A}_1^{(1)}$ is an inverse query, the probability of our interest is

$$\Pr[\{s\} \cap \mathcal{T} \neq \emptyset \mid S[i] \cap \mathcal{T} = \emptyset],$$

where s is the i-the query made by $\mathcal{A}_1^{(1)}$ and the probability is taken over the randomness of the experiment.

Let us fix the index i, the set $S[i]$ and also the query (s_1, s_2) or s. We remark that the choice of $\mathcal{T}$ is dependent on $S[i]$ only by having empty intersection with it. Therefore, for every $y \in \mathcal{Y} \setminus S[i]$ the following holds

$$\Pr[y \in \mathcal{T} \mid S[i] \cap \mathcal{T} = \emptyset] = \frac{|\mathcal{T}|}{|\mathcal{Y} \setminus S[i]|},$$

where the probability is taken over the randomness of the experiment. We get the following bound for our fixed i-th query:

$$\begin{aligned}
&\Pr[i\text{-th query is the first unexpected query}] \\
&\quad \leq \max\left\{ \Pr_{\mathcal{T}}[\{s_1, s_2\} \cap \mathcal{T} \neq \emptyset \mid S[i] \cap \mathcal{T} = \emptyset], \Pr_{\mathcal{T}}[\{s\} \cap \mathcal{T} \neq \emptyset \mid S[i] \cap \mathcal{T} = \emptyset] \right\} \\
&\quad \leq 2\frac{|\mathcal{T}|}{|\mathcal{Y} \setminus S[i]|} \leq 2\frac{T+1}{N-(P+3T)}.
\end{aligned}$$

where the last inequality uses $|\mathcal{T}| \leq T+1$ (Lemma 6) and $S[i] \leq P+1+3(i-1) < P+3T$, as there are at most T queries made by $\mathcal{A}_1^{(1)}$.

Therefore, by the union bound over the queries of $\mathcal{A}_1^{(1)}$, we can bound the probability that any query was unexpected, and thus, the probability of F, as follows:

$$\begin{aligned}
\Pr[F] &\leq \sum_{i=1}^{T} \Pr[i\text{-th query is the first unexpected query}] \\
&\leq \sum_{i=1}^{T} 2\frac{T+1}{N-(P+3T)} \\
&\leq \frac{2T(T+1)}{N-(P+3T)}.
\end{aligned}$$

Together, we get

$$\Pr[E \cup F] \leq \frac{(T+1)^2}{W} + \frac{(2T+2)\cdot(P+3T+1)}{N} + \frac{2\cdot T\cdot(T+1)}{N-(P+3T)}.$$

Since in the alternative experiment x is chosen at the end of the experiment uniformly at random from the integer values in the interval $[-W/2, W/2]$, the success probability of $\mathcal{A}$ in the alternative experiment is at most $1/W$. By the union bound, we can bound the success probability ϵ' of $\mathcal{A}$ in the standard experiment by the following:

$$\epsilon' \leq \frac{(T+1)^2+1}{W} + \frac{(2T+2)\cdot(P+3T+1)}{N} + \frac{2\cdot T\cdot(T+1)}{N-(P+3T)}.$$

In the rest of the proof, we assume that $N \geq 16$ (required by Proposition 1). This can be done without loss of generality since we are proving an asymptotic bound. Now, we apply Theorem 2 in order to bound the attacker's success probability ϵ in the $\mathsf{AI\text{-}GG}(N, M)$-model. It holds that $T^{\mathsf{comb}}_{G^{\mathsf{DDLog}}} = 2T+2$, and we set $\gamma := 1/W$ and $P := 6(S + \log(W)) \cdot (2T+2)$. By Theorem 2, we get:

$$\begin{aligned}\epsilon \leq & 2 \cdot \epsilon' + \gamma \\ \leq & \frac{2 \cdot (T+1)^2 + 3}{W} + \frac{2 \cdot (2T+2) \cdot (P+3T+1)}{N} + \frac{4 \cdot T \cdot (T+1)}{N - (P+3T)}.\end{aligned}$$

without loss of generality, we assume $T \geq 72$. Furthermore, assume that $N \geq 72 \cdot \max\{S, \log(W), 1\} \cdot T$, otherwise if

$$N < 72 \cdot \max\{S, \log(W), 1\} \cdot T \leq \max\{S, \log(W), 1\} \cdot T^2,$$

then $\frac{\max\{S, \log(W), 1\} \cdot T^2}{N} > 1$ and the theorem's bound is looser than $\epsilon = O(1)$, which holds trivially. Then

$$\begin{aligned}N - (P+3T) &= N - (6(S + \log(W)) \cdot (2T+2) + 3T) \\ &\geq N - (12 \cdot \max\{S, \log(W), 1\} \cdot (2T+2) + 3T) \\ &\geq N - 36 \cdot \max\{S, \log(W), 1\} \cdot T \\ &\geq N/2.\end{aligned}$$

Consequently, for the sum of the second and the third term, we have

$$\begin{aligned}\frac{2 \cdot (2T+2) \cdot (P+1+3T)}{N} + & \frac{4 \cdot T \cdot (T+1)}{N - (P+3T)} \\ &\leq \frac{2 \cdot (2T+2) \cdot (P+1+3T)}{N} + \frac{8 \cdot T \cdot (T+1)}{N} \\ &= \frac{4 \cdot (T+1)(P+1+5T)}{N} \\ &= \frac{4 \cdot (T+1)(6 \cdot (S + \log W)(2T+2) + 1 + 5T)}{N} \\ &\leq \frac{192 \cdot \max\{S, \log(W), 1\} \cdot T^2}{N}.\end{aligned}$$

Therefore, for ϵ

$$\begin{aligned}\epsilon \leq & \frac{3T^2}{W} + \frac{192 \cdot \max\{S, \log(W), 1\} \cdot T^2}{N} \\ = & O\left(\frac{T^2}{W} + \frac{\max\{S, \log(W)\} \cdot T^2}{N}\right).\end{aligned}$$

Furthermore, if $N \geq (W \cdot \log(W))$ then we get:

$$\begin{aligned}\epsilon \leq & \frac{3T^2}{W} + \frac{192 \cdot \max\{S, 1\} \cdot T^2}{N} + \frac{192 \cdot \log(W) \cdot T^2}{N} \\ \leq & \frac{3T^2}{W} + \frac{192 \cdot \max\{S, 1\} \cdot T^2}{N} + \frac{192 \cdot T^2}{W} \\ = & O\left(\frac{T^2}{W} + \frac{S \cdot T^2}{N}\right).\end{aligned}$$

Which concludes the proof of Theorem 5. □

In Theorem 5, we derived an upper bound for the success probability of a distributed (S, T)-attacker. Assuming $N \geq W \log W$, our bound for a big preprocessing advice ($S = \Omega(N/W)$) translates into a bound on time-space tradeoff in the DDLog problem $ST^2 = \Omega(\epsilon N)$. For a small preprocessing advice ($S = O(N/W)$), our bound translates to a lower bound on the time complexity of the DDLog problem $T^2 = \Omega(\epsilon W)$. Interestingly, if we consider a constant success probability in the DDLog problem, our bound for a distributed attacker with preprocessing with a small advice matches the time complexity of the algorithm with no preprocessing running in time $T^2 = O(\frac{W}{1-\epsilon})$ from [DKK20]. This implies that if we want to achieve a constant success probability in the DDLog problem, then allowing the attacker to precompute a preprocessing advice of order $S = O(N/W)$ does not asymptotically help to save computation time in the online phase of the DDLog problem.

3. HSS from Joye-Libert encryption scheme

3.1 Modified Joye-Libert public-key encryption scheme

In this section, we first introduce several standard definitions and theorems about quadratic residues and then we define the Modified Joye-Libert public-key encryption scheme, our modified version of the scheme introduced in [JL13].

Definition 20 (Quadratic Residue)**.** *Let $N \in \mathbb{N}$, $y \in \mathbb{Z}$, $\gcd(y, N) = 1$. We say y is* a quadratic residue modulo N *if there exists $x \in \mathbb{Z}_N^*$ such that $y = x^2 \mod N$. We say y is* a quadratic non-residue modulo N *if it is not a quadratic residue modulo N.*

Definition 21 (Legendre symbol)**.** *Let p be a prime. We define* the Legendre symbol modulo p *as a function $\left(\frac{\cdot}{p}\right) : \mathbb{Z} \to \{-1, 0, 1\}$ defined as follows:*

$$\left(\frac{a}{p}\right) = \begin{cases} 1 & \text{if } a \text{ is a quadratic residue modulo } p \\ 0 & \text{if } p \mid a \\ -1 & \text{if } a \text{ is a quadratic non-residue modulo } p \end{cases}$$

Definition 22 (Jacobi symbol)**.** *Let $N \in \mathbb{N}$ and $N = p_1^{e_1} \cdot \dots \cdot p_n^{e_n}$ be the prime factorization of N. We define the* Jacobi symbol modulo N *as a function $\left(\frac{\cdot}{N}\right) : \mathbb{Z} \to \{-1, 0, 1\}$, such that for every $a \in \mathbb{Z}$ we have $\left(\frac{a}{N}\right) = \left(\frac{a}{p_1}\right)^{e_1} \cdot \dots \cdot \left(\frac{a}{p_n}\right)^{e_n}$, where the symbols at the right hand side of the equation are the Legendre symbols.*

Proposition 7 ([Ros84, Theorem 9.5])**.** *Let $N \in \mathbb{N}$ and $a, b \in \mathbb{Z}$. Then*

$$\left(\frac{a \cdot b}{N}\right) = \left(\frac{a}{N}\right) \cdot \left(\frac{b}{N}\right).$$

Proof. For the proof we refer to the proof of [Ros84, Theorem 9.5]. □

Proposition 8. *Let p, q be primes and $N = p \cdot q$. Denote $\mathbb{J}_N \coloneqq \{a \in \mathbb{Z}_N^* \mid \left(\frac{a}{N}\right) = 1\}$ and $\mathbb{QR}_N \coloneqq \{a \in \mathbb{Z}_N^* \mid \exists b \in \mathbb{Z}_N^* : a = b^2 \mod N\}$. Then:*

1. *$\mathbb{J}_N$ form a subgroup of $\mathbb{Z}_N^*$.*
2. *$\mathbb{QR}_N$ form a subgroup of $\mathbb{J}_N$.*

Proof.

1. By an easy observation $1 \in \mathbb{J}_N$ as $\left(\frac{1}{N}\right) = \left(\frac{1}{p}\right) \cdot \left(\frac{1}{q}\right) = 1$, because $1 = 1^2$ both modulo p and modulo q. By Proposition 7, for $a, b \in \mathbb{J}_N$, we have $a \cdot b \in \mathbb{J}_N$, as $\left(\frac{a \cdot b}{N}\right) = \left(\frac{a}{N}\right) \cdot \left(\frac{b}{N}\right) = 1$. Consider $a \in \mathbb{J}_N$, if $a^{-1} \notin \mathbb{J}_N$ then $\left(\frac{1}{N}\right) = \left(\frac{a \cdot a^{-1}}{N}\right) = 1 \cdot (-1) = -1$, but we know $\left(\frac{1}{N}\right) = 1$, therefore $a^{-1} \in \mathbb{J}_N$.

2. If $a \in \mathbb{Z}_N^*$ is a quadratic residue modulo N, then a is also a quadratic residue modulo p and modulo q. Then, for every $a \in \mathbb{QR}_N : \left(\frac{a}{N}\right) = \left(\frac{a}{p}\right) \cdot \left(\frac{a}{q}\right) = 1$, therefore $a \in \mathbb{J}_N$. Clearly $1 \in \mathbb{QR}_N$ and let $a, b \in \mathbb{QR}_N$, $a = x^2 \mod N$, $b = y^2 \mod N$ for some $x, y \in \mathbb{Z}_N^*$. Then, $ab = (xy)^2 \in \mathbb{QR}_N$ and $a^{-1} = (x^{-1})^2 \in \mathbb{QR}_N$.

□

Notation. We use the notation for the groups from Proposition 8, i.e., $\mathbb{J}_N$ for the group of elements with Jacobi symbol modulo N equal to 1 and $\mathbb{QR}_N$ for the group of quadratic residues modulo N, for $N \in \mathbb{N}$. We also denote $\overline{\mathbb{J}_N} \coloneqq \mathbb{Z}_N^* \setminus \mathbb{J}_N$, the set of elements from $\mathbb{Z}_N^*$ with Jacobi symbol -1.

Proposition 9 ([KL20, Proposition 15.21])**.** *Let p,q be two distinct primes and $N = p \cdot q$. Then, for every $x \in \mathbb{Z}_N^*$, $x \in \mathbb{QR}_N$ if and only if $x \in \mathbb{QR}_p$ and $x \in \mathbb{QR}_q$.*

Proof. For the proof we refer to the proof of [KL20, Proposition 15.21]. □

Proposition 10 ([KL20, Proposition 15.23])**.** *Let p, q be two distinct odd primes and $N = p \cdot q$. Then:*

1. $|\mathbb{J}_N| = \frac{1}{2}|\mathbb{Z}_N^*|$.
2. $|\mathbb{QR}_N| = \frac{1}{2}|\mathbb{J}_N|$.

Proof. For the proof we reference to the proof of [KL20, Proposition 15.23]. □

Now, we present an encryption scheme based on the Joye-Libert public-key encryption scheme introduced by [JL13]. In order for the scheme to support an efficient DDLog protocol, we introduce several modifications in the original scheme. We call this new scheme *Modified Joye-Libert encryption scheme* (mJL).

Definition 23 (Modified Joye-Libert encryption scheme (mJL))**.** *Let $\lambda \in \mathbb{N}$ be a security parameter and $k \geq 1$. We define the modified Joye-Libert public-key encryption scheme as a scheme consisting of algorithms* $(\mathsf{KeyGen}_{\mathsf{mJL}}, \mathsf{Enc}_{\mathsf{mJL}}, \mathsf{Dec}_{\mathsf{mJL}})$ *described below.*

$\mathsf{KeyGen}_{\mathsf{mJL}}(1^\lambda, k)$**:** *On input 1^λ and $k \geq 1$, $\mathsf{KeyGen}_{\mathsf{mJL}}$ samples primes p, q of the form $p = 2^k p' + 1$ and $q = 2^k q' + 1$, where p', q' are primes. It sets $N \coloneqq p \cdot q$, $d \coloneqq p' \cdot q'$ and samples $g \leftarrow \mathbb{J}_N \setminus \mathbb{QR}_N$ and it outputs the pair $(\mathsf{pk}, \mathsf{sk})$, where $\mathsf{pk} = (g, g^d, k, N)$ and $\mathsf{sk} = d$.*

$\mathsf{Enc}_{\mathsf{mJL}}(\mathsf{pk}, m)$**:** *On input pk and the message $m \in \mathbb{Z}_{2^k}$, $\mathsf{Enc}_{\mathsf{mJL}}$ samples a random $r \leftarrow \mathbb{Z}_N^*$ and it outputs $g^m \cdot r^{2^k} \mod N$.*

$\mathsf{Dec}_{\mathsf{mJL}}(\mathsf{sk}, c)$**:** *On input sk and a ciphertext $c \in \mathbb{Z}_N^*$, $\mathsf{Dec}_{\mathsf{mJL}}$ calculates $\gamma \coloneqq c^d \mod N$ and extracts the message bit by bit from γ using the algorithm* $\mathsf{Extract}$ *(Algorithm 1).*

Algorithm 1: $\mathsf{Extract}(\gamma, \mathsf{pk})$

$m = 0$;
for $i = 0, \ldots, k-1$ **do**
  $t \coloneqq (\frac{\gamma}{g^{d \cdot m}})^{2^{k-i-1}} \mod N$;
  if $t = 1$ **then**
    $m^{(i)} \coloneqq 0$;
  else if $t = -1$ **then**
    $m^{(i)} \coloneqq 1$;
  $m \coloneqq m + m^{(i)} \cdot 2^i$;
return m

In the original Joye-Libert encryption scheme, the only requirement on the primes p and q is that p is equal 1 modulo 2^k. In our modified scheme, we require this property from both of the primes. Moreover, we require the largest powers of 2 dividing $p-1$ and $q-1$ to be the same. This allows us to perform the decryption algorithm modulo N only; although, in the original scheme, the decryption is performed modulo p. Another change we introduce in the scheme is that we reveal g^d as a part of the public key, which allows a construction of a DDLog protocol.

It might seem that the inclusion of g^d to the public key compromises the security of the scheme. Actually, if we defined $p = 2^k p' + 1$, $q = 2^l q' + 1$, for some $k, l \in \mathbb{N}$, $k < l$, it would imply $\gcd(N, (g^d)^{2^k} - 1) = p$. Nevertheless, in our construction we require the largest powers of 2 dividing $p-1$ and $q-1$ to be the same, and, in this case, $\gcd(N, (g^d)^{2^k} - 1)$ does not reveal the factorisation of N. We discuss the security of the modified Joye-Libert encryption scheme in detail in section 3.2.

The following theorem shows the correctness of the modified Joye-Libert scheme.

Theorem 11. *Let $\lambda \in \mathbb{N}$ be a security parameter and $k \in \mathbb{N}$. For every $m \in \mathbb{Z}_{2^k}$, it holds $m = \mathsf{Dec}_{\mathsf{mJL}}(\mathsf{sk}, \mathsf{Enc}_{\mathsf{mJL}}(\mathsf{pk}, m))$, where $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{KeyGen}_{\mathsf{mJL}}(1^\lambda, k)$.*

Proof. First, we review the structure of the group $\mathbb{Z}_N^*$.

$$\mathbb{Z}_N^* \cong \mathbb{Z}_p^* \times \mathbb{Z}_q^* \cong \mathbb{Z}_{p-1} \times \mathbb{Z}_{q-1} \cong \mathbb{Z}_{p'} \times \mathbb{Z}_{2^k} \times \mathbb{Z}_{q'} \times \mathbb{Z}_{2^k}.$$

Thus, all of the subgroups in $\mathbb{Z}_N^*$ are of order at most $2^k \cdot p' \cdot q'$. Therefore, we have $\gamma = (\mathsf{Enc}_{\mathsf{mJL}}(\mathsf{pk}, m))^d = g^{m \cdot d} \cdot r^{2^k \cdot d} = g^{m \cdot d} \mod N$.

Next, we prove $(g^d)^{2^{k-1}} = -1 \mod N$. As $g \in \mathbb{J}_N \setminus \mathbb{QR}_N$, by Proposition 9, $\left(\frac{g}{p}\right) = \left(\frac{g}{q}\right) = -1$. As $d = p' \cdot q'$ is an odd number, we have from Proposition 7 $\left(\frac{g^d}{p}\right) = \left(\frac{g}{p}\right)^d = -1$. Therefore, g^d is a quadratic non-residue modulo p and thus $2^k \mid \mathrm{ord}_{\mathbb{Z}_p^*}(g^d)$. Furthermore, as the order of $\mathbb{Z}_p^*$ is $2^k \cdot p'$, $(g^d)^{2^k} = (g^{q'})^{2^k \cdot p'} = 1 \mod p$, that is $\mathrm{ord}_{\mathbb{Z}_p^*}(g^d) \mid 2^k$. Together, we have $\mathrm{ord}_{\mathbb{Z}_p^*}(g^d) = 2^k$. In the same way, we can prove $2^k \mid \mathrm{ord}_{\mathbb{Z}_q^*}(g^d)$ and $\mathrm{ord}_{\mathbb{Z}_q^*}(g^d) \mid 2^k$. Therefore, $\mathrm{ord}_{\mathbb{Z}_q^*}(g^d) = 2^k$. Together with the isomorphism $\mathbb{Z}_N^* \cong \mathbb{Z}_p^* \times \mathbb{Z}_q^*$, it implies $(g^d)^{2^{k-1}} = -1 \mod N$.

Next, let us denote t_i the value of t calculated in the i-th iteration of Algorithm 1 and m_i the i-th least significant bit of m, i.e., $m = \sum_{i=0}^{k-1} 2^i \cdot m_i$. We prove

by induction, that for every $i \in \{0, \dots k-1\}$, $m^{(i)} = m_i$. For $i = 0$ we have $t_0 = (g^{d \cdot 2^{k-1}})^m = (-1)^m \mod N$. Then, $t_0 = 1$ if and only if m is an even number, that is, if $m_0 = 0$ and $t_0 = -1$ if and only if $m_0 = 1$. The Algorithm 1 sets $m^{(0)}$ accordingly. Now suppose $m^{(j)} = m_j$ for all $j \in \{0, \dots, i\}$, we prove $m^{(i+1)} = m_{i+1}$. We have $t_{i+1} = \left(g^{d \cdot 2^{k-i-2}}\right)^{m - \sum_{j=0}^{i} 2^j \cdot m^{(j)}} = \left(g^{d \cdot 2^{k-i-2}}\right)^{\sum_{j=i+1}^{k-1} 2^j \cdot m_j} = \left(g^{d \cdot 2^{k-1}}\right)^{m_{i+1}} \mod N$. As before, we have $t_{i+1} = 1 \iff m_{i+1} = 0$ and $t_{i+1} = -1 \iff m_{i+1} = 1$, which is how the Algorithm 1 sets $m^{(i+1)}$. Finally, as $m_i = m^{(i)}$ for all $i \in \{0, \dots k-1\}$, we have $m = \sum_{i=0}^{k-1} 2^i \cdot m^{(i)}$, and thus, $m = \mathsf{Dec}_{\mathsf{mJL}}(\mathsf{sk}, \mathsf{Enc}_{\mathsf{mJL}}(\mathsf{pk}, m))$. □

3.2 Security of the Modified Joye-Libert encryption scheme

We follow the security proof of Joye-Libert scheme from [BHJL17] to prove the IND-CPA security of the modified scheme. Due to the modification of the scheme, in particular, due to the inclusion of g^d into the public key, we need to introduce a new assumption (k-dRI, Definition 26) and modify the assumptions used in the original proof of security in [BHJL17].

The following definition is a modification of a standard Quadratic Residuosity assumption [BHJL17, Definition 2]. In the standard case, the assumption says that a PPT attacker is not able to distinguish between a quadratic residue modulo N and an element from $\mathbb{J}_N \setminus \mathbb{QR}_N$. Our modified version assumes that the attacker is not able to distinguish between these distributions even if he is given an element x^d, where $x \leftarrow \mathbb{J}_N \setminus \mathbb{QR}_N$ and d is the secret key of the modified Joye-Libert scheme.

Definition 24 (Modified Quadratic Residuosity Assumption, k-QRm)**.** *Let $N = p \cdot q$ be generated by $\mathsf{KeyGen}_{\mathsf{mJL}}(1^\lambda, k)$ from Definition 23 alongside with $d = p' \cdot q'$. The* Modified Quadratic Residuosity Assumption (k-QRm) *states that for every PPT distinguisher $\mathcal{D}$, the advantage $\mathsf{Adv}^{\mathsf{k\text{-}QRm}}_{\mathcal{D}}(\lambda)$ is a negligible function in the security parameter λ. Where the advantage $\mathsf{Adv}^{\mathsf{k\text{-}QRm}}_{\mathcal{D}}(\lambda)$ is defined as the advantage of $\mathcal{D}$ in the indistinguishability experiment $\mathsf{Exp}^{D_0, D_1}_{\mathcal{D}}(1^\lambda)$ for the distributions $D_0 = \{(y, x^d, k, N) \mid y, x \leftarrow \mathbb{J}_N \setminus \mathbb{QR}_N\}$ and $D_1 = \{(y, x^d, k, N) \mid y \leftarrow \mathbb{QR}_N, x \leftarrow \mathbb{J}_N \setminus \mathbb{QR}_N\}$, i.e.,*

$$\mathsf{Adv}^{\mathsf{k\text{-}QRm}}_{\mathcal{D}}(\lambda) = \Big| \Pr[\mathcal{D}(1^\lambda, (y, x^d, k, N)) = 1 \mid y \leftarrow \mathbb{QR}_N, x \leftarrow \mathbb{J}_N \setminus \mathbb{QR}_N] - \Pr[\mathcal{D}(1^\lambda, (y, x^d, k, N)) = 1 \mid y, x \leftarrow \mathbb{J}_N \setminus \mathbb{QR}_N] \Big|,$$

where the probability is taken over the choice of (N, p, q, d) and the choice of x and y from the appropriate sets.

The Squared Jacobi Symbol Assumption [BHJL17, Definition 3] formalizes the inability of a PPT attacker of distinguishing between a square of a random element from $\mathbb{J}_N$ and a square of a random element from $\overline{\mathbb{J}_N}$. Again, we modify the assumption by giving x^d, where $x \leftarrow \mathbb{J}_N \setminus \mathbb{QR}_N$, as an extra input to the attacker.

Definition 25 (Modified Squared Jacobi Symbol Assumption, k-SJSm)**.** *Let* $N = p \cdot q$ *be generated by* $\mathsf{KeyGen}_{\mathsf{mJL}}(1^\lambda, k)$ *from Definition 23 alongside with* $d = p' \cdot q'$*. The* Modified Squared Jacobi Symbol Assumption (k-SJSm) *states that for every PPT distinguisher* $\mathcal{D}$*, the advantage* $\mathsf{Adv}^{\mathsf{k\text{-}SJSm}}_{\mathcal{D}}(\lambda)$ *is a negligible function in the security parameter* λ*. Where the advantage* $\mathsf{Adv}^{\mathsf{k\text{-}SJSm}}_{\mathcal{D}}(\lambda)$ *is defined as the advantage of* $\mathcal{D}$ *in the indistinguishability experiment* $\mathsf{Exp}^{D_0,D_1}_{\mathcal{D}}(1^\lambda)$ *for the distributions* $D_0 = \{(y^2, x^d, k, N) \mid y \leftarrow \overline{\mathbb{J}_N}, x \leftarrow \mathbb{J}_N \setminus \mathbb{QR}_N\}$ *and* $D_1 = \{(y^2, x^d, k, N) \mid y \leftarrow \mathbb{J}_N, x \leftarrow \mathbb{J}_N \setminus \mathbb{QR}_N\}$*, i.e.,*

$$\mathsf{Adv}^{\mathsf{k\text{-}SJSm}}_{\mathcal{D}}(\lambda) = \Big| \Pr[\mathcal{D}(1^\lambda, (y^2, x^d, k, N)) = 1 \mid y \leftarrow \mathbb{J}_N, x \leftarrow \mathbb{J}_N \setminus \mathbb{QR}_N] - \Pr[\mathcal{D}(1^\lambda, (y^2, x^d, k, N)) = 1 \mid y \leftarrow \overline{\mathbb{J}_N}, x \leftarrow \mathbb{J}_N \setminus \mathbb{QR}_N] \Big|,$$

where the probability is taken over the choice of (N, p, q, d) *and the choice of* x *and* y *from the appropriate sets.*

We introduce a new assumption, we call it the *d-th Root Indistinguishability Assumption*. The d-th Root Indistinguishability Assumptions assumes inability of a PPT attacker of distinguishing between a pair (x, x^d) and (y, x^d), where $x, y \leftarrow \mathbb{J}_N \setminus \mathbb{QR}_N$ and d is the secret key of the mJL scheme.

Definition 26 (d-th Root Indistinguishability Assumption, k-dRI)**.** *Let* $N = p \cdot q$ *be generated by* $\mathsf{KeyGen}_{\mathsf{mJL}}(1^\lambda, k)$ *from Definition 23 alongside with* $d = p' \cdot q'$*. The* d-th Root Indistinguishability Assumption (k-dRI) *states that for every PPT distinguisher* $\mathcal{D}$*, the advantage* $\mathsf{Adv}^{\mathsf{k\text{-}dRI}}_{\mathcal{D}}(\lambda)$ *is a negligible function in the security parameter* λ*. Where the advantage* $\mathsf{Adv}^{\mathsf{k\text{-}dRI}}_{\mathcal{D}}(\lambda)$ *is defined as the advantage of* $\mathcal{D}$ *in the indistinguishability experiment* $\mathsf{Exp}^{D_0,D_1}_{\mathcal{D}}(1^\lambda)$ *for the distributions* $D_0 = \{(y, x^d, k, N) \mid y, x \leftarrow \mathbb{J}_N \setminus \mathbb{QR}_N\}$ *and* $D_1 = \{(x, x^d, k, N) \mid x \leftarrow \mathbb{J}_N \setminus \mathbb{QR}_N\}$*, i.e.,*

$$\mathsf{Adv}^{\mathsf{k\text{-}dRI}}_{\mathcal{D}}(\lambda) = \Big| \Pr[\mathcal{D}(1^\lambda, (x, x^d, k, N)) = 1 \mid x \leftarrow \mathbb{J}_N \setminus \mathbb{QR}_N] - \Pr[\mathcal{D}(1^\lambda, (y, x^d, k, N)) = 1 \mid y, x \leftarrow \mathbb{J}_N \setminus \mathbb{QR}_N] \Big|,$$

where the probability is taken over the choice of (N, p, q, d) *and the choice of* x *and* y *from the appropriate sets.*

To construct the proof of the IND-CPA security of the Joye-Libert encryption scheme, [BHJL17] introduced the *Special Quadratic Residuosity Assumption* [BHJL17, Definition 5], the *Special Squared Jacobi Symbol Assumption* [BHJL17, Definition 6], and the *Gap* 2^k*-Residuosity Assumption* [BHJL17, Definition 4]. In Definition 27, 28 and 29, we give their modified versions, with the modification of providing x^d as an additional input to the attacker, where $x \leftarrow \mathbb{J}_N \setminus \mathbb{QR}_N$.

Definition 27 (Modified Special Quadratic Residuosity Assumption, k-QRm*)**.** *Let* $N = p \cdot q$ *be generated by* $\mathsf{KeyGen}_{\mathsf{mJL}}(1^\lambda, k)$ *from Definition 23 alongside with* $d = p' \cdot q'$*. The* Modified Special Quadratic Residuosity Assumption (k-QRm*)

states that for every PPT distinguisher $\mathcal{D}$ the advantage $\mathsf{Adv}_{\mathcal{D}}^{\mathsf{k\text{-}QRm}^*}(\lambda)$ is a negligible function in the security parameter λ. Where the advantage $\mathsf{Adv}_{\mathcal{D}}^{\mathsf{k\text{-}QRm}^*}(\lambda)$ is defined as the advantage of $\mathcal{D}$ in the indistinguishability experiment $\mathsf{Exp}_{\mathcal{D}}^{D_0,D_1}(1^\lambda)$ for the distributions $D_0 = \{(y, x^d, k, N) \mid y, x \leftarrow \mathbb{J}_N \backslash \mathbb{QR}_N\}$ and $D_1 = \{(y^2, x^d, k, N) \mid y \leftarrow \mathbb{J}_N, x \leftarrow \mathbb{J}_N \setminus \mathbb{QR}_N\}$, i.e.,

$$\mathsf{Adv}_{\mathcal{D}}^{\mathsf{k\text{-}QRm}^*}(\lambda) = \Big| \Pr[\mathcal{D}(1^\lambda, (y^2, x^d, k, N)) = 1 \mid y \leftarrow \mathbb{J}_N, x \leftarrow \mathbb{J}_N \setminus \mathbb{QR}_N] - \Pr[\mathcal{D}(1^\lambda, (y, x^d, k, N)) = 1 \mid y, x \leftarrow \mathbb{J}_N \setminus \mathbb{QR}_N] \Big|,$$

where the probability is taken over the choice of (N, p, q, d) and the choice of x and y from the appropriate sets.

Definition 28 (Modified Special Squared Jacobi Symbol Assumption, $\mathsf{k\text{-}SJSm}^*$)**.** *Let $N = p \cdot q$ be generated by $\mathsf{KeyGen}_{\mathsf{mJL}}(1^\lambda, k)$ from Definition 23 alongside with $d = p' \cdot q'$. The* Modified Special Squared Jacobi Symbol Assumption ($\mathsf{k\text{-}SJSm}^*$) *states that for every PPT distinguisher $\mathcal{D}$ the advantage $\mathsf{Adv}_{\mathcal{D}}^{\mathsf{k\text{-}SJSm}^*}(\lambda)$ is a negligible function in the security parameter λ. Where the advantage $\mathsf{Adv}_{\mathcal{D}}^{\mathsf{k\text{-}SJSm}^*}(\lambda)$ is defined as the advantage of $\mathcal{D}$ in the indistinguishability experiment $\mathsf{Exp}_{\mathcal{D}}^{D_0,D_1}(1^\lambda)$ for the distributions $D_0 = \{(y^2, x^d, k, N) \mid y \leftarrow \overline{\mathbb{J}_N}, x \leftarrow \mathbb{J}_N \setminus \mathbb{QR}_N\}$ and $D_1 = \{(y^2, x^d, k, N) \mid y, x \leftarrow \mathbb{J}_N \setminus \mathbb{QR}_N\}$, i.e.,*

$$\mathsf{Adv}_{\mathcal{D}}^{\mathsf{k\text{-}SJSm}^*}(\lambda) = \Big| \Pr[\mathcal{D}(1^\lambda, (y^2, x^d, k, N)) = 1 \mid y, x \leftarrow \mathbb{J}_N \setminus \mathbb{QR}_N] - \Pr[\mathcal{D}(1^\lambda, (y^2, x^d, k, N)) = 1 \mid y \leftarrow \overline{\mathbb{J}_N}, x \leftarrow \mathbb{J}_N \setminus \mathbb{QR}_N] \Big|,$$

where the probability is taken over the choice of (N, p, q, d) and the choice of x and y from the appropriate sets.

Definition 29 (Modified Gap 2^k-Residuosity Assumption, $\mathsf{Gap}2^k\text{-}\mathsf{Resm}$)**.** *Let $N = p \cdot q$ be generated by $\mathsf{KeyGen}_{\mathsf{mJL}}(1^\lambda, k)$ from Definition 23 alongside with $d = p' \cdot q'$. The* Modified Gap 2^k-Residuosity Assumption ($\mathsf{Gap}2^k\text{-}\mathsf{Resm}$) *states that for every PPT distinguisher $\mathcal{D}$ the advantage $\mathsf{Adv}_{\mathcal{D}}^{\mathsf{Gap}2^k\text{-}\mathsf{Resm}}(\lambda)$ is a negligible function in the security parameter λ. Where the advantage $\mathsf{Adv}_{\mathcal{D}}^{\mathsf{Gap}2^k\text{-}\mathsf{Resm}}(\lambda)$ is defined as the advantage of $\mathcal{D}$ in the indistinguishability experiment $\mathsf{Exp}_{\mathcal{D}}^{D_0,D_1}(1^\lambda)$ for the distributions $D_0 = \{(y^{2^k}, x^d, k, N) \mid y \leftarrow \mathbb{Z}_N^*, x \leftarrow \mathbb{J}_N \setminus \mathbb{QR}_N\}$ and $D_1 = \{(y, x^d, k, N) \mid y, x \leftarrow \mathbb{J}_N \setminus \mathbb{QR}_N\}$, i.e.,*

$$\mathsf{Adv}_{\mathcal{D}}^{\mathsf{Gap}2^k\text{-}\mathsf{Resm}}(\lambda) = \Big| \Pr[\mathcal{D}(1^\lambda, (y, x^d, k, N)) = 1 \mid y, x \leftarrow \mathbb{J}_N \setminus \mathbb{QR}_N] - \Pr[\mathcal{D}(1^\lambda, (y^{2^k}, x^d, k, N)) = 1 \mid y \leftarrow \mathbb{Z}_N^*, x \leftarrow \mathbb{J}_N \setminus \mathbb{QR}_N] \Big|,$$

where the probability is taken over the choice of (N, p, q, d) and the choice of x and y from the appropriate sets.

Analogously to [BHJL17, Lemma 1], we prove "$\mathsf{k\text{-}QRm}$+$\mathsf{k\text{-}SJSm} \implies \mathsf{k\text{-}QRm}^*$+ $\mathsf{k\text{-}SJSm}^*$".

Lemma 12. *Let λ denote the security parameter, let $k \in \mathbb{N}$ and $N = p \cdot q$ be generated by $\mathsf{KeyGen}_{\mathsf{mJL}}(1^\lambda, k)$. For every PPT distinguisher $\mathcal{A}$ against $\mathsf{k\text{-}QRm}^*$ or $\mathsf{k\text{-}SJSm}^*$, $\mathcal{A}$ is also a distinguisher against $\mathsf{k\text{-}QRm}$ or $\mathsf{k\text{-}SJSm}$ and there exists a distinguisher $\mathcal{B}$ against $\mathsf{k\text{-}QRm}$ with comparable running time to $\mathcal{A}$ such that the following holds:*

$$\mathsf{Adv}^{\mathsf{k\text{-}QRm}^*}_{\mathcal{A}}(\lambda) \leq \mathsf{Adv}^{\mathsf{k\text{-}QRm}}_{\mathcal{A}}(\lambda) + \frac{1}{2}\mathsf{Adv}^{\mathsf{k\text{-}SJSm}}_{\mathcal{A}}(\lambda),$$
$$\mathsf{Adv}^{\mathsf{k\text{-}SJSm}^*}_{\mathcal{A}}(\lambda) \leq \mathsf{Adv}^{\mathsf{k\text{-}SJSm}}_{\mathcal{A}}(\lambda) + \frac{1}{2}\mathsf{Adv}^{\mathsf{k\text{-}QRm}}_{\mathcal{B}}(\lambda).$$

Proof. Let $\mathcal{A}$ be the adversary against $\mathsf{k\text{-}QRm}$ or $\mathsf{k\text{-}SJSm}$ taking as input (a, b, k, N). Let us denote the following probabilities as follows:

$$\epsilon_1 \coloneqq \Pr[\mathcal{A}(1^\lambda, (a, b, k, N)) = 1 \mid b = x^d \ \ \& \ \ x, a \leftarrow \mathbb{J}_N \setminus \mathbb{QR}_N],$$
$$\epsilon_2 \coloneqq \Pr[\mathcal{A}(1^\lambda, (a, b, k, N)) = 1 \mid a = y^2, b = x^d \ \ \& \ \ x, y \leftarrow \mathbb{J}_N \setminus \mathbb{QR}_N],$$
$$\epsilon_3 \coloneqq \Pr[\mathcal{A}(1^\lambda, (a, b, k, N)) = 1 \mid a = y^2, b = x^d \ \ \& \ \ x \leftarrow \mathbb{J}_N \setminus \mathbb{QR}_N \ \ \& \ \ y \leftarrow \mathbb{QR}_N],$$
$$\epsilon_4 \coloneqq \Pr[\mathcal{A}(1^\lambda, (a, b, k, N)) = 1 \mid a = y^2, b = x^d \ \ \& \ \ x \leftarrow \mathbb{J}_N \setminus \mathbb{QR}_N \ \ \& \ \ y \leftarrow \overline{\mathbb{J}_N}].$$

Then, from the definition of the advantages and Proposition 10, we have:

$$\mathsf{Adv}^{\mathsf{k\text{-}QRm}}_{\mathcal{A}}(\lambda) = \left|\epsilon_1 - \frac{1}{4}(\epsilon_2 + \epsilon_3) - \frac{1}{2}\epsilon_4\right|,$$
$$\mathsf{Adv}^{\mathsf{k\text{-}SJSm}}_{\mathcal{A}}(\lambda) = \left|\frac{1}{2}(\epsilon_2 + \epsilon_3) - \epsilon_4\right|,$$
$$\mathsf{Adv}^{\mathsf{k\text{-}QRm}^*}_{\mathcal{A}}(\lambda) = \left|\epsilon_1 - \frac{1}{2}(\epsilon_2 + \epsilon_3)\right|,$$
$$\mathsf{Adv}^{\mathsf{k\text{-}SJSm}^*}_{\mathcal{A}}(\lambda) = \left|\epsilon_2 - \epsilon_4\right|.$$

Therefore, for the advantage against $\mathsf{k\text{-}QRm}^*$ we get

$$\begin{aligned}\mathsf{Adv}^{\mathsf{k\text{-}QRm}^*}_{\mathcal{A}}(\lambda) &= \left|\epsilon_1 - \frac{1}{2}(\epsilon_2 + \epsilon_3)\right| = \left|\epsilon_1 - \frac{1}{4}(\epsilon_2 + \epsilon_3) - \frac{1}{2}\epsilon_4 + \frac{1}{2}\epsilon_4 - \frac{1}{4}(\epsilon_2 + \epsilon_3)\right| \\ &\leq \left|\epsilon_1 - \frac{1}{4}(\epsilon_2 + \epsilon_3) - \frac{1}{2}\epsilon_4\right| + \left|\frac{1}{2}\epsilon_4 - \frac{1}{4}(\epsilon_2 + \epsilon_3)\right| \\ &= \mathsf{Adv}^{\mathsf{k\text{-}QRm}}_{\mathcal{A}}(\lambda) + \frac{1}{2}\mathsf{Adv}^{\mathsf{k\text{-}SJSm}}_{\mathcal{A}}(\lambda).\end{aligned}$$

Furthermore, if $|\epsilon_2 - \epsilon_3|$ is non-negligible, we can construct a $\mathsf{k\text{-}QRm}$ distinguisher $\mathcal{B}$ running $\mathcal{A}$ as a subroutine with a non-negligible advantage $\mathsf{Adv}^{\mathsf{k\text{-}QRm}}_{\mathcal{B}}(\lambda) \geq |\epsilon_2 - \epsilon_3|$ as follows

$\mathcal{B}$:

1. On input $(1^\lambda, (a, b, k, N))$ run $\mathcal{A}(1^\lambda, (a^2, b, k, N))$.
2. On output b' given by $\mathcal{A}$ output also b'.

Then, we get

$$\begin{aligned}
\mathsf{Adv}^{\mathsf{k\text{-}QRm}}_{\mathcal{B}}(1^\lambda) = &\Big| \Pr[\mathcal{B}(1^\lambda, (y, x^d, k, N)) = 1 \mid y \leftarrow \mathbb{QR}_N, x \leftarrow \mathbb{J}_N \setminus \mathbb{QR}_N] \\
&\quad - \Pr[\mathcal{B}(1^\lambda, (y, x^d, k, N)) = 1 \mid y, x \leftarrow \mathbb{J}_N \setminus \mathbb{QR}_N] \Big| \\
= &\Big| \Pr[\mathcal{A}(1^\lambda, (y^2, x^d, k, N)) = 1 \mid y \leftarrow \mathbb{QR}_N, x \leftarrow \mathbb{J}_N \setminus \mathbb{QR}_N] \\
&\quad - \Pr[\mathcal{A}(1^\lambda, (y^2, x^d, k, N)) = 1 \mid y, x \leftarrow \mathbb{J}_N \setminus \mathbb{QR}_N] \Big| \\
= &\Big| \epsilon_3 - \epsilon_2 \Big|.
\end{aligned}$$

Thus, for the advantage against $\mathsf{k\text{-}SJSm}^*$ we get

$$\begin{aligned}
\mathsf{Adv}^{\mathsf{k\text{-}SJSm}^*}_{\mathcal{A}}(\lambda) = \Big| \epsilon_2 - \epsilon_4 \Big| &= \Big| \frac{1}{2}\epsilon_2 - \epsilon_4 + \frac{1}{2}\epsilon_3 - \frac{1}{2}\epsilon_3 + \frac{1}{2}\epsilon_2 \Big| \\
&\leq \Big| \frac{1}{2}\epsilon_2 - \epsilon_4 + \frac{1}{2}\epsilon_3 \Big| + \Big| \frac{1}{2}\epsilon_2 - \frac{1}{2}\epsilon_3 \Big| \\
&\leq \mathsf{Adv}^{\mathsf{k\text{-}SJSm}}_{\mathcal{A}}(\lambda) + \frac{1}{2}\mathsf{Adv}^{\mathsf{k\text{-}QRm}}_{\mathcal{B}}(\lambda),
\end{aligned}$$

which completes the proof. □

Analogously to [BHJL17, Theorem 3], we prove "$\mathsf{k\text{-}QRm} + \mathsf{k\text{-}SJSm} \implies \mathsf{Gap2}^k\text{-}\mathsf{Resm}$".

Lemma 13. *Let λ denote a security parameter, let $k \in \mathbb{N}$ and $N = p \cdot q$ be generated by $\mathsf{KeyGen}_{\mathsf{mJL}}(1^\lambda, k)$. For every PPT distinguisher $\mathcal{B}$ against $\mathsf{Gap2}^k\text{-}\mathsf{Resm}$ there exist a $\mathsf{k\text{-}QRm}$ distinguisher $\mathcal{D}_1$ and a $\mathsf{k\text{-}SJSm}$ distinguisher $\mathcal{D}_2$ with comparable running times to $\mathcal{B}$ such that*

$$\mathsf{Adv}^{\mathsf{Gap2}^k\text{-}\mathsf{Resm}}_{\mathcal{B}}(\lambda) \leq \frac{3}{2}\left((k - \frac{1}{3}) \cdot \mathsf{Adv}^{\mathsf{k\text{-}QRm}}_{\mathcal{D}_1}(\lambda) + (k-1) \cdot \mathsf{Adv}^{\mathsf{k\text{-}SJSm}}_{\mathcal{D}_2}(\lambda) \right).$$

Proof. Let us denote the following distributions for $i \in \{0, \ldots, k-1\}$ as follows

$$\begin{aligned}
D_i &:= \{(y^{2^i}, x^d, k, N) \mid x, y \leftarrow \mathbb{J}_N \setminus \mathbb{QR}_N\}, \\
D'_i &:= \{(y^{2^i}, x^d, k, N) \mid y \leftarrow \overline{\mathbb{J}_N}, x \leftarrow \mathbb{J}_N \setminus \mathbb{QR}_N\}, \\
R_k &:= \{(y^{2^k}, x^d, k, N) \mid y \leftarrow \mathbb{Z}^*_N, x \leftarrow \mathbb{J}_N \setminus \mathbb{QR}_N\}.
\end{aligned}$$

We remark that

$$\begin{aligned}
\mathsf{Adv}^{\mathsf{Gap2}^k\text{-}\mathsf{Resm}}_{\mathcal{B}}(\lambda) = &\Big| \Pr[\mathcal{B}(1^\lambda, (a, b, k, N)) = 1 \mid (a, b, k, N) \leftarrow D_0] \\
&\quad - \Pr[\mathcal{B}(1^\lambda, (a, b, k, N)) = 1 \mid (a, b, k, N) \leftarrow R_k] \Big|.
\end{aligned}$$

In order to prove the statement, we connect the distribution D_0 and R_k by a chain of distributions which we prove are all computationally indistinguishable under k-QRm and k-SJSm assumptions.

$$D_0 \overset{c}{\approx} D'_1 \overset{c}{\approx} D_1 \overset{c}{\approx} D'_2 \overset{c}{\approx} \ldots \overset{c}{\approx} D_{k-2} \overset{c}{\approx} D'_{k-1} \overset{c}{\approx} D_{k-1} \overset{c}{\approx} R_k$$

First, we show that under $\mathsf{k\text{-}QRm}^*$ assumption, $D_{i-1} \overset{c}{\approx} D'_i$ for every $i \in [k-1]$:

Let $\mathcal{D}$ be a distinguisher that distinguishes between the distributions D_{i-1} and D'_i with advantage $\mathsf{Adv}_{\mathcal{D}}^{D_{i-1},D'_i}(\lambda)$. We construct a $\mathsf{k\text{-}QRm}^*$ distinguisher $\mathcal{A}_i^{\mathsf{k\text{-}QRm}^*}$ with the same advantage.

$\mathcal{A}_i^{\mathsf{k\text{-}QRm}^*}$ on input $(1^\lambda, (a, b, k, N))$ chooses a random $z \leftarrow \overline{\mathbb{J}_N}$ and runs $\mathcal{D}$ on input $(1^\lambda, (z^{2^i} \cdot a^{2^{i-1}}, b, k, N))$. In case $a = y^2$, $y \leftarrow \mathbb{J}_N$, then $z^{2^i} \cdot a^{2^{i-1}} = z^{2^i} \cdot y^{2^i} = (zy)^{2^i}$ which is an element from the distribution $\{y^{2^i}, y \leftarrow \overline{\mathbb{J}_N}\}$. Otherwise, if $a \leftarrow \mathbb{J}_N \setminus \mathbb{QR}_N$, then $z^{2^i} \cdot a^{2^{i-1}} = (z^2 \cdot a)^{2^{i-1}}$, which is an element from the distribution $\{y^{2^{i-1}}, y \leftarrow \mathbb{J}_N \setminus \mathbb{QR}_N\}$. $\mathcal{A}_i^{\mathsf{k\text{-}QRm}^*}$ outputs the output of $\mathcal{D}$. Then, $\mathsf{Adv}^{\mathsf{k\text{-}QRm}^*}_{\mathcal{A}_i^{\mathsf{k\text{-}QRm}*}}(\lambda) = \mathsf{Adv}_{\mathcal{D}}^{D_{i-1},D'_i}(\lambda)$.

Next, we show that under $\mathsf{k\text{-}SJSm}^*$ assumption $D'_i \overset{c}{\approx} D_i$, for every $i \in [k-1]$:

Let $\mathcal{D}$ be a distinguisher that distinguishes between the distributions D'_i and D_i with advantage $\mathsf{Adv}_{\mathcal{D}}^{D'_i,D_i}(\lambda)$. We construct a $\mathsf{k\text{-}SJSm}^*$ distinguisher $\mathcal{A}_i^{\mathsf{k\text{-}SJSm}^*}$ with the same advantage.

$\mathcal{A}_i^{\mathsf{k\text{-}SJSm}^*}$ on input (a, b, k, N) runs $\mathcal{D}$ on input $(a^{2^{i-1}}, b, k, N)$ and outputs the output of $\mathcal{D}$. In case $a = y^2, y \leftarrow \mathbb{J}_N \setminus \mathbb{QR}_N$, then $a^{2^{i-1}} = y^{2^i}$ is an element from the distribution $\{y^{2^i}, y \leftarrow \mathbb{J}_N \setminus \mathbb{QR}_N\}$. Otherwise, if $a = y^2, y \leftarrow \overline{\mathbb{J}_N}$, then $a^{2^{i-1}} = y^{2^i}$ is an element from the distribution $\{y^{2^i}, y \leftarrow \overline{\mathbb{J}_N}\}$. Then, $\mathsf{Adv}^{\mathsf{k\text{-}SJSm}^*}_{\mathcal{A}_i^{\mathsf{k\text{-}SJSm}*}}(\lambda) = \mathsf{Adv}_{\mathcal{D}}^{D'_i,D_i}(\lambda)$.

Next, we show that under the k-QRm assumption $D_{k-1} \overset{c}{\approx} R_k$:

Let $\mathcal{D}$ be a distinguisher that distinguishes between the distributions D_{k-1} and R_k with advantage $\mathsf{Adv}_{\mathcal{D}}^{D_{k-1},R_k}(\lambda)$. We construct a k-QRm distinguisher $\mathcal{A}^{\mathsf{k\text{-}QRm}}$ with the same advantage.

$\mathcal{A}^{\mathsf{k\text{-}QRm}}$ on input (a, b, k, N) runs $\mathcal{D}$ on input $(a^{2^{k-1}}, b, k, N)$ and outputs the output of $\mathcal{D}$. In case $a \leftarrow \mathbb{QR}_N$, then $a^{2^{k-1}} = x^{2^k}$ for some $x \in \mathbb{Z}_N^*$, which gives a random element of $\{y^{2^k}, y \leftarrow \mathbb{Z}_N^*\}$, hence $(a^{2^{k-1}}, b, k, N)$ is chosen from distribution R_k. Otherwise, if $a \leftarrow \mathbb{J}_N \setminus \mathbb{QR}_N$, then $a^{2^{k-1}}$ is a random element of $\{y^{2^{k-1}}, y \leftarrow \mathbb{J}_N \setminus \mathbb{QR}_N\}$, and thus, $(a^{2^{k-1}}, b, k, N)$ is chosen from the distribution D_{k-1}. Then, $\mathsf{Adv}^{\mathsf{k\text{-}QRm}^*}_{\mathcal{A}^{\mathsf{k\text{-}QRm}}}(\lambda) = \mathsf{Adv}_{\mathcal{D}}^{D_{k-1},R_k}(\lambda)$.

Finally, we define the distinguishers $\mathcal{D}_1$, $\mathcal{D}_2$:

$\mathcal{D}_1$: Chooses and runs one of the distinguishers with following probabilities

- $\mathcal{A}_i^{\mathsf{k\text{-}QRm}^*}$ with probability $\frac{2}{3k-1}$ for every $i \in [k-1]$,
- The adversary $\mathcal{B}$ defined in Lemma 12 for $\mathcal{A} \coloneqq \mathcal{A}_i^{\mathsf{k\text{-}SJSm}^*}$ (we will denote this distinguisher $\mathcal{B}_i$) with probability $\frac{1}{3k-1}$ for every $i \in [k-1]$,
- $\mathcal{A}^{\mathsf{k\text{-}QRm}}$ with probability $\frac{2}{3k-1}$,

and outputs the output of the chosen distinguisher.

$\mathcal{D}_2$: Chooses and runs one of the distinguishers with following probabilities

- $\mathcal{A}_i^{\mathsf{k\text{-}QRm}^*}$ with probability $\frac{1}{3k-3}$ for every $i \in [k-1]$,
- $\mathcal{A}_i^{\mathsf{k\text{-}SJSm}^*}$ with probability $\frac{2}{3k-3}$ for every $i \in [k-1]$,

and outputs the output of the chosen distinguisher.

Now, we have

$$\begin{aligned}
\mathsf{Adv}_{\mathcal{B}}^{\mathsf{Gap2^k\text{-}Resm}}(\lambda) \leq & \sum_{i=1}^{k-1} \mathsf{Adv}_{\mathcal{A}_i^{\mathsf{k\text{-}QRm}^*}}^{\mathsf{k\text{-}QRm}^*}(\lambda) + \sum_{i=1}^{k-1} \mathsf{Adv}_{\mathcal{A}_i^{\mathsf{k\text{-}SJSm}^*}}^{\mathsf{k\text{-}SJSm}^*}(\lambda) + \mathsf{Adv}_{\mathcal{A}^{\mathsf{k\text{-}QRm}}}^{\mathsf{k\text{-}QRm}}(\lambda) \\
\leq & \sum_{i=1}^{k-1} \mathsf{Adv}_{\mathcal{A}_i^{\mathsf{k\text{-}QRm}^*}}^{\mathsf{k\text{-}QRm}}(\lambda) + \frac{1}{2} \sum_{i=1}^{k-1} \mathsf{Adv}_{\mathcal{B}_i}^{\mathsf{k\text{-}QRm}}(\lambda) + \mathsf{Adv}_{\mathcal{A}^{\mathsf{k\text{-}QRm}}}^{\mathsf{k\text{-}QRm}}(\lambda) + \\
& + \sum_{i=1}^{k-1} \mathsf{Adv}_{\mathcal{A}_i^{\mathsf{k\text{-}SJSm}^*}}^{\mathsf{k\text{-}SJSm}}(\lambda) + \frac{1}{2} \sum_{i=1}^{k-1} \mathsf{Adv}_{\mathcal{A}_i^{\mathsf{k\text{-}QRm}^*}}^{\mathsf{k\text{-}SJSm}}(\lambda) \\
= & \frac{3k-1}{2} \cdot \mathsf{Adv}_{\mathcal{D}_1}^{\mathsf{k\text{-}QRm}}(\lambda) + \frac{3k-3}{2} \cdot \mathsf{Adv}_{\mathcal{D}_2}^{\mathsf{k\text{-}SJSm}}(\lambda),
\end{aligned}$$

where the first inequality is due to

$$\begin{aligned}
\mathsf{Adv}_{\mathcal{B}}^{\mathsf{Gap2^k\text{-}Resm}}(\lambda) &= \mathsf{Adv}_{\mathcal{B}}^{D_0,R_k}(\lambda) \\
&\leq \sum_{i=1}^{k-1} \mathsf{Adv}_{\mathcal{B}}^{D_{i-1},D_i'}(\lambda) + \sum_{i=1}^{k-1} \mathsf{Adv}_{\mathcal{B}}^{D_i',D_i}(\lambda) + \mathsf{Adv}_{\mathcal{B}}^{D_{k-1},R_k}(\lambda)
\end{aligned}$$

and as proved above, this induces the distinguishers $\mathcal{A}_i^{\mathsf{k\text{-}QRm}^*}$, $\mathcal{A}_i^{\mathsf{k\text{-}SJSm}^*}$, $\mathcal{A}^{\mathsf{k\text{-}QRm}}$ respectively. The second inequality is the application of Lemma 12 and the following equality is due to the definition of $\mathcal{D}_1, \mathcal{D}_2$. □

Now, following the proof of IND-CPA security of the Joye-Libert scheme [BHJL17, Theorem 1], we prove IND-CPA security of the modified Joye-Libert encryption scheme.

Theorem 14. *Let λ be a security parameter. For every* IND-CPA *adversary $\mathcal{A}$ for the encryption scheme* $(\mathsf{KeyGen}_{\mathsf{mJL}}, \mathsf{Enc}_{\mathsf{mJL}}, \mathsf{Dec}_{\mathsf{mJL}})$ *with parameter $k \geq 1$, there exists a* k-QRm *distinguisher $\mathcal{D}_1$, a* k-SJSm *distinguisher $\mathcal{D}_2$ and a* k-dRI *distinguisher $\mathcal{D}_3$ such that*

$$\mathsf{Adv}_{\mathcal{A}}^{\mathsf{IND\text{-}CPA}}(\lambda) \leq \frac{3k-1}{2} \cdot \mathsf{Adv}_{\mathcal{D}_1}^{\mathsf{k\text{-}QRm}}(\lambda) + \frac{3k-3}{2} \cdot \mathsf{Adv}_{\mathcal{D}_2}^{\mathsf{k\text{-}SJSm}}(\lambda) + \mathsf{Adv}_{\mathcal{D}_3}^{\mathsf{k\text{-}dRI}}(\lambda).$$

Proof. We first introduce 3 hybrid distributions:

Hybrid 0 : $H_0 \coloneqq \{(g, g^d, k, N) \mid g \leftarrow \mathbb{J}_N \setminus \mathbb{QR}_N\}$,

Hybrid 1 : $H_1 \coloneqq \{(g, y^d, k, N) \mid g, y \leftarrow \mathbb{J}_N \setminus \mathbb{QR}_N\}$,

Hybrid 2 : $H_2 \coloneqq \{(g, y^d, k, N) \mid y \leftarrow \mathbb{J}_N \setminus \mathbb{QR}_N, g = x^{2^k}, x \leftarrow \mathbb{Z}_N^*\}$.

We point out that the distributions of the Hybrid 0 and the Hybrid 1 are indistinguishable under the k-dRI assumption (follows directly from the definition of the k-dRI assumption) and the distributions of the Hybrid 1 and the Hybrid 2 are indistinguishable under the k-QRm and k-SJSm assumptions (follows from Lemma 13). More precisely, using the Union bound (Lemma 4), we get that for every PPT distinguisher $\mathcal{D}$ between the distributions H_0 and H_1 there exists a k-QRm distinguisher $\mathcal{D}_1$, a k-SJSm distinguisher $\mathcal{D}_2$ and a k-dRI distinguisher $\mathcal{D}_3$ such that

$$\mathsf{Adv}_{\mathcal{D}}^{H_0,H_2}(\lambda) \leq \frac{3k-1}{2} \cdot \mathsf{Adv}_{\mathcal{D}_1}^{\text{k-QR}}(\lambda) + \frac{3k-3}{2} \cdot \mathsf{Adv}_{\mathcal{D}_2}^{\text{k-SJS}}(\lambda) + \mathsf{Adv}_{\mathcal{D}_3}^{\mathsf{k\text{-}dRI}}(\lambda).$$

Further, we show that for every PPT IND-CPA attacker $\mathcal{A}$ on the mJL scheme, there exists a PPT distinguisher $\mathcal{D}$ for the distributions H_0 and H_2, such that $\mathsf{Adv}_{\mathcal{A}}^{\mathsf{IND\text{-}CPA}}(\lambda) = \mathsf{Adv}_{\mathcal{D}}^{H_0,H_2}(\lambda)$, by constructing a H_0, H_2 distinguisher which uses $\mathcal{A}$ as a subroutine and has the same advantage.

$\mathcal{D}(1^\lambda, (a, b, k, N))$:

1. $\mathcal{D}$ runs $\mathcal{A}$ on input $(1^\lambda, \mathsf{pk} = (a, b, k, N))$.

2. On receiving $\mathcal{A}$'s query (m_0, m_1), $\mathcal{D}$ chooses a bit $\beta \leftarrow \{0, 1\}$ uniformly at random, samples $r \leftarrow \mathbb{Z}_N^*$, calculates $c \coloneqq a^{m_\beta} \cdot r^{2^k}$ and gives the response c to $\mathcal{A}$.

3. After $\mathcal{A}$ outputs β', $\mathcal{D}$ outputs 1 if $\beta = \beta'$ and 0 otherwise.

We remark that in case $\mathcal{D}$'s input (a, b, k, N) has been chosen from H_0, $\mathcal{D}$ acts as the challenger for $\mathcal{A}$ in the IND-CPA experiment. If (a, b, k, N) has been chosen from H_2, the distributions of encryptions of messages m_0 and m_1, $\{c = a^{m_0} \cdot r^{2^k} \mid r \leftarrow \mathbb{Z}_N^*\}$ and $\{c = a^{m_1} \cdot r^{2^k} \mid r \leftarrow \mathbb{Z}_N^*\}$ respectively, are identical. Therefore, in this case, the probability that $\mathcal{A}$ guesses β is $\frac{1}{2}$. We have:

$$\begin{aligned}
\mathsf{Adv}_{\mathcal{D}}^{H_0,H_2}(\lambda) &= \left| \Pr[\beta = \beta' \mid (a, b, k, N) \leftarrow H_2] - \Pr[\beta = \beta' \mid (a, b, k, N) \leftarrow H_0] \right| \\
&= \left| \Pr[\beta = \beta' \mid (a, b, k, N) \leftarrow H_0] - \frac{1}{2} \right| \\
&= \mathsf{Adv}_{\mathcal{A}}^{\mathsf{IND\text{-}CPA}}(\lambda).
\end{aligned}$$

The theorem follows. □

Theorem 14 shows that the modified Joye-Libert encryption scheme is IND-CPA secure under the k-QRm, k-SJSm and k-dRI assumptions.

Algorithm 2: $\mathsf{DDLog}_{\mathsf{mJL}}(h, \omega)$

$\alpha = 0$;
for $i = 0, \ldots, k-1$ **do**
  $t \coloneqq (\frac{h}{\omega^{\alpha}})^{2^{k-i-1}}$;
  $a^{(i)} \coloneqq \left[t > \frac{N}{2}\right]$;
  $\alpha \coloneqq \alpha + a^{(i)} \cdot 2^i$;
return α

3.3 DDLog for the Modified Joye-Libert encryption scheme

In this section, we consider the distributed discrete logarithm problem for the modified Joye-Libert scheme. Let $\lambda, k \in N$ and $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{KeyGen}_{\mathsf{mJL}}(1^{\lambda}, k)$, $\mathsf{sk} = d$ $\mathsf{pk} = (g, g^d, k, N)$. As we introduced earlier, the DDLog problem is the problem of two players, P_0 and P_1, that are holding $h_0 \in \mathbb{Z}_N^*$ and $h_1 \in \mathbb{Z}_N^*$ respectively, such that $\frac{h_1}{h_0} = \omega^m$ for some subgroup generator ω, and want to convert their shares, without any communication in between them, to shares a_0, a_1, such that $a_1 - a_0 = m$. In the case of the distributed discrete logarithm problem for the mJL scheme, ω will be a generator of a subgroup of order 2^k, more precisely $\omega = g^d$.

In Lemma 15, we review the distributed discrete logarithm procedure for the Goldwasser-Micali scheme [OSY21, Algorithm 5.5]. It serves as a base for the construction of our DDLog procedure for the modified Joye-Libert scheme.

Lemma 15. *Let $N \in \mathbb{N}$ be odd and $h_0, h_1 \in \mathbb{Z}_N^*$, such that $h_1 = h_0 \cdot (-1)^i \mod N$, for some $i \in \mathbb{Z}_2$. Set $z_0 \coloneqq [h_0 > \frac{N}{2}]$ and $z_1 \coloneqq [h_1 > \frac{N}{2}]$. Then $z_0 + z_1 = i \mod 2$.*

Proof. If $i = 0$ then $h_0 = h_1$ and therefore $z_0 = z_1$ and thus $z_0 + z_1 = 0 \mod 2$. If $i = 1$ then $h_0 = -h_1$ and therefore $z_0 \neq z_1$ and thus $z_0 + z_1 = 1 \mod 2$. □

We propose a procedure for solving the distributed discrete logarithm problem for mJL scheme in Algorithm 2. In Theorem 16, we prove the correctness of Algorithm 2.

Theorem 16. *Let $k \in \mathbb{N}$, $k \geq 2$ and $N \in \mathbb{N}$ be a modulus generated by $\mathsf{KeyGen}_{\mathsf{mJL}}(1^{\lambda}, k)$. Let ω be an element of the order 2^k in $\mathbb{Z}_N^*$, let $h_0, h_1 \in \mathbb{Z}_N^*$ and $h_1/h_0 = \omega^a \mod N$ for some $a \in \mathbb{Z}_{2^k}$. Let $a_0 = \sum_{i=0}^{k-1} a_0^{(i)} \cdot 2^i$, $a_1 = \sum_{i=0}^{k-1} a_1^{(i)} \cdot 2^i$ be the outputs of $\mathsf{DDLog}_{\mathsf{mJL}}(h_0, \omega)$, $\mathsf{DDLog}_{\mathsf{mJL}}(h_1, \omega)$ respectively. Then*

$$a_1 - a_0 = a \mod 2^k.$$

Proof. Let us denote $t_0^{(i)}, t_1^{(i)}$ the t value set in the i-th iteration of the algorithm's cycle for $i \in \{0, \ldots, k-1\}$, and $\alpha_0^{(i)}, \alpha_1^{(i)}$ the α value set in the $(i-1)$-th iteration of the algorithm's cycle, for $i \in [k]$, of $\mathsf{DDLog}_{\mathsf{mJL}}(h_0, \omega)$, $\mathsf{DDLog}_{\mathsf{mJL}}(h_1, \omega)$ respectively.

We prove by induction on i that $\alpha_1^{(i)} - \alpha_0^{(i)} = a \mod 2^i$ for every $i \in [k]$. We remark that, even though the i in the algorithm is running from 0 to $k-1$, our

induction runs from $i = 1$ to $i = k$. Notice that $\alpha_0^{(k)}, \alpha_1^{(k)}$ are well defined, even thought there is no k-th iteration of the cycle.

For $i = 1$ we have

$$\frac{t_1^{(0)}}{t_0^{(0)}} = \left(\frac{h_1}{h_0}\right)^{2^{k-1}} = \left(\frac{h_0 \cdot \omega^a}{h_0}\right)^{2^{k-1}} = \left(\omega^{2^{k-1}}\right)^a = (-1)^a \mod N,$$

where the last equality holds because $\omega^{2^{k-1}}$ is an element of order 2 in $\mathbb{Z}_N^*$ with Jacobi symbol equal to 1 (because $k \geq 2$) and only -1 fulfils both of these properties. By Lemma 15, we get $a_0^{(0)} + a_1^{(0)} = a \mod 2$. Therefore $-\alpha_0^{(1)} + \alpha_1^{(1)} = a_0^{(0)} + a_1^{(0)} = a \mod 2$.

Now, we suppose $-\alpha_0^{(i)} + \alpha_1^{(i)} = a \mod 2^i$. That means there exists $b^{(i)} \in \mathbb{Z}$ such that $-\alpha_0^{(i)} + b^{(i)} \cdot 2^i = a - \alpha_1^{(i)}$.

For $t_0^{(i)}$, $t_1^{(i)}$ we have

$$\begin{aligned} t_0^{(i)} &= \left(\frac{h_0}{\omega^{\alpha_0^{(i)}}}\right)^{2^{k-i-1}} = (h_0 \cdot \omega^{-\alpha_0^{(i)}})^{2^{k-i-1}} \mod N, \\ t_1^{(i)} &= \left(\frac{h_1}{\omega^{\alpha_1^{(i)}}}\right)^{2^{k-i-1}} = (h_0 \cdot \omega^{a-\alpha_1^{(i)}})^{2^{k-i-1}} = (h_0 \cdot \omega^{-\alpha_0^{(i)} + b^{(i)} \cdot 2^i})^{2^{k-i-1}} \\ &= t_0^{(i)} \cdot \omega^{b^{(i)} \cdot 2^{k-1}} = t_0^{(i)} \cdot (-1)^{b^{(i)}} \mod N. \end{aligned}$$

If $b^{(i)}$ is even, then $t_1^{(i)} = t_0^{(i)} \mod N$ and therefore $a_0^{(i)} = a_1^{(i)}$. It means that in the i-th step of both algorithms $\mathsf{DDLog}_{\mathsf{mJL}}(h_0, \omega)$, $\mathsf{DDLog}_{\mathsf{mJL}}(h_1, \omega)$ we are adding the same value to the α. In other words, $\alpha_0^{(i+1)} - \alpha_0^{(i)} = \alpha_1^{(i+1)} - \alpha_1^{(i)}$. Therefore,

$$a + \alpha_0^{(i+1)} - \alpha_1^{(i+1)} = a + \alpha_0^{(i)} - \alpha_1^{(i)} = b^{(i)} \cdot 2^i = 0 \mod 2^{i+1},$$

where the last equality follows from the assumption that $b^{(i)}$ was even.

Let $b^{(i)}$ be odd, then $t_1^{(i)} = -t_0^{(i)} \mod N$, and therefore, either

$$\alpha_0^{(i+1)} = \alpha_0^{(i)} + 2^i \quad \text{and} \quad \alpha_1^{(i+1)} = \alpha_1^{(i)},$$

or

$$\alpha_0^{(i+1)} = \alpha_0^{(i)} \quad \text{and} \quad \alpha_1^{(i+1)} = \alpha_1^{(i)} + 2^i.$$

Therefore,

$$a + \alpha_0^{(i+1)} - \alpha_1^{(i+1)} = a + \alpha_0^{(i)} - \alpha_1^{(i)} \pm 2^i = b^{(i)} \cdot 2^i \pm 2^i = \frac{b^{(i)} \pm 1}{2} \cdot 2^{i+1} = 0 \mod 2^{i+1},$$

where the last equality holds because $\frac{b^{(i)} \pm 1}{2} \in \mathbb{Z}$, as $b^{(i)}$ is an odd number.

Finally, $a_1 - a_0 = \alpha_1^{(k)} + \alpha_0^{(k)} = a \mod 2^k$. □

3.4 Homomorphic Secret Sharing from the Modified Joye-Libert encryption scheme

In this section, we propose algorithms Share, Eval and Dec and prove these form a Homomorphic Secret Sharing for RMS programs with a magnitude bound 2^k.

Notation. In our HSS construction we use two different representations of elements. More precisely, we represent an element $x \in \mathbb{Z}_{2^k}$ by its mJL ciphertext, as well as its additive secret sharing. To denote these two representations we use the same notation as [BGI16]. For $x \in \mathbb{Z}_{2^k}$ we denote:

1. $[\![x]\!]$ a modified Joye-Libert encryption of the element x, where the encryption uses the public key generated by $\mathsf{KeyGen}(1^\lambda, k+s)$.
2. $\langle x \rangle$ an additively secret shared element over $\mathbb{Z}_{2^{k+s}}$. More specifically two elements $\langle x \rangle_0, \langle x \rangle_1 \in \mathbb{Z}_{2^{k+s}}$, held by the two evaluation parties P_0 and P_1 respectively, such that $\langle x \rangle_1 - \langle x \rangle_0 = x \mod 2^{k+s}$.

As well as the HSS constructions by [BGI16] and [OSY21] our construction of HSS relies on pseudorandom functions. The pseudorandom functions has been introduced by [GGM86]. We review the definition of pseudorandom function from [KL20].

Definition 30. *Let $\lambda \in \mathbb{N}$. Let $F : \{0,1\}^{\mathsf{key}} \times \{0,1\}^{\mathsf{in}} \to \{0,1\}^{\mathsf{out}}$ be a function. We say F is a pseudorandom function (PRF), if for every PPT distinguisher $\mathcal{D}$, there exists a negligible function ϵ such that*

$$\left| \Pr[1 \leftarrow \mathcal{D}^{F(k,\cdot)}(1^\lambda)] - \Pr[1 \leftarrow \mathcal{D}^{f(\cdot)}(1^\lambda)] \right| \leq \epsilon(\lambda),$$

where the key k is chosen uniformly at random from $\{0,1\}^{\mathsf{key}}$, f is a function chosen uniformly at random from all functions with domain $\{0,1\}^{\mathsf{in}}$ and range $\{0,1\}^{\mathsf{out}}$ and $\mathcal{D}^{g(\cdot)}$ denotes that the distinguisher $\mathcal{D}$ is granted an oracle access to an oracle responding to queries $x \in \{0,1\}^{\mathsf{in}}$ by $g(x)$.

We use the notation from [BGI16] and we denote by $\phi \leftarrow \mathsf{PRFGen}(1^\lambda)$ the generation of the key defining the pseudorandom function ϕ.

In the following theorem, we show the correctness of the HSS scheme introduced in Figure 8.

Theorem 17. *Let $\lambda \in \mathbb{N}$, p be a polynomial and $m \leq p(\lambda)$. Proposed algorithms $\mathsf{Share}, \mathsf{Eval}, \mathsf{Dec}$ defined in Figure 8 satisfy the δ-correctness property of Homomorphic Secret Sharing (Definition 6) for RMS programs of size m and magnitude bound $M = 2^k$ with the probability of error δ at most $m \cdot (l+1) \cdot 2^{-s}$. In particular, if s is a polynomial function in λ, then $(\mathsf{Share}, \mathsf{Eval}, \mathsf{Dec})$ satisfy the statistical correctness property of HSS.*

Proof. We follow the structure of the proof of correctness of the HSS scheme in [BGI16, Lemma 3.10].

Let P be a RMS program of size $m \in \mathbb{N}$ and magnitude bound $M = 2^k$ and denote

$$\langle dy_j \rangle_0 \coloneqq \sum_{t=1}^{l} 2^{t-1} \cdot \langle d^{(t)} y_j \rangle_0,$$

$$\langle dy_j \rangle_1 \coloneqq \sum_{t=1}^{l} 2^{t-1} \cdot \langle d^{(t)} y_j \rangle_1.$$

We show that the following two properties hold for all of the memory values after each step of the homomorphic evaluation of program P except with some specified error probability.

$\mathsf{Share}(1^\lambda, (w_1, \ldots, w_n))$**:** 1. Run $\mathsf{KeyGen}_{\mathsf{mJL}}(1^\lambda, k+s)$, denote $\mathsf{pk} = (N, k+s, g, g^d)$ and $\mathsf{sk} = d$ the public and the secret key for the modified Joye-Libert scheme. Let $l = \lceil \log(N) \rceil$ and elements $d^{(t)}$ for $t \in [l]$ be such that $d = \sum_{t=1}^{l} 2^{t-1} d^{(t)}$.

2. Sample a PRF $\phi \leftarrow \mathsf{PRFGen}(1^\lambda)$, $\phi : \{0,1\}^{\lceil \log m \rceil} \times \{0,1\}^l \to \{0,1\}^{k+s}$.

3. For every $i = 1, \ldots, n$:

   (a) $[\![w_i]\!] \leftarrow \mathsf{Enc}_{\mathsf{mJL}}(\mathrm{pk}, w_i)$

   (b) for every $t \in [l] : [\![d^{(t)}]\!] \leftarrow \mathsf{Enc}_{\mathsf{mJL}}(\mathsf{pk}, d^{(t)})$, $r_{t,i} \leftarrow \mathbb{Z}_N^*$, $[\![d^{(t)} w_i]\!] := [\![d^{(t)}]\!]^{w_i} \cdot r_{t,i}^{2^{k+s}}$

4. Sample $\langle w_i \rangle_0, \langle w_i \rangle_1 \leftarrow \mathbb{Z}_{2^{k+s}}$ such that $w_i = \langle w_i \rangle_1 - \langle w_i \rangle_0 \mod 2^{k+s}$.

5. Sample $\langle d^{(t)} w_i \rangle_0, \langle d^{(t)} w_i \rangle_1 \leftarrow \mathbb{Z}_{2^{k+s}}$ such that $d^{(t)} w_i = \langle d^{(t)} w_i \rangle_1 - \langle d^{(t)} w_i \rangle_0 \mod 2^{k+s}$ for every $t \in [l]$.

6. For $b \in \{0,1\}$ output

$$\mathsf{share}_b = \{\phi, ([\![w_i]\!], \{[\![d^{(t)} w_i]\!]\}_{t \in [l]}, \langle w_i \rangle_b, \{\langle d w_i \rangle_b\}_{t \in [l]})_{i \in [n]}\}.$$

$\mathsf{Eval}(\mathsf{P}, b, \mathsf{share}_b)$**:**

Parse $\mathsf{share}_b = \{\phi, ([\![w_i]\!], \{[\![d^{(t)} w_i]\!]\}_{t \in [l]}, \langle w_i \rangle, \{\langle d w_i \rangle\}_{t \in [l]})_{i \in [n]}\}$. For each instruction in P based on its type do:

- Load input value into memory $(\mathsf{id}, y_i \leftarrow w_i)$:
  $\langle y_i \rangle := \langle w_i \rangle$,
  $\langle d^{(t)} y_i \rangle := \langle d^{(t)} w_i \rangle$ for $t \in [l]$.
- Add two memory values $(\mathsf{id}, y_u \leftarrow y_i + y_j)$:
  $\langle y_u \rangle := \langle y_i \rangle + \langle y_j \rangle$,
  $\langle d^{(t)} y_u \rangle := \langle d^{(t)} y_i \rangle + \langle d^{(t)} y_j \rangle$ for $t \in [l]$.
- Multiply an input value by a memory value $(\mathsf{id}, y_u \leftarrow w_i \cdot y_j)$:
  $\langle d y_j \rangle := \sum_{t=1}^{l} 2^{t-1} \cdot \langle d^{(t)} y_j \rangle$,
  $\langle y_u \rangle := \mathsf{DDLog}_{\mathsf{mJL}}([\![w_i]\!]^{\langle d y_j \rangle}, g^d) + \phi(\mathsf{id}, 0) \mod 2^{k+s}$,
  $\langle d^{(t)} y_u \rangle := \mathsf{DDLog}_{\mathsf{mJL}}([\![d^{(t)} w_i]\!]^{\langle d y_j \rangle}, g^d) + \phi(\mathsf{id}, t) \mod 2^{k+s}$ for every $t \in [l]$.
- Output value from memory as an element of $\mathbb{Z}_\beta$ $(\mathsf{id}, \beta, \mathsf{out}_j \leftarrow y_i)$:
  $\langle \mathsf{out}_j \rangle := \langle y_i \rangle \mod \beta$.

$\mathsf{Dec}(\langle \mathsf{out} \rangle_0, \langle \mathsf{out} \rangle_1, \beta)$**:**

Output $\mathsf{out} := \langle \mathsf{out} \rangle_1 - \langle \mathsf{out} \rangle_0 \mod \beta$.

Figure 8 – Homomorphic secret sharing from modified Joye-Libert scheme.

1. $\langle y \rangle_1 - \langle y \rangle_0 = y$ in $\mathbb{Z}$
2. $\langle dy \rangle_1 - \langle dy \rangle_0 = dy$ in $\mathbb{Z}$

At the beginning of the evaluation there are no memory values. Therefore the above claim is satisfied. We analyse every type of instruction that can occur in P.

1. *Load input value into memory* $(\mathsf{id}, y_i \leftarrow w_i)$*:* We assign directly the additive shares mod 2^{k+s} produced by Share to the additive shares of y_i and $d^{(t)}y_i$. Using the same reasoning as in [OSY21], if $a_1 - a_0 = a \mod 2^{k+s}$, $a_0, a_1 \in \mathbb{Z}_{2^{k+s}}$, then $a_1 - a_0 = a$ in $\mathbb{Z}$ if $a_1 - a \geq 0$. Therefore, there are a possible choices of a_1 such that $a_1 - a_0 \neq a$ in $\mathbb{Z}$. As all of the values that occur during the computation, including the input values, are bounded by 2^k, by the random choice of $\langle y \rangle_1$, the probability that $\langle y \rangle_1 - \langle y \rangle_0 \neq y$ in $\mathbb{Z}$ is $\leq \frac{2^k}{2^{k+s}} = 2^{-s}$. The same holds for the shares of $d^{(t)}y$ for every $t \in [l]$. Therefore, $\langle dy \rangle_1 - \langle dy \rangle_0 \neq dy$ in $\mathbb{Z}$ with probability $\leq l \cdot 2^{-s}$ by the union bound (Lemma 4).

2. *Add two memory values* $(\mathsf{id}, y_u \leftarrow y_i + y_j)$*:* Suppose that both properties hold for all memory values before the execution of this instruction. Then

$$\begin{aligned}
\langle y_u \rangle_1 - \langle y_u \rangle_0 &= (\langle y_i \rangle_1 + \langle y_j \rangle_1) - (\langle y_i \rangle_0 + \langle y_j \rangle_0) \\
&= y_i + y_j = y_u \text{ in } \mathbb{Z} \\
\langle dy_u \rangle_1 - \langle dy_u \rangle_0 &= \sum_{t=1}^{l} 2^{t-1} \cdot \langle d^{(t)} y_u \rangle_1 - \sum_{t=1}^{l} 2^{t-1} \cdot \langle d^{(t)} y_u \rangle_0 \\
&= \sum_{t=1}^{l} 2^{t-1} \cdot (\langle d^{(t)} y_i \rangle_1 + \langle d^{(t)} y_j \rangle_1) - \sum_{t=1}^{l} 2^{t-1} \cdot (\langle d^{(t)} y_i \rangle_0 + \langle d^{(t)} y_j \rangle_0) \\
&= (dy_i + dy_j) = dy_u \text{ in } \mathbb{Z}
\end{aligned}$$

3. *Multiply an input value by a memory value* $(\mathsf{id}, y_u \leftarrow w_i \cdot y_j)$*:*

   Suppose that both properties hold for all memory values before the execution of this instruction. We have

$$\frac{[\![w_i]\!]^{\langle dy_j \rangle_1}}{[\![w_i]\!]^{\langle dy_j \rangle_0}} = [\![w_i]\!]^{dy_j} = g^{dw_i y_j} \cdot r^{2^{k+s} dy_j} = (g^d)^{w_i y_j} \mod N.$$

   Where the first equality holds because $\langle dy_j \rangle_1 - \langle dy_j \rangle_0 = dy_j$ in $\mathbb{Z}$, the second equality is only the definition of the mJL ciphertext and the third equality holds because $r^{2^{k+s} d} = 1 \mod N$ for every $r \in \mathbb{Z}_N^*$.

   Therefore, by Theorem 16:

$$\begin{aligned}
\langle y_u \rangle_1 - \langle y_u \rangle_0 =& \mathsf{DDLog}_{\mathsf{mJL}}([\![w_i]\!]^{\langle dy_j \rangle_1}, g^d) + \phi(\mathrm{id},0) \\
&- (\mathsf{DDLog}_{\mathsf{mJL}}([\![w_i]\!]^{\langle dy_j \rangle_0}, g^d) + \phi(\mathrm{id},0)) = w_i y_j \mod 2^{k+s}.
\end{aligned}$$

   As every intermediate computation value, including y_u, is bounded by 2^k and as the choice of $\langle y_u \rangle_1$, $\langle y_u \rangle_0$ in $\mathbb{Z}_{2^{k+s}}$ is thanks to the addition of a

pseudorandom value $\phi(\mathsf{id}, 0)$ indistinguishable from uniformly random conditioned on $\langle y_u \rangle_1 - \langle y_u \rangle_0 = y_u \mod 2^{k+s}$, we can apply the same analysis as described in the step 1. We are getting that the probability of the first property not holding is at most $2^{-s} + \mathrm{negl}(\lambda)$.

For the 2. property we have for every $t \in [l]$:

$$\frac{[\![d^{(t)} w_i]\!]^{\langle dy_j \rangle_1}}{[\![d^{(t)} w_i]\!]^{\langle dy_j \rangle_0}} = [\![d^{(t)} w_i]\!]^{dy_j} = g^{dd^{(t)} w_i y_j} \cdot r^{2^k dy_j} = (g^d)^{d^{(t)} w_i y_j}.$$

Therefore, again by Theorem 16:

$$\begin{aligned}\mathsf{DDLog}_{\mathsf{mJL}}([\![d^{(t)} w_i]\!]^{\langle dy_j \rangle_1}, g^d) + \phi(\mathrm{id,t}) - (\mathsf{DDLog}_{\mathsf{mJL}}([\![d^{(t)} w_i]\!]^{\langle dy_j \rangle_0}, g^d) + \phi(\mathrm{id,t})) \\ = d^{(t)} w_i y_j \mod 2^{k+s}.\end{aligned}$$

By the same analysis as before, we get the probability of the above equation not holding in $\mathbb{Z}$ at most $2^{-s} + \mathrm{negl}(\lambda)$.

Finally, by the union bound we get the following equation, and thus, the property 2 except with probability $\leq l \cdot (2^{-s} + \mathrm{negl}(\lambda)) = l \cdot 2^{-s} + \mathrm{negl}(\lambda)$.

$$\begin{aligned}\langle dy_u \rangle_1 - \langle dy_u \rangle_0 &= \sum_{t=1}^{l} 2^{t-1} (\mathsf{DDLog}_{\mathsf{mJL}}([\![d^{(t)} w_i]\!]^{\langle dy_j \rangle_1}, g^d) + \phi(\mathsf{id}, t)) \\ &\quad - \sum_{t=1}^{l} 2^{t-1} (\mathsf{DDLog}_{\mathsf{mJL}}([\![d^{(t)} w_i]\!]^{\langle dy_j \rangle_0}, g^d) + \phi(\mathsf{id}, t)) \\ &= \sum_{t=1}^{l} 2^{t-1} d^{(t)} w_i y_j = d w_i y_j \text{ in } \mathbb{Z}\end{aligned}$$

4. *Output value from memory as an element of $\mathbb{Z}_\beta$ $(\mathsf{id}, \beta, \mathsf{out}_j \leftarrow y_i)$:* This instruction does not alter any memory values, therefore, if the properties hold before executing this instruction, then they hold also afterwards.

   We want to show that the homomorphic execution of this instruction also produces a correct output value, that is $\langle \mathsf{out}_j \rangle_1 - \langle \mathsf{out}_j \rangle_0 = y_i \mod \beta$.

   Supposing the property 1 is true for all of the memory values before the execution of this instruction, then

   $$\langle y_i \rangle_1 - \langle y_i \rangle_0 = y_i \text{ in } \mathbb{Z},$$

   then we also have this equation in $\mathbb{Z}_\beta$ and therefore

   $$\langle \mathsf{out}_j \rangle_1 - \langle \mathsf{out}_j \rangle_0 = (\langle y_i \rangle_1 \mod \beta) - (\langle y_i \rangle_0 \mod \beta) = y_i \mod \beta.$$

Finally, using the union bound, we have

$$\mathsf{Dec}(\mathsf{out}_0, \mathsf{out}_1, \beta) = (\mathsf{out}_1 - \mathsf{out}_0) \mod \beta = \mathsf{out} \mod \beta,$$

where $(\mathsf{out} \mod \beta) = P(w_1, \ldots, w_n)$, except with error probability $\leq m \cdot (l+1) \cdot 2^{-s} + \mathrm{negl}(\lambda)$.

Moreover, if s is a polynomial function in λ, then $m \cdot (l+1) \cdot 2^{-s} + \mathrm{negl}(\lambda) = \mathrm{negl}(\lambda)$, as m and l are also polynomial functions in λ, and the algorithms $(\mathsf{Share}, \mathsf{Eval}, \mathsf{Dec})$ satisfy the statistical correctness property of HSS. □

In order to prove that our HSS scheme is secure, we need to assume that the underlying encryption scheme, in our case the modified Joye-Libert scheme, remains secure even if we use it to encrypt functions of its secret key. This notion is formalized by the KDM security (Definition 5).

Theorem 18. *Assuming the* IND-CPA *and the* KDM *security of the modified Joye-Libert scheme over the set of bit selecting functions* $(\{f_i \mid f_i(x) = x_i, i \in [l]\}$, *where* x_i *is the i-th bit of* x *and* $l = \lceil \log N \rceil)$, *the algorithm* Share *as defined above satisfies the security requirement from Definition 6.*

Proof. We closely follow the structure of the proofs of [BGI16, Lemma 3.11] and [OSY21, Theorem 4.4]. We introduce hybrid models in which we alter the way of producing the share for the party $b \in \{0, 1\}$ to bridge the distribution of shares of $w_1, \ldots, w_n$ and the distribution of shares of $v_1, \ldots, v_n$. We show these two distributions are computationally indistinguishable. Fix $b \in \{0, 1\}$ and let H_i denote the output distribution of Hybrid i, for $i = 0, 1, 2, 3$.

Hybrid 0 : Shares are produced from $w_1, \ldots, w_n$ by the Share algorithm.

1. $(\mathsf{share}_0, \mathsf{share}_1) \leftarrow \mathsf{Share}(1^\lambda, (w_1, \ldots, w_n))$.
2. Output $\mathsf{share}_b = \{\phi, (\llbracket w_i \rrbracket, \{\llbracket d^{(t)} w_i \rrbracket\}_{t\in[l]}, \langle w_i \rangle_b, \{\langle d^{(t)} w_i \rangle_b\}_{t\in[l]})_{i\in[n]}\}$.

Hybrid 1 : Additive shares are replaced by random values.

1. $(\mathsf{share}_0, \mathsf{share}_1) \leftarrow \mathsf{Share}(1^\lambda, (w_1, \ldots, w_n))$.
2. Parse $\mathsf{share}_b = \{\phi, (\llbracket w_i \rrbracket, \{\llbracket d^{(t)} w_i \rrbracket\}_{t\in[l]}, \langle w_i \rangle_b, \{\langle d^{(t)} w_i \rangle_b\}_{t\in[l]})_{i\in[n]}\}$.
3. Sample random $r_i, r_{i,t} \leftarrow \mathbb{Z}_{2^{k+s}}$ for $i \in [n]$, $t \in [l]$.
4. Output $\mathsf{share}_b = \{\phi, (\llbracket w_i \rrbracket, \{\llbracket d^{(t)} w_i \rrbracket\}_{t\in[l]}, r_i, \{r_{i,t}\}_{t\in[l]})_{i\in[n]}\}$.

Hybrid 2 : Encryptions of the key bits times w_i values are replaced by encryptions of zero.

1. $(\mathsf{share}_0, \mathsf{share}_1) \leftarrow \mathsf{Share}(1^\lambda, (w_1, \ldots, w_n))$.
2. Parse $\mathsf{share}_b = \{\phi, (\llbracket w_i \rrbracket, \{\llbracket d^{(t)} w_i \rrbracket\}_{t\in[l]}, \langle w_i \rangle_b, \{\langle d^{(t)} w_i \rangle_b\}_{t\in[l]})_{i\in[n]}\}$.
3. Sample random $r_i, r_{i,t} \leftarrow \mathbb{Z}_{2^{k+s}}$ for $i \in [n]$, $t \in [l]$.
4. Sample $\llbracket 0 \rrbracket_{i,t} \leftarrow \mathsf{Enc}_{\mathsf{mJL}}(\mathsf{pk}, 0)$ for every $i \in [n]$, $t \in [l]$.
5. Output $\mathsf{share}_b = \{\phi, (\llbracket w_i \rrbracket, \{\llbracket 0 \rrbracket_{i,t}\}_{t\in[l]}, r_i, \{r_{i,t}\}_{t\in[l]})_{i\in[n]}\}$.

Hybrid 3 : Encryptions of input values w_i are replaced by encryptions of zero.

1. Run $\mathsf{KeyGen}_{\mathsf{mJL}}(1^\lambda, k + s)$, denote $\mathsf{pk} = (N, k + s, g, g^d)$ and $\mathsf{sk} = d$ the public and the secret key for the modified Joye-Libert scheme.
2. Sample a PRF $\phi \leftarrow \mathsf{PRFGen}(1^\lambda)$, $\phi : \{0, 1\}^{\lceil \log m \rceil} \times \{0, 1\}^l \to \{0, 1\}^{k+s}$.
3. Sample random $r_i, r_{i,t} \leftarrow \mathbb{Z}_{2^{k+s}}$ for $i \in [n]$, $t \in [l]$.
4. Sample $\llbracket 0 \rrbracket_{i,t} \leftarrow \mathsf{Enc}_{\mathsf{mJL}}(\mathsf{pk}, 0)$ for every $i \in [n]$, $t \in [l]$.
5. Sample $\llbracket 0 \rrbracket_i \leftarrow \mathsf{Enc}_{\mathsf{mJL}}(\mathsf{pk}, 0)$ for every $i \in [n]$.
6. Output $\mathsf{share}_b = \{\phi, (\llbracket 0 \rrbracket_i, \{\llbracket 0 \rrbracket_{i,t}\}_{t\in[l]}, r_i, \{r_{i,t}\}_{t\in[l]})_{i\in[n]}\}$.

We remark that H_0 represents the honest sharing of $w_1, \ldots, w_n$ and H_3 is an entirely simulated sharing, independent of $w_1, \ldots, w_n$. We prove the output distributions in these hybrid experiments are computationally indistinguishable.

First, we notice that the output distributions of Hybrid 0 and Hybrid 1 are identical, due to the uniformly random choice of additive shares.

Next, let $\mathcal{D}^{H_2,H_1}$ be a distinguisher that is able to distinguish between the output distributions of Hybrid 2 and Hybrid 1 with non-negligible probability. We construct $\mathcal{A}^{\mathsf{KDM}}$, an attacker on the KDM security of the modified Joye-Libert scheme, which succeeds with non-negligible probability.

$\mathcal{A}^{\mathsf{KDM}}(1^\lambda)$:

1. $\mathcal{A}^{\mathsf{KDM}}$ is given the mJL public key pk.
2. $\mathcal{A}^{\mathsf{KDM}}$ sends the set of bit selecting functions $f_1, \ldots f_l$ to the challenger.
3. On challenger's response $c_1, \ldots, c_l$, $\mathcal{A}^{\mathsf{KDM}}$ generates:
   (a) $[\![w_i]\!] \leftarrow \mathsf{Enc}_{\mathsf{mJL}}(\mathsf{pk}, w_i)$ for every $i \in [n]$,
   (b) $r_i, r_{i,t} \leftarrow \mathbb{Z}_{2^{k+s}}$ for every $i \in [n]$, $t \in [l]$,
   (c) For every $i \in [n]$, for every $t \in [l]$ samples $r'_{i,t} \leftarrow \mathbb{Z}_N^*$ and sets $x_{i,t} \coloneqq c_i^{w_i} \cdot (r'_{i,t})^{2^{k+s}} \mod N$.
   (d) $\phi \leftarrow \mathsf{PRFGen}(1^\lambda)$, $\phi : \{0,1\}^{\lceil \log m \rceil} \times \{0,1\}^l \to \{0,1\}^{k+s}$.
4. $\mathcal{A}^{\mathsf{KDM}}$ runs $\mathcal{D}^{H_2,H_1}$ on input $(1^\lambda, \mathsf{share}_b = \{\phi, ([\![w_i]\!], \{x_{i,t}\}_{t\in[l]}, r_i, \{r_{i,t}\}_{t\in[l]})_{i\in[n]}\})$.
5. $\mathcal{A}^{\mathsf{KDM}}$ outputs the output of $\mathcal{D}^{H_2,H_1}$.

We notice that in case $c_1, \ldots, c_l$ are encryptions of bits of the secret key, we have $x_{i,t} = (c_i)^{w_i} \cdot (r'_{i,t})^{2^{k+s}} = g^{d^{(i)} \cdot w_i} \cdot (\hat{r}_i^{w_i} \cdot r'_{i,t})^{2^{k+s}} \mod N$, which for a uniformly random choice of $r'_{i,t}$ gives a random encryption of $d^{(i)} \cdot w_i$. Therefore, the distribution of share_b in this case is identical to the one from Hybrid 1.

In the other case, if $c_1, \ldots, c_l$ are encryptions of zero, we have $x_{i,t} = (c_i)^{w_i} \cdot (r'_{i,t})^{2^{k+s}} = g^0 \cdot (\hat{r}_i^{w_i} \cdot r'_{i,t})^{2^{k+s}} \mod N$, which for a uniformly random choice of $r'_{i,t}$ gives a random encryption of 0. Thus, we get the distribution of share_b identical to the one from Hybrid 2.

Therefore, we have constructed an KDM-security attacker $\mathcal{A}^{\mathsf{KDM}}$ with the following advantage:

$$\begin{aligned}
\mathsf{Adv}^{\mathsf{KDM}}_{\mathcal{A}^{\mathsf{KDM}}}(\lambda) &= \left| \Pr[\mathsf{Exp}^{\mathsf{KDM}}_{\mathcal{A}^{\mathsf{KDM}}}(1^\lambda) = 1] - \frac{1}{2} \right| = \left| \Pr[b'_{\mathsf{KDM}} = b_{\mathsf{KDM}}] - \frac{1}{2} \right| \\
&= \left| \Pr[b'_{\mathsf{KDM}} = 1, b_{\mathsf{KDM}} = 1] + \Pr[b'_{\mathsf{KDM}} = 0, b_{\mathsf{KDM}} = 0] - \frac{1}{2} \right| \\
&= \frac{1}{2} \left| \Pr[b'_{\mathsf{KDM}} = 1 \mid b_{\mathsf{KDM}} = 1] - \Pr[b'_{\mathsf{KDM}} = 1 \mid b_{\mathsf{KDM}} = 0] \right| \\
&= \frac{1}{2} \left| \Pr[b'_{\mathsf{IND}} = 1 \mid b_{\mathsf{IND}} = 1] - \Pr[b'_{\mathsf{IND}} = 1 \mid b_{\mathsf{IND}} = 0] \right| = \frac{1}{2} \mathsf{Adv}^{H_2,H_1}_{\mathcal{D}^{H_2,H_1}}(\lambda),
\end{aligned}$$

where $b_{\mathsf{KDM}}, b'_{\mathsf{KDM}}$ denote the bits b, b' from the KDM security experiment and $b_{\mathsf{IND}}, b'_{\mathsf{IND}}$ denote the bits b, b' from the indistinguishability experiment $\mathsf{Exp}^{H_2,H_1}_{\mathcal{D}^{H_2,H_1}}(1^\lambda)$.

Next, we show for every non-uniform PPT distinguisher $\mathcal{D}^{H_2,H_3}$ that his advantage is negligible. We introduce another hybrid argument, where we define for $i \in \{0,\ldots,n\}$ H'_i as the output distribution of Hybrid 2 where we replace $[\![w_j]\!]$ by random encryption of zero for all $j \in [i]$. Then $H'_0 = H_2$ and $H'_n = H_3$.

Now, for every $i \in [n]$ a distinguisher $\mathcal{D}^{H'_{i-1},H'_i}$ for the distribution of H'_{i-1} and H'_i can be used to construct an attacker on the $\mathsf{IND\text{-}CPA}$ security of the underlying modified Joye-Libert scheme $\mathcal{A}_i^{\mathsf{IND\text{-}CPA}}$.

$\mathcal{A}_i^{\mathsf{IND\text{-}CPA}}$:

1. On receiving the challenge $(1^\lambda, \mathsf{pk})$, $\mathcal{A}_i^{\mathsf{IND\text{-}CPA}}$ chooses $m_0 = w_i$ and $m_1 = 0$ and sends these to the challenger.

2. $\mathcal{A}_i^{\mathsf{IND\text{-}CPA}}$ generates:

   - $r_j, r_{j,t} \leftarrow \mathbb{Z}_{2^{k+s}}$ for $j \in [n]$, $t \in [l]$,
   - $[\![0]\!]_{j,t} \leftarrow \mathsf{Enc}_{\mathsf{mJL}}(\mathsf{pk}, 0)$ for $j \in [n]$, $t \in [l]$,
   - $[\![0]\!]_j \leftarrow \mathsf{Enc}_{\mathsf{mJL}}(\mathsf{pk}, 0)$ for $j \in [i-1]$,
   - $[\![w_j]\!] \leftarrow \mathsf{Enc}_{\mathsf{mJL}}(\mathsf{pk}, w_j)$ for $j \in \{i+1,\ldots,n\}$,
   - $\phi \leftarrow \mathsf{PRFGen}(1^\lambda)$, $\phi : \{0,1\}^{\lceil \log m \rceil} \times \{0,1\}^l \to \{0,1\}^{k+s}$.

3. On receiving the response ciphertext c, $\mathcal{A}_i^{\mathsf{IND\text{-}CPA}}$ runs $\mathcal{D}^{H'_{i-1},H'_i}$ on input

$$\{\phi, ([\![0]\!]_j, \{[\![0]\!]_{j,t}\}_{t\in[l]}, r_j, \{r_{j,t}\}_{t\in[l]})_{j\in[i-1]},$$
$$(c, \{[\![0]\!]_{i,t}\}_{t\in[l]}, r_i, \{r_{i,t}\}_{t\in[l]}), ([\![w_j]\!], \{[\![0]\!]_{j,t}\}_{t\in[l]}, r_j, \{r_{j,t}\}_{t\in[l]})_{j\in\{i+1,\ldots,n\}}\}.$$

4. $\mathcal{A}_i^{\mathsf{IND\text{-}CPA}}$ outputs whatever $\mathcal{D}^{H'_{i-1},H'_i}$ outputs.

We remark that if $c = \mathsf{Enc}_{\mathsf{mJL}}(\mathsf{pk}, w_i)$, $\mathcal{D}^{H'_{i-1},H'_i}$ receives an element from H'_{i-1} as an input and if $c = \mathsf{Enc}_{\mathsf{mJL}}(\mathsf{pk}, 0)$, $\mathcal{D}^{H'_{i-1},H'_i}$ receive an element from H'_i as an input. Therefore, if we denote $b_{\mathsf{IND\text{-}CPA}}, b'_{\mathsf{IND\text{-}CPA}}$ the bits b, b' from the $\mathsf{IND\text{-}CPA}$ security experiment and $b_{\mathsf{IND}}, b'_{\mathsf{IND}}$ denote the bits b, b' from the indistinguishability experiment $\mathsf{Exp}^{H'_{i-1},H'_i}_{\mathcal{D}^{H'_{i-1},H'_i}}(1^\lambda)$, we have

$$\begin{aligned}
\mathsf{Adv}^{\mathsf{IND\text{-}CPA}}_{\mathcal{A}_i^{\mathsf{IND\text{-}CPA}}}(\lambda) &= \left| \Pr[\mathsf{Exp}^{\mathsf{IND\text{-}CPA}}_{\mathcal{A}_i^{\mathsf{IND\text{-}CPA}}}(1^\lambda) = 1] - \frac{1}{2} \right| = \left| \Pr[b'_{\mathsf{IND\text{-}CPA}} = b_{\mathsf{IND\text{-}CPA}}] - \frac{1}{2} \right| \\
&= \left| \Pr[b'_{\mathsf{IND\text{-}CPA}} = 1, b_{\mathsf{IND\text{-}CPA}} = 1] + \Pr[b'_{\mathsf{IND\text{-}CPA}} = 0, b_{\mathsf{IND\text{-}CPA}} = 0] - \frac{1}{2} \right| \\
&= \frac{1}{2} \left| \Pr[b'_{\mathsf{IND\text{-}CPA}} = 1 \mid b_{\mathsf{IND\text{-}CPA}} = 1] - \Pr[b'_{\mathsf{IND\text{-}CPA}} = 1 \mid b_{\mathsf{IND\text{-}CPA}} = 0] \right| \\
&= \frac{1}{2} \left| \Pr[b'_{\mathsf{IND}} = 1 \mid b_{\mathsf{IND}} = 1] - \Pr[b'_{\mathsf{IND}} = 1 \mid b_{\mathsf{IND}} = 0] \right| \\
&= \frac{1}{2} \mathsf{Adv}^{H'_{i-1},H'_i}_{\mathcal{D}^{H'_{i-1},H'_i}}(\lambda).
\end{aligned}$$

Together we have

$$\mathsf{Adv}^{H_2,H_3}_{\mathcal{D}^{H_2,H_3}}(\lambda) \leq \sum_{i=1}^{n} \mathsf{Adv}^{H'_{i-1},H'_i}_{\mathcal{D}^{H'_{i-1},H'_i}}(\lambda) \leq 2 \cdot \sum_{i=1}^{n} \mathsf{Adv}^{\mathsf{IND\text{-}CPA}}_{\mathcal{A}_i^{\mathsf{IND\text{-}CPA}}}(\lambda) \leq \mathrm{negl}(\lambda).$$

Thus, the indistinguishability of distributions H_2 and H_3 follows from the semantic security of the modified Joye-Libert scheme.

Therefore, for every PPT distinguisher $\mathcal{D}$ we have

$$\mathsf{Adv}^{H_0,H_3}_{\mathcal{D}}(\lambda) \leq \mathsf{Adv}^{H_0,H_1}_{\mathcal{D}}(\lambda) + \mathsf{Adv}^{H_1,H_2}_{\mathcal{D}}(\lambda) + \mathsf{Adv}^{H_2,H_3}_{\mathcal{D}}(\lambda) \leq \mathrm{negl}(\lambda).$$

We have shown that one party's share distribution of input $w_1, \ldots, w_n$ is indistinguishable from entirely simulated distribution independent of the input values. Together with the same argumentation for the shares of $v_1, \ldots, v_n$, it implies that, for every $b \in \{0, 1\}$, the distributions of share_b and $\mathsf{share}_b{}'$ are computationally indistinguishable, where $(\mathsf{share}_0, \mathsf{share}_1) \leftarrow \mathsf{Share}(1^\lambda, (w_1, \ldots, w_n))$ and $(\mathsf{share}_0{}', \mathsf{share}_1{}') \leftarrow \mathsf{Share}(1^\lambda, (v_1, \ldots, v_n))$. This concludes the proof. □

Theorem 17 and Theorem 18 imply that the algorithms $(\mathsf{Share}, \mathsf{Eval}, \mathsf{Dec})$ form a statistically correct Homomorphic Secret Sharing under the k-QRm, k-SJSm and k-dRI assumptions, assuming KDM security of the modified Joye-Libert encryption scheme over the set of bitselecting functions, and setting the paramater s to be a polynomial function of the security parameter.

Conclusion

In this thesis, we focused on constructions of homomorphic secret sharing from assumptions not known to imply the fully homomorphic encryption. Such constructions have been introduced in [BGI16] and [OSY21]. We studied in detail the *distributed discrete logarithm* problem, the common denominator of this group of HSS constructions. The thesis contains two independent results, the first one being a negative result limiting the effectiveness of solving the DDLog problem in prime order groups in the preprocessing model and the second one being a new HSS construction.

The first result is introduced in the Chapter 2. Motivated by the HSS construction by [BGI16], which reaches a non-negligible correctness error induced by an error in the solution of the DDLog problem, and the follow-up work by [DKK20], showing an optimal DDLog protocol reaching an error $O(W/T^2)$, we analysed the DDLog problem for prime order groups in the preprocessing model. Adapting the technique comparing the auxiliary input and bit-fixing model introduced by [CDG18] we derived an upper bound for the success probability in the DDLog problem with preprocessing. Assuming $N \geq W \log W$, our bound for a big preprocessing advice ($S = \Omega(N/W)$) translates into a bound on time-space trade-off in the DDLog problem $ST^2 = \Omega(\epsilon N)$. For a small preprocessing advice ($S = O(N/W)$), our bound translates to a lower bound on the time complexity of the DDLog problem $T^2 = \Omega(\epsilon W)$. For a constant probability of success our bound for the distributed discrete logarithm problem with preprocessing with a small advice matches the time complexity of the algorithm with no preprocessing running in time $T^2 = O(\frac{W}{1-\epsilon})$ from [DKK20]. Therefore, this algorithm can be considered as optimal even if a small preprocessing is allowed. As an open question we leave the time complexity bounds on the distributed discrete logarithm problem in generic groups of orders which are not prime.

The second result is a new HSS construction for RMS programs with magnitude bound 2^k introduced in the Chapter 3. We based our HSS construction on Joye-Libert encryption scheme [JL13]. To be able to construct a DDLog protocol, we made several changes in the JL scheme, resulting in the *Modified Joye-Libert* (mJL) encryption scheme. Due to the modifications we made, we needed to adapt the security proof of the scheme, which includes modifying assumptions from the original proof (k-QRm, k-SJSm) and introducing a new assumption (k-dRI). An important question, which stays open, is whether these assumptions are secure. Our DDLog protocol for the mJL scheme enjoys the probability of success 1 and gives a possibility of constructing an HSS protocol with a negligible error probability. A downside of our HSS construction is that in order to achieve a negligible error probability for RMS programs with magnitude bound 2^k, we cannot use the modified Joye-Libert scheme with the message space $\mathbb{Z}_{2^k}$, which would suffice to encrypt all of the computation values. Instead, we need to use the mJL scheme with the message space of size $\mathbb{Z}_{2^{k+s}}$, where s is a polynomial function in the security parameter. The possibility of removing this issue that influences the protocol's complexity stays an open question. Our HSS construction's security is based on the KDM security of mJL scheme over the set of bitselecting functions. As an open question, we leave whether this property holds.

Bibliography

[BGI15] Elette Boyle, Niv Gilboa, and Yuval Ishai. Function secret sharing. In Elisabeth Oswald and Marc Fischlin, editors, *Advances in Cryptology - EUROCRYPT 2015 - 34th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Sofia, Bulgaria, April 26-30, 2015, Proceedings, Part II*, volume 9057 of *Lecture Notes in Computer Science*, pages 337–367. Springer, 2015.

[BGI16] Elette Boyle, Niv Gilboa, and Yuval Ishai. Breaking the circuit size barrier for secure computation under DDH. In Matthew Robshaw and Jonathan Katz, editors, *Advances in Cryptology - CRYPTO 2016 - 36th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 14-18, 2016, Proceedings, Part I*, volume 9814 of *Lecture Notes in Computer Science*, pages 509–539. Springer, 2016.

[BGI+18] Elette Boyle, Niv Gilboa, Yuval Ishai, Huijia Lin, and Stefano Tessaro. Foundations of homomorphic secret sharing. In Anna R. Karlin, editor, *9th Innovations in Theoretical Computer Science Conference, ITCS 2018, January 11-14, 2018, Cambridge, MA, USA*, volume 94 of *LIPIcs*, pages 21:1–21:21. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2018.

[BHJL17] Fabrice Benhamouda, Javier Herranz, Marc Joye, and Benoît Libert. Efficient cryptosystems from 2^k-th power residue symbols. *J. Cryptol.*, 30(2):519–549, 2017.

[BKS19] Elette Boyle, Lisa Kohl, and Peter Scholl. Homomorphic secret sharing from lattices without FHE. In Yuval Ishai and Vincent Rijmen, editors, *Advances in Cryptology - EUROCRYPT 2019 - 38th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Darmstadt, Germany, May 19-23, 2019, Proceedings, Part II*, volume 11477 of *Lecture Notes in Computer Science*, pages 3–33. Springer, 2019.

[BRS02] John Black, Phillip Rogaway, and Thomas Shrimpton. Encryption-scheme security in the presence of key-dependent messages. In Kaisa Nyberg and Howard M. Heys, editors, *Selected Areas in Cryptography, 9th Annual International Workshop, SAC 2002, St. John's, Newfoundland, Canada, August 15-16, 2002. Revised Papers*, volume 2595 of *Lecture Notes in Computer Science*, pages 62–75. Springer, 2002.

[CDG18] Sandro Coretti, Yevgeniy Dodis, and Siyao Guo. Non-uniform bounds in the random-permutation, ideal-cipher, and generic-group models. In Hovav Shacham and Alexandra Boldyreva, editors, *Advances in Cryptology - CRYPTO 2018 - 38th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 19-23, 2018, Proceedings, Part I*, volume 10991 of *Lecture Notes in Computer Science*, pages 693–721. Springer, 2018.

[DKK20] Itai Dinur, Nathan Keller, and Ohad Klein. An optimal distributed discrete log protocol with applications to homomorphic secret sharing. *J. Cryptol.*, 33(3):824–873, 2020.

[Gen09] Craig Gentry. Fully homomorphic encryption using ideal lattices. In Michael Mitzenmacher, editor, *Proceedings of the 41st Annual ACM Symposium on Theory of Computing, STOC 2009, Bethesda, MD, USA, May 31 - June 2, 2009*, pages 169–178. ACM, 2009.

[GGM86] Oded Goldreich, Shafi Goldwasser, and Silvio Micali. How to construct random functions. *J. ACM*, 33(4):792–807, 1986.

[GI14] Niv Gilboa and Yuval Ishai. Distributed point functions and their applications. In Phong Q. Nguyen and Elisabeth Oswald, editors, *Advances in Cryptology - EUROCRYPT 2014 - 33rd Annual International Conference on the Theory and Applications of Cryptographic Techniques, Copenhagen, Denmark, May 11-15, 2014. Proceedings*, volume 8441 of *Lecture Notes in Computer Science*, pages 640–658. Springer, 2014.

[JL13] Marc Joye and Benoît Libert. Efficient cryptosystems from 2^k-th power residue symbols. In Thomas Johansson and Phong Q. Nguyen, editors, *Advances in Cryptology - EUROCRYPT 2013, 32nd Annual International Conference on the Theory and Applications of Cryptographic Techniques, Athens, Greece, May 26-30, 2013. Proceedings*, volume 7881 of *Lecture Notes in Computer Science*, pages 76–92. Springer, 2013.

[KL20] Jonathan Katz and Yehuda Lindell. *Introduction to Modern Cryptography*. Chapman & Hall/CRC Cryptography and Network Security Series. CRC Press, 2020.

[OSY21] Claudio Orlandi, Peter Scholl, and Sophia Yakoubov. The rise of Paillier: Homomorphic secret sharing and public-key silent OT. In Anne Canteaut and François-Xavier Standaert, editors, *Advances in Cryptology - EUROCRYPT 2021 - 40th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Zagreb, Croatia, October 17-21, 2021, Proceedings, Part I*, volume 12696 of *Lecture Notes in Computer Science*, pages 678–708. Springer, 2021.

[Ros84] Kenneth H. Rosen. *Elementary Number Theory and Its Applications*. Addison Wesley Publishing Company, 1984.

[Sho97] Victor Shoup. Lower bounds for discrete logarithms and related problems. In Walter Fumy, editor, *Advances in Cryptology - EUROCRYPT '97, International Conference on the Theory and Application of Cryptographic Techniques, Konstanz, Germany, May 11-15, 1997, Proceeding*, volume 1233 of *Lecture Notes in Computer Science*, pages 256–266. Springer, 1997.