PhD-FSTM-2020-62

Faculty of Science, Technology and Medicine

DISSERTATION

Defense held on 20/10/2020 in Esch-sur-Alzette, Luxembourg

to obtain the degree of

DOCTEUR DE L'UNIVERSITÉ DU LUXEMBOURG

EN INFORMATIQUE

by

Hilder Vitor LIMA PEREIRA

Born on 15 September 1989 in Pará (Brazil)

HOMOMORPHIC ENCRYPTION AND MULTILINEAR MAPS BASED ON THE APPROXIMATE-GCD PROBLEM

Dissertation Defense Committee:

Dr Jean-Sébastien Coron, thesis supervisor

Professor, Université du Luxembourg

Dr Alex Biryukov, chairman

Professor, Université du Luxembourg

Dr Volker Müller

Associate Professor, Université du Luxembourg

Dr Diego de Freitas Aranha

Associate Professor, Aarhus University

Dr Fréderik Vercauteren

Professor, Katholieke Universiteit Leuven

Affidavit

I hereby confirm that the PhD thesis entitled "Homomorphic encryption and multilinear maps based on the approximate-GCD problem" has been written independently and without any other sources than cited.

Luxembourg, _____ _____

Name

Acknowledgments

Tout d'abord, je remercie professeur Jean-Sébastien Coron, mon directeur de thèse, pour toutes les discussions enrichissantes, pour toute l'aide qu'il m'a donnée pendant mon doctorat et pour m'encourager à être indépendant et à travailler sur les problèmes qui m'intéressaient le plus. J'ai vraiment beaucoup appris avec lui.

Também gostaria de agradecer ao professor Diego de Freitas Aranha, que foi meu orientador de mestrado, sempre tinha ótimas ideias e me ajudou muito durante meu percurso acadêmico. Antes dele, o professor Alfredo Goldman foi essencial para que eu seguisse na academia, por isso, também lhe agradeço enormemente.

I also want to thank my colleagues for all the discussions, for the seminars and other academic activities that we organized together. Ein besonderer Dank geht an Fabienne, die Sekretärin, die immer engagiert, effizient und hilfreich war.

De coração, à minha mãe, Ana Lícia, ao meu pai, Hilmar, e à minha tia, Ana Lúcia.

Index

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Introduction to modern cryptography

Modern cryptography is a broad area of knowledge lying in the intersection of engineering, mathematics, and computer science, and dealing with secure communication, refraining unauthorized people from reading messages, confirming the identity of someone, providing ways to verify if a given value was computed in a predefined way, among many other tasks. Because of its broadness, defining modern cryptography in simple terms is not easy, as simpler definitions tend to be more incomplete and unclear. For example, in the introduction of [DK07], it is said that "cryptography is the science of keeping secrets secret", but one cannot use this sentence as the definition of cryptography, since it focus only on the *confidentiality*, that is, on the fact that one can use cryptography to guarantee that only *authorized* parties can access the content of a message, but it says nothing about, for instance, *data integrity*, which is not related with secrecy, but with checking if the data received by one party is really the data sent by another party or if it was corrupted or modified in any way. In [KL14], instead of trying to provide a precise definition of cryptography, the authors just say that "modern cryptography involves the study of mathematical techniques for securing digital information, systems, and distributed computations against adversarial attacks."

One of the main objects of study in cryptography is encryption, which involves a cryptographic scheme called cipher and, generally, consists in three algorithms: one to

generate the keys, one to encrypt the messages using the keys and generating ciphertexts, and one to decrypt the ciphertexts. In traditional ciphers, like the AES, the only goal is to guarantee the *secrecy*, i.e., a ciphertext c encrypting a message m under a key k totally hides m, in other words, anyone having access to c but not to k cannot learn anything about the content of the message m. However, several more modern encryption schemes aim to provide more than the confidentiality of the messages. For example, in order-preserving encryption [BCLO09], one is not only concerned about security, but also about being able to compare plaintexts using the corresponding ciphertexts, thus, the following property is provided: if $m_1 \leq m_2$, then $\mathsf{Enc}_k(m_1) \leq \mathsf{Enc}_k(m_2)$. Another example is functional encryption [BSW11], a type of scheme that allows authorized parties to learn a function of the encrypted message. In this type of cipher, instead of having a single key to decrypt, one has functional keys, which are secret keys related to some functions. Then, by decrypting a ciphertext c using a key k_f, one obtains $f(m)$ instead of m. When the keys are generated, one can keep the key corresponding to the identity function (the usual decryption key), and distribute to the authorized parties only the keys corresponding to the desired functions.

Among many other ciphers that provide some extra property besides the confidentiality, over the past years much attention has been given to homomorphic encryption (HE) schemes. As functional encryption, HE allows computation over encrypted data, however, instead of having the output in clear, i.e., $f(m)$, one only has a ciphertext $\mathsf{Enc}(f(m))$. In this case, we say that f was evaluated homomorphically. An encryption scheme is homomorphic for one operation $\oplus$ if given two ciphertexts c_1 and c_2 encrypting m_1 and m_2, one can produce, without using the decryption key, an encryption of $m_1 \oplus m_2$. For example, the textbook RSA is homomorphic for the product modulo n, because it encrypts a message m_i as $c_i := m_i^e \pmod{n}$ and one can multiply two ciphertexts to obtain an encryption of the product of messages, i.e., $c := c_1 \cdot c_2 = (m_1 \cdot m_2)^e \pmod{n}$. But notice that the ciphertexts that can be produced homomorphically in this way with RSA always encrypt a message of the form $\prod_{i=0}^{\ell} m_i^{k_i}$, but one cannot, for example, produce an encryption of $m_1 + m_2$. Thus, the set of functions that can be evaluated homomorphically is

very limited. RSA was published in 1977 and constructing a HE scheme that can evaluate any computable function, which is called a fully homomorphic scheme (FHE), remained as an open problem for almost 30 years. In 1999, Paillier [Pai99] proposed a scheme that is homomorphic for the addition modulo n and, additionally, can homomorphically multiply a ciphertext by a plaintext, i.e., given a ciphertext $c := \mathsf{Enc}(m)$ and a message α, one can generate an encryption of $\alpha \cdot m \mod n$. Therefore, the functions that can be homomorphically evaluated by this scheme are of the form $f(m_1, ..., m_k) = \sum_{i=1}^{k} \alpha_i \cdot m_i$ for known values α_i's. In 2005, Boneh, Goh, and Nissim [BGN05] used cryptographic bilinear maps to construct the first scheme homomorphic for two operations, that is, which can homomorphically multiply and add the messages. However, only one homomorphic product was possible. Therefore, the functions that can be evaluated are of the form $f(m_{1,0}, m_{1,1}..., m_{k,0}, m_{k,1}) = \sum_{i=1}^{k} m_{i,0} \cdot m_{i,1}$. It was only in 2009 that the first FHE scheme was proposed [Gen09].

For each type of cryptographic scheme, there are different security notions, which are ways of defining what is meant by "the scheme is secure". For ciphers, the strongest notion is known as security against adaptive chosen-ciphertext attacks (CCA2), that basically states that a scheme is secure if any (polynomial-time probabilistic) adversary $\mathcal{A}$ cannot distinguish encryptions of two different messages even if $\mathcal{A}$ is allowed to encrypt and decrypt arbitrary plaintexts and ciphertexts of their choice. But it is well known that FHE schemes cannot be CCA2-secure [Gen09], thus, the security notion that a FHE scheme is commonly supposed to satisfy is the security against adaptive chosen-plaintext attacks (IND-CPA), which differs from CCA2 security because the attacker does not have access to a decryption oracle, thus, $\mathcal{A}$ can only obtain encryptions of any message m and has to tell if a ciphertext c encrypts m_0 or m_1 of their choice. Instead of saying that a scheme is IND-CPA-secure, we simply say that it is *semantically secure*.

Proving that a scheme Ω is CCA- or CPA-secure is usually done by defining some mathematical problem $\mathcal{P}$ and proving a statement of the form

$$\mathcal{P} \text{ is computationally hard} \implies \Omega \text{ is semantically secure.}$$

Then, if $\mathcal{P}$ is believed to be hard (for instance, if all known algorithms to solve $\mathcal{P}$ run in exponential time), the security proof is meaningful and the scheme is assumed to be secure.

Therefore, studying the hardness of mathematical problems is an important part of modern cryptography. For example, in 2005 Regev introduced the LWE problem [Reg05] and analyzed its hardness, showing that any algorithm that solves random instances of the LWE problem with non-negligible probability can be turned in an algorithm that solves worst-case instances of lattice problems, as the shortest-vector problem (SVP). Because these lattice problems were already well studied and no efficient algorithm to solve them is known, we conclude that the LWE problem is also hard. Thereafter, several cryptographic schemes were built relying on the hardness of the LWE problem, including FHE schemes. Another hard problem that has been used to construct FHE schemes is the Approximate-GCD problem (AGCD). An important feature of the LWE and the AGCD problems is that they are believed to be quantum-secure, i.e., there is no quantum algorithm that can efficiently solve these problems. This is not the case for the integer factorization problem, for example, which is the underlying hard problem used in the RSA cryptosystems.

1.2 Overview of the thesis

We start this thesis by studying the AGCD problem, which consists in trying to find a secret integer p given many "approximate multiples" of p of the form $c_i = pq_i + r_i$, where the terms r_i's are usually small compared to p. This problem and several variants of it have been used as a simpler alternative to the LWE problem in the construction of several cryptographic primitives. However, in spite of its simplicity, the AGCD problem usually requires huge parameters in order to be both secure and useful for cryptographic applications, which limits its uses. For example, in [DGHV10], an FHE scheme based on the AGCD problem is proposed and to guarantee a security level of λ bits, each ciphertext encrypts a single bit into λ^5 bits, which is a huge ciphertext expansion. Thus, our first contribution is to notice that by randomizing the AGCD problem, one obtains a

problem that is potentially harder. Hence, in Chapter 3, we propose a new cryptographic problem which we name the randomized vector AGCD problem (VAGCD) and we perform extensive cryptanalysis on it. The VAGCD problem consists in grouping m approximate multiples of p into a vector $\mathbf{c} := (pq_1 + r_1, ..., pq_m + r_m)$, then publishing $\tilde{\mathbf{c}} := \mathbf{c} \cdot \mathbf{K}$, where $\mathbf{K} \in \mathbb{Z}^{m \times m}$ is a hidden random matrix. Because the VAGCD problem is a randomized version of the original AGCD problem, it is clear that it cannot be easier than the AGCD problem. Moreover, we noticed that all the attacks against the AGCD problem, when adapted to the VAGCD problem, become much more expensive, therefore, we can select smaller parameters for the same security level. This has the potential of yielding more efficient cryptographic schemes enjoying the same functionalities of the ones constructed using the AGCD problem.

Thus, in Chapter 4, we show that the program obfuscator of [GGH13a] instantiated with the CLT13 multilinear map [CLT13] requires much smaller parameters when cryptanalyzed with the VAGCD problem instead of the usual multi-prime AGCD problem. Hence, we propose an N-party One-round key-exchange protocol based on multilinear maps and indistinguishability obfuscator schemes. Moreover, we provide an implementation of this protocol and execute it for $N = 4$ parties. Although the memory requirements and the time needed to generate the keys are huge, the key derivation is still feasible, taking around one minute per party on a single core of a regular commercial computer. We notice that without using our cryptanalysis of the VAGCD problem and basing the security solely on the original AGCD problem, the size of public parameters would be larger than 100 TB, therefore, the protocol would be totally unpractical. Moreover, to the best of our knowledge, this is the first implementation of a non-interactive N-party key exchange resistant against all the known attacks.

In Chapter 5, we show how to use the VAGCD problem to construct another type of cryptosystem: we propose a *leveled* homomorphic encryption scheme for vectors and matrices (HEVAM). Therewith, one can homomorphically add vectors and matrices, and perform homomorphic matrix-matrix and vector-matrix multiplications. Thanks to a ciphertext decomposition technique adapted from [GSW13], homomorphic products in-

crease the noise only (approximately) additively, thus, we can perform several multiplications before reaching the maximum amount of noise supported by the scheme. Furthermore, because we are using the VAGCD problem, we are able to instantiate the scheme with small parameters and to obtain an efficient implementation. For example, we can perform a sequence of more than one hundred homomorphic products between 128-dimensional vectors and 128×128 matrices in less than one second for a security level of $\lambda = 100$. We conclude Chapter 5 by showing how our scheme performs in two practical applications:

- Homomorphic evaluation of nondeterministic finite automata (NFA): we suppose that a server has a textual database and a client wants to recover the entries corresponding to a query Q, which is a regular expression. Then, Q is represented as an NFA, which is then represented by vectors and matrices, that are finally encrypted with our scheme and sent to the server. The server can then homomorphically test if any text matches Q.

- Homomorphic Naïve Bayes classifier: we assume that the server has a model and the client has the data to be classified. The client represents the data by vectors and matrices, encrypt and send them to server. Finally, the server can homomorphically classify and send back to the client an encryption of the class.

Hence, we show that the VAGCD problem can be efficiently used on problems whose message space is $\mathbb{Z}$ or some the modular ring $\mathbb{Z}_N$. However, for some applications it is better to work over polynomial rings, e.g. $R := \mathbb{Z}[x]/\langle x^N + 1\rangle$, and when we use our HE scheme for vectors and matrices to work over R, the ciphertext expansion is not satisfactory. Therefore, in Chapter 6 we propose another variant of the AGCD problem by grouping N approximate multiples $c_i = pq_i + r_i \in \mathbb{Z}$ into a polynomial $c(x) := c_{N-1} \cdot x^{N-1} + ... + c_1 \cdot x + c_0$, then randomizing $c(x)$ with a hidden random polynomial $k(x)$, obtaining $\tilde{c}(x) := c(x) \cdot k(x)$. We call this problem the Randomized Polynomial AGCD problem (RAGCD). As in the case of the VACGD problem, the RAGCD problem cannot be easier than the original AGCD problem, as the latter can be viewed as a particular case

of the RAGCD problem with $N = k(x) = 1$. Moreover, by cryptanalyzing the RAGCD problem, we see that we can also choose smaller parameters for it than for the AGCD problem. Putting all this together, we propose a homomorphic encryption scheme that works natively over polynomial rings and can efficiently perform homomorphic polynomial multiplications.

Finally, in Chapter 7, we investigate the new fast bootstrapping techniques available for HE scheme based on the (R)LWE problem. The current scenario was the following: bootstrapping RLWE-based schemes can be done in less than one second in usual commercial computers [DM15, CGGI16a, CGGI19], however, FHE over the integers (schemes based on the AGCD problem) required minutes to be bootstrapped. Thus, there is a gap between these two main types of FHE schemes. While in the decryption function of (R)LWE-based schemes one has to work modulo a public integer q, in the decryption of AGCD-based schemes one has to perform a reduction modulo a secret integer p. Moreover, p is exponentially large in the security parameter λ. Thus, the fast bootstrapping techniques of [DM15, CGGI16a] cannot be applied immediately to AGCD-based schemes. Hence, in Chapter 7 we propose fast bootstrapping methods for FHE over the integers. By evaluating the decryption function using our homomorphic scheme for polynomials proposed in Chapter 6, we can, for the first time, bootstrap an AGCD-based scheme in less than one second using a security level of 100 bits.

1.3 Academic production

In this section, the papers, presentations, and implementations made during my doctoral studies are presented in chronological order.

Publications

- [CP19a]: *On Kilian's Randomization of Multilinear Map Encodings.*
  - Author(s): Jean-Sébastien Coron and Hilder Vitor Lima Pereira.

  - Published at: The 25th Annual International Conference on the Theory and Application of Cryptology and Information Security (Asiacrypt 2019).
  - Brief discussion: In this article, we present the first functional lattice attack against the multi-prime AGCD problem; we define and analyze the multi-prime version of the VAGCD problem with a noiseless sample x_0; in the cryptanalysis, we generalize the attacks on the AGCD problem to this version of the VAGCD problem, concluding that they are more expensive than the original attacks. Finally, we propose our N-party One-round key exchange and present practical results. This article partially overlaps the chapters 3 and 4. Namely, in Chapter 3 we define and cryptanalyze several variants of the VAGCD problem, including the one defined in this article, and in Chapter 4, we present the same key-exchange protocol proposed in this article.

- [Per20b]: *Efficient AGCD-based homomorphic encryption for matrix and vector arithmetic.*
  - Author(s): Hilder Vitor Lima Pereira.
  - Published at: The 18th International Conference on Applied Cryptography and Network Security (ACNS 2020).
  - Brief discussion: In this article, we use the VAGCD problem to propose a leveled homomorphic encryption scheme that operates natively on vectors and matrices, then use this scheme in two applications: the homomorphic evaluation of NFA and of a Naïve Bayes Classifier (this same scheme and the two applications are presented in Chapter 5 of this thesis). Moreover, we present a more general GCD attack that applies to all variants of the (single prime) AGCD and VAGCD problem, and we also show some theoretical evidence to justify our cryptanalysis of the VAGCD problem. This part of the paper is presented in Chapter 3.

- [Per20a]: *Bootstrapping fully homomorphic encryption over the integers in less than one second.*

  - Author(s): Hilder Vitor Lima Pereira.
  - Published at: Not yet published. Available on IACR e-print archive (2020).
  - Brief discussion: In this article, we propose fast bootstrapping methods for AGCD-based FHE schemes. Firstly, we introduce the RAGCD problem, then we use it to construct an encryption scheme that operates homomorphically on the ring $R := \mathbb{Z}[x]/\langle x^N + 1\rangle$ (thus, this part of the article appears in Chapter 6). Then, we propose a functional key-switching transformation and a hidden modulus-switching, which enables us to perform a fast bootstrapping as in [DM15]. An AGCD-based scheme encrypting a single bit is then bootstrapped in less than one second. We also present a multi-bit bootstrapping procedure. Both bootstrapping methods are presented in Chapter 7 of this thesis.

Presentations

- ASIACRYPT 2019. Kobe, Japan. Presentation of [CP19a].
- ACNS 2020. Planned to take place in Rome, Italy, but done remotely due to the COVID-19 pandemic. Presentation of [Per20b].

Source code

- Implementation, in SAGE, of the attacks on the AGCD and on the VAGCD problem, and of our multipartite key-exchange protocol. Available in [CP19b].
- Implementation, in C++, of our leveled homomorphic encryption scheme for vectors and matrices, and some applications. Available in [Per20c].
- Implementation, in C++, of our RAGCD-based leveled homomorphic encryption scheme for polynomials, and the bootstrapping procedures. Available in [Per20e].

Chapter 2

Theoretical background

2.1 Notation

In this section we define the notation that will be used in this thesis. We use the notation with double brackets for integer intervals, e.g., an integer interval open on a is $[\![a, b[\![:= \mathbb{Z} \cap]a, b]$. For $n \in \mathbb{N}$, we denote by $\mathbb{Z}_n$ the ring of integers modulo n. Moreover, we use $a \bmod n$ or $[a]_n$ to denote the reduction modulo n. If $n_1, ..., n_\ell$ are coprime integers, then the direct product $\mathbb{Z}_{n_1} \times \cdots \times \mathbb{Z}_{n_\ell}$ is isomorphic to $\mathbb{Z}_n$, where $n := \prod_{i=1}^{\ell} n_i$, and the map isomorphism is $\mathsf{CRT}_{(n_1,...,n_\ell)}(a_1, ..., a_\ell)$, which is the unique $x \in [\![0, n-1]\!]$ that satisfy $x = a_i \pmod{n_i}$ for all $i \in [\![1, \ell]\!]$. If the number of factors, ℓ, is clear from the context, we simplify the notation by writing $\mathsf{CRT}_{(n_i)}(a_i)$.

Vectors will be denoted by bold lowercase letters, like $\mathbf{v}$. The infinity norm of an m-dimensional vector $\mathbf{v}$, defined as $\|\mathbf{v}\|_\infty := \max\{|v_1|, ..., |v_m|\}$, is denoted simply by $\|\mathbf{v}\|$. The other norms are written explicitly, e.g., the Euclidean norm is denoted by $\|\mathbf{v}\|_2$. The inner product, also known as dot product, is simply written as $\mathbf{u} \cdot \mathbf{v}$. Matrices are denoted by bold uppercase letters, e.g., $\mathbf{A} \in \mathbb{R}^{n \times m}$ is an $n \times m$ matrices with real entries and $\mathbf{A}^T$ is its transpose. A diagonal matrix whose diagonal elements are $a_1, ..., a_n$ is denoted by $\mathsf{diag}(a_1, ..., a_n)$. In particular, the $n \times n$ identity matrix, denoted by $\mathbf{I}_n$, is $\mathsf{diag}(1, ..., 1)$. The norm of a polynomial $a(x) = \sum_{i=0}^{N} a_i x^i$ is defined as the infinity norm of its coefficient vector, that is, $\|a(x)\| := \max\{|a_0|, ..., |a_N|\}$.

For matrices, we use the max-norm $\|\mathbf{A}\| := \|\mathbf{A}\|_{\max} = \max\{\|a_{i,j}\| : a_{i,j} \text{ is an entry of } \mathbf{A}\}$.

Notice that for matrices whose entries are real numbers, it holds that $\|\mathbf{A}\cdot\mathbf{B}\| \leq m\,\|\mathbf{A}\|\cdot\|\mathbf{B}\|$, where m is the number of columns of $\mathbf{A}$. If the entries of at least one of the matrices are polynomials with degree up to N, then it holds that $\|\mathbf{A}\cdot\mathbf{B}\| \leq mN\,\|\mathbf{A}\|\cdot\|\mathbf{B}\|$.

2.2 Linear Algebra: eigenvalues and tensor products

A linear space over a field $\mathbb{K}$ generated by $\mathcal{V} := \{\mathbf{v}_1, ..., \mathbf{v}_n\}$ is the set $V := \{\alpha_1\cdot\mathbf{v}_1 + ... + \alpha_n\cdot\mathbf{v}_n : \alpha_i \in \mathbb{K}\}$. If all the vectors of $\mathcal{V}$ are linearly independent, that is, it is not possible to write one $\mathbf{v}_i \in \mathcal{V}$ as a $\mathbb{K}$-linear combination of the other vectors in $\mathcal{V}$, then, $\mathcal{V}$ is a basis and $|\mathcal{V}|$ is the dimension of V.

For a square matrix $\mathbf{A} \in \mathbb{K}^{n\times n}$, a scalar $\lambda \in \mathbb{K}$ and a vector $\mathbf{v} \in \mathbb{K}^n$ satisfying the equation $\mathbf{A}\mathbf{v} = \lambda\mathbf{v}$ are called an eigenvalue and an eigenvector, respectively. The characteristic polynomial of $\mathbf{A}$ is defined as $\mathsf{charPoly}_{\mathbf{A}}(x) := \det(x\cdot\mathbf{I}_n - \mathbf{A}) \in \mathbb{K}[x]$ and the eigenvalues of $\mathbf{A}$ are actually the roots of $\mathsf{charPoly}_{\mathbf{A}}(x)$. It is clear that when $\mathbf{A}$ is a diagonal matrix, we have $c(x) = \prod_{i=1}^{n}(x - a_{i,i})$, thus, in this case, the eigenvalues of $\mathbf{A}$ are the diagonal elements. Moreover, if $\mathbf{A}$ is diagonal per blocks, that is, $\mathbf{A} = \mathsf{diag}(\mathbf{A}_1, ..., \mathbf{A}_n)$ where each $\mathbf{A}_i$ is a matrix, then, the characteristic polynomial of $\mathbf{A}$ decomposes as a product of the characteristic polynomial of its blocks, i.e., $\mathsf{charPoly}_{\mathbf{A}}(x) = \prod_{i=1}^{n}\mathsf{charPoly}_{\mathbf{A}_i}(x)$. Two $n\times n$ matrices $\mathbf{A}$ and $\mathbf{B}$ are called similar if there is an $n\times n$ invertible matrix $\mathbf{C}$ such that $\mathbf{A} = \mathbf{C}\mathbf{B}\mathbf{C}^{-1}$. It is easy to see that two similar matrices have the same characteristic polynomial (and thus, the same eigenvalues):

$$
\begin{aligned}
\mathsf{charPoly}_{\mathbf{A}}(x) &= \det(x\cdot\mathbf{I}_n - \mathbf{A}) && \text{(By definition)}\\
&= \det(x\cdot\mathbf{I}_n - \mathbf{C}\mathbf{B}\mathbf{C}^{-1}) && \text{(By similarity)}\\
&= \det(\mathbf{C}(x\cdot\mathbf{I}_n - \mathbf{B})\mathbf{C}^{-1}) && \text{(Since } \mathbf{I}_n = \mathbf{C}\mathbf{I}_n\mathbf{C}^{-1})\\
&= \det(\mathbf{C})\det(x\cdot\mathbf{I}_n - \mathbf{B})\det(\mathbf{C}^{-1}) && \text{(Since } \det(\mathbf{X}\mathbf{Y}) = \det(\mathbf{X})\det(\mathbf{Y}))\\
&= \mathsf{charPoly}_{\mathbf{B}}(x) && \text{(Because } \det(\mathbf{C})\det(\mathbf{C}^{-1}) = 1).
\end{aligned}
$$

Finally, the Cayley-Hamilton theorem guarantees that every square matrix $\mathbf{A}$ is a zero of its own characteristic polynomial, that is, $\mathsf{charPoly}_{\mathbf{A}}(\mathbf{A}) = \mathbf{0}$.

The (Kronecker) tensor product of two matrices $\mathbf{A} \in \mathbb{K}^{n\times m}$ and $\mathbf{B} \in \mathbb{K}^{p\times q}$, denoted by $\mathbf{A} \otimes \mathbf{B}$ is the $np \times mq$ matrix obtained by replacing each entry $a_{i,j}$ of $\mathbf{A}$ by the matrix $a_{i,j} \cdot \mathbf{B}$. For instance,

$$\mathbf{A} = \begin{pmatrix} -1 & 0 \\ 2 & 1 \end{pmatrix} \text{ and } \mathbf{B} = \begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{pmatrix} \implies \mathbf{A} \otimes \mathbf{B} = \begin{pmatrix} -1 & -2 & -3 & 0 & 0 & 0 \\ -4 & -5 & -6 & 0 & 0 & 0 \\ 2 & 4 & 6 & 1 & 2 & 3 \\ 8 & 10 & 12 & 4 & 5 & 6 \end{pmatrix}.$$

The tensor product is not commutative. If $\mathbf{A}$ and $\mathbf{B}$ are $n \times n$ and $m \times m$ matrices, respectively, then it is known that $\det(\mathbf{A} \otimes \mathbf{B}) = \det(\mathbf{A}^{-1})^m \det(\mathbf{B}^{-1})^n$. Moreover, $(\mathbf{A} \otimes \mathbf{B})^{-1} = \mathbf{A}^{-1} \otimes \mathbf{B}^{-1}$. We define $\mathsf{vec}(\mathbf{A}) \in \mathbb{K}^{nm}$ as the operator that transforms an $n \times m$ matrix into a vector by stacking the columns. For instance, considering the matrices defined in the last example, we have $\mathsf{vec}(\mathbf{A}) = (-1 \;\; 2 \;\; 0 \;\; 1)$ and $\mathsf{vec}(\mathbf{B}) = (1 \;\; 4 \;\; 2 \;\; 5 \;\; 3 \;\; 6)$. The following equality is known as the *vec trick*: $\mathsf{vec}(\mathbf{ABC}) = (\mathbf{C}^T \otimes \mathbf{A})\,\mathsf{vec}(\mathbf{B})$, where $\mathsf{vec}(\mathbf{B})$ is interpreted as a column vector.

2.3 Lattices

A lattice can be viewed as a discrete linear vector space, since it is also defined by taking linear combinations of basis vectors, however, only integer coefficients are used in the combinations. That is, the lattice generated by linearly independent vectors $\mathbf{b}_1, ..., \mathbf{b}_n \in \mathbb{R}^m$ is

$$\mathcal{L}(\mathbf{b}_1, ..., \mathbf{b}_n) := \{a_1 \cdot \mathbf{b}_1 + ... + a_n \cdot \mathbf{b}_n : a_i \in \mathbb{Z}\}.$$

We say that $\{\mathbf{b}_1, ..., \mathbf{b}_n\}$ is a basis of $\mathcal{L}$. The dimension and the rank of $\mathcal{L}$ are defined as $\mathsf{dim}(\mathcal{L}) := m$ and $\mathsf{rank}(\mathcal{L}) := n$. Moreover, we say that $\mathcal{L}$ is full rank if $\mathsf{dim}(\mathcal{L}) = \mathsf{rank}(\mathcal{L})$. It is common to represent the basis by a matrix $\mathbf{B} \in \mathbb{R}^{m\times n}$ defined by $\mathsf{col}_j(\mathbf{B}) = \mathbf{b}_j$, and the elements of $\mathcal{L}$ are then of the form $\mathbf{Bz}$ for $\mathbf{z} \in \mathbb{Z}^n$; in this case, we say that $\mathcal{L}$ is generated by the columns of $\mathbf{B}$.[1] The fundamental region defined by a basis $\mathbf{B} \in \mathbb{R}^{m\times n}$ is

[1] Alternatively, one can define $\mathbf{B}$ by $\mathsf{row}_i(\mathbf{B}) = \mathbf{b}_i$, write $\mathbf{v} \in \mathcal{L}$ as $\mathbf{v} = \mathbf{zB}$, and say that $\mathcal{L}$ is generated by the rows of $\mathbf{B}$. But in this thesis, we will use the column notation.

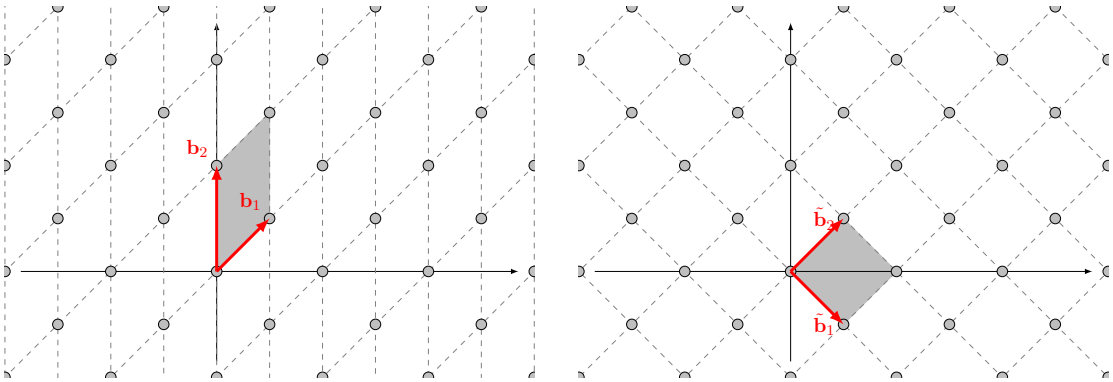

Figure 2.1: Two different bases of $\mathcal{L}$: $\mathbf{b}_1 := (1, 1)$ and $\mathbf{b}_2 := (0, 2)$, and $\tilde{\mathbf{b}}_1 := (1, -1)$ and $\tilde{\mathbf{b}}_2 := (1, 1)$; and their respective fundamental regions (the gray parallelepipeds). Notice that the points are the same.

the set $\mathcal{F}(\mathbf{B}) := \{\sum_{i=1}^{n} x_i \cdot \mathbf{b}_i : 0 \leq x_i < 1\}$. By placing one copy of $\mathcal{F}(\mathbf{B})$ at each point of the lattice, we cover the entire space defined by the span of $\mathbf{B}$ over the real numbers. Two $m \times n$ matrices $\mathbf{B}$ and $\mathbf{B}'$ are bases of the same lattice if, and only if, there is a unimodular matrix $\mathbf{U}$ such that $\mathbf{B} = \mathbf{B}' \cdot \mathbf{U}$. Remember that an $n \times n$ matrix $\mathbf{U}$ is unimodular if all its entries are integers and $\det(\mathbf{U}) \in \{-1, 1\}$.

For example, consider the following matrices:

$$\mathbf{B} := \begin{pmatrix} 1 & 0 \\ 1 & 2 \end{pmatrix}, \quad \tilde{\mathbf{B}} := \begin{pmatrix} 1 & 1 \\ -1 & 1 \end{pmatrix}, \quad \text{and} \quad \mathbf{U} := \begin{pmatrix} 1 & 1 \\ -1 & 0 \end{pmatrix}.$$

It is clear that $\mathbf{U}$ is unimodular, moreover, $\tilde{\mathbf{B}} = \mathbf{B} \cdot \mathbf{U}$, thus, by defining the lattice $\mathcal{L} := \mathcal{L}(\mathbf{B})$, i.e., the lattice with basis $\mathbf{B}$, we have that $\tilde{\mathbf{B}}$ is also a basis of $\mathcal{L}$. In Figure 2.1, we show a graphical representation of $\mathcal{L}$, the bases $\mathbf{B}$ and $\tilde{\mathbf{B}}$, and the fundamental regions $\mathcal{F}(\mathbf{B})$ and $\mathcal{F}(\tilde{\mathbf{B}})$. Notice how these regions tile $\mathbb{R}^2$ completely.

We define the determinant of a lattice $\mathcal{L}$ with basis $\mathbf{B}$ as $\det(\mathcal{L}) := \sqrt{\det(\mathbf{B}^T \cdot \mathbf{B})}$. If $\mathcal{L}$ is full rank, then $\mathbf{B}$ is square, and the expression is simplified to $\det(\mathcal{L}) := |\det(\mathbf{B})|$. Although the determinant is defined in terms of a basis, it is easy to see that its value is independent of the basis used to compute it. Namely, consider two basis $\mathbf{B}$ and $\tilde{\mathbf{B}}$ of $\mathcal{L}$. Then, $\tilde{\mathbf{B}} = \mathbf{B}\mathbf{U}$ for some unimodular $\mathbf{U}$. Thus,

$$\det(\tilde{\mathbf{B}}^T\tilde{\mathbf{B}}) = \det(\mathbf{U}^T\mathbf{B}^T\mathbf{B}\mathbf{U}) = \det(\mathbf{U}^T)\det(\mathbf{U})\det(\mathbf{B}^T\mathbf{B}) = (\det(\mathbf{U}))^2 \det(\mathbf{B}^T\mathbf{B})$$

but because $\det(\mathbf{U}) = \pm 1$, we have $\det(\tilde{\mathbf{B}}^T\tilde{\mathbf{B}}) = \det(\mathbf{B}^T\mathbf{B})$.

For any set of linearly independent vectors $\mathcal{V} := \{\mathbf{v}_1, ..., \mathbf{v}_n\} \subset \mathbb{Z}^m$, we define the lattice orthogonal to these vectors as

$$\mathcal{L}^\perp(\mathcal{V}) := \{\mathbf{u} \in \mathbb{Z}^m : \mathbf{u} \cdot \mathbf{v}_i = 0 \text{ for } 1 \leq i \leq n\}.$$

It is known that $\mathsf{rank}(\mathcal{L}^\perp(\mathcal{V})) = m - n$ [NS97]. For a lattice $\mathcal{L} \subset \mathbb{Z}^m$, let $\mathbf{b}_1, ..., \mathbf{b}_n$ be any basis of $\mathcal{L}$, then the lattice orthogonal to $\mathcal{L}$ is $\mathcal{L}^\perp := \mathcal{L}^\perp(\mathbf{b}_1, ..., \mathbf{b}_n)$. Notice that the vectors of $\mathcal{L}^\perp$ are orthogonal to all vectors of $\mathcal{L}$. Define $\bar{\mathcal{L}} := (\mathcal{L}^\perp)^\perp$, then it holds that $\bar{\mathcal{L}} = \mathbb{Z}^m \cap \mathsf{span}_{\mathbb{Q}}(\mathbf{b}_1, ..., \mathbf{b}_n)$ [NS97], that is, $\bar{\mathcal{L}}$ is composed by all the $\mathbb{Q}$-linear combinations of $\mathbf{b}_1, ..., \mathbf{b}_n$ that results in integral vectors. Furthermore, it is also known that $\det(\mathcal{L}^\perp) = \det(\bar{\mathcal{L}})$, and $\det(\mathcal{L})$ is expected to be close to $\det(\bar{\mathcal{L}})$, therefore, we have $\det(\mathcal{L}) \approx \det(\mathcal{L}^\perp)$. Additionally, by the Hadamard inequality, we expect that in general $\det(\mathcal{L}) \approx \prod_{i=1}^n \|\mathbf{b}_i\|_2$.

Another type of lattice that is often used in cryptography is the "q-ary lattice", which is defined as the orthogonal lattice, but with dot products computed in $\mathbb{Z}_q$, i.e., for an integer q and n linearly independent vectors $\mathbf{V} := [\mathbf{v}_1 \ ... \ \mathbf{v}_n] \in \mathbb{Z}^{m\times n}$, we define

$$\mathcal{L}_q^\perp(\mathbf{V}) := \{\mathbf{u} \in \mathbb{Z}^m : \mathbf{u} \cdot \mathbf{V} = \mathbf{0} \pmod q\}.$$

By writing $\mathbf{V}$ by blocks, as $\mathbf{V}^T := [\mathbf{V}_1^T \ \mathbf{V}_2^T]$, where $\mathbf{V}_1 \in \mathbb{Z}^{(m-n)\times n}$ and $\mathbf{V}_2 \in \mathbb{Z}^{n\times n}$, and assuming that $\mathbf{V}_2$ is invertible, we see that any vector $\mathbf{u} := [\mathbf{u}_1 \ \mathbf{u}_2] \in \mathcal{L}_q^\perp(\mathbf{V})$ satisfies

$$\begin{aligned}\mathbf{u} \in \mathcal{L}_q^\perp(\mathbf{V}) &\iff \mathbf{u_1}\mathbf{V_1} + \mathbf{u_2}\mathbf{V}_2 = \mathbf{0} \pmod q \\ &\iff \mathbf{u_1}\mathbf{V_1}{\mathbf{V}_2}^{-1} + \mathbf{u_2} = \mathbf{0} \pmod q\end{aligned}$$

therefore, the following matrix is a basis of $\mathcal{L}_q^\perp(\mathbf{V})$:

$$\mathbf{B} = \begin{pmatrix} \mathbf{I}_{m-n} & \mathbf{0} \\ -\mathbf{V}_2^{-T}\mathbf{V}_1^T & q\mathbf{I}_n \end{pmatrix} \in \mathbb{Z}^{m\times m}.$$

Notice that $\mathbf{u} := \mathbf{B}\mathbf{z} = [\mathbf{z}_1 \ \ -\mathbf{V}_2^{-T}\mathbf{V}_1^T\mathbf{z}_1 + q\mathbf{z}_2]$, thus, because $-\mathbf{V}_2^{-T}\mathbf{V}_1^T\mathbf{z}_1 = -\mathbf{z}_1\mathbf{V}_1\mathbf{V}_2^{-1}$, we have $\mathbf{u}\mathbf{V} = \mathbf{z}_1\mathbf{V}_1 - \mathbf{z}_1\mathbf{V}_1\mathbf{V}_2^{-1}\mathbf{V}_2 + q\mathbf{z}_2\mathbf{V}_2 = \mathbf{0} \pmod q$, as expected. From the basis $\mathbf{B}$, we can see that $\mathcal{L}_q^\perp(\mathbf{V})$ is full-rank of rank m and that $\det(\mathcal{L}_q^\perp(\mathbf{V})) = q^n$.

Since a lattice is discrete, the distance between two vectors cannot be arbitrarily small. Thus, we define the shortest distance of a lattice $\mathcal{L}$ as

$$\lambda_1(\mathcal{L}) := \min\left\{\|\mathbf{u}-\mathbf{v}\|_2 : \mathbf{u},\mathbf{v} \in \mathcal{L} \text{ and } \mathbf{u} \neq \mathbf{v}\right\}.$$

Because the difference of two vectors of $\mathcal{L}$ is again in $\mathcal{L}$, there exist $\mathbf{w} \in \mathcal{L}$ such that $\|\mathbf{w}\| = \lambda_1(\mathcal{L})$. Such $\mathbf{w}$ is called a *shortest non-zero vector* of $\mathcal{L}$. By the first Minkowski Theorem [MG02], we know that $\lambda_1(\mathcal{L}) \leq \sqrt{n}\det(\mathcal{L})^{1/n}$, where $n = \mathsf{rank}(\mathcal{L})$.

Additionally to $\lambda_1(\mathcal{L})$, we also define other minima distances in lattices. Namely, a shortest vector $\mathbf{w}$ defines a linear vector space of dimension one, then, if we consider all the lattice vectors that are not in this space, we have again shortest non-zero vectors and their norms are then $\lambda_2(\mathcal{L})$. We also define $\lambda_3(\mathcal{L})$ by considering the vectors that are not in the space of the first two shortest vectors, and so on. In general, we define $\lambda_i(\mathcal{L})$ to be the smallest value $r > 0$ such that $\mathcal{L}$ contains at least i linearly independent vectors whose (Euclidean) norm is bounded by r. We call $\lambda_i(\mathcal{L})$'s the successive minima of $\mathcal{L}$. It is clear that $\lambda_i(\mathcal{L}) \leq \lambda_{i+1}(\mathcal{L})$. By Minkowski's Second Theorem, we have $(\prod_{i=1}^{n}\lambda_i(\mathcal{L}))^{1/n} \leq \sqrt{n}\det(\mathcal{L})^{1/n}$, where $n = \mathsf{rank}(\mathcal{L})$.

Given a basis $\mathbf{B}$ of a lattice $\mathcal{L}$, finding a vector whose norm equals $\lambda_1(\mathcal{L})$ is known as the shortest non-zero vector problem (SVP) and it is regarded as a hard problem, since all known algorithms to solve it are exponential time in the rank of the lattice. Another important hard problem in lattices is the closest vector problem (CVP), which is defined as follows: given a basis $\mathbf{B} \in \mathbb{R}^{m\times n}$ and a vector $\mathbf{t} \in \mathbb{R}^m$, find $\mathbf{u} \in \mathcal{L}(\mathbf{B})$ such that $\|\mathbf{u}-\mathbf{t}\|_2 \leq \|\mathbf{v}-\mathbf{t}\|_2$ for all $\mathbf{v} \in \mathcal{L}(\mathbf{B})$. Algorithms to solve CVP, or approximate versions of it, typically performs better if the given basis is *reduced*, that is, the vectors therein are short or nearly orthogonal. The two main lattice-basis reduction algorithms are the LLL and the BKZ. They receive an arbitrary basis $\mathbf{B} \in \mathbb{R}^{m\times n}$ and output a basis $\tilde{\mathbf{B}}$ satisfying several properties, in particular,

$$\left\|\tilde{\mathbf{b}}_1\right\|_2 \leq \delta^n \cdot \det(\mathcal{L}(\mathbf{B}))^{1/n}.$$

The value δ is called the root-Hermite factor of the lattice-basis reduction algorithm.

LLL runs in polynomial time, but it outputs vectors that are larger than the shortest vector by an exponential factor. Namely, $\left\|\tilde{\mathbf{b}}_1\right\|_2 \leq 2^{(n-1)/2} \cdot \lambda_1(\mathcal{L}(\mathbf{B}))$ and $\left\|\tilde{\mathbf{b}}_1\right\|_2 \leq 2^{(n-1)/4} \cdot \det(\mathcal{L}(\mathbf{B}))^{1/n}$. Although the proved approximation factor of LLL is $2^{(n-1)/4} \approx 2^{n/4} \approx (1.19)^n$, it is known that LLL performs better in practice and achieves root-Hermite factor $\delta_{LLL} = 1.02$. BKZ has a parameter β, called block size, which is the dimension of the sublattices used during the basis reduction. Larger values of β produce shorter vectors, that is, the root-Hermite factor δ_β obtained is smaller, but the reduction takes more time. The running time of BKZ is exponential, and it is generally assumed that it takes time $2^{\Omega(\beta)}$ to achieve root-Hermite factor $\delta = \beta^{1/\beta}$ using block size β [CS15, HPS11]. The root-Hermite factor is sometimes approximated by $\delta = 2^{1/\beta}$ to ease the analyses [APS15].

Several cryptographic schemes can be attacked by finding short vectors in lattices. For instance, in some encryption schemes, one can use the public key to construct a lattice $\mathcal{L}$, then, run BKZ with block size β to recover a short vector $\mathbf{v}$ of norm approximately $\delta_\beta^n \cdot \det(\mathcal{L})^{1/n}$. Then, if $\mathbf{v}$ is short enough, that is, if $\delta_\beta^n \cdot \det(\mathcal{L})^{1/n} \leq B$, where B is a known bound, $\mathbf{v}$ reveals the private key. Thus, to guarantee a security level of λ bits, when instantiating the scheme, one has to choose the parameters such that the value δ_β required for the inequality to have a solution is very small, say, so small that one needs a block size $\beta \approx \lambda$, which means that an attacker will need time $2^{\Omega(\lambda)}$ to break the scheme by running BKZ.

2.4 Distributions

For a probability distribution χ over a set Ω, we denote by $x \leftarrow \chi$ a random element sampled from Ω following χ. The probability of an event $E \subset \Omega$ is denoted by $\chi(E)$. For any nonempty set X, we denote the uniform distribution on X by $\mathcal{U}(X)$. Because several cryptographic problems deal with distinguishing two probability distributions, it is useful to have some measure on how close the distributions are and to relate this measure with the chances that an attacker can distinguish them. Thus, we define the *statistical distance* for finite sets (also known as total variation distance) as follows:

Definition 2.4.1 (Statistical Distance)**.** *Let u and v be two probability distributions on Ω. We define the statistical distance between u and v as the maximum of the difference over all possible events, that is,*

$$\Delta(u, v) := \max_{E \subset \Omega} |u(E) - v(E)|.$$

The following two lemmas will be used to provide equivalent definitions of the statistical distance:

Lemma 2.4.2. *Let u and v be two probability distributions on Ω, and $M := \{x \in \Omega : u(x) \geq v(x)\}$. Then, for all event $E \subset \Omega$ such that $0 \leq u(E) - v(E)$, we have*

$$|u(E) - v(E)| \leq |u(M) - v(M)|.$$

Proof. By definition of M, we know that $u(M) - v(M) \geq 0$, then, because $u(E) - v(E) = -(u(\bar{E}) - v(\bar{E}))$, we have $u(\bar{M}) - v(\bar{M}) \leq 0$ and consequently, $u(E \cap \bar{M}) - v(E \cap \bar{M}) \leq 0$. Thus,

$$\begin{aligned} u(E) - v(E) &= u(E \cap M) + u(E \cap \bar{M}) - (v(E \cap M) + v(E \cap \bar{M})) \\ &= u(E \cap M) - v(E \cap M) + (u(E \cap \bar{M}) - v(E \cap \bar{M})) \\ &\leq u(E \cap M) - v(E \cap M) \\ &\leq u(M) - v(M) = |u(M) - v(M)|. \end{aligned}$$

□

Lemma 2.4.3. *Let u and v be two probability distributions on Ω, and $M = \{x \in \Omega : u(x) \geq v(x)\}$. Then, for all event $E \subset \Omega$ such that $0 > u(E) - v(E)$, we have*

$$|u(E) - v(E)| \leq |u(M) - v(M)|.$$

Proof. If $0 > u(E) - v(E)$, then, because $u(E) - v(E) = -(u(\bar{E}) - v(\bar{E}))$, we know that $0 < u(\bar{E}) - v(\bar{E})$, then, by Lemma 2.4.2, we have

$$|u(\bar{E}) - v(\bar{E})| \leq |u(M) - v(M)|.$$

Since $|u(E) - v(E)| = |u(\bar{E}) - v(\bar{E})|$, it holds that $|u(\bar{E}) - v(\bar{E})| = |u(E) - v(E)|$. □

Since the number of subsets of a Ω is exponential in the cardinality of Ω, the following characterizations of the statistical distance can be more useful for computational purposes.

Theorem 2.4.4 (Alternative characterization of statistical distance)**.** *Let u and v be two probability distributions on Ω, and $M := \{x \in \Omega : u(x) \geq v(x)\}$. Then, the statistical distance between u and v is given by*

$$\Delta(u, v) = u(M) - v(M).$$

Proof. Using lemmas 2.4.2 and 2.4.3, we see that for all $E \subset \Omega$, we have $|u(E) - v(E)| \leq |u(M) - v(M)|$. Since $M \subset \Omega$, we have

$$\Delta(u, v) = \max_{E \subset \Omega} |u(E) - v(E)| = |u(M) - v(M)|.$$

□

Theorem 2.4.5 (Statistical distance as a sum of differences)**.** *The statistical distance between two probability distributions u and v on Ω is given by*

$$\Delta(u, v) := \frac{1}{2} \sum_{x \in \Omega} |u(x) - v(x)|.$$

Proof. As before, define $M = \{x \in \Omega : u(x) \geq v(x)\}$. We see that

$$\begin{aligned}
\sum_{x \in \Omega} |u(x) - v(x)| &= \sum_{x \in M} |u(x) - v(x)| + \sum_{x \in \bar{M}} |u(x) - v(x)| \\
&= \sum_{x \in M} u(x) - v(x) + \sum_{x \in \bar{M}} -(u(x) - v(x)) \\
&= u(M) - v(M) - (u(\bar{M}) - v(\bar{M})) && \text{(By def. of } M\text{)} \\
&= u(M) - v(M) + u(M) - v(M) \\
&= 2(u(M) - v(M)) \\
&= 2\Delta(u, v) && \text{(By Theorem 2.4.4)}
\end{aligned}$$

□

We now state basic properties of the statistical distance and prove that the advantage of any adversary in distinguishing two distributions u and v is bounded $\Delta(u, v)$.

Lemma 2.4.6 (Triangular inequality)**.** *Let u, v and w be three probability distributions on Ω, then*

$$\Delta(u, w) \leq \Delta(u, v) + \Delta(v, w).$$

Proof. We know that for all x in Ω, $|u(x) - w(x)| = |u(x) - v(x) + v(x) - w(x)| \leq |u(x) - v(x)| + |v(x) - w(x)|$, therefore, by Theorem 2.4.5, we have

$$\Delta(u, w) \leq \frac{1}{2} \sum_{x \in \Omega} |u(x) - v(x)| + \frac{1}{2} \sum_{x \in \Omega} |v(x) - w(x)| = \Delta(u, v) + \Delta(v, w).$$

□

Lemma 2.4.7 (A third independent distribution does not change the statistical distance)**.** *Let u, v and w be three probability distributions on Ω with w being independent of u and v. Also, define $\alpha := (u, w)$ and $\beta := (v, w)$ to be distributions on $\Omega \times \Omega$. Then,*

$$\Delta(\alpha, \beta) = \Delta(u, v).$$

Proof. Because of the independence property, we have $\alpha(x, y) = u(x)w(y)$ and $\beta(x, y) = v(x)w(y)$. Thus,

$$\begin{aligned}
\Delta(\alpha, \beta) &= \frac{1}{2} \sum_{(x,y) \in \Omega^2} |\alpha(x, y) - \beta(x, y)| \\
&= \frac{1}{2} \sum_{(x,y) \in \Omega^2} |u(x)w(y) - v(x)w(y)| \\
&= \frac{1}{2} \sum_{(x,y) \in \Omega^2} |u(x) - v(x)|w(y) \\
&= \frac{1}{2} \left(\sum_{x \in \Omega} |u(x) - v(x)| \right) \cdot \sum_{y \in \Omega} w(y) \\
&= \frac{1}{2} \left(\sum_{x \in \Omega} |u(x) - v(x)| \right) \cdot 1 \\
&= \Delta(u, v)
\end{aligned}$$

□

Lemma 2.4.8. *Let u and v be two probability distributions on Ω. Then, for any function $f : \Omega \to \{0,1\}$, if we sample x and x' following u and v, respectively, we have*

$$|\Pr[f(x) = 1] - \Pr[f(x') = 1]| \leq \Delta(u, v).$$

Proof. Define the event $E := \{x \in \Omega : f(x) = 1\}$. We know that

$$\begin{aligned}
\Pr[f(x) = 1] &= \Pr[f(x) = 1 \mid x \in E] \Pr[x \in E] + \Pr[f(x) = 1 \mid x \notin E] \Pr[x \notin E] \\
&= \Pr[f(x) = 1 \mid x \in E] \Pr[x \in E] + 0 \\
&= 1 \cdot \Pr[x \in E] \\
&= u(E).
\end{aligned}$$

Analogously, we can see that $\Pr[f(x') = 1] = v(E)$. Thus, by definition of statistical distance, we have

$$|\Pr[f(x) = 1] - \Pr[f(x') = 1]| = |u(E) - v(E)| \leq \Delta(u, v).$$

□

Corollary 2.4.9. *The advantage of any (possibly probabilistic) adversary $\mathcal{A}$ in distinguishing two distributions u and v is upper bounded by their statistical distance (regardless the running time of $\mathcal{A}$), that is,*

$$\forall \mathcal{A},\ \mathrm{Adv}(\mathcal{A}) := \left| \Pr_{x \leftarrow u}[\mathcal{A}(x) = 1] - \Pr_{x' \leftarrow v}[\mathcal{A}(x') = 1] \right| \leq \Delta(u, v).$$

Proof. If $\mathcal{A}$ is deterministic, then the result follows directly from Lemma 2.4.8.

Otherwise, if $\mathcal{A}$ is probabilistic, then it is equivalent to a deterministic adversary $\mathcal{B}$ that receives the random coins w used by $\mathcal{A}$. In this case, $\mathcal{B}$ tries to distinguish (u, w) from (v, w), so we have

$$\begin{aligned}
\mathrm{Adv}(\mathcal{A}) &= \mathrm{Adv}(\mathcal{B}) \\
&\leq \Delta((u, w), (v, w)) && \text{(By Lemma 2.4.8)} \\
&= \Delta(u, v) && \text{(By Lemma 2.4.7)}
\end{aligned}$$

□

Proving that a scheme is secure often reduces to proving that no PPT adversary can distinguish two distributions, e.g., the distribution of ciphertexts encrypting a zero and that of encryptions of one, hence, Corollary 2.4.9 is used as follows: we first prove that the statistical distance of the two distributions is smaller than $2^{-\lambda}$, therefore, negligible in λ, thus, Corollary 2.4.9 implies that the advantage of the adversary is also negligible.

Finally, we state a simplified version of the well-known leftover hash lemma (LHL) that can be used to compute an upper bound to the statistical distance [BBL17].

Definition 2.4.10 (2-universal family of hash functions)**.** *A set $\mathcal{H} := \{h : X \to Y\}$ of functions from a finite set X to a finite set Y is a 2-universal family of hash functions if $\forall x, x' \in X, x \neq x' \Rightarrow \Pr_{h \leftarrow \mathcal{H}}[h(x) = h(x')] = \frac{1}{|Y|}$.*

Lemma 2.4.11 (Matrix product as a 2-universal hash)**.** *Let $n, m, N, p \in \mathbb{N}$ with p being prime. Define $X := \{0, ..., N-1\}^n$ and $Y := \mathbb{Z}_p^m$. For any matrix $\mathbf{B}$, let $h_{\mathbf{B}}(\mathbf{x}) = \mathbf{xB} \pmod p$. Then, the set $\mathcal{H} := \{h_{\mathbf{B}} : \mathbf{B} \in \mathbb{Z}_p^{n \times m}\}$ is a 2-universal family of hash functions from X to Y.*

Lemma 2.4.12 (LHL)**.** *Let $\mathcal{H}$ be a 2-universal family of hash functions from X to Y. Suppose that $h \leftarrow \mathcal{U}(\mathcal{H})$ and $x \leftarrow \mathcal{U}(X)$ independently. Then, the statistical distance between $(h, h(x))$ and the uniform $\mathcal{U}(\mathcal{H} \times Y)$ is at most $\frac{1}{2}\sqrt{\frac{|Y|}{|X|}}$.*

2.5 Representing polynomial rings with vectors and matrices

In this section, we consider "modular polynomial rings" of the form $R := \mathbb{Z}[x]/\langle f \rangle$ where f is a degree-n integral polynomial, and discuss how R can be represented with vectors and matrices. This representation is used in chapter 6 and 7. Firstly, notice that every element $g + \langle f \rangle$ of R has the unique representation $g \mod f$, which is a polynomial of degree smaller than n. Therefore, we can represent the elements of R as $g = \sum_{i=0}^{n-1} g_i x^i$. This allows us to define the coefficient vector of any $g \in R$ as $\boldsymbol{\phi}(g) = (g_0, g_1, ..., g_{n-1})$. Moreover, the norm of a polynomial is then defined as the norm of its coefficient vector,

i.e., $\|g\| := \|\boldsymbol{\phi}(g)\|$. It is easy to see that $\boldsymbol{\phi}$ is an isomorphism between R, seen as an additive group, and the group $(\mathbb{Z}^n, +)$, where the addition in $\mathbb{Z}^n$ is done componentwise. However, if we use only the map $\boldsymbol{\phi}$, then we lose the multiplicative property of R. Thus, we want another map $\boldsymbol{\Phi}$ such that $\boldsymbol{\Phi}(g)$ is an $n \times n$ matrix and, for all $h \in R$, it holds that $\boldsymbol{\phi}(h)\boldsymbol{\Phi}(g) = \boldsymbol{\phi}(h \cdot g)$, where the product $h \cdot g$ is done modulo f.

But notice that if this equality holds for all h, then, we have the particular case $\boldsymbol{\phi}(x^i \cdot g) = \boldsymbol{\phi}(x^i)\boldsymbol{\Phi}(g)$, and since $\boldsymbol{\phi}(x^i) = \mathbf{e}_i \in \{0,1\}^n$, i.e., the elementary vector with one in the i-th coordinate and zeros elsewhere, we have $\boldsymbol{\phi}(x^i \cdot g) = \mathbf{e}_i\boldsymbol{\Phi}(g)$. Finally, because multiplying $\mathbf{e}_i$ by any matrix $\mathbf{M}$ results in the i-th row of $\mathbf{M}$, we have $\boldsymbol{\phi}(x^i \cdot g) = \mathsf{row}_i(\boldsymbol{\Phi}(g))$, that is, the i-th row of $\boldsymbol{\Phi}(g)$ is equal to $\boldsymbol{\phi}(x^i \cdot g)$.

Therefore, the desired map can be defined as follows:

Definition 2.5.1. *For all $g \in R$, we define $\boldsymbol{\Phi}(g)$ as the $n \times n$ matrix such that the i-th row is equal to $\boldsymbol{\phi}(x^i g)$, that is, for $0 \le i \le n-1$, $\mathsf{row}_i(g) = \boldsymbol{\phi}(x^i g)$. Expressly,*

$$\boldsymbol{\Phi}(g) = \begin{pmatrix} - & \boldsymbol{\phi}(g) & - \\ - & \boldsymbol{\phi}(x \cdot g) & - \\ & \cdots & \\ - & \boldsymbol{\phi}(x^{n-1} \cdot g) & - \end{pmatrix} \in \mathbb{Z}^{n \times n}.$$

Some special cases that are often used in cryptographic works are the following:

- The "NTRU ring" $R = \mathbb{Z}[x]/\langle x^N - 1\rangle$. In this case, $\boldsymbol{\phi}(x^i g)$ is just a cyclic rotation of $\boldsymbol{\phi}(g)$ and the matrix $\boldsymbol{\Phi}(g)$, which is then called *a circulant matrix*, is simply

$$\boldsymbol{\Phi}(g) = \begin{pmatrix} g_0 & g_1 & \cdots & g_{N-1} \\ g_{N-1} & g_0 & \cdots & g_{N-2} \\ \vdots & \vdots & \ddots & \vdots \\ g_1 & g_2 & \cdots & g_0 \end{pmatrix} \in \mathbb{Z}^{N \times N}.$$

- The "anticirculant ring" $R = \mathbb{Z}[x]/\langle x^N + 1\rangle$. (which is a cyclotomic ring when N is a power of two). In this case, because $x^N = -1$ in R, $\boldsymbol{\phi}(x^i g)$ is also a cyclic rotation of $\boldsymbol{\phi}(g)$, but the rotated entries are multiplied by -1. Thus, $\boldsymbol{\Phi}(g)$ is called

the *anticirculant matrix* of g and has to following format:

$$\mathbf{\Phi}(g) = \begin{pmatrix} g_0 & g_1 & g_2 & \cdots & g_{N-1} \\ -g_{N-1} & g_0 & g_1 & \cdots & g_{N-2} \\ -g_{N-2} & -g_{N-1} & g_0 & \cdots & g_{N-3} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ -g_1 & -g_2 & -g_3 & \cdots & g_0 \end{pmatrix} \in \mathbb{Z}^{n \times n}.$$

Of course, for other rings $R = \mathbb{Z}[x]/\langle f \rangle$ with a polynomial f less sparse then the two last cases, the vector $\boldsymbol{\phi}(x^i g)$, and consequently the matrix $\mathbf{\Phi}(g)$, can be very complicated.

We now prove some basic properties about $\boldsymbol{\phi}$ and $\mathbf{\Phi}$.

Lemma 2.5.2. *For all $g, h \in R$, it holds that $\boldsymbol{\phi}(g)\mathbf{\Phi}(h) = \boldsymbol{\phi}(gh)$.*

Proof. Because $\boldsymbol{\phi}$ is additive, we have $\boldsymbol{\phi}(g) = \sum_{i=0}^{n-1} \boldsymbol{\phi}(g_i x^i) = \sum_{i=0}^{n-1} g_i \boldsymbol{\phi}(x^i)$.

Hence

$$\boldsymbol{\phi}(g)\mathbf{\Phi}(h) = \sum_{i=0}^{n-1} g_i \boldsymbol{\phi}(x^i)\mathbf{\Phi}(h) = \sum_{i=0}^{n-1} g_i \boldsymbol{\phi}(x^i h) = \sum_{i=0}^{n-1} \boldsymbol{\phi}(g_i x^i h) = \boldsymbol{\phi}\left(\sum_{i=0}^{n-1} g_i x^i h\right) = \boldsymbol{\phi}(gh)$$

□

Lemma 2.5.3 ($\mathbf{\Phi}$ is additive)**.** $\forall g, h \in R$, $\mathbf{\Phi}(g + h) = \mathbf{\Phi}(g) + \mathbf{\Phi}(h)$

Proof. For each row $0 \leq i \leq n - 1$, we have

$$\begin{aligned} \mathsf{row}_i(\mathbf{\Phi}(g) + \mathbf{\Phi}(h)) &= \mathsf{row}_i(\mathbf{\Phi}(g)) + \mathsf{row}_i(\mathbf{\Phi}(h)) && \text{(matrix addition)} \\ &= \boldsymbol{\phi}(x^i g) + \boldsymbol{\phi}(x^i h) && \text{(by definition of } \mathbf{\Phi}) \\ &= \boldsymbol{\phi}(x^i g + x^i h) && \text{(by additivity of } \boldsymbol{\phi}) \\ &= \mathsf{row}_i(\mathbf{\Phi}(g + h)) && \text{(by definition of } \mathbf{\Phi}(g + h)) \end{aligned}$$

□

Lemma 2.5.4 ($\mathbf{\Phi}$ is multiplicative)**.** $\forall g, h \in R$, $\mathbf{\Phi}(gh) = \mathbf{\Phi}(g)\mathbf{\Phi}(h)$

Proof. For $0 \leq i \leq n-1$, we have

$$\begin{aligned}
\mathsf{row}_i(\mathbf{\Phi}(g) \cdot \mathbf{\Phi}(h)) &= \mathsf{row}_i(\mathbf{\Phi}(g)) \cdot \mathbf{\Phi}(h) && \text{(property of matrix multiplication)} \\
&= \phi(x^i \cdot g) \cdot \mathbf{\Phi}(h) && \text{(by definition of } \mathbf{\Phi}) \\
&= \phi(x^i \cdot g \cdot h) && \text{(by Lemma 2.5.2)} \\
&= \mathsf{row}_i(\mathbf{\Phi}(g \cdot h)) && \text{(by definition of } \mathbf{\Phi})
\end{aligned}$$

Thus, each row of $\mathbf{\Phi}(g)\mathbf{\Phi}(h)$ equals the corresponding row of $\mathbf{\Phi}(gh)$, i.e., $\mathbf{\Phi}(gh) = \mathbf{\Phi}(g)\mathbf{\Phi}(h)$. □

2.6 Fully homomorphic encryption

With homomorphic encryption schemes, it is possible to perform computation over encrypted data as if one had access to the data in clear. That is, consider some family of functions $\mathcal{F}_\lambda$, then a homomorphic encryption scheme for $\mathcal{F}_\lambda$ is a tuple $\mathcal{E} = (\mathsf{HE.KeyGen}, \mathsf{HE.Enc}, \mathsf{HE.Dec}, \mathsf{HE.Eval})$, where the key generation, encryption, and decryption are defined as usual, and $\mathsf{HE.Eval}$ is a procedure that receives a function $f \in \mathcal{F}_\lambda$ on t variables, t ciphertexts c_i's encrypting messages m_i's, and possibly the public key, and outputs a ciphertext $c = \mathsf{HE.Eval}(c_1, ..., c_t, f)$ encrypting $f(m_1, ..., m_t)$. Notice that $\mathsf{HE.Eval}$ does not use the secret key. For example, the Paillier cryptosystem [Pai99] encrypts a message $m_i \in \mathbb{Z}_n$ into a ciphertext $c_i := g^{m_i} \cdot r_i^n \mod n^2$, thus, multiplying two ciphertexts yields $c := c_1 \cdot c_2 = g^{m_1+m_2} \cdot (r_1 r_2)^n \mod n^2$, which is an encryption of $m_1 + m_2 \mod n$. Thus, we say that Paillier is an additive homomorphic encryption scheme, or that it is homomorphic for the family of functions $f_t(x_1, ..., x_t) := \sum_{i=1}^{t} x_i \mod n$ for all $t \in \mathbb{N}^*$. If there is a set of parameters for which the scheme is homomorphic for all circuits, then, we have a fully homomorphic encryption scheme (FHE). If for each set of parameters, there is an integer L such that we can homomorphically evaluate all circuits of depth bounded by L, then we have a leveled FHE. Notice that a leveled FHE scheme can be used to evaluate any circuit, but the evaluation must be planned in advance, that is, one has to decide beforehand a maximum depth for the circuits that will be evaluated,

then generate the parameters that support such depth.

Applications of FHE

FHE schemes can be used neatly to solve a long list of practical and theoretical problems. The most trivial application of FHE is outsourcing computation to untrusted third parties. For instance, nowadays a data holder can encrypt their data to securely store it in the cloud, so that the service provider do not learn the data that is being stored. However, when any computation is needed, the data must be downloaded, decrypted, and only then processed by the data holder. If FHE is used, then the cloud itself can process the data and the data holder only needs to download and then decrypt the output of the computation. Besides of being practical, this also allows the data holder to use services that are offered by the cloud, e.g., a trained neural network that could be used as a classifier. It is likely that clients using the cloud services do not have enough resources (e.g., data, expertise, computational power) to train complex machine learning models. Another applications include reducing proof size in Non-interactive zero-knowledge proofs [GGI+15], constructing e-voting systems [CGGI16b], creating verifiable oblivious transfer protocols [Lip03], amplifying obfuscation schemes to create obfuscators for all circuits from obfuscators for depth-bounded circuits [GGH+13b], among many others.

Construction of FHE

All the best-known FHE schemes (e.g., [Gen09, DGHV10, BGV12, GSW13]) perform encryption following the same strategy: the secret key sk is used to compute a key-dependent value, for instance, $a_i \cdot \mathsf{sk}$, which is then added to the message m_i along with a random noise term r_i, producing thus a ciphertext like $c_i = a_i \cdot \mathsf{sk} + r_i + m_i$. To decrypt, we use the secret key to remove the key-dependent term, then, apply some simple error-correction technique to remove r_i and recover m. However, the decryption only works if the noise is small, say, if $|r_i| < B$, for a known bound B. Then, the homomorphic

addition is typically performed simply by adding two ciphertexts, obtaining

$$c_{add} := c_1 + c_2 = \underbrace{(a_1 + a_2)}_{a_{add}} \cdot \mathsf{sk} + \underbrace{r_1 + r_2}_{r_{add}} + m_1 + m_2.$$

Notice that the new noise term is potentially bigger than the noises r_1 and r_2. The homomorphic product varies greatly from one FHE scheme to another and, in general, more complex procedures increase less the noise of the ciphertext when compared to simpler procedures. In some schemes, a homomorphic multiplication is performed via a tensor product and then the application of a dimension-reduction function using a especial public key, called evaluation key. In other schemes, one of the operands is decomposed in small values before being multiplied. Considering that the decryption does not work if the noise surpasses a predefined bound and that each homomorphic operation increases the noise, it is clear that this type of scheme can only evaluate circuits of bounded depth. To transform a "bounded" homomorphic encryption scheme $\mathcal{E}$ into a fully homomorphic one, the original idea proposed by Gentry [Gen09] was using $\mathcal{E}$ to execute its own decryption function homomorphically. Given an encryption c of m with a large noise r, and the secret key sk, decryption is a function D on two variables, c and sk, which removes r and outputs m, thus, to evaluate it homomorphically, we need encryptions of the inputs, that is, we encrypt c and sk obtaining $\tilde{c}$ and $\tilde{\mathsf{sk}}$, then, we compute a new ciphertext $c' = \mathsf{HE.Eval}(c, \tilde{\mathsf{sk}}, D)$ encrypting $m = \mathsf{HE.Dec}(c, \mathsf{sk})$. This process is illustrated in Figure 2.6. Moreover, the noise of c' is independent of r, depending only on the noise accumulated by the evaluation of the function $\mathsf{HE.Dec}$. Then, by construction, this evaluation accumulates a noise that is smaller than r, therefore, we obtain a ciphertext encrypting the same message, but with less noise. This technique is called *bootstrapping*. Notice that it allows the evaluation of any circuit, since we can proceed performing a sequence of homomorphic operations and bootstrapping to reduce the noise, until we performed all the operations present in the circuit.

However, bootstrapping is not simple, it is generally computationally expensive, and also requires additional security hypothesis, as supposing that the scheme remains secure even when it encrypts its own secret key (circular security hypothesis). Thus, since the

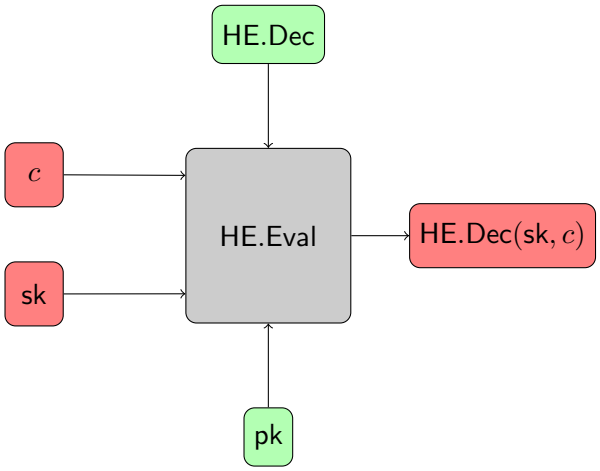

Figure 2.2: Red boxes represent encrypted data while green boxes represent known values. We obtain a new encryption of $m = \mathsf{HE.Dec}(\mathsf{sk}, c)$.

first FHE scheme was proposed [Gen09], most of the work done in homomorphic encryption was devoted to reducing the noise growth due to the homomorphic operations, which decreases the amount of bootstrappings that must be done when a circuit is evaluated; and to finding more efficient bootstrapping procedures.

History of FHE

Regarding the noise growth during the homomorphic operations, the various HE schemes proposed to date can be classified in the three following groups [Hal17]:

First generation: It begins with the first fully homomorphic encryption scheme, which Gentry proposed in 2009 [Gen09]. The main common characteristic of schemes from this generation is that the noise growth is exponential in the multiplicative depth of the function that is evaluated. Roughly speaking, if a ciphertext output by the encryption function has a noise term of size B, then, after a sequence of L products, the noise becomes B^L. Since all the parameters, including the initial noise magnitude B are, in general, at least linear in λ, that noise growth means that, without using the bootstrapping, we cannot evaluate in polynomial time (in λ) functions that have multiplicative depth bigger than logarithmic.

Second generation: Those schemes introduced new techniques, such as re-linearization (or key switching) and modulus reduction [BV11], to reduce the noise growth during homomorphic operations, along with other improvements, like *batching* encryption [BGV12, GHS12]. Using all those tools, the noise growth was reduced to polynomial in the multiplicative depth (simply stated, from B^L in the first generation to L^c for some constant c). Also, it became possible to perform parallel operations on the plaintext with a single homomorphic operation.

Third generation: In 2013, Gentry, Sahai, and Waters introduced a HE scheme that uses a decomposition technique to reduce the noise growth [GSW13]. Basically, each ciphertext has the format $c_i = a_i + r_i$, where r_i is the noise, and the homomorphic multiplication is performed by first applying some kind of decomposition G^{-1} to one of the ciphertexts, i.e., $c_{mult} := G^{-1}(c_i)c_j = G^{-1}(c_i)(a_j + r_j) = a' + G^{-1}(c_i)r_j$. But $G^{-1}(c_i)$ is guaranteed to be very small, thus, the new noise term is just slightly bigger then the noises in c_i and in c_j. Moreover, the noise in c_i has little impact on the noise of c_{mult} which means that we can choose to apply G^{-1} to the ciphertext that has a bigger noise in order to minimize the new noise.

We call the schemes using that decomposition technique, GSW-like schemes. Examples of such schemes are the FHEW [DM15] and the TFHE [CGGI19], which exhibits very fast bootstrapping procedures. Some works proposed ways to adapt that decomposition technique to schemes from the first and second generation. For example, in [BBL17], the authors proposed a GSW-like scheme based on the HE scheme over the integers from first generation [DGHV10].

Chapter 3

Randomized vector AGCD problem

3.1 Motivation

We commence this chapter by defining the Approximate-Greatest Common Divisor (AGCD) problem and some variants that have already been used in several cryptographic schemes, then we review the main attacks against these problems. In particular, we notice that the attack on the multi-prime variant described in [CLT13] does not work in general, thus, our first contribution is to describe the first known lattice attack on the multi-prime AGCD problem. Then, we define and cryptanalyze a new variant of the AGCD problem, which we call randomized vector AGCD problem (VAGCD).

Basically, for a fixed η-bit p, we define an AGCD sample as a γ-bit integer of the form $c_i = pq_i + r_i$, where r_i is a ρ-bit integer called the noise. Then, in the AGCD problem, one has to recover p given many AGCD samples c_i's. Because of its simple definition, involving only products and additions on $\mathbb{Z}$, the AGCD problem is seen as a good underlying problem to construct complex schemes in a simpler way. For instance, in [DGHV10], the authors propose an FHE scheme based on the AGCD problem and they write that their "main motivation is conceptual simplicity". Of course, a simple construction is desirable not only because it is elegant or because we expend less time to understand it, there are also practical aspects, like the implementation of such scheme. For example, it is more likely that one can find efficient and high optimized libraries to sample ρ-bit integers in constant time than to find libraries for constant-time discrete

Gaussian sampling, as it is usually needed in LWE-based schemes.

However, despite its simplicity, the AGCD problem usually requires large parameters to be secure. Existing attacks can be divided in two types:

- GCD attacks: the basic strategy consists in trying to remove the noise terms r_i's from the AGCD samples to obtain multiples of p, then compute GCDs of these multiples to recover p. This type of attack usually requires around 2^ρ integer operations, therefore, to guarantee security of λ bits, we must use $\eta > \rho \approx \lambda$.

- Lattice attacks: roughly speaking, this type of attack can be described as attacks that try to find short integer vectors $\mathbf{v}_i$'s orthogonal to a vector $\mathbf{r} = (r_1, ..., r_t)$ of noise terms, then to use the equations $0 = \mathbf{v}_i \cdot \mathbf{r} = \sum_{j=1}^{t} v_{i,j} \cdot r_j$ to recover the terms r_j's and finally p. The cost of this type of attack is essentially $2^{\Omega(\gamma/(\eta-\rho)^2)}$, thus, we must use $\gamma \approx \lambda \cdot (\eta - \rho)^2$.

Moreover, we typically have to choose η much bigger than ρ to guarantee the correctness of the scheme. To illustrate it, consider FHE schemes: when we perform homomorphic operations, we generate new ciphertexts with larger noise, and we must guarantee that the noise of a ciphertext is always smaller than p, otherwise, we cannot decrypt it, thus, if the final noise is, for example, $2^{10\rho}$, then, we have to use $\eta > 10\rho \approx 10\lambda$. Hence, we could set $\rho = \lambda$, $\eta = 11 \cdot \lambda$, and $\gamma = \lambda(\eta - \rho)^2 = 10^2 \cdot \lambda^3$. Therefore, for a security level of 100 bits, the AGCD samples, and thus, the ciphertexts, have approximately $\gamma = 10^8$ bits. For comparison, the RSA cryptosystem, which is usually regarded as a primitive that needs long keys, offers more than 100 bits of security if we use the modulus n bigger than 2^{2048}, which means that the bit length of the ciphertexts is $2048 \approx 2 \cdot 10^3$.

Hence, finding more "efficient" variants of the AGCD problem is an interesting question with practical effects. In view of this, as our second contribution in this chapter, we propose a randomized version of the AGCD problem, called VAGCD, which can be defined as follows: In addition to the integer p, we have a secret random matrix $\mathbf{K} \in \mathbb{Z}^{m \times m}$. Then, a VAGCD sample is a vector $\tilde{\mathbf{c}} \in \mathbb{Z}^m$ of the form $\mathbf{c} \cdot \mathbf{K}$, where each entry of $\mathbf{c}$ is an AGCD sample, i.e., $c_i = pq_i + r_i$. The VAGCD problem is then the problem of finding p

given many VAGCD samples $\tilde{\mathbf{c}}_i$'s. We extend the attacks on the AGCD problem to the VAGCD problem and conclude that their costs increase: the GCD attacks cost $2^{\Omega(m\cdot\rho)}$ and the lattice attacks cost $2^{\Omega(\gamma\cdot m/(\eta-\rho)^2)}$ instead of $2^{\Omega(\rho)}$ and $2^{\Omega(\gamma/(\eta-\rho)^2)}$, respectively. Therefore, when using the VAGCD problem, we can divide both ρ and γ by m and maintain the same security level. To illustrate how significant this reduction in the parameters can be, consider again the example of the FHE scheme given above. For a security level of 100 bits, now we could use $\rho = 100/m$. By setting $m = 10$, we have $\rho = 10$. Thus, $\eta = 11\rho = 110$ and $\gamma = \lambda(\eta-\rho)^2/m = 100\cdot 100^2/10 = 10^5$. A VAGCD sample is an m dimensional vector, thus, its bit size can be approximated by $m\cdot\gamma = 10^6$. Before, an AGCD sample had 10^8 bits. Moreover, in one AGCD sample, we can encrypt one message, while in a VAGCD sample, we can encrypt m messages (one in each entry of the vector before randomizing). Thus, in this example, considering that the message space is simple $\{0,1\}$, we would use 10^6 bits to encrypt 10 bits, which represents a ciphertext expansion of 10^5, one thousand times smaller than the ciphertext expansion obtained in the example above using the AGCD problem.

3.2 Approximate-GCD and common variants

The main goal of this section is to formally define the Approximate Greatest Common Divisor (AGCD) problem, introduced in [HG01], some variants used in several cryptographic schemes, and also their underlying distributions.

Definition 3.2.1. *Let ρ, η, γ, and p be integers such that $\gamma > \eta > \rho > 0$ and p has η bits. The distribution $\mathcal{D}_{\gamma,\rho}(p)$, whose support is $[\![0, 2^\gamma - 1]\!]$ is defined as*

$$\mathcal{D}_{\gamma,\rho}(p) := \{\textit{Sample } q \leftarrow [\![0, 2^\gamma/p\,[\![\textit{ and } r \leftarrow]\!] - 2^\rho, 2^\rho[\![\ : \ \textit{Output } x := pq + r\}.$$

The distribution $\mathcal{D}_{\gamma,\rho}(p)$ is the underlying distribution of the AGCD problem. Notice that the elements sampled from $\mathcal{D}_{\gamma,\rho}(p)$ are near multiples of p, because their distance to an element of $p\mathbb{Z}$ is upper-bounded by 2^ρ, which is typically small when compared with 2^η. Then the AGCD problem is defined as follows [DGHV10]:

Definition 3.2.2 (AGCD)**.** *The (ρ, η, γ)-approximate-GCD problem is the problem of finding p, given arbitrarily many samples from $\mathcal{D}_{\gamma,\rho}(p)$.*

It is also possible to define a decisional version of the AGCD problem:

Definition 3.2.3 (dAGCD)**.** *The (ρ, η, γ)-decisional-approximate-GCD (dAGCD) problem is the problem of distinguishing between $\mathcal{D}_{\gamma,\rho}(p)$ and $\mathcal{U}(\llbracket 0, 2^\gamma \rrbracket)$.*

Notice that in the AGCD problem, all the samples share the same fixed value p, thus, one has to recover p given samples of the form $x_i := pq_i + r_i$. The values r_i are referred to as the noise terms and the bit length of each x_i is smaller than γ. It is also common to require that the secret integer p be a prime number, although no known attack exploits the fact that p is composite [CS15].

A common variant of the AGCD problem consists in defining x_0 as a multiple of p, that is, a noiseless AGCD sample $x_0 := pq_0$ [CMNT11, BBL17]. The main motivation to define this variant is to have a public modulus so that we can operate with AGCD samples then reduce modulo x_0 to obtain new AGCD samples with the same bit length. For instance, when we multiply x_i by x_j, we obtain a $x := p(q_i x_j + r_i q_j) + r_i r_j$, which can be viewed as a new AGCD sample with noise term $r_i r_j$ and bit length around 2γ. If we had $x_0 = pq_0 + r_0$ and we reduced x modulo x_0, we would obtain $[x]_{x_0} = x - ux_0 = pq' + r_i r_j - ur_0$ for some integer $u \approx 2^\gamma$, thus, the new noise term, $r_i r_j - ur_0$ would probably be larger than p, which means that $[x]_{x_0}$ is an ill-defined AGCD sample. However, this problem is solved if $r_0 = 0$, i.e., by using $x_0 = pq_0$, it holds that $[x]_{x_0}$ is a γ-bit AGCD sample with noise term equal to $r_i r_j$. This variant is defined as follows:

Definition 3.2.4 (Partial AGCD)**.** *The (ρ, η, γ)-partially-approximate-GCD (partial AGCD) problem is the problem of finding p, given arbitrarily many samples from $\mathcal{D}_{\gamma,\rho}(p)$ and a value $x_0 := pq_0$.*

Another common variant of the AGCD problem, which is also motivated by the homomorphic schemes and multilinear maps, is the multi-prime AGCD problem. In this version of the problem, instead of a single secret p, there are n secret integers $p_1, ..., p_n$

that must be recovered and each sample is defined by combining n noise terms $r_1, ..., r_n$ using the Chinese remainder theorem (CRT). Formally,

Definition 3.2.5 (Underlying distribution of Multi-prime AGCD)**.** *Let ρ, η, γ and n be integers such that $\gamma > \eta > \rho > 0$. Let p_1, ..., p_n be η-bit primes. The distribution $\mathcal{D}_{\gamma,\rho}(p_1, ..., p_n)$, whose support is $[\![0, 2^\gamma - 1]\!]$ is defined as*

$$\mathcal{D}_{\gamma,\rho}(p_1, ..., p_n) := \{\textit{Sample } r_i \leftarrow [\![-2^\rho, 2^\rho[\![\;\; \textit{for } 1 \leq i \leq n : \;\; \textit{Output } x := \mathsf{CRT}_{(p_i)}(r_i)\}.$$

Definition 3.2.6 (Multi-prime AGCD)**.** *Let $p_1, ..., p_n$ be n random η-bit prime integers. The (ρ, η, γ, n)-multi-prime AGCD problem is the problem of finding each p_i given arbitrarily many samples of $\mathcal{D}_{\gamma,\rho}(p_1, ..., p_n)$.*

Finally, one can also define the multi-prime AGCD problem with an additional noiseless sample x_0.

3.3 Main attacks against the AGCD problem

In this section, we recall the two main families of attacks against the AGCD problem, namely, the orthogonal lattice attacks [DGHV10, CS15] and the GCD attacks [CMNT11, CN12, LS14]. It is worth noting that no attack directly on the decisional version of AGCD is known, thus, to the best of the current knowledge, it can only be solved by solving the search version first, that is, by finding p and then reducing the samples x_i modulo p, which results in the small noise terms r_i's when x_i's are AGCD samples, but yields random η-bit integers when x_i's are uniform.

3.3.1 Orthogonal lattice attack

This type of attack is based in the orthogonal lattice attacks against a variant of the subset sum problem proposed in [NS01]. They were adapted to the AGCD problem in [DGHV10]. Given AGCD samples $x_1, ..., x_t$ for some $t \in \mathbb{N}$, one can define the basis

matrix

$$\mathbf{B} := \begin{pmatrix} x_1 & x_2 & \dots & x_t \\ 2^\rho & & & \\ & 2^\rho & & \\ & & \ddots & \\ & & & 2^\rho \end{pmatrix} \in \mathbb{Z}^{(t+1)\times t}$$

and the lattice L generated by the columns of $\mathbf{B}$.

Now, for each vector $\mathbf{v} \in L$, there is a $\mathbf{u} = (u_1, ..., u_t) \in \mathbb{Z}^t$ such that

$$\mathbf{v} = \mathbf{Bu} = \left(\sum_{i=1}^{t} u_i x_i,\ 2^\rho u_1,\ \cdots,\ 2^\rho u_t \right) \in \mathbb{Z}^{t+1}.$$

Therefore, by defining $\mathbf{w} := (1, r_1/2^\rho, ..., r_t/2^\rho) \in \mathbb{Q}^{t+1}$, we see that all $\mathbf{v} \in L$ satisfy $\mathbf{vw} = v_0 - \sum_{i=1}^{t} \frac{v_i r_i}{2^\rho} = \sum_{i=1}^{t} u_i(x_i - r_i) = p \sum_{i=1}^{t} u_i q_i$, which means $\mathbf{vw} = 0 \bmod p$. But then, if the norm of $\mathbf{v}$ is small enough, the only multiple of p that satisfies this equation is the 0, in other words, the equation holds over $\mathbb{Z}$ without the reduction modulo p. That is to say, $|\mathbf{vw}| < p \implies \mathbf{vw} = 0$. But for $|\mathbf{vw}| < p$ to hold, it is sufficient to have $\|\mathbf{v}\|_2 < p/\|\mathbf{w}\|_2 \approx 2^\eta$. Thus, this attack consists in running a lattice basis reduction algorithm in L to recover t short vectors $\mathbf{v}_i$'s orthogonal to $\mathbf{w}$, then defining $\mathbf{V} \in \mathbb{Z}^{t\times(t+1)}$ such that $\mathsf{row}_i(\mathbf{V}) = \mathbf{v}_i$, and solving the equation $\mathbf{Vw} = \mathbf{0}$ to recover the noise terms r_i's. Notice that because $w_1 = 1$, there are only t variables, thus, one can find $\mathbf{w}$ with simple linear algebra. Once $\mathbf{w}$ is found, one can recover the noise terms r_i's, then compute $\gcd(x_1 - r_1, ..., x_t - r_t)$, which reveals p.

Because L has rank t and its determinant is $2^{\rho(t-1)}\sqrt{2^{2\rho} + x_1^2 + ... + x_t^2} \approx 2^{\rho t+\gamma}$ [GGM16], a lattice basis reduction is expected to output vectors $\mathbf{v}_i$'s such that $\|\mathbf{v}_i\| < 2^{\alpha t}(\det(L))^{1/t} \approx 2^{\alpha t} \cdot 2^{\gamma/t+\rho}$, where 2^α is the root-Hermite factor of the reduction algorithm. Hence, one needs $2^{\alpha t}(\det(L))^{1/t} \approx 2^{\alpha t} \cdot 2^{\gamma/t+\rho} < 2^\eta$ in order to obtain vectors orthogonal to $\mathbf{w}$. Thus, t must satisfy at least $2^{\gamma/t+\rho} < 2^\eta \implies t > \gamma/(\eta - \rho)$. Then, the root-Hermite factor must satisfy at least

$$2^{\alpha t} \cdot 2^\rho < 2^\eta \implies \alpha < (\eta - \rho)/t < (\eta - \rho)^2/\gamma.$$

Using the "rule of thumb" that we need time $2^{\Omega(\lambda)}$ to achieve root Hermite factor of $2^{O(\log(\lambda)/\lambda)}$ [HPS11, CS15, APS15], we obtain finally $\alpha = O(\log(\lambda)/\lambda) < (\eta-\rho)^2/\gamma$, thus, we have to choose $\gamma = \Omega(\lambda(\eta-\rho)^2/\log(\lambda))$ to guarantee that the attack takes exponential time in λ [CS15].

It is worth noting that this attack does not benefit of the noiseless sample x_0, i.e., its time and memory complexities are the same if x_0 is public or private.

3.3.2 GCD attacks

The basic idea of this family of attacks is to perform an exhaustive search of the noise terms r_i's of some samples x_i's to obtain multiples of p, then to compute the GCD of these multiples to recover p. The naïve version of this attack can then be formulated as follows: Given AGCD samples $x_i := pq_i + r_i$ for $1 \leq i \leq t$, one first obtain a multiple of p by fixing x_1 and x_2 and computing $d_{a,b} = \gcd(x_1 - a, x_2 - b)$ for all $a, b \in [\![-2^\rho, 2^\rho]\!]$. If the bit length of $d_{a,b}$ is bigger than η, then $d_{a,b}$ is probably a multiple of p and $(a, b) = (r_1, r_2)$. Then, define $d_2 := d_{a,b}$ and continue computing $d_i = \gcd(d_{i-1}, x_i - r)$ for $3 \leq i \leq t$ and $r \in [\![-2^\rho, 2^\rho]\!]$ until d_i has η bits. The complexity of the first step is $O\left(2^{2\rho}\right)$ and the second step costs $O\left(t2^\rho\right)$, thus, the complexity of this naïve attack is $O\left(2^{2\rho}\right)$. Notice that if an attacker has access to a noiseless sample x_0, as it is the case in the partial AGCD problem, then, the first step is not necessary and the complexity of is reduced to $O\left(t2^\rho\right) = O\left(2^\rho\right)$.

The Chen-Nguyen's attack

At Eurocrypt 2012, Chen and Nguyen proposed an improved GCD attack against the partial AGCD problem [CN12] that has memory and time complexity $\tilde{O}\left(2^{\rho/2}\right)$. The two key concepts in this attack are the use of a noiseless sample x_0 to perform all the operations over $\mathbb{Z}_{x_0}$, avoiding thus that the bit length of the operands increases and guaranteeing that all scalar operations cost $\tilde{O}(\gamma)$, and the use of multipoint polynomial evaluation algorithms, which can evaluate a degree-d univariate polynomial $f(x)$ at points a_1, ..., a_d using $\tilde{O}(d)$ scalar operations over $\mathbb{Z}_{x_0}$ (notice that using simple evaluation algorithms, as the Horner's method, d times to obtain $f(a_1)$, ..., $f(a_d)$ takes time quadratic on d).

Firstly, let's assume that we have a sample $c := pq + r \leftarrow \mathcal{D}_{\gamma,\rho}(p)$ with $r \in [\![0, 2^\rho - 1]\!]$. If the noise term of c belongs to $[\![-2^\rho + 1, 0]\!]$ instead, then the attack will fail and we can simply run it again using $-c$. Moreover, assume that ρ is even. If this is not the case, then, just use $\rho + 1$ instead of ρ.

The starting point of the attack is the following: Using the noiseless sample x_0, define $d := \prod_{i=0}^{2^\rho - 1}(c - i) \mod x_0$, then the equation $p = \gcd(x_0, d)$ holds with high probability (or, at least, this GCD yields a multiple of p that is short enough to be factorized).

Now notice that by defining the polynomial $f(x) = \prod_{i=0}^{2^{\rho/2}-1}(c-(x+i)) \mod x_0$ one can rewrite the above product as $\prod_{i=0}^{2^\rho - 1}(c-i) = \prod_{k=0}^{2^{\rho/2}-1} f(2^{\rho/2}k) \pmod{x_0}$. Therefore, using a multipoint polynomial evaluation algorithm, one can compute $d_k := f(2^{\rho/2}k) \mod x_0$ for $0 \leq k \leq 2^{\rho/2} - 1$ using $\tilde{O}(2^{\rho/2})$ operations over $\mathbb{Z}_{x_0}$. The memory complexity is also $\tilde{O}(2^{\rho/2})$. Then, computing the product $d = \prod_{k=0}^{2^{\rho/2}-1} d_k \mod x_0$ can also be done with $\tilde{O}(2^{\rho/2})$ ring operations. Finally, because the bit length of both x_0 and d is upper bounded by γ, one can compute $\gcd(x_0, d)$ in time $\tilde{O}(\gamma) \subset \tilde{O}(2^{\rho/2})$. Therefore, the time complexity of this attack is $\tilde{O}(2^{\rho/2})$.

The Lee-Seo attack

At Crypto 2014, Lee and Seo generalized Chen-Nguyen attack to the partial AGCD problem with masked samples [LS14]. In detail, suppose that there is a secret mask $z \in \mathbb{Z}_{x_0}$, where $x_0 := q_0 p$ is a noiseless AGCD sample, and that we are given masked AGCD samples x_i's such that $x_i = r_i \cdot z \bmod p$ for $r_i \in [\![-2^\rho + 1, 2^\rho - 1]\!]$.

The main idea is to construct two lists of masked AGCD samples $A, B \subset \{x_1, ..., x_t\}$ and look for two elements $(a, b) \in A \times B$ that have the same noise term, i.e., $r_a = r_b$. Notice that in this case, the following holds: $r_a = r_b \bmod p \iff r_a z = r_b z \bmod p \iff a = b \bmod p$, therefore, $p|(a - b)$ and we can recover p by computing $\gcd(a - b, x_0)$. If these lists have around $2^{\rho/2}$ elements, then such pair (a, b) exists with high probability. Hence, $d := \prod_{(a,b)\in A\times B}(a-b) \mod x_0 \in p\mathbb{Z}$ because at least one of its factors is a multiple of p.

Similarly to Chen-Nguyen attack, d can be computed using $\tilde{O}(2^{\rho/2})$ operations over

$\mathbb{Z}_{x_0}$. Namely, one can define the polynomial $f(x) := \prod_{a\in A}(a - x) \bmod x_0$, whose degree is $|A| \approx 2^{\rho/2}$, and use multipoint evaluation to obtain $f(b_1), ..., f(b_{|B|})$, then compute $d = \prod_{b_j\in B} f(b_j) \bmod x_0$. Finally, once d is computed, one recovers p by computing $\gcd(d, x_0)$. Therefore, this attack can be performed in time and memory $\tilde{O}(2^{\rho/2})$.

3.3.3 New orthogonal lattice attack against partial multi-prime AGCD

In [CLT13], an attack to recover all the n primes p_i's is presented, however, in its last step, a lattice L orthogonal to the noise terms of the AGCD samples is constructed and one must run a lattice-basis reduction algorithm on L to recover the short vectors, which are likely to be the noise terms. However, the rank of L is n, therefore, finding short vectors of L is only viable in small dimension. Hence, for general values of n, solving the partial multi-prime AGCD problem was still considered an open problem [GGM16].

Thus, in this section, we present a new attack against the multi-prime AGCD problem with a noise-free element x_0. The first step is similar to the one of [CLT13] and it outputs a lattice orthogonal to the noise terms. Then, in the second step, we define a matrix $\mathbf{W}$ with a special structure and recover the noise terms by computing the eigenvalues of $\mathbf{W}$, similarly to [CHL+15]. Finally, we obtain the secret primes by computing GCDs.

In detail, consider that one has a noiseless sample $x_0 := \prod_{i=0}^{n} p_i$ and AGCD samples $\mathbf{x} := (x_1, ..., x_t)$ such that $x_j \bmod p_i = r_{i,j} \in [\![-2^\rho, 2^\rho]\!]$. Then, it is possible to construct the lattice of vectors orthogonal to $\mathbf{x}$ modulo x_0, that is, $L := \{\mathbf{u} \in \mathbb{Z}^t : \mathbf{ux} = \mathbf{0} \pmod{x_0}\}$. Moreover, let $\mathbf{r}_i := (r_{i,1}, ..., r_{i,t}) = \mathbf{x} \bmod p_i$ and consider the lattice generated by the vectors $\mathbf{r}_i$'s, i.e., $L_{\mathbf{r}} = \{\sum_{i=1}^{n} a_i\mathbf{r}_i : (a_1, ..., a_n) \in \mathbb{Z}^n\}$. Consider also $L_{\mathbf{r}}^{\perp}$, the orthogonal lattice of $L_{\mathbf{r}}$.

Notice that if a vector $\mathbf{u}$ is orthogonal to all $\mathbf{r}_i$'s, then, it is orthogonal to $\mathbf{x}$ over $\mathbb{Z}_{x_0}$, i.e., if $\mathbf{u} \cdot \mathbf{r}_i = 0 \in \mathbb{Z}$ for $1 \leq i \leq n$, then $\mathbf{u} \cdot \mathbf{x} = 0 \pmod{x_0}$, in other words, $L_{\mathbf{r}}^{\perp} \subset L$. Thus, the first step consists in running a lattice basis reduction on L to recover a reduced basis $\{\mathbf{u}_i\}_{i=1}^{t}$, where the first $t - n$ vectors form a basis of the sublattice $L_{\mathbf{r}}^{\perp}$.

In order to do so, notice that if $\mathbf{u} \in L$, then it holds that $\mathbf{u} \cdot \mathbf{r}_i = 0 \pmod{p_i}$, but if $\mathbf{u}$ is short enough, more precisely, if $\|\mathbf{u}\|_2 \|\mathbf{r}_i\|_2 < p_i$, then $\mathbf{u} \cdot \mathbf{r}_i = 0$ over $\mathbb{Z}$. Hence,

considering that $\|\mathbf{r}_i\|_2 \approx 2^\rho$ and $p_i \approx 2^\eta$, we want to recover $\mathbf{u}$ such that $\|\mathbf{u}\|_2 < 2^{\eta-\rho}$. Since $L_{\mathbf{r}}^\perp \subset L$, $\mathsf{dim}(L_{\mathbf{r}}^\perp) = t-n$ (assuming that the vectors $\mathbf{r}_i$'s are linearly independent), and $\mathsf{dim}(L) = t$, when we apply a lattice basis reduction on L, we expect to recover vectors of norm

$$2^{\alpha \cdot \mathsf{dim}(L)} \lambda_1(L_{\mathbf{r}}^\perp) \approx 2^{\alpha t} \left(\prod_{i=1}^{n} \|\mathbf{r}_i\|_2 \right)^{1/(t-n)} \approx 2^{\alpha t} \cdot 2^{\rho n/(t-n)}$$

where 2^α is the root-Hermite factor of the lattice basis reduction algorithm. Thus, this first step should work if $2^{\alpha t + \rho n/(t-n)} < 2^{\eta-\rho}$, or, equivalently,

$$\alpha t + \rho n/(t-n) < \eta - \rho. \tag{3.1}$$

Hence, after finding a basis $\mathbf{u}_1, ..., \mathbf{u}_{t-n}$ of the sublattice $L_{\mathbf{r}}^\perp$, we can finally construct $\bar{L}_{\mathbf{r}} := (L_{\mathbf{r}}^\perp)^\perp$ by computing the orthogonal lattice of $L_{\mathbf{r}}^\perp$. Notice that $\bar{L}_{\mathbf{r}} = \mathsf{span}_{\mathbb{Q}}(L_{\mathbf{r}}) \cap \mathbb{Z}^t$, therefore, it contains the short vectors $\mathbf{r}_i$'s. However, we cannot recover them directly in general, because $\mathsf{rank}(\bar{L}_{\mathbf{r}}) = \mathsf{dim}(L_{\mathbf{r}}^\perp) - \mathsf{rank}(L_{\mathbf{r}}^\perp) = t - (t-n) = n$, which can be too large for a lattice basis reduction algorithm to recover the short vectors. But the main observation is that a basis $\mathbf{B}$ of $\bar{L}_{\mathbf{r}}$ can be rewritten in terms of the vectors $\mathbf{r}_i$'s, that is, since each column $\mathbf{b}_j$ of $\mathbf{B}$ can be written as a $\mathbb{Q}$-linear combination $\sum_{i=1}^n a_{i,j} \cdot \mathbf{r}_i$, we can write $\mathbf{B} = \mathbf{RA}$, where $\mathbf{R} = [\mathbf{r}_1 \dots \mathbf{r}_n] \in \mathbb{Z}^{t \times n}$ and $\mathbf{A} \in \mathbb{Q}^{n \times n}$ where each entry (i,j) of $\mathbf{A}$ is $a_{i,j}$. Thus, $\mathbf{B}$ provides us with linear equations on the noise terms, hence, as in the attack of [CHL+15], we can exploit these equations to recover the secret p_i's.

Notice that this first step can be viewed as a black box that receives a vector $\mathbf{x}$ and outputs a matrix $\mathbf{B} = \mathbf{RA}$ if Inequality 3.1 is satisfied. Consider then that we have $n+1$ AGCD samples and let $y := x_{n+1}$. Also, let $s_i = y \bmod p_i$. Then, define $\mathbf{z} := (\mathbf{x},\, y \cdot \mathbf{x}) \in \mathbb{Z}^{2n}$. We see that $\mathbf{z} \bmod p_i = (\mathbf{r}_i,\, s_i \cdot \mathbf{r}_i) \in \mathbb{Z}^{2n}$. Therefore, the "matrix of noise terms" is

$$\mathbf{Z} = \begin{bmatrix} \mathbf{r}_1 & \cdots & \mathbf{r}_n \\ s_1 \cdot \mathbf{r}_1 & \cdots & s_n \cdot \mathbf{r}_n \end{bmatrix} = \begin{bmatrix} \mathbf{R} \\ \mathbf{R} \cdot \mathbf{S} \end{bmatrix}$$

instead of simply being $\mathbf{R} \in \mathbb{Z}^{n \times n}$, where $\mathbf{S} = \mathsf{diag}(s_1, ..., s_n)$.

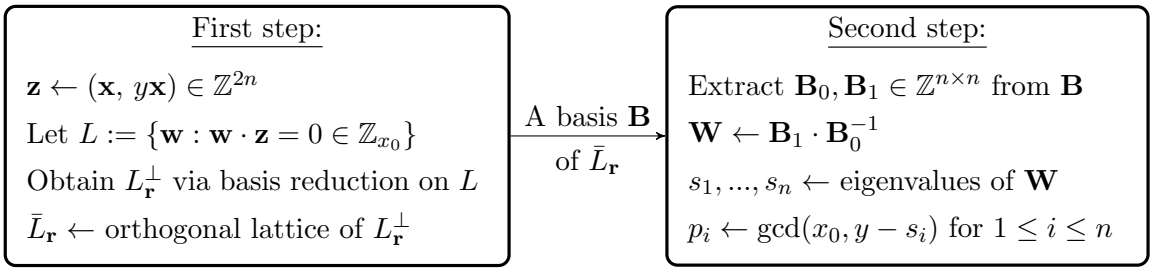

Figure 3.1: Two main steps of our attack against multi-prime AGCD.

Hence, instead of running the first step on $\mathbf{x}$, we run it on $\mathbf{z}$, obtaining

$$\mathbf{B} = \mathbf{ZA} = \begin{bmatrix} \mathbf{R} \cdot \mathbf{A} \\ \mathbf{R} \cdot \mathbf{S} \cdot \mathbf{A} \end{bmatrix}$$

for some matrix $\mathbf{A} \in \mathbb{Q}^{n \times n}$.

Finally, the second step of our attack is the following: We extract the first n rows of $\mathbf{B}$, which can be written as $\mathbf{B}_0 = \mathbf{RA}$, and also the last n rows, which we write as $\mathbf{B}_1 = \mathbf{RSA}$. Assuming that both $\mathbf{R}$ and $\mathbf{A}$ are invertible over $\mathbb{Q}$, we can then proceed as in [CHL+15] and compute

$$\mathbf{W} = \mathbf{B}_1 \cdot \mathbf{B}_0^{-1} = \mathbf{RSR}^{-1}.$$

But we know that the eigenvalues of $\mathbf{W}$ are exactly those of $\mathbf{S}$, which in turn, are the noise terms s_i's, since $\mathbf{S}$ is a diagonal matrix. Therefore, we can recover $s_1, ..., s_n$ by computing the eigenvalues of $\mathbf{W}$. After that, it is sufficient to compute $\gcd(y - s_i, x_0)$ to recover each p_i. On Figure 3.1 we see a summary of the two steps of this attack.

Because we are using $\mathbf{z} \in \mathbb{Z}^{2n}$ as input of the first step, we have $t = 2n$ in Inequality 3.1, which gives the condition $2n\alpha + 2\rho < \eta$, from which we can derive $\alpha < \frac{\eta}{2n}$. But achieving root Hermite-factor of 2^{α} (heuristically) requires time $2^{\Omega(1/\alpha)}$ [HPS11], thus, the cost of this attack is $2^{\Omega(n/\eta)} = 2^{\Omega(\gamma/\eta^2)}$, since $\gamma = n \cdot \eta$.

3.4 The randomized vector AGCD Problem

In this section, a new variant of the AGCD problem is introduced and cryptanalyzed. The main motivation is to increase the cost of the attacks by increasing the dimension of the samples, i.e., instead of having scalars of the form $x := pq + r$, we have vectors $\mathbf{x} := (p\mathbf{q}+\mathbf{r})\mathbf{K}$, for a secret random $m \times m$ matrix $\mathbf{K}$. Because $\mathbf{K}$ acts as a mask, it is not possible to access each scalar AGCD sample $pq_j + r_j$, since each entry x_j of x is actually of the form $\sum_{i=1}^{n}(pq_i + r_i)k_{i,j}$, that is, a random linear combination of m AGCD samples.

We define the vector approximate-GCD (VAGCD) problem as follows:

Definition 3.4.1 (VAGCD)**.** *For a random η-bit prime integer p, generate a random $m \times m$ matrix $\mathbf{K}$ invertible modulo p. Given many samples $\tilde{\mathbf{v}} := \mathbf{v} \cdot \mathbf{K}$ where $\mathbf{v} \leftarrow (\mathcal{D}_{\gamma,\rho}(p))^m$, output p.*

We can also define a multi-prime version of this problem:

Definition 3.4.2 (Multi-prime VAGCD problem)**.** *For random η-bit prime integers $p_1, ..., p_n$, let $x_0 = \prod_{i=1}^{n} p_i$. Let $\mathbf{K}$ be a random $m \times m$ matrix invertible modulo x_0. Given many vectors $\tilde{\mathbf{v}} = \mathbf{v} \cdot \mathbf{K}$, where $\mathbf{v} \leftarrow (\mathcal{D}_{\gamma,\rho}(p_1, ..., p_n))^m$, output the primes p_i's.*

Partial versions of these problems are defined in the same way, except that $x_0 := q_0 p$ (or $x_0 := \prod_{i=1}^{n} p_i$ in the multi-prime case) is also made public. It is clear that the vector variants of the AGCD problem cannot be easier than the original problem, since given many scalars $x_1, ..., x_t$ constituting an instance of the AGCD problem, one can sample a random $m \times m$ matrix $\mathbf{K}$, define vectors $\mathbf{v}_i$'s using the scalars x_i's, and randomize each $\mathbf{v}_i$ obtaining $\tilde{\mathbf{v}}_i = \mathbf{v}_i \cdot \mathbf{K}$, which are valid instances of the vector AGCD problem with overwhelming probability, since $\mathbf{K}$ is likely to be invertible modulo p (or modulo $\prod_{i=1}^{n} p_i$). Moreover, the vector variants can be harder to solve, so that smaller parameters could be used. In fact, we adapted the orthogonal lattice and the GCD attacks to the vector AGCD problem and they are indeed more expensive, as it is shown in Section 3.5. Thus, basing the security on the vector AGCD problem, we can construct cryptographic schemes that are more efficient than their equivalents based on the original AGCD problem.

3.5 Cryptanalysis of the VAGCD problem

In this section, we adapt the attacks against the traditional variants of the AGCD problem to the variants of the vector AGCD problem defined in Section 3.4.

3.5.1 Orthogonal lattice attack on the partial VAGCD problem

In this attack, one has access to a noiseless AGCD sample $x_0 := pq_0$ and to VAGCD samples of the form $\tilde{\mathbf{c}}_i := \mathbf{c}_i \cdot \mathbf{K} \in \mathbb{Z}^m$, where $\mathbf{c}_i := p\mathbf{q}_i + \mathbf{r}_i \in \mathbb{Z}^m$, and the goal is to recover the secret p (recovering the matrix $\mathbf{K}$ would also work, since it would allow to disclose p by running the known attacks against the original AGCD problem). By defining a matrix $\tilde{\mathbf{C}} \in \mathbb{Z}^{t\times m}$ such that $\mathsf{row}_i(\tilde{\mathbf{C}}) := \tilde{\mathbf{c}}_i$, we see that $\tilde{\mathbf{C}} = (p\mathbf{Q} + \mathbf{R})\mathbf{K}$ for some matrices $\mathbf{Q}$ and $\mathbf{R}$. Thus, one can proceed as in the original orthogonal lattice attack and try to use lattice basis reduction algorithms to recover vectors orthogonal to the matrix of noise terms $\mathbf{R} \in \mathbb{Z}^{t\times m}$, and from those orthogonal vectors, it is possible to construct a lattice $\bar{L}$ that contains the columns of $\mathbf{R}$. However, the rank of $\bar{L}$ is m, which can be very big, hence, finding the short vectors of $\bar{L}$ (that is, the columns of $\mathbf{R}$) may be infeasible. Thus, instead of trying to recover $\mathbf{R}$ and then using it to reveal p, we proceed as in the orthogonal lattice attack on the multi-prime AGCD problem and perform an algebraic step, but using the Cayley–Hamilton theorem to obtain several multiples of p, then computing the GCD of these multiples to recover p.

In detail, the attack is as follows: suppose that we can obtain linearly independent vectors $\mathbf{v}_1, ..., \mathbf{v}_{t-m} \in \mathbb{Z}^t$ such that $\mathbf{v}_i \cdot \mathbf{R} = \mathbf{0} \in \mathbb{Z}^m$ (we will explain soon how to do it). Then, we can construct the lattice of vectors orthogonal to these $\mathbf{v}_i$'s, that is, $L := \{\mathbf{u} \in \mathbb{Z}^t : \mathbf{u} \cdot \mathbf{v}_i = 0 \text{ for } 1 \leq i \leq t-m\}$. Notice that $\mathsf{rank}(L) = t - (t-m) = m$. Thus, let $\mathbf{B} \in \mathbb{Z}^{t\times m}$ be a basis of L. Since each column $\mathbf{r}_j$ of $\mathbf{R}$ belongs to L, for $1 \leq j \leq m$ there exists for some $\mathbf{a}_j \in \mathbb{Z}^m$ such that $\mathbf{r}_j = \mathbf{B} \cdot \mathbf{a}_j$. Therefore, we can write $\mathbf{R} = \mathbf{B} \cdot \mathbf{A}$ for $\mathbf{A} := [\ \mathbf{a}_1 \mid \ ... \ \mid \mathbf{a}_m\ ] \in \mathbb{Z}^{m\times m}$. In particular, by writing $\mathbf{R}$ and $\mathbf{B}$ by blocks, with the

first two blocks being $m \times m$ matrices, we have

$$\mathbf{R} \cdot \mathbf{A}^{-1} = \begin{bmatrix} \mathbf{R}_1 \cdot \mathbf{A}^{-1} \\ \mathbf{R}_2 \cdot \mathbf{A}^{-1} \\ \mathbf{R}_3 \cdot \mathbf{A}^{-1} \end{bmatrix} = \mathbf{B} = \begin{bmatrix} \mathbf{B}_1 \\ \mathbf{B}_2 \\ \mathbf{B}_3 \end{bmatrix}.$$

Hence, given $\mathbf{B}$, we can extract the first two blocks $\mathbf{B}_i$'s and compute $\mathbf{W} := \mathbf{B}_1 \cdot \mathbf{B}_2^{-1} \in \mathbb{Q}^{m \times m}$. Assuming that $\mathbf{R}_2$ is invertible, it holds that $\mathbf{W} = \mathbf{R}_1 \cdot \mathbf{R}_2^{-1} \in \mathbb{Q}^{m \times m}$. Then, let f be the characteristic polynomial of $\mathbf{W}$. By the Cayley–Hamilton theorem, we know that $f(\mathbf{W}) = \mathbf{0}$. Thus, by defining $\tilde{\mathbf{C}}_1 \in \mathbb{Z}^{m \times m}$ as the matrix formed by the first m rows of $\tilde{\mathbf{C}}$ and $\tilde{\mathbf{C}}_2 \in \mathbb{Z}^{m \times m}$ formed by the next m rows, and computing $\mathbf{M} = \tilde{\mathbf{C}}_1 \cdot \tilde{\mathbf{C}}_2^{-1} \in \mathbb{Q}^{m \times m}$, we have $\mathbf{M} \bmod p = \tilde{\mathbf{R}}_1 \cdot \tilde{\mathbf{R}}_2^{-1} = \mathbf{W}$, which means $f(\mathbf{M}) = f(\mathbf{W}) = \mathbf{0} \mod p$, where each rational a/b is interpreted in the obvious way in $\mathbb{Z}_p$ (i.e., $a/b \pmod p$ is $a \cdot b^{-1}$ with b^{-1} being the inverse of b modulo p). In other words, for each entry $a_{i,j}/b_{i,j}$ of $f(\mathbf{M})$, it holds that $a_{i,j} \in p\mathbb{Z}$, thus we can recover p by computing $\gcd(a_{0,0}, ..., a_{m,m})$.

It remains to show how to use $\tilde{\mathbf{C}}$ to compute $\mathbf{v}_1, ..., \mathbf{v}_{t-m} \in \mathbb{Z}^t$ satisfying $\mathbf{v}_i \cdot \mathbf{R} = \mathbf{0} \in \mathbb{Z}^m$. We start by constructing the lattice of vectors orthogonal to the columns of $\tilde{\mathbf{C}}$ modulo x_0, i.e.,

$$L_{x_0}^{\perp} := \{\mathbf{v} \in \mathbb{Z}^t : \mathbf{v} \cdot \tilde{\mathbf{C}} = \mathbf{0} \pmod{x_0}\}.$$

Since $L_{x_0}^{\perp}$ is a "q-ary lattice", as discussed in Section 2.3, it holds that $\mathsf{rank}(L^{\perp}) = t$ and $\det(L_{x_0}^{\perp}) = x_0^m$. Thus, once we have $L_{x_0}^{\perp}$, we apply a lattice basis reduction algorithm to obtain a short basis $\mathbf{v}_1, ..., \mathbf{v}_t$. Now, because $\mathbf{K}$ is invertible modulo p, we notice that $\mathbf{v} \in L_{x_0}^{\perp}$ implies

$$\mathbf{v} \cdot (p\mathbf{Q} + \mathbf{R}) = \mathbf{0} \pmod{x_0} \implies \mathbf{v} \cdot \mathbf{R} = \mathbf{0} \pmod{p}$$

Therefore, for each column $\mathbf{r}_j$ of $\mathbf{R}$, $\|\mathbf{v}\mathbf{r}_j\|_2 \in p\mathbb{Z}$. Thus, if $\|\mathbf{v}\|_2 \|\mathbf{r}_j\|_2 < p$, then $\mathbf{v}\mathbf{r}_j = 0$ over $\mathbb{Z}$. Hence, assuming that $\|\mathbf{r}_j\|_2 \approx 2^{\rho}$, a vector $\mathbf{v}_i$ of the reduced basis is orthogonal to the columns of $\mathbf{R}$ if

$$\|\mathbf{v}_i\|_2 < \frac{p}{\max\{\|\mathbf{r}_j\|_2\}_{j=1}^{m}} \approx 2^{\eta - \rho}.$$

Assuming that we need time $2^{\Omega(\lambda)}$ to achieve root Hermite factor $\delta := \lambda^{O(1/\lambda)}$ [APS15,

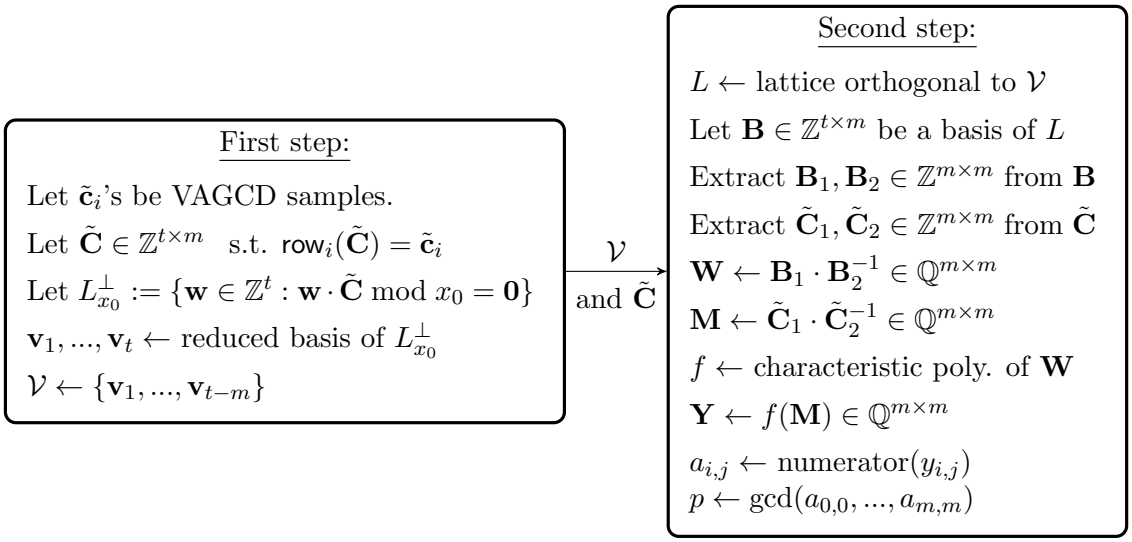

Figure 3.2: Our orthogonal lattice attack against the vector AGCD problem.

CS15, HPS11], we expect that the norms of the vectors of the reduced basis will be

$$\|\mathbf{v}_i\|_2 \approx \delta^t \cdot \det(L^{\perp}_{x_0})^{1/t} = \delta^t \cdot x_0^{m/t} \approx \lambda^{t/\lambda} \cdot 2^{m\gamma/t},$$

and we want the $\mathbf{v}_i$'s to be shorter than $2^{\eta-\rho}$, thus, for the attack to work, we have the condition

$$t\log(\lambda)/\lambda + \gamma m/t < \eta - \rho \iff t^2\log(\lambda)/\lambda - t(\eta-\rho) + \gamma/t < 0.$$

Thus, for this attack to be more expensive than $2^{\Omega(\lambda)}$, we can choose γ such that no t satisfies this inequality. To do so, it is sufficient to choose

$$\gamma = \Omega\left(\frac{\lambda(\eta-\rho)^2}{m\log(\lambda)}\right)$$

since it implies that the discriminant $(\eta-\rho)^2 - 4\log(\lambda)/\lambda\gamma$ is negative.

An overview of the attack is shown in Figure 3.2. Furthermore, we implemented this attack in Sage and it is available in [Per20d]. Letting γ' be the parameter obtained in the analysis of the original orthogonal lattice attack against the AGCD problem (Section 3.3.1), we see that in the VAGCD problem, we can set $\gamma = \gamma'/m$. Notice that the bit-length of each vector AGCD sample is about $m \cdot \gamma$, which equals the bit-length of the original AGCD samples, γ'. However, because now each sample is a vector, applications

using the vector AGCD problem can encode m integers instead of a single one into γ' bits.

3.5.2 GCD attack on the VAGCD problem

The GCD attacks on the partial AGCD problem run in time and memory $\tilde{O}(2^{\rho/2})$ [CN12, LS14], and against the original AGCD problem, when no noiseless sample x_0 is available, the time and memory complexities become $\tilde{O}(2^{\rho})$ [CNT12]. However, those attacks are not directly applicable to the VAGCD problem. In particular, in the attack described in [CNT12], one has to subtract "candidate" noise terms from an AGCD sample to possibly obtain a multiple of p, i.e., given a AGCD sample $x := pq+r$, one has to compute $x-u$ for all $u \in [\![-2^{\rho}, 2^{\rho}]\!]$, which certainly produces an integer in $p\mathbb{Z}$, since at some point u is equal to the noise term r. But considering a vector AGCD sample $\mathbf{x} := (p\mathbf{q}+\mathbf{r})\mathbf{K} \in \mathbb{Z}^m$, we see that $\mathbf{x} = p\mathbf{q}' + [\mathbf{r} \cdot \mathbf{K}]_p$ for some $\mathbf{q}'$, but $[\mathbf{r} \cdot \mathbf{K}]_p$ is a random vector from $\mathbb{Z}_p^m$, thus, fixing any entry x_i of $\mathbf{x}$ and trying to compute $x_i - u$ to obtain multiple a multiple of p would require to test all $u \in \mathbb{Z}_p$, therefore, the number of operations would be lower-bounded by 2^{η}. Hence, in this section we show how to generalize the attack of [LS14] to dimension m and how to combine it with the attack of [CNT12] to create a GCD attack against the VAGCD problem that is applicable even when x_0 is not known.

Consider VAGCD samples $\tilde{\mathbf{c}}_i := \mathbf{c}_i\mathbf{K} = (p\mathbf{q}_i + \mathbf{r}_i)\mathbf{K} \in \mathbb{Z}^m$. We say that two samples $\tilde{\mathbf{c}}_i$ and $\tilde{\mathbf{c}}_j$ form a colliding pair if they are equal modulo p, in other words, if the difference $\tilde{\mathbf{c}}_i - \tilde{\mathbf{c}}_j$ gives m multiples of p. Notice that because $\mathbf{K}$ is invertible over $\mathbb{Z}_p$, this is equivalent to having the same noise terms, i.e.,

$$\tilde{\mathbf{c}}_i = \tilde{\mathbf{c}}_j \pmod{p} \iff \mathbf{c}_i = \mathbf{c}_j \pmod{p} \iff \mathbf{r}_i = \mathbf{r}_j.$$

Because each noise vector is sampled from $\mathcal{U}([\![-2^{\rho}, 2^{\rho}[\![ )$, the probability that two independent AGCD samples collide is $\Pr[\mathbf{r}_i = \mathbf{r}_j] = 2^{-m\rho+1} \approx 2^{-m\rho}$. Thus, given two lists L_1 and L_2 of independent AGCD samples, by the union bound, the probability that there is a colliding pair $(\tilde{\mathbf{c}}_i, \tilde{\mathbf{c}}_j) \in L_1 \times L_2$ is

$$\Pr[(\tilde{\mathbf{c}}_1, \tilde{\mathbf{c}}_2) \in L_1 \times L_2 : \tilde{\mathbf{c}}_1 = \tilde{\mathbf{c}}_2 \pmod{p}] \leq |L_1| \cdot |L_2| \cdot \Pr[\mathbf{r}_i = \mathbf{r}_j] \approx |L_1| \cdot |L_2| \cdot 2^{-m\rho}.$$

Therefore, if $|L_1| = |L_2| \approx 2^{-m\rho/2}$, this probability is close to one and we expect a colliding pair to exist. But in this case, for $1 \leq k \leq m$, at least one of the terms of the following product is a multiple of p:

$$y_k := \prod_{i=1}^{|L_1|} \prod_{j=1}^{|L_2|} (\tilde{\mathbf{c}}_i[k] - \tilde{\mathbf{c}}_j[k]).$$

But if some $(\tilde{\mathbf{c}}_i[k] - \tilde{\mathbf{c}}_j[k]) \in p\mathbb{Z}$, then $y_k \in p\mathbb{Z}$. That is to say, if $|L_1| = |L_2| \approx 2^{-m\rho/2}$, then we expect all $y_1, ..., y_m$ to be multiples of p.

Once we have sufficient elements of $p\mathbb{Z}$, we can compute the GCD of them to obtain a much smaller multiple of p, an integer of the form $g = p\prod z_i$. Furthermore, by repeating the analysis of the probabilities done in [CNT12], we can see that all the terms z_i's are bounded by $2^{\rho m}$ if we have $m\rho$ elements of $p\mathbb{Z}$. Thus, using for instance, Bernstein's algorithm [Ber04], we can compute the $2^{\rho m}$-smooth part of g to obtain $\prod z_i$ in time $\tilde{O}(2^{\rho m})$, and finally recover p. We present the attack in detail in Algorithm 1.

As for the time (and memory) complexity, we can use polynomial multi-point evaluation to compute each y_k using $\tilde{O}(2^{m\rho/2})$ integer operations, however, the bit length of each y_k is approximately $\gamma 2^{m\rho}$, therefore, computing each $\gcd(y_k, y_{k'})$ takes time $\tilde{O}(2^{m\rho})$ instead of $\tilde{O}(2^{m\rho/2})$. Thus, the this attack runs im time and memory $\tilde{O}(2^{m\rho})$. Nevertheless, these asymptotic complexities can be improved if the attacker has access to a scalar AGCD sample $x_0 = pq_0 + r_0$ whose bit length, $\hat{\gamma}$, is polynomial in ρ. Firstly, consider that x_0 is noiseless, i.e., $x_0 = pq_0$. In this case, all the products performed to compute each y_k can be done modulo x_0, which means that the bit-length of y_k is upper bounded by $\hat{\gamma}$. Thus, computing the GCDs takes time $\tilde{O}(\hat{\gamma})$. Moreover, because x_0 is already a multiple of p, $\gcd(y_t, x_0)$ is also so and it is smaller than or equal to x_0, being equal if, and only if, x_0 divides y_t. Thus, instead of initializing g with one, it is possible to initialize it with x_0 and proceed as before, updating g as $g \leftarrow \gcd(g, y_t)$, which would always reduce the bit length of g and we would not need to remove its S-smooth part. Hence, the time complexity of the whole algorithm is $\tilde{O}(2^{m\rho/2})$, for the multi-point evaluation, plus $\tilde{O}(\hat{\gamma})$, for the GCDs, which results in $\tilde{O}(2^{m\rho/2})$. The same holds for the memory complexity.

The other possible scenario is that x_0 is a noisy AGCD sample. Thus, consider that

Algorithm 1: GCD attack on the vector AGCD problem

Input: $(\rho+1)\cdot 2^{m\rho/2}$ vector-AGCD samples $\tilde{\mathbf{c}}_i \in \mathbb{Z}^m$
Output: The secret prime p

Let L_1 be a list with $2^{\lceil m\rho/2 \rceil}$ samples.
for $t = 1$ until m **do**
  Let $f_t(x)$ be the polynomial $\prod_{\tilde{\mathbf{c}}\in L_1}(x - \tilde{\mathbf{c}}[t])$.
$s \leftarrow \rho$; $S \leftarrow 2^{m\rho(ms+1)/(ms-1)}$
$g \leftarrow 1$ ▷ `Variable` g `stores` p `at the end of the execution`
for $i = 1$ until s **do**
  Let L_2 be a list with $2^{\lceil m\rho/2 \rceil}$ samples.
  for $t = 1$ until m **do**
    Use multi-point evaluation to compute $d_i \leftarrow f_t(\tilde{\mathbf{c}}_i[t])$ for each $\tilde{\mathbf{c}}_i \in L_2$.
    $y_t \leftarrow \prod_{i=1}^{|L_2|} d_i$
    if $g = 1$ **then**
      Let y' be the S-smooth part of y_t
      $g \leftarrow y_t/y'$
    else
      $g \leftarrow \gcd(g, y_t)$
      if $\eta - 1 \leq \log_2 g \leq \eta$ **then**
        return g
return g

$x_0 = pq_0 + r_0$ for some $r_0 \in]\!] -2^{\rho_0}, 2^{\rho_0} [\![$, then we could repeat the attack 2^{ρ_0+1} times using $x_0' = x_0 - r$ (for $-2^{\rho_0} < r < 2^{\rho_0}$) as the modulus, obtaining then time complexity $\tilde{O}(2^{\rho_0+m\rho/2})$.

Therefore, if a scalar x_0 is available, we have the cost

$$T_{GCD,x_0}(\eta,\rho,\rho_0,\gamma,m) := (m\rho)^2 \cdot 2^{\rho_0+m\rho/2}\gamma\log\gamma$$

where $\rho_0 = 0$ if x_0 is noiseless; and if x_0 is not public, then we have

$$T_{GCD}(\eta,\rho,\gamma,m) := (m\rho)^2 2^{m\rho}\gamma\log\gamma.$$

Resembling our orthogonal lattice attack, where we concluded that the parameter γ can be divided by m, when comparing this GCD attack with the GCD attacks on the variants of the AGCD problem, we see that we can divide the bit length of the noise terms

by m, i.e., to obtain a security level of λ bits with the original AGCD problem, we must choose $\rho \in \Omega(\lambda)$, while with the vector AGCD problem, we can choose $\rho \in \Omega(\lambda/m)$.

3.5.3 Distribution of the noise term of VAGCD samples

In Section 3.5.1 and Section 3.5.1, we saw that the costs of the attacks against the VAGCD problem are essentially the m-th power of the costs of the respective attacks against the AGCD problem, e.g., GCD attacks on AGCD problem cost $\tilde{O}(2^{\rho})$ and the GCD attacks generalized to the VAGCD problem cost $\tilde{O}(2^{m\rho})$. But one could wonder if these attacks could be improved, so that we have a much smaller value multiplying the exponent, for instance, $(\log m)\rho$ instead of $m\rho$.

In this section, we present some theoretical evidences that corroborate our practical analysis and argue that, for typical parameters, if any improvement on those attacks can be done, the factor m in the exponent can only be replaced by $\Theta(m)$ (e.g., improving from $m\rho$ to $m\rho/2$), but it will not be possible to replace the factor m by any function asymptotically smaller.

In fact, in the vector AGCD we have samples $\mathbf{x} := (p\mathbf{q} + \mathbf{r})\mathbf{K} \in \mathbb{Z}^m$, and the matrix $\mathbf{K}$ is secret. It is then easy to see that each entry x_j of $\mathbf{x}$ is of the form $x_j = p\tilde{q}_j + \tilde{r}_j$ where $\tilde{r}_j = \mathbf{r} \cdot \mathsf{col}_j(\mathbf{K}) \pmod p$. But as we will see in Lemma 3.5.1, each $\tilde{r}_j$ is distributed uniformly on $\mathbb{Z}_p$, which means that one cannot hope to treat each x_j as an instance of the AGCD problem and apply the known attacks against AGCD, since such distribution of the noise term erases all the information that x_j carries about p.

But the *joint* distribution of $(\tilde{r}_1, ..., \tilde{r}_n)$ is different from $\mathcal{U}(\mathbb{Z}_p^n)$ since they are all defined with the same vector $\mathbf{r}$, which implies some correlation among them. Consequently, to solve the randomized AGCD problem, we indeed need attacks "in higher dimension", that is, we must consider more than one entry of each instance $\mathbf{x}$ in order to try to exploit the correlation in the errors.

Thus, let's consider ℓ entries of $\mathbf{x}$. Without loss of generality, take the ℓ first entries, denoted here by $\mathbf{x}^{(\ell)} := (x_1, ..., x_\ell)$. Likewise, let's consider the first ℓ columns of $\mathbf{K}$, denoted by the matrix $\mathbf{K}^{(\ell)} := [\mathbf{K}_1 \, ... \, \mathbf{K}_\ell] \in \mathbb{Z}^{m \times \ell}$. Now, the error term of $\mathbf{x}^{(\ell)}$ is $\mathbf{r}^{(\ell)} =$

$\mathbf{r}\mathbf{K}^{(\ell)} \bmod p$.

In which follows, we prove that for specific parameters, even when we consider ℓ as a constant fraction of m, like $m/2$, the noise term $\mathbf{r}^{(\ell)}$ is still statistically close to ℓ independent samples of $\mathbb{Z}_p$.

Lemma 3.5.1 (Distribution of $\mathbf{r}^{(\ell)}$)**.** *Let $\mathbf{K} \leftarrow \mathcal{U}(\mathbb{Z}_p^{m\times m})$. If $\ell \leq (\rho m + 2 - 2\lambda)/\eta$, then the statistical distance between $\mathbf{r}^{(\ell)} = \mathbf{r} \cdot \mathbf{K}^{(\ell)} \bmod p$ and $\mathcal{U}(\mathbb{Z}_p^{\ell})$ is negligible in λ.*

Proof. Substituting N by 2^ρ and $\mathbf{B}$ by $\mathbf{K}^{(\ell)}$ in Lemma 2.4.11, we have $h_{\mathbf{B}}(\mathbf{x}) = \mathbf{r}^{(\ell)}$. Therefore, by the LHL, the statistical distance between $\mathbf{r}^{(\ell)}$ and $\mathcal{U}(\mathbb{Z}_p^{\ell})$ is upper bounded by

$$\Delta := \frac{1}{2}\sqrt{\frac{|Y|}{|X|}} = \frac{1}{2}\sqrt{\frac{p^\ell}{2^{n\rho}}} \leq 2^{(\ell\eta - m\rho)/2 - 1}.$$

But $\ell \leq (\rho m + 2 - 2\lambda)/\eta \implies (\ell\eta - m\rho)/2 - 1 \leq -\lambda$, therefore, $\Delta \leq 2^{-\lambda}$, which is negligible in λ. □

Thus, since we usually set $\eta \geq \lambda$, we see that the minimum ℓ that we need to take to make it possible to attack the randomized AGCD problem is $\ell_{\min} \approx (\rho/\eta)m$. In particular, the following holds:

Corollary 3.5.2. *If $\eta = \lambda$ and $\rho = \lambda/2$, then $\mathbf{r}^{(\ell)}$ is statistically close to $\mathcal{U}(\mathbb{Z}_p^{\ell})$ for all $\ell \leq m/2 - 2$.*

Proof. Notice that for these parameters, we have $(\rho m + 2 - 2\lambda)/\eta = m/2 + 2/\lambda - 2 > m/2 - 2$. Thus, if $\ell \leq m/2 - 2$, Lemma 3.5.1 applies. □

3.6 Cryptanalysis of the multi-prime VAGCD problem

Generalizing our attacks on the (single-prime) VAGCD problem to the multi-prime variant is not straightforward. For example, in the first step of the orthogonal lattice attack presented in Section 3.5.1, we find vectors orthogonal to an "AGCD matrix" $\tilde{\mathbf{C}}$, but because they are short, they end up being orthogonal to $\tilde{\mathbf{C}} \bmod p = \mathbf{R}$. When we have

n primes, those recovered vectors are orthogonal to each $\mathbf{R}_i := \tilde{\mathbf{C}} \bmod p_i$, therefore, they are orthogonal to the matrix

$$\begin{bmatrix} \mathbf{R}_1 & \dots & \mathbf{R}_n \end{bmatrix} \in \mathbb{Z}^{t \times nm}.$$

Thus, in the second step, when we compute $\mathbf{B}_1 \cdot \mathbf{B}_2^{-1}$, we have something like

$$\begin{bmatrix} \mathbf{R}'_1 & \dots & \mathbf{R}'_n \end{bmatrix} \cdot \begin{bmatrix} \mathbf{R}''_1 & \dots & \mathbf{R}''_n \end{bmatrix}^{-1} \in \mathbb{Z}^{nm \times nm},$$

which no longer corresponds to $\tilde{\mathbf{C}}_1 \cdot \tilde{\mathbf{C}}_2^{-1}$ modulo any prime.

We solve this problem by obtaining block diagonal matrices such that each $m \times m$ block corresponds to one prime, that is, when we compute $\mathbf{W} := \mathbf{B}_1 \cdot \mathbf{B}_2^{-1}$, we obtain

$$\mathbf{V} \cdot \begin{bmatrix} \mathbf{R}'_1 \cdot (\mathbf{R}''_1)^{-1} & \mathbf{0} & \dots & \mathbf{0} \\ \mathbf{0} & \mathbf{R}'_2 \cdot (\mathbf{R}''_2)^{-1} & \dots & \mathbf{0} \\ \vdots & \vdots & \ddots & \vdots \\ \mathbf{0} & \mathbf{0} & \dots & \mathbf{R}'_n \cdot (\mathbf{R}''_n)^{-1} \end{bmatrix} \cdot \mathbf{V}^{-1} \in \mathbb{Z}^{nm \times nm},$$

where $\mathbf{V}$ is a random $nm \times nm$ matrix. Then, the characteristic polynomial of $\mathbf{W}$ can be factored and each factor corresponds to one block $\mathbf{R}'_i \cdot (\mathbf{R}''_i)^{-1}$, which corresponds to $\tilde{\mathbf{C}}_1 \cdot \tilde{\mathbf{C}}_2^{-1} \bmod p_i$, allowing thus to recover p_i as previously.

However, to obtain $\mathbf{W}$ multiplied by this matrix $\mathbf{V}$ (and its inverse) in the second step, instead of constructing a lattice orthogonal to $\tilde{\mathbf{C}}$, we have to start the first step with a product $\tilde{\mathbf{V}} \cdot \tilde{\mathbf{C}}$, where the norm of each $\mathbf{V}_i := \tilde{\mathbf{V}} \bmod p_i \in \mathbb{Z}^{nm \times m}$ is small and the matrix $\mathbf{V} = [\, \mathbf{V}_1 \dots \mathbf{V}_n \,] \in \mathbb{Z}^{nm \times nm}$ is invertible. But, of course, we cannot construct $\tilde{\mathbf{V}}$, since we do not know the primes $p_1, ..., p_n$. Thus, our attack works only in scenarios where such matrix $\tilde{\mathbf{V}}$ is given. For example, in the multilinear map proposed in [CLT13], we have matrices of the form $\tilde{\mathbf{C}} := \mathbf{K} \cdot \mathbf{C} \cdot \mathbf{K}'$ and a matrix $\tilde{\mathbf{V}} := \mathbf{V} \cdot \mathbf{K}^{-1}$, such that both $\mathbf{C}$ and $\mathbf{V}$ have small norm modulo each p_i. Thus, $\tilde{\mathbf{V}} \cdot \tilde{\mathbf{C}} = \mathbf{V} \cdot \mathbf{C} \cdot \mathbf{K}'$ has the desired form.

3.6.1 Orthogonal lattice attack on the multi-prime VAGCD problem

Suppose that we have multi-prime VAGCD samples $\tilde{\mathbf{v}}_j$'s of the form $\mathbf{v}_j \cdot \mathbf{K} \in \mathbb{Z}^m$ such that, for each secret prime p_i, it holds that $\mathbf{v}_j = \mathbf{r}_{i,j} \mod p_i$, with $\|\mathbf{r}_{i,j}\|_\infty < 2^\rho$. To

simplify the description of the attack, suppose also that we have access to the noiseless sample $x_0 := \prod_{i=1}^{n} p_i$. Finally, assume that we have access to matrices $\tilde{\mathbf{C}}_j := \mathbf{K}^{-1}\mathbf{C}_j\mathbf{K}'$ such that the entries of $\mathbf{R}_{i,j} := \mathbf{C}_j \bmod p_i$ are noise terms sampled from $\rrbracket -2^\rho, 2^\rho \llbracket$. We stress that this last hypothesis is not always satisfied, since it is not implied from the multi-prime VAGCD problem only. However, we do have access to such matrices $\tilde{\mathbf{C}}_j$'s and can use this attack in some applications (e.g., in iO schemes that use the Kilian randomization on top of the [CLT13] multilinear map).

Thus, define $\tilde{\mathbf{V}} = \mathbf{V} \cdot \mathbf{K} \in \mathbb{Z}^{nm \times m}$ such that $\mathsf{row}_i(\tilde{\mathbf{V}}) = \tilde{\mathbf{v}}_i$ and also the following matrices, for $j \in \{1, 2\}$:

$$\tilde{\mathbf{D}}_j := \tilde{\mathbf{V}} \cdot \tilde{\mathbf{C}}_j = \mathbf{V} \cdot \mathbf{C}_j \cdot \mathbf{K}' \pmod{x_0}.$$

At last, we define

$$\tilde{\mathbf{D}} := \begin{bmatrix} \tilde{\mathbf{D}}_1 \\ \tilde{\mathbf{D}}_2 \end{bmatrix} \in \mathbb{Z}^{2nm \times m} \quad \text{and} \quad \mathbf{D} := \begin{bmatrix} \mathbf{V} \cdot \mathbf{C}_1 \\ \mathbf{V} \cdot \mathbf{C}_2 \end{bmatrix} \in \mathbb{Z}^{2nm \times m}$$

Similarly to the first step of the attack described in Section 3.5.1, we can then construct the lattice

$$\tilde{L} := \{\mathbf{u} \in \mathbb{Z}^{2nm} : \mathbf{u} \cdot \tilde{\mathbf{D}} = \mathbf{0} \pmod{x_0}\}$$

which equals

$$L := \{\mathbf{u} \in \mathbb{Z}^{2nm} : \mathbf{u} \cdot \mathbf{D} = \mathbf{0} \pmod{x_0}\}$$

because $\mathbf{K}'$ is invertible modulo x_0 (otherwise, we can recover a non-trivial factor of x_0, which would disclose some p_i's).

As before, for a vector $\mathbf{u}$ satisfies $\mathbf{u} \cdot \tilde{\mathbf{D}} = \mathbf{0} \pmod{p_i}$ for all $i \in [\![1, n]\!]$, then, $\mathbf{u} \in L$. And if $\mathbf{u}$ is short enough, then $\mathbf{u} \cdot \tilde{\mathbf{D}} = \mathbf{0} \pmod{p_i}$ implies $\mathbf{u} \cdot [\tilde{\mathbf{D}}]_{p_i} = \mathbf{0}$ over $\mathbb{Z}$. Therefore, by defining $\mathbf{E} \in \mathbb{Z}^{2nm \times nm}$ as a block matrix such that each "column" is given by $\mathbf{D} \bmod p_i$, i.e.,

$$\mathbf{E} := \begin{bmatrix} \mathbf{S}_1 \cdot \mathbf{R}_{1,1} & \dots & \mathbf{S}_n \cdot \mathbf{R}_{n,1} \\ \mathbf{S}_1 \cdot \mathbf{R}_{1,2} & \dots & \mathbf{S}_n \cdot \mathbf{R}_{n,2} \end{bmatrix},$$

we see that L contains a sublattice $L^\perp$ of vectors orthogonal to the columns of $\mathbf{E}$. It holds

that $\mathsf{rank}(L^\perp) = 2nm - nm = nm$. Thus, we run a lattice basis reduction of L to recover short vectors $\mathbf{v}_1, ..., \mathbf{v}_{nm}$ that form a basis of $L^\perp$. Then, each column of $\mathbf{E}$ belongs to $\bar{L} := (L^\perp)^\perp$, hence, by computing a basis $\mathbf{B}$ of $\bar{L}$, we see that $\mathbf{E} = \mathbf{B} \cdot \mathbf{A}$ for some matrix $\mathbf{A} \in \mathbb{Z}^{nm \times nm}$.

By defining the $nm \times nm$ matrices $\mathbf{S} := [\,\mathbf{S}_1 ... \mathbf{S}_n\,]$, $\mathbf{R}_1 := \mathsf{diag}(\mathbf{R}_{1,1}, ..., \mathbf{R}_{n,1})$, and $\mathbf{R}_2 := \mathsf{diag}(\mathbf{R}_{1,2}, ..., \mathbf{R}_{n,2})$ we have then

$$\mathbf{B} = \mathbf{E} \cdot \mathbf{A}^{-1} = \begin{bmatrix} \mathbf{S} \cdot \mathbf{R}_1 \cdot \mathbf{A}^{-1} \\ \mathbf{S} \cdot \mathbf{R}_2 \cdot \mathbf{A}^{-1} \end{bmatrix}$$

which means that, from $\mathbf{B}$, we can obtain two blocks $\mathbf{B}_i = \mathbf{S} \cdot \mathbf{R}_i \cdot \mathbf{A}^{-1}$ and compute

$$\mathbf{W} = \mathbf{B}_1 \cdot \mathbf{B}_2^{-1} = \mathbf{S} \cdot \mathbf{R}_1 \cdot \mathbf{R}_2^{-1} \cdot \mathbf{S}^{-1}.$$

The characteristic polynomial f of $\mathbf{W}$ is equal to the one of $\mathbf{R}_1 \cdot \mathbf{R}_2^{-1}$. But because $\mathbf{R}_1 \cdot \mathbf{R}_2^{-1} = \mathsf{diag}(\mathbf{R}_{1,1} \cdot \mathbf{R}_{1,2}^{-1}, ..., \mathbf{R}_{n,1} \cdot \mathbf{R}_{n,2}^{-1})$, f factors in n characteristic polynomials f_i's corresponding to each block $\mathbf{R}_{i,1} \cdot \mathbf{R}_{i,2}^{-1}$. By the Cayley-Hamilton theorem, we have $f_i(\mathbf{R}_{i,1} \cdot \mathbf{R}_{i,2}^{-1}) = \mathbf{0}$, which implies $f_i(\tilde{\mathbf{C}}_1 \cdot \tilde{\mathbf{C}}_2^{-1} \pmod{x_0}) = \mathbf{0} \pmod{p_i}$. Therefore, if all the f_i's are irreducible, we can obtain them by factoring f, then we can recover each p_i by computing GCDs of the entries of $f_i(\tilde{\mathbf{C}}_1 \cdot \tilde{\mathbf{C}}_2^{-1} \pmod{x_0})$. If some f_i is not irreducible, then factoring f will produce irreducible factors $f'_1, ..., f'_N \in \mathbb{Q}[x]$ for some $N \in \mathbb{N}$. Then, for each $k \in [\![1, N]\!]$, we can define $F_k := f/f'_k \in \mathbb{Q}[x]$ and scale it to the integers, i.e., $G_k := d_k \cdot F_k \in \mathbb{Z}[x]$ where d_k is the common denominator of the coefficients of F_k. Because G_k corresponds to f without some factors, we have that $G_k(\tilde{\mathbf{C}}_1 \cdot \tilde{\mathbf{C}}_2^{-1} \pmod{x_0}) \neq \mathbf{0}$ at least for one prime and it equals $\mathbf{0}$ for the others, therefore, we can still recover the secret primes by computing the GCD of the entries of $G_k(\tilde{\mathbf{C}}_1 \cdot \tilde{\mathbf{C}}_2^{-1} \pmod{x_0})$ and x_0.

We expect that the norm of the shortest vectors of $L^\perp$ to be approximately

$$\lambda_i = \left(\prod_{j=1}^{nm} \|\mathsf{col}_j(\mathbf{E})\|_2 \right)^{1/\mathsf{rank}(L^\perp)} \approx \left(\prod_{j=1}^{nm} 2^{2\rho} \right)^{1/(nm)} = 2^{2\rho}$$

and the norm of the vectors $\mathbf{v}_1, ..., \mathbf{v}_{nm}$ recovered with a lattice basis reduction to be approximately $2^{\alpha \mathsf{rank}(L)} \cdot \lambda_i = 2^{2\alpha nm + 2\rho}$, where 2^α is the root Hermite factor. Finally, we

need these recovered vectors to be small enough for their inner products with the columns of $\mathbf{E}$ to be smaller than all p_i's, thus, we have the condition

$$2^{2\alpha nm+2\rho} < 2^{\eta-2\rho}, \tag{3.2}$$

which implies $\alpha < \eta/(2nm)$. Therefore, this orthogonal lattice attack has time complexity $2^{\Omega(1/\alpha)} = 2^{\Omega(nm/\eta)} = 2^{\Omega(\gamma m/\eta^2)}$, where the last equality is derived from $\gamma = n\eta$.

3.6.2 GCD attack on the multi-prime VAGCD problem

In this section, we show how our GCD attack presented in Section 3.5.2 can be adapted to the multi-prime scenario. Consider distinct η-bit primes p_1, ..., p_n. Each sample $\tilde{\mathbf{c}}_i$ now has n noise terms $\mathbf{r}_{i,\ell} := \tilde{\mathbf{c}}_i \bmod p_\ell$, thus, we say that a pair of samples $\tilde{\mathbf{c}}_i$ and $\tilde{\mathbf{c}}_j$ collide if at least one of their corresponding noise terms are equal, that is, if $\tilde{\mathbf{c}}_i = \tilde{\mathbf{c}}_j \pmod{p_\ell}$ at least for one $\ell \in [\![1, n]\!]$. For a fixed prime p_ℓ, the probability that $\tilde{\mathbf{c}}_i = \tilde{\mathbf{c}}_j \pmod{p_\ell}$ is about $2^{-m\rho}$, hence, the probability that two samples form a colliding pair is about $n \cdot 2^{-m\rho}$. Therefore, given two lists of samples L_1 and L_2, the probability that there exist a colliding pair $(\tilde{\mathbf{c}}_i, \tilde{\mathbf{c}}_j) \in L_1 \times L_2$ is about $|L_1| \cdot |L_2| \cdot n \cdot 2^{-m\rho}$, which is close to one if $|L_1| \approx |L_2| \approx 2^{-m\rho/2}/\sqrt{n}$.

Therefore, to reveal one of the n primes, we can apply the attack described in Algorithm 1, but using lists of length $2^{-m\rho/2}/\sqrt{n}$ instead of $2^{-m\rho/2}$. The running time and the memory usage of the attack is then divided by $\sqrt{n}$. Moreover, a scalar x_0 can be used in the same way as described in Section 3.5.2. Therefore, we have the following: if a scalar x_0 is available, we the cost of this attack is

$$T_{mp\_GCD,x_0}(\eta, \rho, \rho_0, \gamma, m, n) := \frac{(m\rho)^2}{\sqrt{n}} \cdot 2^{\rho_0 + m\rho/2} \cdot \gamma \log \gamma,$$

where $\rho_0 = 0$ if x_0 is noiseless; and if the attacker has no scalar x_0, then

$$T_{mp\_GCD}(\eta, \rho, \gamma, m, n) := \frac{(m\rho)^2}{\sqrt{n}} \cdot 2^{m\rho} \cdot \gamma \log \gamma.$$

3.7 Conclusion

In this section, we recapitulated the main known attacks against the AGCD problem. In particular, we noticed that the proposed orthogonal-lattice algorithms to solve the multi-prime AGCD problem were actually incomplete, since they only work when n, the number of primes, is small. Then, we proposed a lattice attack that works for any value of n. We also proposed a new variant of the AGCD problem, called VAGCD, in which a sample is an m-dimensional vector instead of an integer. We showed that the VAGCD problem cannot be easier than the original AGCD problem and that it actually seems to be strictly harder. Namely, we adapted the main attacks on the AGCD to the VAGCD problem and concluded that, for the same security level, it is possible to choose smaller parameters for the VAGCD problem, basically by dividing the size of the parameters by m. Moreover, we provided theoretical evidence to support this growth in the cost of the attacks when adapted to the VAGCD problem.

Chapter 4

Multilinear maps, obfuscation and key exchange

In this chapter, the cryptographic primitives known as multilinear maps and indistinguishability obfuscation are formally introduced, then, N-party One-round key-exchange (NOKE) protocols based on multilinear maps are discussed. Finally, we introduce a new NOKE protocol based on a multilinear map proposed by Coron, Lepoint, and Tibouchi [CLT13] and show how the cryptanalysis of the randomized AGCD problem developed in Chapter 3 can be used to make this protocol more efficient.

4.1 Introduction

A (cryptographic) multilinear map is a generalization of the well-known bilinear maps. The first candidates were presented in the papers [GGH13a], [CLT13], and [GGH15]. In principle the most straightforward application of multilinear maps is non-interactive multipartite Diffie-Hellman (DH) key exchange with N users, a natural generalization of the DH protocol for 3 users based on the bilinear pairing. This was originally described for GGH13, CLT13 and GGH15, but it was quickly broken for the three families of multilinear maps. In particular, the key-exchange protocol constructed on top of CLT13 was broken by an attack presented by Cheon *et al.* [CHL+15]. The main question is therefore:

Can we construct a practical N-way non-interactive key-exchange protocol from candidate multilinear maps constructions?

In this chapter we provide a first step in that direction by describing the first implemen-

tation of N-way DH key exchange resistant against all known attacks. Our construction is based on CLT13 multilinear maps and contains many ingredients from the [GGH+13b] and other similar constructions. Moreover, it is secure against the Cheon *et al.* attack and its variants.

In more detail, each party u chooses a secret session key $\mathsf{sk}^{(u)}$, then the shared key derived at the end of the protocol is basically the evaluation of a branching program B at the concatenation of each secret key, that is, $B(\mathsf{sk}^{(1)}||\ldots||\mathsf{sk}^{(N)}) \in \mathbb{Z}^{m\times m}$. The branching program is randomized and encoded as in the indistinguishability obfuscator proposed in [GGH+13b], but using the CLT13 multilinear map. Then, Kilian's randomization is used on the encoding side, so that we end up with a variant of the vector AGCD problem discussed in Chapter 3. At last, in order to prevent the Cheon *et al.* attack against CLT13, when considering input partitioning attacks as in [CGH+15], and its extension with the tensoring attack [CLLT17], the branching program B is defined as the product of k subprograms B_1, ..., B_k that read the same input bit in the same step, thus, $B(\mathsf{sk}^{(1)}||\ldots||\mathsf{sk}^{(N)}) = B_1(\mathsf{sk}^{(1)}||\ldots||\mathsf{sk}^{(N)}) \cdot \ldots \cdot B_k(\mathsf{sk}^{(1)}||\ldots||\mathsf{sk}^{(N)})$, hence, the input bits are repeated k times during the evaluation of B. With such branching program B, we argue that the extended Cheon *et al.* attack has complexity $\Omega(m^{2k-1})$ on our scheme, where m is the matrix dimension and k the number of repetitions.

In Section 4.5.3, we also present a generalization of the straddling set systems from [BGK+14] that can be used in our protocol, with no additional cost, to further constrain the attacker.

For $N = 4$ users and a medium (62 bits) level of security, our implementation requires 18 GB of public parameters, and a few minutes for the derivation of a shared key. We note that without Kilian's randomization of encodings our construction would be completely unpractical, as it would require more than 100 TB of public parameters.

4.2 Cryptographic multilinear maps

In [BS03], Boneh and Silverberg formally analyzed generalizations of bilinear maps, called multilinear maps, showing several applications and also discussing some difficulties that would be encountered when trying to construct such maps. Fixing "source" groups $G_1, ..., G_n$ and a "target" group G_T, a cryptographic multilinear map is defined as a map $e : G_1 \times \cdots \times G_n \to G_T$, such that, for $a_1, \ldots, a_n \in \mathbb{Z}$ and $(g_1, ..., g_n) \in \prod_{i=1}^{n} G_n$, the following three properties hold:

- (Multilinearity) $e(g_1^{a_1}, \ldots, g_n^{a_n}) = e(g_1, \ldots, g_n)^{\prod_{i=1}^{n} a_i}$;
- (Non-degeneracy) If $g_1, ..., g_n$ are generators of $G_1, ..., G_n$, respectively, then G_T is generated by $e(g_1, ..., g_n)$;
- (Computability) The map e is efficiently computable.

Moreover, in order for the map to be useful in cryptographic applications, we require the discrete logarithm problem (DL) in the source groups to be hard. Remember that in the DL problem, one is given an element $h := g^{\alpha}$ and has to recover α. Now, notice that the DL problem in any source group G_i reduces to the DL problem in the target group G_T. Namely, given $h_i := g_i^{\alpha}$, by computing $x := e(g_1, ..., g_n) \in G_T$ and $h = e(g_1, ..., g_{i-1}, h, g_{i+1}, g_n) \in G_T$, we see that $h = x^{\alpha}$. Thus, solving the DL problem in G_T is sufficient to solve this problem in any source group.

Despite the variety of important cryptographic protocols and schemes that can be constructed with multilinear maps, as one-round n-party key exchange and efficient broadcast encryption [BS03], it was only in 2013 that the first candidate multilinear map was proposed [GGH13a]. Actually, this candidate, which is called a *graded encoding system*, is only an approximation of the multilinear maps defined in [BS03], but they are somehow functionally equivalent. In [BS03], one can view a group element g^a as an encoding of an integer a. The encoded plaintexts can be added (modulo the order of the group) via multiplication of encodings, since $g^a \cdot g^b = g^{a+b}$. And, at some point, one can apply the map e to combine all the encodings into a single element of the target group. In [GGH13a], rings

are used instead of groups, therefore, one can add and multiply the plaintexts; an integer a is encoded into a random element instead of a fixed element g^a; and more importantly, the map e can be partially applied generating encodings in intermediary "target" rings. Namely, there are levels associated with the encodings. A plaintext is encoded at level one; encodings at the same level can be added, generating an encoding at the same level, but multiplying two encodings generates an encoding in a higher level and the target ring defines the highest level supported by the multilinear map, thus, multiplication is equivalent to partially applying the map e. Notice that because encodings of the form g^a are deterministic, one can test if two encodings g^a and g^b encode the same value by checking whether $g^a = g^b$. However, in [GGH13a], because the encodings are randomized, one cannot simply perform an equality test, thus, there is a special procedure, called *zero testing*, which checks if a given value encodes a zero. With this, one can test equalities by first subtracting two encodings, then zero testing the result. The zero-testing procedure uses an additional public value, called zero-testing parameter, denoted p_{zt}, and that is computed using private parameters.

Short after the publication of [GGH13a], other two candidate multilinear maps were proposed: one over the integers, based on the partial multi-prime AGCD problem [CLT13], and a graph-induced one, based on lattice problems [GGH15]. Because our key-exchange protocol proposed in this chapter uses the CLT13, we present it in detail now. Notice that understanding how the CLT13 works is enough for having a good idea of how the other candidate multilinear maps work, since their main structures are similar.

- *Instance generation:* Let k be the degree of multilinearity. For all $i \in [\![1, n]\!]$, sample an η-bit prime p_i and an α-bit prime g_i. Define $x_0 := \prod_{i=1}^{n} p_i$, sample $z \leftarrow \mathcal{U}(\mathbb{Z}_{x_0})$ (with overwhelming probability, z is invertible modulo x_0). Compute the zero-testing parameter as

$$p_{zt} := \mathsf{CRT}_{(p_i)}\left(h_i \cdot \frac{x_0}{p_i} \cdot \left(z^k [g_i^{-1}]_{p_i}\right)\right) = \sum_{i=1}^{n} h_i \cdot \frac{x_0}{p_i} \cdot \left(z^k [g_i^{-1}]_{p_i}\right) \mod x_0,$$

  where h_i's are random β-bit integers. Notice that the inverse of g_i is computed modulo p_i. Finally, let s be a seed for a strong randomness extractor and ν be the number of

bits that extracted by the zero-testing procedure. Output the parameters $\mathsf{params} := (n, \eta, \alpha, \rho, \beta, \nu, s, x_0, p_{zt})$.

- *Private encode:* To privately encode n integers $(m_1, ..., m_n) \in \prod_{i=1}^{n} \mathbb{Z}_{g_i}$ in level $\ell \leq k$, sample noise terms $r_i \leftarrow]\!] - 2^\rho, 2^\rho [\![$ and output

$$c := \mathsf{CRT}_{(p_i)} \left(z^{-\ell} \cdot (r_i \cdot g_i + m_i) \right).$$

- *Adding and multiplying encodings:* Because the CRT acts componentwise, it is easy to see that adding two encodings at level ℓ yields an encoding of the sum of plaintexts at the same level ℓ, since

$$c_{add} := [c_1 + c_2]_{x_0} = \mathsf{CRT}_{(p_i)} \left(z^{-\ell} \cdot ((r_{i,1} + r_{i,2}) \cdot g_i + m_{i,1} + m_{i,2}) \right).$$

To see how the multiplication works, notice that

$$z^{-\ell_1}(r_{i,1} \cdot g_i + m_{i,1}) \cdot z^{-\ell_2}(r_{i,2} \cdot g_i + m_{i,2}) = z^{-(\ell_1+\ell_2)} \cdot (r_i \cdot g_i + m_{i,1} \cdot m_{i,2})$$

where $r_i := r_{i,1} g_i r_{i,2} + r_{i,1} m_{i,2} + m_{i,1} r_{i,2}$, therefore, considering that c_1 and c_2 are encodings at level ℓ_1 and ℓ_2, we have

$$c_{mult} := [c_1 \cdot c_2]_{x_0} = \mathsf{CRT}_{(p_i)} \left(z^{-(\ell_1+\ell_2)} \cdot (r_i \cdot g_i + m_{i,1} \cdot m_{i,2}) \right).$$

Therefore, c_{mult} is an encoding of the product of the messages in the n slots. Moreover, the level of c_{mult} is the sum of the levels of the operands. Notice also that the multiplication increases the noise quadratically, from $|r_{i,1}|$ and $|r_{i,2}|$ to $|r_i| \approx |r_{i,1}| \cdot |r_{i,2}| \cdot 2^\alpha$.

- *Zero-testing procedure:* To verify if a given top-level encoding c encodes a zero, multiply c by the zero-testing parameter and check if the result is small. Namely, compute $\omega := p_{zt} \cdot c \mod x_0$ and return *true* if $|\omega| < x_0 \cdot 2^{-\nu}$. Otherwise, return *false*. Notice that by defining $\hat{p}_i := x_0/p_i$, we have

$$\omega = \mathsf{CRT}_{(p_i)} \left(h_i \cdot \hat{p}_i \cdot [g_i^{-1}]_{p_i} (r_i \cdot g_i + m_i) \right).$$

Thus, if $(m_1, ..., m_n) = (0, ..., 0)$, the following holds over the integers

$$\omega = \sum_{i=1}^{n} h_i \cdot \hat{p}_i \cdot r_i.$$

Therefore, because $|h_i| < 2^{\beta}$, considering that the final noise is bounded by 2^{ρ_f}, we have $|\omega| \leq \sum_{i=1}^{n} |h_i \hat{p}_i r_i| < n2^{\beta} \cdot x_0 \cdot 2^{-\eta+1} \cdot 2^{\rho_f} = x_0 \cdot 2^{\beta+\rho_f+\log n+1-\eta}$. Since we want $|\omega|$ be smaller than $x_0 2^{-\nu}$, it is sufficient to choose η sufficiently larger than $\beta + \rho_f + \log n + \nu$ for the zero-testing to be correct.

- *Extraction:* Compute $\omega := p_{zt} \cdot c \mod x_0$, extract the ν most significant bits of ω, and apply a strong randomness generator with seed s. Notice that for two level-k encodings c_1 and c_2 encoding the same message, we have essentially $p_{zt}(c_1 - c_2) \leq x_0 \cdot 2^{-\nu}$, thus, $\omega_1 := p_{zt} \cdot c_1$ and $\omega_2 := p_{zt} \cdot c_2$ agree in their ν most significant bits, thus, the random extractor outputs the same random value. Conversely, if c_1 and c_2 encode different messages, we expect their ν most significant bits to differ by at least one bit, thus, the random values extracted are different.

Notice that in the way we presented the CLT13 multilinear map, one needs the secret primes to encode a plaintext. To overcome this limitation, the parameters must include encodings of zero and of one, which can be publicly combined (added and multiplied) to generate encodings of any desired value, as it is done in the original formulation of this multilinear map in [CLT13].

4.3 Indistinguishability obfuscation

Obfuscating a program aims to hide its implementation while maintaining its functionalities, in other words, we intuitively expect that an obfuscator receive as input a program P and output an equivalent program $\mathcal{O}(P)$ such that, when analyzing $\mathcal{O}(P)$, one cannot learn the code of P. Defining equivalent programs formally is easy: we can simply say that two programs are equivalent if they always output the same result when given the same input and if they have the same size in asymptotic terms. However, defining pre-

cisely what is meant by the "the obfuscated program must hide the implementation of the original program" is tricky.

In [BGI+01], this hiding notion is formalized as virtual black-box (VBB) obfuscation, which basically means that anything that can be efficiently computed from an obfuscated program $\mathcal{O}(P)$ can also be efficiently computed when $\mathcal{O}(P)$ is replaced by oracle access to P. In other words, $\mathcal{O}(P)$ reveals nothing more than the pairs of input and output $(x, P(x))$. However, still in [BGI+01], the authors prove that this strong notion of obfuscability cannot be achieved. In more detail, it is proved that the existence of a VBB obfuscator that runs in polynomial time in the size of the given circuit implies the existence of one-way functions, but it is also proved that if one-way functions exist, then, no VBB obfuscator exist (not even exponential-time obfuscators). Therefore, no efficient (polynomial-time) VBB obfuscator exist. They also prove that VBB obfuscation cannot be achieved even for some particular families of functions, e.g., there are some private-key encryption schemes and signature schemes for which no VBB obfuscation exists. In view of these impossibility results, a weaker notion of obfuscation, called indistinguishability obfuscation (iO), is proposed. In simple terms, iO is weaker than VBB obfuscation because it does not intend to completely hide the program, but just to guarantee that all the obfuscated versions of equivalent programs are indistinguishable. We now present a formal definition [GGH+13b]:

Definition 4.3.1. *(iO) A uniform PPT machine iO called an* indistinguishability obfuscator *for a circuit class C_λ if the following conditions are satisfied:*

- *(Functional equivalency) For all security parameters $\lambda \in \mathbb{N}$, for all $C \in C_\lambda$, for all inputs x, we have that*

$$\Pr[C'(x) = C(x) : C' \leftarrow iO(\lambda, C)] = 1$$

- *(Indistinguishability) For any (not necessarily uniform) PPT distinguisher D, there exists a negligible function α such that the following holds: For all security parameters $\lambda \in \mathbb{N}$, for all pairs of circuits $C_0, C_1 \in C_\lambda$, we have that if $C_0(x) = C_1(x)$ for*

all inputs x, then

$$|\Pr[D(iO(\lambda, C_0)) = 1] - \Pr[D(iO(\lambda, C_1)) = 1]| \leq \alpha(\lambda)$$

Although the definition of iO deals with circuits, the iO candidates obfuscate branching programs. Then, in order to obfuscate a circuit C, one first transforms C in a branching program P, for instance, using Barrington's theorem [Bar89], then obfuscate P. A branching program is composed by two sequences of matrices $(\mathbf{A}_{i,0})_{i=1}^{n}$ and $(\mathbf{A}_{i,1})_{i=1}^{n}$ and a function inp. The evaluation of the branching program on input $x \in \{0,1\}^{\ell}$ is done in n steps as follows: we start with $\mathbf{R}_0 := \mathbf{I}$ and at each step i, we take the input bit $b_i := x[\mathsf{inp}(i)]$ and compute $\mathbf{R}_i := \mathbf{R}_{i-1} \cdot \mathbf{A}_{i,b_i}$, that is, we use the function inp to decide which input bit will be used, then, we use this bit to select one of the two matrices corresponding to the i-th step and multiply it by the current evaluation. Finally, if $\mathbf{R}_n = \mathbf{I}$, we output 0. Otherwise, we output 1. Formally, a branching program is defined as follows:

Definition 4.3.2. *(Oblivious linear branching program) A width-m length-n branching program for ℓ-bit inputs is a tuple*

$$B := (\mathsf{inp}(i), \mathbf{A}_{i,0}, \mathbf{A}_{i,1})_{i=1}^{n}$$

where inp is a function from $[\![1, n]\!]$ to $[\![1, \ell]\!]$ and each $\mathbf{A}_{i,b}$ is an $m \times m$ matrix.

The function computed by B is

$$f_B(x) := \begin{cases} 0 & \text{if } \prod_{i=1}^{n} \mathbf{A}_{i,x[\mathsf{inp}(i)]} = \mathbf{I} \\ 1 & \text{otherwise} \end{cases}$$

The basic structure of the iO scheme presented in [GGH+13b] is the following: Each $m \times m$ matrix $\mathbf{A}_{i,b}$ of a branching program B is embedded in a higher dimensional matrix with random elements in the diagonal

$$\hat{\mathbf{A}}_{i,b} \sim \begin{pmatrix} \$ & & & \\ & \ddots & & \\ & & \$ & \\ & & & \alpha_{i,b} \cdot \mathbf{A}_{i,b} \end{pmatrix}$$

where the values $\alpha_{i,b}$'s are called bundling scalars. Then, we sample random invertible matrices $\mathbf{K}_0, ..., \mathbf{K}_n$, define $\bar{\mathbf{A}}_{i,b} := \mathbf{K}_{i-1} \cdot \hat{\mathbf{A}}_{i,b} \cdot \mathbf{K}_i^{-1}$, and encode each $\bar{\mathbf{A}}_{i,b}$ entrywise, using a multilinear map, obtaining $\tilde{\mathbf{A}}_{i,b}$. We also sample random d-dimensional vectors $\mathbf{s}$ and $\mathbf{t}$ and embed them into vectors $\hat{\mathbf{s}} := (\$ \ ... \ \$ \ 0 \ ... \ 0 \ \mathbf{s})$ and $\hat{\mathbf{t}} := (0 \ ... \ 0 \ \$ \ ... \ \$ \ \mathbf{s})$. Notice that multiplying any matrix $\hat{\mathbf{A}}_{i,b}$ by $\hat{\mathbf{s}}$ and $\hat{\mathbf{t}}$ removes the additional random diagonal values, that is, $\hat{\mathbf{s}} \cdot \hat{\mathbf{A}}_{i,b} \cdot \hat{\mathbf{t}} = \alpha_{i,b} \cdot \mathbf{s} \cdot \mathbf{B}_{i,b} \cdot \mathbf{t}$. Then, we also randomize them, obtaining $\bar{\mathbf{s}} := \hat{\mathbf{s}} \cdot \mathbf{K}_0^{-1}$ and $\bar{\mathbf{t}} := \mathbf{K}_n \cdot \hat{\mathbf{t}}$. Finally, we also encode these two vectors entrywise using a multilinear map, obtaining $\tilde{\mathbf{s}}$ and $\tilde{\mathbf{t}}$.

To evaluate the obfuscated program $iO(B)$, we include an obfuscation of a branching program B' that always evaluate to the identity matrix, thus, the difference between the evaluation of the original branching program and that of B', that is, $\prod_{i=1}^{n} \mathbf{A}_{i,\mathsf{inp}(x)} - \prod_{i=1}^{n} \mathbf{A}'_{i,\mathsf{inp}(x)}$ is zero if, and only if, the function computed by B is zero at x. Moreover,

$$\mathbf{s} \cdot \left(\prod_{i=1}^{n} \mathbf{A}_{i,\mathsf{inp}(x)} - \prod_{i=1}^{n} \mathbf{A}'_{i,\mathsf{inp}(x)} \right) \cdot \mathbf{t} = 0 \iff B(x) = 0$$

Because both B and B' are actually encoded, what we obtain at the end is a top-level encoding of $\mathbf{s} \cdot (B(x) - B'(x)) \cdot \mathbf{t}$, hence, we use the zero-testing procedure of the multilinear map to decide if the evaluation of the branching program is zero or one. Therefore, to complete the obfuscation of B, we create an auxiliary branching program B' of same length as B, but with all the matrices $\mathbf{A}'_{i,b}$'s being $m \times m$ identity matrices. As for the matrices of B, we embed each $\mathbf{A}'_{i,b}$ into higher dimensional matrices, multiply them by $\mathbf{K}_{i-1}$ and $\mathbf{K}_i^{-1}$, and encode them entrywise, obtaining $\tilde{\mathbf{A}}'_{i,b}$. We also create bookend vectors $\hat{\mathbf{s}'} := (\$ \ ... \ \$ \ 0 \ ... \ 0 \ \mathbf{s})$ and $\hat{\mathbf{t}'} := (0 \ ... \ 0 \ \$ \ ... \ \$ \ \mathbf{s})$, multiply them by $\mathbf{K}_0^{-1}$ and $\mathbf{K}_n$, and encode them, obtaining $\tilde{\mathbf{s}}'$ and $\tilde{\mathbf{t}}'$. The obfuscation of B is then defined as

$$iO(B) = \begin{Bmatrix} \tilde{\mathbf{s}}, & \tilde{\mathbf{t}}, & \tilde{\mathbf{A}}_{1,0}, \ ..., \tilde{\mathbf{A}}_{n,0}, & \tilde{\mathbf{A}}_{1,1}, \ ..., \tilde{\mathbf{A}}_{n,1} \\ \tilde{\mathbf{s}}', & \tilde{\mathbf{t}}', & \tilde{\mathbf{A}}'_{1,0}, \ ..., \tilde{\mathbf{A}}'_{n,0}, & \tilde{\mathbf{A}}'_{1,1}, \ ..., \tilde{\mathbf{A}}'_{n,1} \end{Bmatrix}.$$

Hence, to evaluate $iO(B)$ on input x, we compute

$$\tilde{\omega} := \tilde{\mathbf{s}} \cdot \left(\prod_{i=1}^{n} \tilde{\mathbf{A}}_{i,\mathsf{inp}(i)} \right) \cdot \tilde{\mathbf{t}} - \tilde{\mathbf{s}}' \cdot \left(\prod_{i=1}^{n} \tilde{\mathbf{A}}'_{i,\mathsf{inp}(i)} \right) \cdot \tilde{\mathbf{t}}'$$

and use the zero-testing procedure of the multilinear map, outputting $\mathsf{encodesZero?}(\tilde{\omega})$.

Notice that all the random matrices $\mathbf{K}_i$'s and their inverses are canceled out. Furthermore, the extra diagonal elements used when the initial matrices were embedded into higher dimensional matrices are also eliminated. Thus, $\tilde{\omega}$ is an encoding of

$$\omega := \alpha \cdot \mathbf{s} \cdot \left(\prod_{i=1}^{n} \mathbf{A}_{i,\mathsf{inp}(i)} \right) \cdot \mathbf{t} - \alpha' \cdot \mathbf{s} \cdot \mathbf{I} \cdot \mathbf{t},$$

where $\alpha := \prod_{i=1}^{n} \alpha_{i,\mathsf{inp}(i)}$ and $\alpha' := \prod_{i=1}^{n} \alpha'_{i,\mathsf{inp}(i)}$. By the way that the scalars $\alpha_{i,b}$'s are chosen, it holds that $\omega = 0 \iff \prod_{i=1}^{n} \mathbf{A}_{i,\mathsf{inp}(i)} = \mathbf{I}$, therefore, $\tilde{\omega}$ encodes a zero if, and only if, the evaluation of B at x is zero. Thus, by outputting the result of the zero-testing procedure on $\tilde{\omega}$, we have that $iO(B)(x) = B(x)$.

Notice that the iO scheme proposed in [GGH+13b] can be instantiated with any multilinear map. In particular, when instantiated with the CLT13, there are attacks that can exploit input partitions on the branching program, thus, in our key-exchange protocol, presented in Section 4.5, we use a special branching program whose input is repeated several times. Furthermore, we use Kilian randomization in the encoding side, that is, we first encode the matrices, then we multiply them by $\mathbf{K}_{i-1}$ and $\mathbf{K}_i^{-1}$. By doing so, we can base the security of the protocol in the VAGCD problem instead of the original AGCD problem, thus, we are able to select smaller parameters, as discussed in Chapter 3.

4.4 N-party one-round key-exchange protocol

With a multipartite key-exchange protocol, N users have a common preestablished setting (e.g. a group from which they will sample random values) and they exchange messages via a public channel in order to agree on a common value that is shared by all the N users but is unknown to any other party. In addition to that, in a one-round key-exchange protocol, each user just need to send their messages one time before deriving the secret shared value, i.e., there is only one round of communication in which each user broadcasts their values to all the other users.

Following the notation of [BS03], such protocol can be described with three randomized probabilistic polynomial-time algorithms as follows.

- $\mathsf{Setup}(1^\lambda, N)$: This algorithm runs in polynomial time in the security parameter $\lambda \in \mathbb{N}$ and in the number of parties N, and outputs the public parameters params.

- $\mathsf{Publish}(\mathsf{params}, u)$: Given a party $u \in [\![1, N]\!]$, this algorithm generates a pair of keys $(\mathsf{sk}^{(u)}, \mathsf{pk}^{(u)})$ for party u. Then u broadcasts $\mathsf{pk}^{(u)}$ and keeps $\mathsf{sk}^{(u)}$ secret.

- $\mathsf{KeyGen}(\mathsf{params}, v, \mathsf{sk}^{(v)}, \{\mathsf{pk}^{(u)}\}_{u \neq v})$: Party v uses its secret $\mathsf{sk}^{(v)}$ and all the values $\mathsf{pk}^{(u)}$'s broadcasted by other parties to generate a session key $s^{(v)}$.

The protocol is said to be correct if $s^{(1)} = s^{(2)} = \cdots = s^{(N)}$, i.e., if all the parties share the same value at the end. The protocol is secure if no probabilistic polynomial-time adversary can distinguish the shared value from a random string given the public parameters params and the broadcasted values $\mathsf{pk}^{(1)}, \ldots, \mathsf{pk}^{(N)}$.

4.5 Our construction

In this section, we describe our N-party one-round key-exchange protocol. We start with the Setup procedure, which is run a single time by a trusted party to generate the public parameters. Setup samples a branching program $B := \{\mathbf{B}_{i,b} \in \mathbb{Z}^{m \times m} : b \in \{0,1\} \text{ and } 1 \leq i \leq \ell\}$, then, B is encoded and randomized N times, generating for each party v two sequences of matrices $(\mathbf{C}_{i,b}^{(v)})_{i=1,\ldots,\ell}$ for $b \in \{0,1\}$, as illustrated in Table 4.1. Then, in the $\mathsf{Publish}$ procedure, each party u samples a secret string $\mathsf{sk}^{(u)} \in \{0,1\}^\mu$, where $\mu \in \mathbb{N}^*$ is a parameter, and compute partial evaluations of the obfuscated branching programs corresponding to the rows of the other parties (the matrices of rows v of Table 4.1 for $v \neq u$) using as input their own secret $\mathsf{sk}^{(u)}$.

In more detail, we have $\ell := \mu N k$, and B has a special structure, as its input is supposed to be the following concatenation of all the secret strings:

$$\mathsf{sk} := (\underbrace{\mathsf{sk}^{(1)}, \ldots, \mathsf{sk}^{(N)}}_{\text{First repetition}}, \underbrace{\mathsf{sk}^{(1)}, \ldots, \mathsf{sk}^{(N)}}_{\text{Second repetition}}, \ldots, \underbrace{\mathsf{sk}^{(1)}, \ldots, \mathsf{sk}^{(N)}}_{k\text{-th repetition}}) \in \{0,1\}^\ell. \tag{4.1}$$

Table 4.1: Public matrices for $N = 3$ generated during the Setup procedure.

Party 1	$\mathbf{C}^{(1)}_{1,0}$ $\mathbf{C}^{(1)}_{1,1}$	$\mathbf{C}^{(1)}_{2,0}$ $\mathbf{C}^{(1)}_{2,1}$	$\dots$ $\dots$	$\mathbf{C}^{(1)}_{\ell,0}$ $\mathbf{C}^{(1)}_{\ell,1}$
Party 2	$\mathbf{C}^{(2)}_{1,0}$ $\mathbf{C}^{(2)}_{1,1}$	$\mathbf{C}^{(2)}_{2,0}$ $\mathbf{C}^{(2)}_{2,1}$	$\dots$ $\dots$	$\mathbf{C}^{(2)}_{\ell,0}$ $\mathbf{C}^{(2)}_{\ell,1}$
Party 3	$\mathbf{C}^{(3)}_{1,0}$ $\mathbf{C}^{(3)}_{1,1}$	$\mathbf{C}^{(3)}_{2,0}$ $\mathbf{C}^{(3)}_{2,1}$	$\dots$ $\dots$	$\mathbf{C}^{(3)}_{\ell,0}$ $\mathbf{C}^{(3)}_{\ell,1}$

Thus, in the Publish procedure, each party u evaluates the program B on its own secret $\mathsf{sk}^{(u)}$ in each of the k repetitions. But because B is obfuscated, u has to perform this evaluation for each other party v using the matrices $\mathbf{C}^{(v)}_{i,b}$'s corresponding to party v.

To simplify the notation, we define the following function that will be used as the index to access the matrices of Table 4.1:

$$\forall (i,u,r) \in [\![1,\mu]\!] \times [\![1,N]\!] \times [\![1,k]\!],\ \psi(i,u,r) := i + (u-1)\mu + (r-1)N\mu. \tag{4.2}$$

One can read $\psi(i,u,r)$ as "the index of the i-th matrix of party u in the r-th repetition". Thus, for example, to compute the partial evaluation of B with input $\mathsf{sk}^{(u)}$ in the first repetition, party u would compute $\prod_{i=1}^{\mu} \mathbf{B}_{\psi(i,u,1),b_i}$, where $b_i := \mathsf{sk}^{(u)}[i]$ is the i-th bit of $\mathsf{sk}^{(u)}$, but because party u actually has only access to the obfuscated programs in Table 4.1, they would instead compute $\prod_{i=1}^{\mu} \mathbf{C}^{(v)}_{\psi(i,u,1),b_i}$ for all $v \neq u$.

In the KeyGen procedure, each party v will generate the shared key using the products $\prod_{i=1}^{\mu} \mathbf{C}^{(v)}_{\psi(i,u,r),\mathsf{sk}^{(u)}[i]}$ on his row v published by the other parties. Thus, the final product, in other words, the total evaluation of the obfuscated program in row v, is computed according to the secret-key $\mathsf{sk}^{(v)}$ of party v and the secret-keys $\mathsf{sk}^{(u)}$'s of the other parties. These three procedures are the following:

Setup$(1^\lambda, N)$: given the security parameter λ and the number of participants N, we set the length μ of each party's secret, the number of repetitions k, and the dimension m of the matrices, with m being a multiple of 3. We then instantiate the CLT13 multilinear

map with degree of multilinearity $\ell + 2$ with $\ell := \mu N k$. Let $g = \prod_{i=1}^{n} g_i$ be the integer defining the message space $\mathbb{Z}_g$. Let ν be the number of high-order bits that can be extracted by CLT13 from a zero-tested value.

To ensure that all users $1 \leq u \leq N$ compute the same session-key, we define $\mathbf{A}_{i,b}^{(u)}$ as a larger matrix embedding a matrix $\mathbf{B}_{i,b}$ that is the same for all users, with some random block padding in the diagonal and the multiplicative bundling scalars $\alpha_{i,b}^{(u)}$ to prevent the adversary from switching the corresponding bits b_i's between the k repetitions of the secret keys:

$$\mathbf{A}_{i,b}^{(u)} \sim \begin{pmatrix} \$ & \dots & \$ & & & & & \\ \vdots & \ddots & \vdots & & & & & \\ \$ & \dots & \$ & & & & & \\ & & & \$ & \dots & \$ & & \\ & & & \vdots & \ddots & \vdots & & \\ & & & \$ & \dots & \$ & & \\ & & & & & & \alpha_{i,b}^{(u)} \cdot \mathbf{B}_{i,b} & \end{pmatrix} \tag{4.3}$$

More precisely, we first sample 2ℓ random invertible matrices $\mathbf{B}_{i,b}$ in $\mathbb{Z}_g^{m' \times m'}$ where $m' = m/3$, for $1 \leq i \leq \ell$ and $b \in \{0,1\}$. For each $u \in [\![1, N]\!]$, we additionally sample $2\ell N$ scalars $\alpha_{i,b}^{(u)}$ in $\mathbb{Z}_g^\star$ and $4\ell N$ random invertible matrices $\mathbf{S}_{i,b}^{(u)}$ and $\mathbf{T}_{i,b}^{(u)}$ in $\mathbb{Z}_g^{m' \times m'}$, for $1 \leq i \leq \ell$ and $b \in \{0,1\}$. As illustrated in (4.3), we let

$$\mathbf{A}_{i,b}^{(u)} := \mathsf{diag}(\mathbf{S}_{i,b}^{(u)},\ \mathbf{T}_{i,b}^{(u)},\ \alpha_{i,b}^{(u)} \cdot \mathbf{B}_{i,b}) \in \mathbb{Z}_g^{m \times m}. \tag{4.4}$$

Notice that because multiplying matrices that are diagonal per blocks is equivalent to multiplying the corresponding blocks, it holds that for any binary string $\mathbf{b} := (b_1, ..., b_\ell)$,

$$\mathbf{A}^{(u)} := \prod_{i=1}^{\ell} \mathbf{A}_{i,b_i}^{(u)} = \mathsf{diag}(\mathbf{S}^{(u)}, \mathbf{T}^{(u)},\ \alpha^{(u)} \cdot \mathbf{B})$$

where $\mathbf{S}^{(u)} := \prod_{i=1}^{\ell} \mathbf{S}_{i,b_i}^{(u)}$ and $\mathbf{T}^{(u)} := \prod_{i=1}^{\ell} \mathbf{T}_{i,b_i}^{(u)}$ are two random matrices, $\mathbf{B} := \prod_{i=1}^{\ell} \mathbf{B}_{i,b_i}$ is a matrix that do not depend on u, and $\alpha^{(u)} := \prod_{i=1}^{\ell} \alpha_{i,b_i}^{(u)}$. But we want all the parties to obtain the same matrix in the third block, thus, we need the product $\alpha^{(u)}$ to not depend

on u neither. Therefore, the scalars $\alpha_{i,b}^{(u)}$'s must satisfy the following condition:

$$\forall u, v, w \in [\![1, N]\!], \forall i \in [\![1, \mu]\!], \forall b \in \{0,1\}, \quad \prod_{r=1}^{k} \alpha_{\psi(i,w,r),b}^{(u)} = \prod_{r=1}^{k} \alpha_{\psi(i,w,r),b}^{(v)} \pmod{g}.$$

In words, if we fix a value b for the i-th bit of the key $\mathsf{sk}^{(w)}$, then, in any pair of rows (u, v) of Table 4.1, the products of the $\alpha_{i,b}^{(u)}$'s and of the $\alpha_{i,b}^{(v)}$'s corresponding to this bit in all the k repetitions must yield the same value modulo g.

In addition, we sample the vectors $\mathbf{s}^*, \mathbf{t}^*$ uniformly from $\mathbb{Z}_g^{m'}$, and for each $u \in [\![1, N]\!]$ we define a left bookend vector

$$\mathbf{s}^{(u)} := (0, \ldots, 0, \$, \ldots, \$, \mathbf{s}^*) \in \mathbb{Z}_g^m$$

where the block of 0's and the block of randoms have the same length $m' = m/3$ as $\mathbf{s}^*$, and similarly a right bookend vector $\mathbf{t}^{(u)} := (\$, \ldots, \$, 0, \ldots, 0, \mathbf{t}^*) \in \mathbb{Z}_g^m$.

We let $\tilde{\mathbf{A}}_{i,b}^{(u)} \in \mathbb{Z}_{x_0}^{m \times m}$ be the matrix obtained by encoding each entry of $\mathbf{A}_{i,b}^{(u)}$ independently. Similarly we encode $\mathbf{s}^{(u)}$ and $\mathbf{t}^{(u)}$ entrywise, obtaining $\tilde{\mathbf{s}}^{(u)}$ and $\tilde{\mathbf{t}}^{(u)}$. For each $u \in [\![1, N]\!]$, we sample uniformly random invertible matrices $\mathbf{K}_i^{(u)} \in \mathbb{Z}_{x_0}^{m \times m}$ for $0 \leq i \leq \ell$. Then, we use Kilian's randomization on the encoding side and define:

$$\mathbf{C}_{i,b}^{(u)} := \mathbf{K}_{i-1}^{(u)} \tilde{\mathbf{A}}_{i,b}^{(u)} \left(\mathbf{K}_i^{(u)}\right)^{-1} \pmod{x_0}.$$

Similarly, we define $\bar{\mathbf{s}}^{(u)} := \tilde{\mathbf{s}}^{(u)} \left(\mathbf{K}_0^{(u)}\right)^{-1} \pmod{x_0}$ and $\bar{\mathbf{t}}^{(u)} := \mathbf{K}_\ell^{(u)} \tilde{\mathbf{t}}^{(u)} p_{zt} \pmod{x_0}$. The steps performed in this procedure are summarized in Figure 4.1. Finally we output params, which is defined as the set containing all the matrices $\mathbf{C}_{i,b}^{(u)}$'s, the bookend vectors $\bar{\mathbf{s}}^{(u)}$ and $\bar{\mathbf{t}}^{(u)}$, and the scalars $\mu, k, N, \ell, x_0, \nu$ and m.

$\mathsf{Publish}(\mathsf{params}, u)$: Party u samples a bit string $\mathsf{sk}^{(u)} \leftarrow \{0,1\}^\mu$ and for each $v \in [\![1, N]\!]$ such that $u \neq v$, party u computes k products using matrices from the row of party v. This ensures that from the extraction procedure of the multilinear map scheme, each user u can derive the shared key from their own $\mathsf{sk}^{(u)}$ by computing on their row u the partial products corresponding to $\mathsf{sk}^{(u)}$, combined with the partial matrix products published by the other users. More precisely, by defining $b_i := \mathsf{sk}^{(u)}[i]$, party u computes and broadcasts

$$\mathbf{s}^* \xrightarrow[\text{dimensional vector}]{\text{Embed in higher}} \mathbf{s}^{(u)} \xrightarrow[\text{with CLT13}]{\text{Encode entrywise}} \tilde{\mathbf{s}}^{(u)} \xrightarrow[\text{by } \left(\mathbf{K}_0^{(u)}\right)^{-1}]{\text{Multiply on the right}} \bar{\mathbf{s}}^{(u)}$$

$$\mathbf{t}^* \xrightarrow[\text{dimensional vector}]{\text{Embed in higher}} \mathbf{t}^{(u)} \xrightarrow[\text{with CLT13}]{\text{Encode entrywise}} \tilde{\mathbf{t}}^{(u)} \xrightarrow[\text{by } \mathbf{K}_\ell^{(u)} \cdot p_{zt}]{\text{Multiply on the left}} \bar{\mathbf{t}}^{(u)}$$

$$\mathbf{B}_{i,b} \xrightarrow[\text{multiply by } \alpha_{i,b}^{(u)}]{\text{Embed and}} \mathbf{A}_{i,b}^{(u)} \xrightarrow[\text{with CLT13}]{\text{Encode entrywise}} \tilde{\mathbf{A}}_{i,b}^{(u)} \xrightarrow[\text{and by } \left(\mathbf{K}_i^{(u)}\right)^{-1}]{\text{Multiply by } \mathbf{K}_{i-1}^{(u)}} \mathbf{C}_{i,b}^{(u)}$$

Figure 4.1: Summary of how the bookend vectors and the matrices of the matrix branching program are obfuscated.

Table 4.2: Products published by all the parties at the end of Publish arranged as a table. Example with three users ($N = 3$) and two repetitions ($k = 2$).

	$u=1$	$u=2$	$u=3$	$u=1$	$u=2$	$u=3$
$v=1$		$\mathbf{D}_1^{(2\to1)}$	$\mathbf{D}_1^{(3\to1)}$		$\mathbf{D}_2^{(2\to1)}$	$\mathbf{D}_2^{(3\to1)}$
$v=2$	$\mathbf{D}_1^{(1\to2)}$		$\mathbf{D}_1^{(3\to2)}$	$\mathbf{D}_2^{(1\to2)}$		$\mathbf{D}_2^{(3\to2)}$
$v=3$	$\mathbf{D}_1^{(1\to3)}$	$\mathbf{D}_1^{(2\to3)}$		$\mathbf{D}_2^{(1\to3)}$	$\mathbf{D}_2^{(2\to3)}$	

the following products:

$$\mathbf{D}_r^{(u\to v)} := \prod_{i=1}^{\mu} \mathbf{C}_{\psi(i,u,r),b_i}^{(v)} \pmod{x_0} \tag{4.5}$$

for each $v \neq u$ and $r \in [\![1,k]\!]$. The notation $u \to v$ stands for "computed by u to be used by v". We let $\mathsf{pk}^{(u)} = \{\mathbf{D}_r^{(u\to v)} : v \in [\![1,N]\!],\ v \neq u,\ r \in [\![1,k]\!]\}$. Notice that after all the parties have executed Publish, the partial evaluations can be arranged as in Table 4.2.

KeyGen$(\mathsf{params}, v, \mathsf{sk}^{(v)}, \{\mathsf{pk}^{(u)}\}_{u\neq v})$: Using secret $\mathsf{sk}^{(v)}$, party v computes the products $\mathbf{D}_r^{(v\to v)}$ for all $r \in [k]$ using Equation (4.5), and then the product

$$\tilde{z}^{(v)} := \bar{\mathbf{s}}^{(v)} \left(\prod_{r=1}^{k} \left(\prod_{u=1}^{N} \mathbf{D}_r^{(u\to v)} \right) \right) \bar{\mathbf{t}}^{(v)} \pmod{x_0}. \tag{4.6}$$

In other words, party v completes the v-th row of Table 4.2 by computing $\mathbf{D}_1^{(v\to v)}$, ..., $\mathbf{D}_k^{(v\to v)}$, then, multiply all the matrices $\mathbf{D}_r^{(u\to v)}$'s of that row, together with the bookend vectors, obtaining the scalar $\tilde{z}^{(v)}$.

Finally, the shared key is obtained by applying a strong randomness extractor to the ν most-significant bits of $z^{(v)}$.

4.5.1 Correctness of our construction

Let $\mathsf{sk} \in \{0,1\}^\ell$ be the concatenation of the N secret keys repeated k times, as in Equation (4.1). Moreover, let $b_i := \mathsf{sk}[i]$ be the i-th bit of sk. Then, by the definitions of $\mathbf{D}_r^{(u\to v)}$ and of ψ, i.e., equations (4.5) and (4.2), it is easy to see that

$$\prod_{r=1}^{k}\left(\prod_{u=1}^{N}\mathbf{D}_r^{(u\to v)}\right) = \prod_{r=1}^{k}\left(\prod_{u=1}^{N}\left(\prod_{i=1}^{\mu}\mathbf{C}^{(v)}_{\psi(i,u,r),b_i}\right)\right) = \prod_{j=1}^{\ell}\mathbf{C}^{(v)}_{i,b_i}.$$

Therefore, from the cancellation of Kilian's randomization on the encoding side:

$$\tilde{z}^{(v)} = \bar{\mathbf{s}}^{(v)}\left(\prod_{i=1}^{\ell}\mathbf{C}^{(v)}_{i,b_i}\right)\bar{\mathbf{t}}^{(v)} = \tilde{\mathbf{s}}^{(v)}\left(\prod_{i=1}^{\ell}\tilde{\mathbf{A}}^{(v)}_{i,b_i}\right)\tilde{\mathbf{t}}^{(v)}p_{zt} \pmod{x_0}.$$

This corresponds to a zero-tested encoding of:

$$z^{(v)} := \mathbf{s}^{(v)}\cdot\left(\prod_{i=1}^{\ell}\mathbf{A}^{(v)}_{i,\mathsf{sk}[i]}\right)\cdot\mathbf{t}^{(v)} = \mathbf{s}^*\cdot\left(\prod_{i=1}^{\ell}\alpha^{(v)}_{i,\mathsf{sk}[i]}\right)\cdot\left(\prod_{i=1}^{\ell}\mathbf{B}_{i,\mathsf{sk}[i]}\right)\cdot\mathbf{t}^* \pmod{g}$$

From the condition satisfied by the scalars $\alpha^{(v)}_{i,b}$'s, the products $\prod_{i=1}^{\ell}\alpha^{(v)}_{i,\mathsf{sk}[i]}$ are independent from v. Therefore, the values $z^{(v)}$'s are also independent from v, thus, at the end of the protocol, each part has a scalar $\tilde{z}^{(v)} \in \mathbb{Z}_{x_0}$ encoding the same value, hence, by the correctness of the extraction procedure of CLT13, each party v will extract from $\tilde{z}^{(v)}$ the same shared key, as required.

4.5.2 Some simple optimizations

In this section we describe a few optimizations that can be used to obtain a concrete implementation of our key-exchange protocol.

Encoding of elements

Instead of using a single parameter ρ to sample the noise terms during the encoding procedure of CLT13, we can use ρ_b, which is used to encode the bookend vectors, and ρ_m, that is used to encode the matrices. Because each bookend vector is an instance of the vector multi-prime AGCD problem, the value of ρ_b must then be chosen considering the GCD attacks presented in Section 3.6.2, hence, for a security level of λ bits, one has to set $\rho_b \in \Omega(\lambda/(2m))$. However, the matrices can be encoded using a much smaller encoding noise, because they are randomized by two Kilian matrices, one on the left and one on the right. Specifically, we have matrices of the form $\tilde{\mathbf{A}} := \mathbf{KAK'}$, thus, to transform them in vector AGCD instances, we use the operator vec and the Kronecker (tensor) product, as discussed in Section 2.2, i.e., we have

$$\mathsf{vec}(\tilde{\mathbf{A}}) = \mathsf{vec}(\mathbf{A}) \cdot (\mathbf{K'} \otimes \mathbf{K}^T) \in \mathbb{Z}^{m^2},$$

where $(\mathbf{K'} \otimes \mathbf{K}^T)$ is a random $m^2 \times m^2$ matrix, $\mathsf{vec}(\tilde{\mathbf{A}})$ is a vector of dimension m^2 formed by stacking the columns of $\tilde{\mathbf{A}}$ on top of one another, and similarly for $\mathsf{vec}(\mathbf{A})$. Because we are obtaining vector AGCD instances of dimension m^2, we can choose $\rho_m \in \Omega(\lambda/(2m^2))$. Remember that using smaller noise to encode implies smaller final noise accumulated by the operations performed on the CLT13 encodings i.e., the matrix multiplications performed in our key-exchange protocol, therefore, choosing smaller ρ_m allows to also reduce the sizes of η and γ, producing thus a more efficient instantiation of the CLT13 multilinear map.

Number of matrices per level

For security reasons, each secret key $\mathsf{sk}^{(u)}$ sampled in $\mathsf{Publish}$ must have enough bits to prevent an attacker of simply guessing $\mathsf{sk}^{(u)}$ and breaking the key-exchange protocol. In our construction, we fixed a parameter μ as the length of $\mathsf{sk}^{(u)}$. But notice that the degree of multilinearity of CLT13 must be essentially $\ell := N\mu k$, and the other parameters, as η and γ, increase as ℓ increases. Thus, large values of μ tend to make CLT13 less efficient.

But a simply way of reducing μ is by turning $\mathsf{sk}^{(u)}$ in a non-binary string. Hence, let τ

be another parameter of our construction. Instead of sampling $\mathsf{sk}^{(u)}$ from $\{0,1\}^{\mu}$, sample it from $[\![0, 2^{\tau}-1]\!]^{\mu}$. Therewith, each entry $\mathsf{sk}^{(u)}[i]$ has τ bits, thus, guessing $\mathsf{sk}^{(u)}$ costs $2^{\tau\mu}$ and we can divide μ by τ maintaining the same security level.

This also implies that at each step of the evaluation of the branching program, instead of choosing between only two matrices $\mathbf{A}_{i,0}^{(u)}$, $\mathbf{A}_{i,1}^{(u)}$, one has 2^{τ} possible matrices, therefore, the number of encoded matrices is multiplied by a factor $2^{\tau}/\tau$.

4.5.3 Additional safeguard: Straddling Set System

In [BGK+14], the authors introduced a concept called Straddling Set System, which, roughly speaking, is a family of sets that can be used as the levels of a multilinear map when applied to indistinguishability obfuscation so that anyone evaluating an obfuscated branching program B is forced to follow the expected structure of B. That is to say, when evaluating B, if one tries to multiply matrices out of order, or not corresponding to the current input bit, or if one performs only a partial evaluation of the program by not multiplying the entire chain of matrices, then, the resulting evaluation is expected to look random and so is the output of the zero-testing procedure.

In this section, we propose a generalization of the notion of Straddling Set System [BGK+14] so that it can be applied on branching programs with more than two matrices per step, as discussed in Section 4.5.2, and we show how this generalization can be used in our construction.

Definition 4.5.1 ((n,t)-Straddling Set System)**.** *An (n,t)-Straddling Set System over a universe U is a collection of $n \cdot t$ sets $\mathbb{S}_{n,t} := \{S_{i,b} \subset U : i \in [\![1, N]\!], b \in \{0, ..., t-1\}\}$, such that*

$$\bigcup_{1 \le i \le n} S_{i,0} = \bigcup_{1 \le i \le n} S_{i,1} = \cdots = \bigcup_{1 \le i \le n} S_{i,t-1} = U$$

and for every distinct non-empty sets $C, D \subset \mathbb{S}_{n,t}$ we have that if:

1. *(Disjoint sets) C contains only disjoint sets and D also contains only disjoint sets.*
2. *(Collision) $\cup_{S \in C} S = \cup_{S \in D} S$.*

Then, $\exists b_1, b_2 \in \{0, \ldots, t-1\}$ *such that* $b_1 \neq b_2$, $C = \{S_{j,b_1} : j \in [\![1, N]\!]\}$, *and* $D = \{S_{j,b_2} : j \in [\![1, N]\!]\}$.

Thus, the original definition presented on [BGK+14] is an $(n, 2)$-Straddling Set System. Moreover, the only exact cover of the universe is $\{S_{i,0} : i \in [\![1, N]\!]\}, \ldots, \{S_{i,t-2} : i \in [\![1, N]\!]\}$, and $\{S_{i,t-1} : i \in [\![1, N]\!]\}$.

For concreteness, we construct the following Straddling Set System:

Definition 4.5.2 (Canonical (n, t)-Straddling Set System)**.** *For any* $n \in \mathbb{N}^*$ *and* $t \leq 2$, *let the universe be* $U := [\![1, (n-1)t + 1]\!]$ *and* $\mathbb{S}_{n,t} := \{S_{i,b} : i \in [\![1, N]\!], b \in \{0, ..., t-1\}\}$ *where:*

- $S_{1,b} := [\![1, 1 + b]\!]$.
- $S_{i,b} := [\![(i-2)t + 2 + b, \ldots, (i-1)t + 1 + b]\!]$, *for* $2 \leq i \leq n-1$.
- $S_{n,b} := [\![(n-2)t + 2 + b, \ldots, (n-1)t + 1]\!]$.

We now prove that this canonical construction satisfies the definition of (n, t)-Straddling Set System.

Lemma 4.5.3. *The Canonical* (n, t)*-Straddling Set System satisfies Definition 4.5.1*

Proof. For any fixed $b \in \{0, ..., t-1\}$, notice that $\min(S_{i+1,b}) = \max(S_{i,b}) + 1$ for all $i \in [\![1, n-1]\!]$. Therefore, $S_{i,b} \cup S_{i+1,b} = [\![\min(S_i, b), \max(S_i, b)]\!] \cup [\![\max(S_i, b) + 1, \max(S_{i+1}, b)]\!] = [\![\min(S_i, b), \max(S_{i+1}, b)]\!]$. From this, we conclude that

$$\cup_{i \in [\![1,n]\!]} S_{i,b} = [\![\min(S_1, b), \max(S_n, b)]\!] = U.$$

Furthermore, let C and D be two distinct non-empty subsets of $\mathbb{S}_{n,t}$ that satisfy the disjointness and collision properties. In this case, C cannot have two elements $S_{i,b}$ and $S_{i,b'}$ (for same i and different b and b'), because $(i-1)\tau + 1 + \min(b, b') \in S_{i,b} \cap S_{i,b'}$, that is, the sets $S_{i,b}$ and $S_{i,b'}$ are not disjoint. Therefore, there exists b such that $C = \{S_{i_1,b}, \ldots, S_{i_N,b}\}$. By the same argument, $D = \{S_{j_1,b'}, \ldots, S_{j_M,b'}\}$ for some b'.

If $b = b'$, then the collision property implies that $C = D$, which contradicts the fact that they are distinct sets. Thus, without loss of generality, let $b' = b + d$, for some $d \in [\![1, t-1]\!]$.

Notice that if $S_{i,b} \in C$ for $i \geq 2$, then $(i-2)t + 2 + b \in \cup_{S \in C} S = \cup_{S \in D} S$. But $(i-2)t+2+b = (i-2)t+2+b'-d \in S_{i-1,b'}$, then $S_{i-1,b'} \in D$. Using a similar argument, we can show that $S_{i-1,b'} \in D \Rightarrow S_{i-1,b} \in C$. Putting these two implications together, we have

$$S_{i,b} \in C \implies S_{i-1,b} \in C.$$

Similarly, if for some $i \leq n-1$, $S_{i,b} \in C$, then $S_{i,b'} \in D$, which implies $S_{i+1,b} \in C$. Therefore,

$$S_{i,b} \in C \implies S_{i+1,b} \in C.$$

Finally, because C is non-empty by hypothesis, we know that at least one set $S_{i,b}$ belongs to C, thus, all sets $S_{i,b}$'s for the same b belong to C, that is, $C = \{S_{1,b}, \ldots, S_{n,b}\}$. The same argument applies to D, thus, it also holds that $D = \{S_{1,b'}, \ldots, S_{n,b'}\}$. □

When used in multilinear maps, each set $S_{i,b}$ represents a level, hence, for CLT13, it is represented by a product of different z_j's, for $j \in S_{i,b}$. Specifically, for our key-exchange protocol, we want to avoid the related bits to be flipped independently, thus, we use $t = 2^\tau$ and for each $i \in [\![1, \mu]\!]$ and $u, v \in [\![1, N]\!]$, we sample random invertible elements $z_j^{(v,u,i)} \in \mathbb{Z}_{x_0}$ for $j \in U = [\![1, (n-1)t+1]\!]$ (the universe set). Then, each of the related matrices, that is, $\mathbf{C}^{(v)}_{\phi(i,u,r),b}$ for $r \in [\![1, k]\!]$, is encoded at level $S_{r,b}$ by diving the encoding by

$$\prod_{j \in S_{r,b}} z_j^{(v,u,i)} \pmod{x_0}.$$

To illustrate, consider that one sets the first bit of $\mathsf{sk}^{(2)}$ as zero. Then, when the branching programs are evaluated, all the k matrices corresponding to this bit are supposed to be selected also using the bit zero, that is, one is supposed to use the matrices $\mathbf{C}^{(v)}_{\psi(1,2,1),0}$, ..., $\mathbf{C}^{(v)}_{\psi(1,2,k),0}$. If one does it, then the product of the $z_j^{(v,2,1)}$'s will form an exact cover of the universe U and will be correctly removed by the zero-testing procedure.

Otherwise, if at least one of the matrices is selected with respect to a bit one, that is, a matrix $\mathbf{C}^{(v)}_{\psi(1,2,r),1}$ is used, then, the values $z_j^{(v,2,1)}$'s will not be canceled out by the p_{zt} and the resulting zero-tested value will look random.

Hence, the zero-testing parameter, p_{zt}, is multiplied by all the z_j's in each universe. Namely, p_{zt} is multiplied by

$$\prod_{u=1}^{N}\prod_{v=1}^{N}\prod_{i=1}^{\mu}\prod_{j\in S_{r,b}} z_j^{(u,v,i)} \pmod{x_0}.$$

4.6 Cryptanalysis

In this section, we cryptanalyze our N-party one-round key-exchange protocol and show how the parameters can be chosen to guarantee a security level of λ bits.

4.6.1 Practical experiments.

We have run our orthogonal lattice attacks on the multi-prime AGCD problem and its vector variant, as described in sections 3.3.3 and 3.6.1. The source code is provided in [CP19b]. We summarize the running times for various values of n in tables 4.3 and 4.4. We see that the running time of the lattice step in the vector variant is roughly the same as in the non-vector variant, when the number of primes n is divided by m in the vector variant, as predicted by our asymptotic analysis of Chapter 3. For the algebraic step of the non-vector problem, it is significantly more efficient to compute the matrix kernel and eigenvalues modulo some arbitrary prime integer q of size η, instead of over the rationals.

Table 4.3: Running time of the LLL step and the algebraic step for solving the multi-prime AGCD problem, on a 3.2 GHz Intel Core i5.

n	η	ρ	lat. dim.	Time LLL	Time alg.
20	335	80	40	1.5 s	0.6 s
30	335	80	60	9 s	0.7 s
40	335	80	80	37 s	1.5 s
60	335	80	120	4 min	4 s
80	335	80	160	20 min	8 s

Table 4.4: Running time of the LLL step and the algebraic step for solving the multi-prime VAGCD problem, on a 3.2 GHz Intel Core i5.

n	m	η	ρ	lat. dim.	Time LLL	Time alg.
4	5	335	80	40	1.4 s	2.3 s
8	5	335	80	80	32 s	2 min
12	5	335	80	120	4 min	6 min
16	5	335	80	160	12 min	12 min
20	5	335	80	200	44 min	37 min

4.6.2 LLL and BKZ practical complexity.

To derive concrete parameters for our construction from Section 4.5, we have run more experiments with LLL and BKZ lattice reduction algorithms applied to a lattice similar to the lattice L constructed in the orthogonal lattice attacks. Recall that we must apply lattice-basis reduction on the lattice:

$$L := \{\mathbf{u} \in \mathbb{Z}^t \mid \mathbf{u} \cdot \tilde{\mathbf{C}} = \mathbf{0} \pmod{x_0}\}$$

with $t = 2nm$. Consider that $\mathbf{u} = [\mathbf{u}_1, \mathbf{u}_2]$ with $\mathbf{u}_1 \in \mathbb{Z}^{t-m}$ and $\mathbf{u}_2 \in \mathbb{Z}^m$. Similarly, let $\tilde{\mathbf{C}} = \begin{bmatrix} \mathbf{C} \\ \mathbf{W} \end{bmatrix}$, where $\mathbf{W}$ is an $m \times m$ matrix. We can assume that $\mathbf{W}$ is invertible modulo x_0, otherwise we can partially factor x_0. Thus, we obtain

$$\begin{aligned} \mathbf{u} \in L &\iff \mathbf{u_1 A} + \mathbf{u_2 W} \equiv 0 \pmod{x_0} \\ &\iff \mathbf{u_1 A W}^{-1} + \mathbf{u_2} \equiv 0 \pmod{x_0} \end{aligned}$$

Therefore, a basis of L is given by rows of the following matrix:

$$L = \begin{bmatrix} \mathbf{I}_{t-m} & -\mathbf{AW}^{-1} \\ & x_0\mathbf{I}_m \end{bmatrix}$$

For simplicity, we have performed our experiments on a simpler lattice:

$$L' = \begin{bmatrix} \mathbf{I}_{t-m} & \mathbf{A}' \\ & x_0\mathbf{I}_m \end{bmatrix}$$

where the entries of $\mathbf{A}'$ are sampled uniformly from $\mathbb{Z}_{x_0}$. Experimentally, we observed the following running time (expressed in number of clock cycles) for the LLL lattice-basis

	LLL	BKZ-60	BKZ-80	BKZ-100
Root-Hermite factor $= 2^{\alpha}$	1.021	1.011	1.01	1.009
Running time parameter $b(\beta)$	–	10^3	$6 \cdot 10^4$	$3 \cdot 10^6$

Table 4.5: Experimental values of running time and root-Hermite factor for LLL and BKZ as a function of the blocksize β. The parameters for $\beta = 80, 100$ are extrapolated.

reduction provided by Sage:

$$T_{LLL}(t, \gamma, m) := 2 \cdot t^{3.3} \cdot \gamma \cdot m. \tag{4.7}$$

We used the BKZ 2.0 [CN11] implementation included in Sage to estimate the running-time of BKZ-β (in number of clock cycles) against our key-exchange protocol as follows:

$$T_{BKZ}(t, \beta) := b(\beta) \cdot t^{4.3} \tag{4.8}$$

where the observed constant $b(\beta)$ and the root-Hermite factor are given in Table 4.5. However we were not able to obtain experimental results for block-sizes $\beta > 60$, so for BKZ-80 and BKZ-100 we used extrapolated values, assuming that the cost of BKZ sieving with blocksize β is $\mathsf{poly}(t) \cdot 2^{0.292\beta + \circ(\beta)}$ (see [BDGL16]). The Hermite factors for BKZ-80 and BKZ-100 are from [CN11].

Setting concrete parameters.

As explained in Section 3.6.1, when applying LLL or BKZ with blocksize β on the original lattice L, we obtain an orthogonal vector $\mathbf{u}$ if the Inequality (3.2) is satisfied, which gives

$$\alpha \cdot 2nm + 4\rho < \eta \tag{4.9}$$

Therefore we must run LLL or BKZ-β with a large enough blocksize β so that α is small enough this inequality to hold. For security level λ, we require that $T_{lat}(t, \gamma) \geq 2^{\lambda}$, with $t = 2nm$, where the running time (in number of clock cycles) $T_{lat}(t, \gamma)$ is given by (4.7) or (4.8), for $\gamma = \eta \cdot n$.

4.6.3 The Cheon et al. attack and its generalization using tensor products

At Eurocrypt 2015, Cheon *et al.* described in [CHL+15] a total break of the basic key-exchange protocol of CLT13. The attack was then extended and applied to several constructions based on CLT13. In this section, we argue that the complexity of the Cheon *et al.* attack against our construction is $\Omega(m^{2k-1})$, where m is the matrix dimension and k the number of times that the input is repeated in our branching program. Therefore, this attack is prevented by using a large enough k.

The original Cheon et al. attack

The Cheon et al. attack [CHL+15] against CLT13 consists in multiplying the level-one encodings of zero available in the original CLT13 by other encodings to obtain top-level encodings of zero, which are then zero-tested to provide equations over $\mathbb{Z}$ instead of $\mathbb{Z}_{x_0}$. Then, these equations are used to recover all secret primes $p_1, \ldots, p_n$ from the public parameters.

More precisely, assume that CLT13 is instantiated with only $\ell = 2$ levels. Given level-one encodings of zero $a_1, ..., a_n$, any level-zero encoding b_0, and any level-one encodings $c_1, ..., c_n$, notice that any $a_i \cdot b_0 \cdot c_j$ and $a_i \cdot c_j$ is a level-2 (thus, a top-level) encoding of zero. The attack also works for higher multilinear degree, by defining the encoding c_i's as a product of $\ell - 1$ level-one encodings, so that these products are still top-level encodings of zero. The attacker then defines $w_{i,j} := [a_i \cdot b_0 \cdot c_j \cdot p_{zt}]_{x_0}$ and $w'_{i,j} := [a_i \cdot c_j \cdot p_{zt}]_{x_0}$, And then computes two matrices $\mathbf{W}_0, \mathbf{W}_1 \in \mathbb{Z}_{x_0}^{n \times n}$ whose entries are defined as $\mathbf{W}_0[i,j] := w_{i,j}$ and $\mathbf{W}_1[i,j] := w'_{i,j}$.

From the definition of p_{zt}, we obtain $w_{i,j} = \sum_{k=1}^{n} a_{i,k} b_{0,k} c_{j,k} \xi_k \pmod{x_0}$, where $a_{i,k}$, $b_{0,k}$ and $c_{j,k}$ represent the the numerator of a_i, b_0, and c_j modulo p_k, respectively, and ξ_k gathers the terms from p_{zt}. Since we obtain zero-tested encodings of zero, the equation also holds over $\mathbb{Z}$, hence it can be rewritten as

$$w_{i,j} = [a_{i,1}\ a_{i,2} \dots a_{i,n}] \cdot \begin{bmatrix} \xi_1 b_{0,1} & & & \\ & \xi_2 b_{0,2} & & \\ & & \ddots & \\ & & & \xi_n b_{0,n} \end{bmatrix} \cdot \begin{bmatrix} c_{j,1} \\ c_{j,2} \\ \vdots \\ c_{j,n} \end{bmatrix}.$$

Therefore we can write $\mathbf{W}_0 = \mathbf{A}\mathbf{B}_0\mathbf{C}$, where the rows of $\mathbf{A} \in \mathbb{Z}^{n\times n}$ are the vectors in the left (for $i \in [\![1, n]\!]$), $\mathbf{B}_0$ is the diagonal matrix in the middle, and $\mathbf{C} \in \mathbb{Z}^{n\times n}$ is the matrix whose columns are the vectors in the right (for $j \in [\![1, n]\!]$). By the same argument, $\mathbf{W}_1 = \mathbf{A}\mathbf{B}_1\mathbf{C}$, where $\mathbf{B}_1 = \mathsf{diag}(\xi_1, \dots, \xi_n)$. Thus, the attacker can compute over $\mathbb{Q}$:

$$\mathbf{W} = \mathbf{W}_0\mathbf{W}_1^{-1} = (\mathbf{A}\mathbf{B}_0\mathbf{C})(\mathbf{A}\mathbf{B}_1\mathbf{C})^{-1} = \mathbf{A}\mathbf{B}_0\mathbf{B}_1^{-1}\mathbf{A}^{-1}.$$

The eigenvalues of $\mathbf{W}$ are the same as those of $\mathbf{B}_0\mathbf{B}_1^{-1}$ and are equal to $b_{0,1}, \dots, b_{0,n}$. The attacker can therefore recover the $b_{0,i}$'s and then the primes p_i's by computing GCDs. We provide an implementation of the attack in [CP19b].

Variant modulo q.

Since the eigenvalues $b_{0,i}$ are small, they can be computed modulo a small prime q of size η bits. Therefore it suffices to compute the matrix $\mathbf{W}_0\mathbf{W}_1^{-1}$ modulo q only. The characteristic polynomial of $\mathbf{W}_0\mathbf{W}_1^{-1}$ is computed modulo q and then factored to recover the $b_{0,i}$'s modulo q. Experimentally, computing the two matrices $\mathbf{W}_0$ and $\mathbf{W}_1$ takes time $\mathcal{O}(n^{3.5})$. Computing the full $\mathbf{W}_0\mathbf{W}_1^{-1}$ over $\mathbb{Q}$ takes time $\mathcal{O}(n^6)$, whereas computing $\mathbf{W}_0\mathbf{W}_1^{-1} \bmod q$ and recovering the eigenvalues modulo q takes only $\mathcal{O}(n^3)$. Therefore the variant attack modulo q is much faster, and its dominant cost is to compute the two matrices $\mathbf{W}_0$ and $\mathbf{W}_1$. We also provide an implementation of the variant in [CP19b].

Generalization to matrices

The previous attack was extended to matrices of encodings in [CGH+15] as follows: we define an *attack set of dimension* d as 3 sets of matrices $\mathcal{A} := \{\mathbf{A}_i \in \mathbb{Z}_{x_0}^{d\times d} : i \in [\![1, nd]\!]\}$, $\mathcal{B} = \{\mathbf{B}_\sigma \in \mathbb{Z}_{x_0}^{d\times d} : \sigma \in \{0, 1\}\}$, and $\mathcal{C} := \{\mathbf{C}_j \in \mathbb{Z}_{x_0}^{d\times d} : j \in [\![1, nd]\!]\}$, and two vectors $\mathbf{s} \in$

$\mathbb{Z}_{x_0}^d$ and $\mathbf{t} \in \mathbb{Z}_{x_0}^d$ such that the value $w_{i,\sigma,j} := \mathbf{s}\mathbf{A}_i\mathbf{B}_\sigma\mathbf{C}_j\mathbf{t} \pmod{x_0}$ is a zero-tested top-level encoding of zero. The attack then proceeds as previously by computing two matrices $\mathbf{W}_\sigma \in \mathbb{Z}^{nd\times nd}$ (for $\sigma \in \{0,1\}$) whose each entry is defined as $\mathbf{W}_\sigma[i,j] = w_{i,\sigma,j}$, then computing the matrix $\mathbf{W} := \mathbf{W}_0\mathbf{W}_1^{-1}$ over $\mathbb{Q}$. As previously we can write:

$$\mathbf{W}_\sigma = \mathbf{A}\bar{\mathbf{B}}_\sigma\mathbf{C}$$

where the matrix $\bar{\mathbf{B}}_\sigma$ of dimension nd is block-diagonal with the matrices $\xi_i \cdot (\mathbf{B}_\sigma \bmod p_i) \in \mathbb{Z}^{d\times d}$ in the diagonal. We obtain:

$$\mathbf{W} = \mathbf{W}_0\mathbf{W}_1^{-1} = \mathbf{A}\bar{\mathbf{B}}_0\bar{\mathbf{B}}_1^{-1}\mathbf{A}^{-1}$$

The characteristic polynomial $f(X)$ of $\mathbf{W}$ is the same as the characteristic polynomial of $\bar{\mathbf{B}}_0\bar{\mathbf{B}}_1^{-1}$, which is the product of the n characteristic polynomials $f_i(X)$ of the matrices $\tilde{\mathbf{B}}_i = (\mathbf{B}_0 \bmod p_i) \cdot (\mathbf{B}_1 \bmod p_i)^{-1}$. By the Cayley-Hamilton theorem, we must have $f_i(\tilde{\mathbf{B}}_i) = \mathbf{0}$ for all $1 \le i \le n$. This implies $f_i(\mathbf{B}_0 \cdot \mathbf{B}_1^{-1} \bmod x_0) = 0 \pmod{p_i}$. Therefore, if the polynomials $f_i(X)$ are irreducible, they can be recovered by computing $f(X)$ and factoring $f(X)$ into irreducible polynomials. Then each prime p_i can be recovered by computing the GCD of the entries of $\mathbf{M_i} = f_i(\mathbf{B}_0 \cdot \mathbf{B}_1^{-1} \bmod x_0)$ with x_0. We provide the source code of the attack in [CP19b].

Alternatively, if the polynomials $f_i(X)$ are not irreducible, one can still factor $f(X)$ into monic irreducible factors $f'_1, \ldots, f'_N \in \mathbb{Q}[X]$. Then for $k \in [\![1, N]\!]$, the attacker defines $F_k := f/f'_k \in \mathbb{Q}[X]$ and $G_k = F_k \cdot d_k \in \mathbb{Z}[X]$, where d_k is the common denominator of F_k's coefficients. As previously, by the Cayley-Hamilton theorem we have that $G_k(\mathbf{B}_0 \cdot \mathbf{B}_1^{-1} \bmod x_0) = \mathbf{0}$ modulo all primes except one, and therefore the remaining prime p_i can be recovered by computing the GCD of the entries of $\mathbf{M}_k = G_k(\mathbf{B}_0 \cdot \mathbf{B}_1^{-1}) \bmod x_0$ with x_0.

Variant without $\mathbf{B}_0\mathbf{B}_1^{-1} \bmod x_0$.

We describe an alternative attack in which one does not need to compute the matrix $\mathbf{B}_0\mathbf{B}_1^{-1} \bmod x_0$; only the matrix $\mathbf{W}$ is used. This alternative attack will be useful in the

context of the tensoring attack from [CLLT17]; in that case we will not have to compute tensors explicitly as in [CLLT17], which makes the attack slightly simpler.

Our variant attack is as follows. We define the polynomials $G_k(X)$ as previously, and instead of computing the matrices $\mathbf{M}_k = G_k(\mathbf{B}_0 \cdot \mathbf{B}_1^{-1}) \bmod x_0$, we compute the matrices:

$$\mathbf{M}'_k = G_k(\mathbf{W}) \cdot \mathbf{W}_0 \bmod x_0$$

Then as previously each prime p_i can be recovered by computing the gcd of the entries of $\mathbf{M'}_k$ with x_0. Namely we have:

$$\begin{aligned}\mathbf{M}'_k &= G_k\left(\mathbf{A}\bar{\mathbf{B}}_0\bar{\mathbf{B}}_1^{-1}\mathbf{A}^{-1}\right) \cdot \mathbf{W}_0 \pmod{x_0} \\ &= \mathbf{A}G_k(\bar{\mathbf{B}}_0 \cdot \bar{\mathbf{B}}_1^{-1})\mathbf{A}^{-1}\mathbf{A}\bar{\mathbf{B}}_0\mathbf{C} \pmod{x_0} \\ &= \mathbf{A}G_k(\bar{\mathbf{B}}_0 \cdot \bar{\mathbf{B}}_1^{-1})\bar{\mathbf{B}}_0\mathbf{C} \pmod{x_0}\end{aligned}$$

The characteristic polynomial of $\mathbf{W}$ is the same as the one of $\bar{\mathbf{B}}_0 \cdot \bar{\mathbf{B}}_1^{-1}$. Therefore, by the Cayley-Hamilton theorem, all the blocks on the diagonal of $G_k(\bar{\mathbf{B}}_0 \cdot \bar{\mathbf{B}}_1^{-1})$ are zero except the block corresponding to $\tilde{\mathbf{B}}_i = (\mathbf{B}_0 \bmod p_i) \cdot (\mathbf{B}_1 \bmod p_i)^{-1}$ for some i. When multiplying by $\bar{\mathbf{B}}_0$, such block is multiplied by $\xi_i \cdot (\mathbf{B}_0 \bmod p_i) \in \mathbb{Z}^{d \times d}$. Therefore, the resulting block is a multiple of ξ_i, while all the other blocks are zero. This implies that all entries of $\mathbf{M}'_k$ are multiple of ξ_i, which is a multiple of all primes except p_i; this enables to recover p_i by gcd. We also provide an implementation of this variant in [CP19b].

Application to our construction

Our attack proceeds as follows. For simplicity we consider the case of 3 users only, since the generalization to N users is straightforward. As in Equation (4.1), we use $\mathsf{sk} \in \{0,1\}^\ell$ to compute the product matrices in each row, with:

$$\mathsf{sk} = (\underbrace{\mathsf{sk}^{(1)}, \mathsf{sk}^{(2)}, \mathsf{sk}^{(3)}}_{\text{First repetition}}, \ldots, \underbrace{\mathsf{sk}^{(1)}, \mathsf{sk}^{(2)}, \mathsf{sk}^{(3)}}_{k\text{-th repetition}}).$$

Remember that each party has an obfuscated program corresponding to the same branching program, thus, when we evaluate the obfuscated programs of two different rows on the same input, we obtain an encoding of the same value, thus, subtracting them results

in a top-level encoding of zero. Hence, using the same session key in the first two rows, we obtain the following zero-tested top-level encoding of zero:

$$\omega := \bar{\mathbf{s}}^{(1)} \prod_{i=1}^{\ell} \mathbf{C}^{(1)}_{i,sk[i]} \bar{\mathbf{t}}^{(1)} - \bar{\mathbf{s}}^{(2)} \prod_{i=1}^{\ell} \mathbf{C}^{(2)}_{i,sk[i]} \bar{\mathbf{t}}^{(2)} \pmod{x_0}.$$

Notice that we can write ω as

$$\omega = \underbrace{\begin{bmatrix} \bar{\mathbf{s}}^{(1)} & -\bar{\mathbf{s}}^{(2)} \end{bmatrix}}_{\mathbf{s}} \underbrace{\begin{bmatrix} \mathbf{C}^{(1)} & \mathbf{0} \\ \mathbf{0} & \mathbf{C}^{(2)} \end{bmatrix}}_{\mathbf{C}} \underbrace{\begin{bmatrix} \bar{\mathbf{t}}^{(1)} \\ \bar{\mathbf{t}}^{(2)} \end{bmatrix}}_{\mathbf{t}} \pmod{x_0} \tag{4.10}$$

where $\mathbf{C}^{(1)} := \prod_{i=1}^{\ell} \mathbf{C}^{(1)}_{i,sk[i]}$ and $\mathbf{C}^{(2)} := \prod_{i=1}^{\ell} \mathbf{C}^{(2)}_{i,sk[i]}$. Hence, in principle, to produce the attack sets $\mathcal{A}$, $\mathcal{B}$, and $\mathcal{C}$ needed for the extended Cheon *et al.* attack, one should find a partition of sk so that its first bits affect only the first matrices, the middle bits affect the matrices in the middle, and the last bits affect only the last matrices, because both $\mathbf{C}^{(1)}$ and $\mathbf{C}^{(2)}$ must be written as $\mathbf{A}_i \cdot \mathbf{B}_\sigma \cdot \mathbf{C}_j$ where the three factors are independent of each other. However, the k repetitions in sk prevents us from constructing such independent sets, because flipping any bit of sk forces the evaluator to flip the other $k-1$ corresponding bits (otherwise, subtracting two rows does not result in a correct zero-tested encoding of zero). Therefore to generate the attack sets, we use the tensoring technique from [CLLT17] to group the matrices that depend on the same input bits.

More precisely, given three secrets $\mathsf{sk}^{(1)}$, $\mathsf{sk}^{(2)}$, $\mathsf{sk}^{(3)}$ and a given row u, we define the matrices $\mathbf{A}^{(u)}_r := \prod_{i=1}^{\mu} \mathbf{C}^{(u)}_{\psi(i,1,r),\mathsf{sk}^{(1)}[i]}$, $\mathbf{B}^{(u)}_r := \prod_{i=1}^{\mu} \mathbf{C}^{(u)}_{\psi(i,2,r),\mathsf{sk}^{(2)}[i]}$, $\mathbf{C}^{(u)}_r := \prod_{i=1}^{\mu} \mathbf{C}^{(u)}_{\psi(i,3,r),\mathsf{sk}^{(3)}[i]}$. Therefore $\mathbf{A}^{(u)}_r$ is the r-th matrix of the first user computed using secret $\mathsf{sk}^{(1)}$ on row u, $\mathbf{B}^{(u)}_r$ is the r-th matrix of the second user computed on row u, and likewise for $\mathbf{C}^{(u)}_r$. Then, define $\mathbf{A}_r := \mathsf{diag}(\mathbf{A}^{(1)}_r, \mathbf{A}^{(2)}_r)$, $\mathbf{B}_r := \mathsf{diag}(\mathbf{B}^{(1)}_r, \mathbf{B}^{(2)}_r)$, and $\mathbf{C}_r := \mathsf{diag}(\mathbf{C}^{(1)}_r, \mathbf{C}^{(2)}_r)$. Thus, given the number of repetitions k, notice that the diagonal matrix $\mathbf{C}$ defined in Equation (4.10) can be written as $\mathbf{C} = \prod_{r=1}^{k} \mathbf{A}_r \mathbf{B}_r \mathbf{C}_r$. Finally, let's assume that $\mathbf{C}_k$ is an $2m$-dimensional column vector (that is obtained by multiplying by the right bookend vector $\mathbf{t}$) and all the other factors are $2m \times 2m$ matrices.

Now, we claim that $\mathbf{C}$ can be written as $\mathbf{ABC}$ where $\mathbf{A}$ is a $2m \times (2m)^{2k-1}$ matrix depending only on $\mathsf{sk}^{(1)}$, $\mathbf{B}$ is a $(2m)^{2k-1} \times (2m)^{2k-1}$ matrix depending only on $\mathsf{sk}^{(2)}$,

and $\mathbf{C}$ is a $(2m)^{2k-1} \times 1$ matrix (column vector) depending only on $\mathsf{sk}^{(3)}$ (we prove this statement in Lemma 4.6.1 at the end of this section). Thus, by setting $d = (2m)^{2k-1}$ and choosing nd different keys $\mathsf{sk}^{(1)}$, two different keys $\mathsf{sk}^{(2)}$, and nd different keys $\mathsf{sk}^{(3)}$, we can construct an attack set of dimension d of Cheon *et al.* attack and obtain the matrices $\mathbf{W}_0$ and $\mathbf{W}_1$ of dimension $nd \times nd = n(2m)^{2k-1} \times n(2m)^{2k-1}$. Then, the extended Cheon *et al.* attack applies. (i.e., to recover the secret primes p_i's, one just has to proceed as before, by computing $\mathbf{W} := \mathbf{W}_0 \cdot \mathbf{W}_1^{-1}$, factoring the the characteristic polynomials and using the Cayley-Hamilton theorem...). Because one has at least to construct $\mathbf{W}_0$ in order to perform the attack, we can safely say that this attack on our key-exchange protocol has complexity $\Omega(m^{2k-1})$.

Notice that the case $N > 3$ reduces to $N = 3$ by simply using $\mathsf{sk}^{(1)}$ and $\mathsf{sk}^{(2)}$ as the session keys of parties 1 and 2, as usual, and merging the session keys of the remaining users into a single session key, i.e., by defining $\tilde{\mathsf{sk}}^{(3)} := (\mathsf{sk}^{(3)}, ..., \mathsf{sk}^{(N)})$ and performing the attack with $\mathsf{sk}^{(1)}, \mathsf{sk}^{(2)}$ and $\tilde{\mathsf{sk}}^{(3)}$. Therefore, it remains only to prove the following lemma:

Lemma 4.6.1. *Let $k, N \in \mathbb{N}^*$, and $\mathbf{A}_i, \mathbf{B}_i, \mathbf{C}_i \in \mathbb{Z}^{N \times N}$, for $1 \leq i \leq k$, except for $\mathbf{C}_k$, which is defined as a column vector, that is, $\mathbf{C}_k \in \mathbb{Z}^{N \times 1}$. Then, the product $\prod_{i=1}^{k} \mathbf{A}_i \mathbf{B}_i \mathbf{C}_i \in \mathbb{Z}^N$ can be written as $\mathbf{ABC}$, where $\mathbf{A} \in \mathbb{Z}^{N \times N^{2k-1}}$ depends only on the terms $\mathbf{A}_i$'s, $\mathbf{B} \in \mathbb{Z}^{N^{2k-1} \times N^{2k-1}}$ depends only on the terms $\mathbf{B}_i$'s, and $\mathbf{C} \in \mathbb{Z}^{N^{2k-1} \times 1}$ depends only on the terms $\mathbf{C}_i$'s.*

Proof. We proceed with a proof by induction on k. The base case with $k = 1$ is clearly true. For $k \geq 2$, we use induction to write

$$\prod_{i=1}^{k} \mathbf{A}_i \mathbf{B}_i \mathbf{C}_i = \mathbf{A}_1 \mathbf{B}_1 \mathbf{C}_1 \left(\prod_{i=2}^{k} \mathbf{A}_i \mathbf{B}_i \mathbf{C}_i \right) = \mathbf{A}_1 \mathbf{B}_1 \mathbf{C}_1 \tilde{\mathbf{A}} \tilde{\mathbf{B}} \tilde{\mathbf{C}}$$

with $\tilde{\mathbf{A}} \in \mathbb{Z}^{N \times N^{2(k-1)-1}}$ depending only on $\mathbf{A}_2, ..., \mathbf{A}_k$, $\tilde{\mathbf{B}} \in \mathbb{Z}^{N^{2(k-1)-1} \times N^{2(k-1)-1}}$ depending only on $\mathbf{B}_2, ..., \mathbf{B}_k$, and $\tilde{\mathbf{B}} \in \mathbb{Z}^{N^{2(k-1)-1} \times 1}$ depending only on $\mathbf{C}_2, ..., \mathbf{C}_k$.

Then, since $\tilde{\mathbf{C}}$ is a column vector, using the fact that $\mathsf{vec}(\mathbf{v}) = \mathbf{v}$ for any vector v and

that $\mathsf{vec}(\mathbf{XYZ}) = (\mathbf{Z}^T \otimes \mathbf{X})\,\mathsf{vec}(\mathbf{Y})$, we have

$$\begin{aligned}
\mathbf{A}_1\mathbf{B}_1\mathbf{C}_1\tilde{\mathbf{A}}\tilde{\mathbf{B}}\tilde{\mathbf{C}} &= \mathbf{A}_1\mathbf{B}_1\,\mathsf{vec}(\mathbf{C}_1\tilde{\mathbf{A}}\tilde{\mathbf{B}}\tilde{\mathbf{C}}) \\
&= \mathbf{A}_1\mathbf{B}_1(\tilde{\mathbf{C}}^T \otimes \mathbf{C}_1)\,\mathsf{vec}(\tilde{\mathbf{A}}\tilde{\mathbf{B}}) \\
&= \mathbf{A}_1\mathbf{B}_1(\tilde{\mathbf{C}}^T \otimes \mathbf{C}_1)\,\mathsf{vec}(\mathbf{I}_N\tilde{\mathbf{A}}\mathbf{B}) \\
&= \mathbf{A}_1\mathbf{B}_1(\tilde{\mathbf{C}}^T \otimes \mathbf{C}_1)(\tilde{\mathbf{B}}^T \otimes \mathbf{I}_N)\,\mathsf{vec}(\tilde{\mathbf{A}}) \\
&= \mathsf{vec}\left(\mathbf{A}_1\mathbf{B}_1(\tilde{\mathbf{C}}^T \otimes \mathbf{C}_1)(\tilde{\mathbf{B}}^T \otimes \mathbf{I}_N)\,\mathsf{vec}(\tilde{\mathbf{A}})\right) \\
&= (\mathsf{vec}(\tilde{\mathbf{A}})^T \otimes \mathbf{A}_1)\,\mathsf{vec}\left(\mathbf{B}_1(\tilde{\mathbf{C}}^T \otimes \mathbf{C}_1)(\tilde{\mathbf{B}}^T \otimes \mathbf{I}_N)\right) \\
&= (\mathsf{vec}(\tilde{\mathbf{A}})^T \otimes \mathbf{A}_1)((\tilde{\mathbf{B}}^T \otimes \mathbf{I}_N)^T \otimes \mathbf{B}_1)\,\mathsf{vec}(\tilde{\mathbf{C}}^T \otimes \mathbf{C}_1) \\
&= (\mathsf{vec}(\tilde{\mathbf{A}})^T \otimes \mathbf{A}_1)((\tilde{\mathbf{B}} \otimes \mathbf{I}_N) \otimes \mathbf{B}_1)\,\mathsf{vec}(\tilde{\mathbf{C}}^T \otimes \mathbf{C}_1)
\end{aligned}$$

By defining $\mathbf{A} := \mathsf{vec}(\tilde{\mathbf{A}})^T \otimes \mathbf{A}_1$, $\mathbf{B} := (\tilde{\mathbf{B}} \otimes \mathbf{I}_N) \otimes \mathbf{B}_1$, and $\mathbf{C} := \mathsf{vec}(\tilde{\mathbf{C}}^T \otimes \mathbf{C}_1)$, we obtain $\prod_{i=1}^{k} \mathbf{A}_i\mathbf{B}_i\mathbf{C}_i = \mathbf{ABC}$. Because $\mathsf{vec}(\tilde{\mathbf{A}})^T \in \mathbb{Z}^{1\times N^{2(k-1)}}$, we see that $\mathbf{A}$ has dimension $N \times N^{2(k-1)+1} = N \times N^{2k-1}$. Furthermore, $\tilde{\mathbf{B}} \otimes \mathbf{I}_N \in \mathbb{Z}^{N^{2(k-1)}\times N^{2(k-1)}}$, therefore, the dimension of $\mathbf{B}$ is $N^{2k-1} \times N^{2k-1}$. Similarly, the dimension of $\mathbf{C}$ is $N^{2k-1} \times 1$, so the result holds. □

Finally, we provide a basic implementation of the attack in [CP19b], including the variant without $\mathbf{B}_0\mathbf{B}_1^{-1} \bmod x_0$ described in the previous section. This variant recovers the primes p_i's from the matrices $\mathbf{W}_0$ and $\mathbf{W}_1$ only, without computing tensors explicitly.

4.6.4 Other attacks

ECM factoring.

A trivial attack against the CLT13, and thus, against our key-exchange protocol, consists in factoring x_0 to recover the secret primes p_i's. The two main factoring algorithms are the ECM and NFS, whose running times are estimated as $T_{ECM}(\eta,\gamma) := 2^{-20}\exp\left(\sqrt{2\eta(\ln\eta)(\ln 2))}\right)\cdot\gamma\log\gamma$ and $T_{NFS}(\gamma) := \exp((64/9)^{1/3}(\gamma\ln 2)^{1/3}\ln(\gamma\ln 2)^{2/3})$. Hence, we must choose the parameters such that $\min(T_{ECM}(\eta,\gamma), T_{NFS}(\gamma)) \leq 2^{\lambda}$.

Meet-in-the-middle attack.

Given any matrix products $\mathbf{D}_r^{(u\to v)}$ published by party u corresponding to his secret $\mathsf{sk}^{(u)}$ in the procedure Publish, there is a meet-in-the-middle attack that can recover $\mathsf{sk}^{(u)}$: By definition, $\mathbf{D}_r^{(u\to v)} = \prod_{i=1}^{\mu} \mathbf{C}^{(v)}_{\psi(i,u,r),\mathsf{sk}^{(u)}[i]}$, but this can be rewritten as

$$\left(\prod_{i=\lfloor \mu/2\rfloor+1}^{\mu} (\mathbf{C}^{(v)}_{\psi(i,u,r),\mathsf{sk}^{(u)}[i]})^{-1}\right) \mathbf{D}_r^{(u\to v)} = \prod_{i=1}^{\lfloor \mu/2\rfloor} \mathbf{C}^{(v)}_{\psi(i,u,r),\mathsf{sk}^{(u)}[i]}.$$

Thus, since the matrices $\mathbf{C}^{(v)}_{i,b}$'s are known and because each $\mathsf{sk}^{(u)}$ has $\mu\cdot\tau$ bits, one can simply compute the $2^{\tau\mu/2}$ possible products on the left and the ones on the right, and check for which value of $\mathsf{sk}^{(u)}$ the equality holds. The cost of the attack is at least $M(m,\gamma)\cdot 2^{\mu\cdot\tau/2}$, where $M(m,\gamma)$ is the time it takes to multiply $m\times m$ matrices with γ-bit entries. When selecting the parameters, we ensure $M(m,\gamma)\cdot 2^{\mu\cdot\tau/2}\geq 2^{\lambda}$.

4.6.5 Concrete parameters and implementation results

In this section we propose concrete parameters for our key-exchange construction with $N=4$ parties. These parameters are generated so that all known attacks have running time $\geq 2^{\lambda}$ clock cycles. In the construction the total number of encoded matrices is $2^{\tau}\cdot\ell\cdot N$ with $\tau=3$, with a total degree $\ell=\mu\cdot k\cdot N$. Therefore, the total number of CLT13 encodings is $N_{CLT13}\simeq 2^{\tau}\cdot\ell\cdot N\cdot m^2$. The size of the secret key is $\tau\mu=3\mu$ bits. The size η of the primes p_i is adjusted so that we extract $\nu=\lambda$ bits. During the publish phase, each party must broadcast $k\cdot(N-1)$ matrices of dimension $m\times m$ and γ-bit entries. The size of those broadcasted values along with the other parameters are shown in Table 4.6.

The main difference with the original (insecure) key-exchange protocol from [CLT13] is that we get a much larger public parameter size; for $\lambda=62$ bits of security, we need 18 GB of public parameters, instead of 70 MB originally. However our construction would be completely unpractical without Kilian's randomization on the encoding side. Namely for $\lambda=62$ and a degree $\ell=168$, one would need primes p_i of size $\eta\approx(\alpha+\rho)\cdot\ell\approx 2.4\cdot 10^4$ with $\alpha=80$ and $\rho=62$ as in [CLT13]. Since $\gamma=\omega(\eta^2\log\lambda)$ in [CLT13], one would need

Table 4.6: Concrete parameters for a 4-party key-exchange.

	Small	Medium	Large	High
λ	52	62	72	82
η	1759	2602	3761	5159
m	6	6	6	9
n	160	294	1349	4188
μ	15	21	27	33
α	11	12	14	16
k	2	2	2	2
$\gamma = n \cdot \eta$	$281 \cdot 10^3$	$764 \cdot 10^3$	$5073 \cdot 10^3$	$21605 \cdot 10^3$
ℓ	120	168	216	264
N_{CLT13}	$1.4 \cdot 10^5$	$1.9 \cdot 10^5$	$2.5 \cdot 10^5$	$6.8 \cdot 10^5$
params	4.8 GB	18.5 GB	157.8 GB	1848.0 GB
broadcast	7.6 MB	20 MB	137 MB	1312 MB

$\gamma \approx 4 \cdot 10^9$. With $N_{CLT13} = 1.9 \cdot 10^5$, that would require 100 TB of public parameter size. Hence Kilian's randomization on the encoding side provides a reduction of the public parameter size by a factor $\approx 10^4$.

We have implemented the key-exchange protocol in SAGE and executed it on a machine with processor Intel Core i5-8600K CPU (3.60GHz), 32 GB of RAM, and Ubuntu 18.04.2 LTS. The execution times are shown in Table 4.7. We could not run the Large and High instantiations ($\lambda = 72$ and $\lambda = 82$) because of the huge parameter size. While the Setup time is significant, since we need to sample all the random values and perform expensive operations like CRT and inverting matrices, the Publish and KeyGen times remain reasonable. In fact, each user just has to multiply $m \times m$ matrices $\mu \cdot k \cdot (N-1)$ times to publish their values and $k \cdot (\mu + N)$ times to derive the shared key. We provide the source code of the key-exchange in [CP19b].

Table 4.7: Timings for a 4-party key-exchange.

	Setup (once)	Publish (per party)	KeyGen (per party)
Small	2 h 20 min	45 s	19 s
Medium	12 h 23 min	3 min 35 s	1 min 24 s

4.7 Conclusion

In this chapter, we introduced indistinguishability obfuscators (iO) and cryptographic multilinear maps, giving special attention to [CLT13]. Then, we showed that by instantiating the iO scheme from [GGH$^+$13b] with the CLT multilinear map, and by adding another layer of randomization, we can consider the VAGCD problem instead of the AGCD problem, thus, we can then select smaller parameters for the CLT multilinear map. Finally, we proposed and implemented the first N-party one-round key-exchange protocol based on multilinear maps that is secure against all known attacks. For $N = 4$, we could execute experiments for two levels of security ($\lambda = 52$ and $\lambda = 62$). We notice that the size of the public parameters generated during the setup represents the main obstruction to make our protocol practical, hence, it would be interesting to study ways of improving our construction by reducing its memory requirements.

Chapter 5

Homomorphic encryption scheme for vector and matrices

5.1 Introduction

With fully homomorphic encryption (FHE) schemes it is possible to evaluate any computable function homomorphically, i.e., given f and a ciphertext c encrypting x, we can compute an encryption of $f(x)$ using only the public parameters, and possibly the public key, available for the FHE scheme. However, with great generality comes great costs: despite several practical and theoretical improvements since the first construction due to Craig Gentry [Gen09], the size of the keys, the ciphertext expansion, and also the evaluation times are, in general, prohibitive for FHE.

Thus it is plausible to consider weaker classes of homomorphic schemes, since they tend to be more efficient than fully homomorphic ones, and, for several applications, they are already sufficient. For example, leveled homomorphic schemes allows us to add and multiply ciphertexts, but only a limited number of times, hence enabling us to evaluate any computable function that can be represented by depth-bounded circuits. The leveled homomorphic encryption (HE) scheme presented in [GGH+19] is able to compute any program that can be represented by a nondeterministic finite automaton (NFA), thus being able to homomorphically accept regular languages, which is a very restricted yet powerful set of languages. One important difference between that scheme

and typical homomorphic schemes is that the program is encrypted instead of the inputs, i.e., we encrypt the automata and evaluate it on a string given in clear. To do so, we represent the automaton as a state vector and a set of transition matrices, then, evaluating the automaton on a string of length k is done by performing k homomorphic vector-matrix products. Of course, we could also do it using existing FHE schemes, however, the efficiency would not be good. For example, even considering that fast bit-level homomorphic schemes like [CGGI16a] can perform homomorphic multiplications over $\mathbb{Z}_2$ in 0.01 second, evaluating a *deterministic* automaton with 1000 states on an input string of length 1024 by multiplying the vector state by the transition matrices modulo two would take at least $1000^2 \cdot 1024 \cdot 0.01$ seconds, that is, about four months, while in [GGH+19], it takes about one minute and forty seconds. Evaluating *non-deterministic* automata would be even slower, since it would require to operate over, say, 32-bit integers instead of $\mathbb{Z}_2$.

However, the scheme for automata from [GGH+19] is based on yet a new security assumption, that the authors named MiNTRU, and also on a circular security assumption. Ideally, we would like to have schemes whose security is based on more standard problems, like the Learning with errors (LWE) or the Approximate Greatest Common Divisor (AGCD). Moreover, the efficiency of [GGH+19] comes mainly from a noise-control technique in which, roughly speaking, one performs a decomposition of the ciphertexts before operating with them homomorphically, so that they are represented with smaller values and their contribution to the noise growth is reduced. That technique was first used in [GSW13] and has become standard since then. There are several proposals of such GSW-like schemes that are based on more standard problems, like LWE or R-LWE. In particular, the GSW-like scheme proposed in [BBL17] is constructed over the integers, which is appealing because of the simplicity, and it is based on the AGCD problem, which is even believed to be quantum hard. On the negative side, the scheme of [BBL17] encrypts a single bit into a high-dimensional vector, therefore, it has a very high ciphertext expansion, which hurts its efficiency.

In this chapter, we propose a scheme that can perform vectorial operations like

[GGH+19], but that is based on the AGCD problem and uses no circular security assumption, like [BBL17]. To solve the problem of ciphertext expansion, we randomize the AGCD instances with a secret matrix, which allows us to reduce the size of parameters, as it was discussed in Chapter 3. Thus, we obtain an efficient scheme that has good encryption, decryption and evaluation times. We implemented it in C++ and ran experiments for two security levels. As applications, we homomorphically evaluated nondeterministic finite automata and also a Naïve Bayes Classifier.

Our scheme

We propose a leveled homomorphic encryption scheme capable of evaluating vector-matrix and matrix-matrix products homomorphically. Basically, we include an AGCD instance $x_0 := pq_0 + r_0$ in the public parameters, and the secret key is composed of the prime p and a random matrix $\mathbf{K}$ invertible modulo x_0. Then, a vector $\mathbf{m}$ is encrypted as $\mathbf{c} := (p\mathbf{q}+\mathbf{r}+\mathbf{m})\mathbf{K}^{-1} \mod x_0$ and a matrix $\mathbf{M}$ is encrypted as $\mathbf{C} := (p\mathbf{Q}+\mathbf{R}+\mathbf{GKM})\mathbf{K}^{-1} \mod x_0$, where $\mathbf{G}$ is a constant matrix that does not depend on the secret values and $\mathbf{r}, \mathbf{q}, \mathbf{R}$, and $\mathbf{Q}$ are random vectors and matrices. Indeed, we are adding instances $pq_i + r_i$ of AGCD to the messages and randomizing them with $\mathbf{K}$, therefore, we can base the security of our scheme on the AGCD problem. To perform homomorphic products, we apply a publicly computable decomposition G^{-1} to one of the operands and multiply them modulo x_0. For any vector or matrix, G^{-1} yields vectors or matrices with small entries and it holds that $G^{-1}(\mathbf{v})\mathbf{G} = \mathbf{v} \mod x_0$.

Hence, our proposed scheme is a GSW-like scheme [GSW13] and, as a result, the noise growth is only linear on the multiplicative degree, i.e., if the initial noise has magnitude 2^ρ, then performing a sequence of L homomorphic products yields ciphertexts whose noise's size is $O(L \cdot 2^\rho)$. The GSW-like scheme of [BBL17] is also based on AGCD, but it works over $\mathbb{Z}_2$ only. In our case, the plaintext space is bigger, containing vectors and matrices with entries bounded by a parameter B. This already improves the ciphertext expansion and increases the efficiency. Moreover, as observed in [CP19a], the cost of the best attacks against AGCD increases when it is randomized with a matrix $\mathbf{K}$, which means that we

can select smaller parameters, reducing even further the size of the ciphertexts. As a result, we have a scheme whose running times are comparable to those of [GGH+19], but that is based on a more standard problem. In Section 5.6.1, we show that for low-dimensional vectors (roughly up to $m = 100$), our scheme performs even better than [GGH+19], although their scheme becomes faster for bigger values of m.

Optimizations, implementation and applications

We implemented our scheme in C++ using the Number Theory Library[1] (NTL). We also tested two applications: homomorphic evaluation of nondeterministic finite automata and a simple machine learning classification method. The scheme is efficient and the running times are comparable to those of [GGH+19]. All the details are presented in Section 5.5. The source code is available on Github. [Per20c] As a simple optimization, we propose to keep x_0 private and to perform the homomorphic operations without the reduction modulo x_0, which causes the ciphertexts to increase during homomorphic evaluation, making the scheme non-compact, but allows us to select smaller parameters and to obtain better timings for big values of m (plaintext dimension). Moreover, we show that for a large set of functions, in particular, for the evaluation of NFAs, the bit-length of the ciphertexts increases only slightly during the homomorphic evaluation.

5.2 Related work

GSW-like leveled HE over integers

Aiming to construct an AGCD-based homomorphic encryption scheme that enjoys of the same slow noise-growth rate as the GSW scheme, in [BBL17], Benarroch, Brakerski, and Lepoint adapt the decomposition technique used in [GSW13] and propose a scheme in which a homomorphic multiplication increases the noise only approximately additively.

Firstly, they propose a simple version of the scheme that works as follows: let γ, η, and ρ be the AGCD parameters. Define $\mathbf{g} := (2^0, 2^1, \ldots, 2^{\gamma-1})$. For any integer with

[1] https://www.shoup.net/ntl/

less than γ bits, that is, $x \in]\!] -2^{\gamma}, 2^{\gamma}[\![$, let $g^{-1}(x) \in \{-1,0,1\}^{\gamma}$ be the signed binary decomposition of x such that $\mathbf{g} \cdot g^{-1}(x) = x$. Moreover, for $\mathbf{x} \in \mathbb{Z}^{\gamma}$, define $\mathbf{G}^{-1}(\mathbf{x}) := [g^{-1}(x_1)|...|g^{-1}(x_{\gamma})] \in \mathbb{Z}^{\gamma \times \gamma}$, i.e., the matrix whose each column j corresponds to the decomposition of the j-th entry of $\mathbf{x}$. Then, a bit m is encrypted as follows: we sample $p\mathbf{q} + \mathbf{r} \leftarrow \mathcal{D}_{\gamma,\rho}(p)^{\gamma}$ and output

$$\mathbf{c} := p\mathbf{q} + \mathbf{r} + m\mathbf{g} \in \mathbb{Z}^{\gamma}.$$

The decryption function is the following:

$$\mathsf{HE.Dec}(\mathbf{c}) = \begin{cases} 0 & \text{if } \left| \left[\mathbf{c} \cdot g^{-1}(\lfloor p/2 \rceil)\right]_p \right| < p/4 \\ 1 & \text{otherwise} \end{cases}$$

To understand why the decryption works, notice that

$$\mathbf{c} \cdot g^{-1}(\lfloor p/2 \rceil) \bmod p = \mathbf{r} \cdot g^{-1}(\lfloor p/2 \rceil) + m\mathbf{g} \cdot g^{-1}(\lfloor p/2 \rceil) = \mathbf{r} \cdot g^{-1}(\lfloor p/2 \rceil) + m \lfloor p/2 \rceil$$

and the absolute value of the inner product $\mathbf{r} \cdot g^{-1}(\lfloor p/2 \rceil)$ is bounded by $\gamma \cdot \|\mathbf{r}\|_{\infty}$, which is smaller than $p/4$ (considering that the noise is small enough). Thus, when $m = 0$, we have $|\left[\mathbf{c} \cdot g^{-1}(\lfloor p/2 \rceil)\right]_p| = |\mathbf{r} \cdot g^{-1}(\lfloor p/2 \rceil)| < p/4$.

Considering ciphertexts $\mathbf{c}_i := p\mathbf{q}_i + \mathbf{r}_i + m_i\mathbf{g}$ for $i \in \{1,2\}$, the homomorphic NAND gate is performed as

$$\mathbf{c} = \mathsf{HE.Nand}(\mathbf{c}_1, \mathbf{c}_2) := \mathbf{g} - \mathbf{c}_1 \cdot \mathbf{G}^{-1}(\mathbf{c}_2).$$

Notice that, $\mathbf{g} \cdot \mathbf{G}^{-1}(\mathbf{c}_2) = \mathbf{c}_2$, therefore, the following holds modulo p

$$\begin{aligned} \mathbf{g} - \mathbf{c}_1 \cdot \mathbf{G}^{-1}(\mathbf{c}_2) &= \mathbf{g} - \mathbf{r}_1 \cdot \mathbf{G}^{-1}(\mathbf{c}_2) - m_1\mathbf{g} \cdot \mathbf{G}^{-1}(\mathbf{c}_2) \\ &= \mathbf{g} - \mathbf{r}_1 \cdot \mathbf{G}^{-1}(\mathbf{c}_2) - m_1(\mathbf{r}_2 + m_2\mathbf{g}) \\ &= \underbrace{-\mathbf{r}_1 \cdot \mathbf{G}^{-1}(\mathbf{c}_2) - m_1\mathbf{r}_2}_{\mathbf{r}_{NAND}} + (1 - m_1m_2)\mathbf{g}. \end{aligned}$$

Thus, since $NAND(m_1, m_2) = 1 - m_1m_2$, we have a valid encryption of the desired message.

The main problem of this scheme is the huge ciphertext expansion. Notice that we encrypt one single bit into a γ-dimensional vector, moreover, each entry of this vector is

a γ-bit integer, hence, the ciphertext expansion is γ^2. Furthermore, the value γ is usually very big (much bigger than the security parameter λ). Notice that the number of integer multiplications needed to perform one homomorphic NAND gate is also quadratic in γ because $\mathbf{G}^{-1}(\mathbf{c}_2)$ is a $\gamma \times \gamma$ matrix, thus, considering that each of those integer products costs $\tilde{\Theta}(\gamma)$, the total cost of computing $NAND(m_1, m_2)$ homomorphically is $\tilde{\Theta}(\gamma^3)$.

Thus, trying to amend this issue, a batched version of the scheme is proposed. This variant uses the multi-prime AGCD problem to encrypt n bits into a single ciphertext. Furthermore, each homomorphic operation acts in parallel on all the n slots, that is, each ciphertext $\mathbf{c}_i$ encrypts a binary vector $(m_{i,1}, ..., m_{i,n})$ and $\mathsf{HE.Nand}(\mathbf{c}_i, \mathbf{c}_j)$ yields an encryption of $(NAND(m_{i,1}, m_{j,1}), ..., NAND(m_{i,n}, m_{j,n}))$. This basically divides the ciphertext expansion and the cost of the homomorphic evaluation by n. However, one should notice that this is packing technique is not always applicable, since in practice a user may want to evaluate a circuit that is not parallelizable. Finally, the authors of [BBL17] say that even this batched version of the scheme is not efficient, since a single homomorphic NAND gate takes several seconds to be performed in a common personal computer.

FHE for nondeterministic finite automata

Genise et al. propose in [GGH+19] a leveled GSW-like homomorphic scheme that is capable of homomorphically evaluating nondeterministic finite automaton. The scheme has parameters n and m to control the dimension of the plaintext and ciphertext matrices, and a parameter q used as the modulus, that is, the messages are $n \times n$ integer matrices and the ciphertext space is $\mathbb{Z}_q^{n \times m}$. The secret key is defined as a pair $(\mathbf{E}, \mathbf{S}) \in \mathbb{Z}^{n \times m} \times \mathbb{Z}^{n \times n}$, with $\mathbf{E}$ having low norm and $\mathbf{S}$ being invertible modulo q.

The authors say that their scheme is similar to the scheme presented in [HAO15], however, differently of [HAO15], the security of their scheme is not based on the LWE problem, but in a new ad hoc assumption. Essentially, Genise et al. define the Matrix-inhomogeneous NTRU problem (MiNTRU) as the a decisional problem in which one has to distinguish between the uniform $\mathcal{U}(\mathbb{Z}_q^{n \times m})$ and $\left[\mathbf{S}^{-1}(\mathbf{G} - \mathbf{E})\right]_q$, where $\mathbf{G} := [\,\mathbf{0} \,|\, 2 \cdot \mathbf{I}_m \,|\, 2^2 \cdot$

$\mathbf{I}_m \,|\, ... \,|\, 2^{m/n-1} \cdot \mathbf{I}_m \,] \in \mathbb{Z}^{n \times m}$. They argue that this problem is similar to an inhomogeneous version of the well-known NTRU problem, but no formal reduction between these two problems is presented.

In order to build an encryption scheme on top of this decisional problem, they use a standard *randomized* decomposition Φ that satisfies $\mathbf{G}^T \cdot \Phi(\mathbf{A}) = \mathbf{A}$ for any $\mathbf{A}$. The function Φ is implemented by sampling from the lattice $L^{\perp}_{\mathbf{G},q} := \{\mathbf{v} \in \mathbb{Z}^m : \mathbf{G} \cdot \mathbf{v} = \mathbf{0} \mod q\}$ vectors following a narrow discrete Gaussian distribution, e.g., [MP12].

Thus, assuming MiNTRU problem, that is, $\mathcal{U}(\mathbb{Z}_q^{n \times nm}) \sim \left[\mathbf{S}^{-1}\left(\mathbf{G} - \mathbf{E}\right)\right]_q$, the following holds modulo q for any $\mathbf{M} \in \mathbb{Z}^{n \times n}$:

$$\begin{aligned}\mathcal{U}(\mathbb{Z}_q^{n \times m}) \cdot \Phi(\mathbf{MG}) &\sim \mathbf{S}^{-1}\left(\mathbf{G} - \mathbf{E}\right) \cdot \Phi(\mathbf{MG}) \\ &= \mathbf{S}^{-1}\left(\mathbf{G} \cdot \Phi(\mathbf{MG}) - \mathbf{E} \cdot \Phi(\mathbf{MG})\right) \\ &= \mathbf{S}^{-1}\left(\mathbf{MG} - \mathbf{E} \cdot \Phi(\mathbf{MG})\right).\end{aligned}$$

But by setting the parameters so that the sampler Φ has enough entropy, we can use the Leftover Hash Lemma to prove that $\mathcal{U}(\mathbb{Z}_q^{n \times m}) \cdot \Phi(\mathbf{MG}) \sim \mathcal{U}(\mathbb{Z}_q^{n \times m})$. Therefore, we conclude that $\mathcal{U}(\mathbb{Z}_q^{n \times m})$ is computationally indistinguishable from $\mathbf{S}^{-1}\left(\mathbf{MG} - \mathbf{E}'\right)$ where $\mathbf{E}' := \mathbf{E} \cdot \Phi(\mathbf{MG})$ is a low-norm matrix, hence, we can use this expression as the encryption function, i.e., we encrypt a matrix $\mathbf{M}$ as $\mathbf{S}^{-1}\left(\mathbf{MG} - \mathbf{E}'\right)$.

However, in order to have homomorphic properties, Genise et al. also suppose circular security, thus, given $\mathbf{M}$, they encrypt $\mathbf{M} \cdot \mathbf{S}$. Hence, the encryption function is finally defined as

$$\mathbf{C} := \mathbf{S}^{-1}\left(\mathbf{MSG} - \mathbf{E}'\right) \mod q.$$

A vector $\mathbf{m} \in \mathbb{Z}^n$ is encrypted as follows:

$$\mathbf{c} := \mathbf{S}^{-1}\left(\mathbf{m} - \mathbf{e}'\right) \mod q.$$

Then, a homomorphic vector-matrix product is performed as $\mathbf{c}_{mult} := \mathbf{C}_i \cdot \Phi(\mathbf{c}_i)$

mod q, which is indeed an encryption of $\mathbf{Mm}$, because over $\mathbb{Z}_q$ we have

$$\begin{aligned}
\mathbf{c}_{mult} &= \mathbf{S}^{-1}(\mathbf{MSG} - \mathbf{E}') \cdot \Phi(\mathbf{c}) \\
&= \mathbf{S}^{-1}(\mathbf{MSG}\Phi(\mathbf{c}) - \mathbf{E}' \cdot \Phi(\mathbf{c})) \\
&= \mathbf{S}^{-1}(\mathbf{MSc} - \mathbf{E}' \cdot \Phi(\mathbf{c})) \\
&= \mathbf{S}^{-1}(\mathbf{MSS}^{-1}(\mathbf{m} - \mathbf{e}') - \mathbf{E}' \cdot \Phi(\mathbf{c})) \\
&= \mathbf{S}^{-1}(\mathbf{M}\left(\mathbf{m} - \mathbf{e}'\right) - \mathbf{E}' \cdot \Phi(\mathbf{c})) \\
&= \mathbf{S}^{-1}(\mathbf{Mm} - \underbrace{\mathbf{Me}' - \mathbf{E}' \cdot \Phi(\mathbf{c})}_{\mathbf{e}'_{mult}}).
\end{aligned}$$

Furthermore, the authors argue that their scheme can be cryptanalyzed by NTRU attacks and say that for 80 and 100 bits of security, one needs to use $n = 750$ and $n = 1024$, respectively. Note, however, that a user aiming to evaluate homomorphically an NFA with few states, say, 50, would need n to be just 50. This implies that a user cannot take advantage of the low number of states to make the homomorphic evaluation faster, as it would be natural. Nevertheless, we note that, when compared to other HE schemes, [GGH+19] is very efficient even for such big values of n.

5.3 Homomorphic scheme for vector and matrix arithmetic

In this section, a new leveled homomorphic encryption scheme is presented. Like [GGH+19], this scheme can perform homomorphic vector-matrix products (and also other vectorial operations), but instead of being based in an ad hoc hardness assumption, our scheme is based in the AGCD problem. Firstly, the basic procedures are presented (as key generation and encryption), then the correctness of the scheme is shown, then the homomorphic operations are analyzed, and finally, the scheme is proven to be secure under the AGCD assumption.

5.3.1 Preliminaries

Our scheme uses a public modulus $x_0 := p \cdot q_0 + r_0$, which is a γ-bit integer. The ciphertexts are defined over $\mathbb{Z}_{x_0}$. To control the noise-growth, elements of $\mathbb{Z}_{x_0}$ are decomposed in a base b. Thus, we will denote by ℓ the number of words that we need to perform such decomposition, i.e., $\ell := \lceil \log_b(2^\gamma) \rceil$, and we will always use $\mathbf{g}$ to represent the column vector $(1, b, b^2, ..., b^{\ell-1})^T$. Usually, b is equal to 2 and we have a binary decomposition, but we can increase b to reduce the dimensions of the encrypted matrices at the expense of increasing the accumulated noise. For any $a \in [\![0, x_0[\![$, let $g^{-1}(a)$ denote the vector whose entries are the signed base-b decomposition of a and such that $g^{-1}(a)\mathbf{g} = a$. As our gadget matrix, we use $\mathbf{G} = \mathbf{I}_m \otimes \mathbf{g} \in \mathbb{Z}^{m\ell \times m}$, where $\otimes$ denotes the tensor product. Thus, $\mathbf{G}$ is a block-matrix with $\mathbf{g}$ in the diagonal. For instance, for $m = 3$, we have

$$\mathbf{G} = \begin{pmatrix} \mathbf{g} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{g} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{g} \end{pmatrix} \in \mathbb{Z}^{3\ell \times 3}.$$

For any $\mathbf{a} \in \mathbb{Z}^m$, we denote by $G^{-1}(\mathbf{a})$ the vector in which we concatenate the decomposition of each entry of $\mathbf{a}$, that is, $G^{-1}(\mathbf{a}) := (g^{-1}(a_1), ..., g^{-1}(a_m)) \in \mathbb{Z}^{\ell m}$. Notice that $G^{-1}(\mathbf{a})\mathbf{G} = (g^{-1}(a_1)\mathbf{g}, ..., g^{-1}(a_m)\mathbf{g}) = (a_1, ..., a_m) = \mathbf{a}$. For a matrix $\mathbf{A} \in \mathbb{Z}^{n \times m}$, we define $G^{-1}(\mathbf{A})$ as an $n \times m\ell$ matrix such that each row i is the decomposition of the i-th row of $\mathbf{A}$, i.e., $\mathsf{row}_i(G^{-1}(\mathbf{A})) := G^{-1}(\mathsf{row}_i(\mathbf{A}))$. Notice that $G^{-1}(\mathbf{A})\mathbf{G} = \mathbf{A}$.

We also define truncated distributions, which are obtained by rejecting samples that are greater than a given value. They are important to formally prove the security of the scheme, because based on the decisional AGCD problem, we can prove properties about distributions over $[\![0, 2^\gamma - 1]\!]$, but in fact, since the encryption scheme performs reductions modulo a public integer x_0, we want to make statements using the interval $[\![0, x_0 - 1]\!]$.

Definition 5.3.1. *Let Ψ be any distribution whose support is contained in $\mathbb{Z}$ and let r be an integer. We define then $\Psi_{<r}$ as the distribution Ψ conditioned on $\Psi < r$. If $\Pr[\Psi < r] = 0$, then $\Psi_{<r}$ is undefined.*

Thus, we define $\mathcal{D}_{<x_0}$ simply by sampling from $\mathcal{D}_{\gamma,\rho}(p)$ and rejecting the sampled

value if it is bigger than or equal to x_0, which occurs with probability less than one half if we choose $x_0 > 2^{\gamma-1}$.

As it was discussed in Section 5.2, the scheme of [BBL17] has a huge ciphertext expansion. Notice that a natural way to improve that is to generalize the scheme to encrypt non-binary vectors and matrices instead of binary scalars. For instance, one could define the plaintext space over $\mathbb{Z}_B$ for some $B \geq 2$, then encrypt a matrix $\mathbf{M} \in \mathbb{Z}_B^{m\times m}$ as

$$\mathbf{C} := p\mathbf{Q} + \mathbf{R} + \mathbf{GM} \in \mathbb{Z}^{m\ell\times m}.$$

With that, we would encrypt $m^2 \log B$ bits into $m^2\ell\gamma$ bits, which represents a ciphertext expansion of $m^2\ell\gamma/(m^2 \log B) \approx \gamma^2/(\log b \log B)$ instead of the original γ^2. The homomorphic product could still be performed if G^{-1} decomposed the entries of the given matrix now in base b and were multiplied by the left.

Moreover, using the cryptanalysis done in Section 3.5, we see that if we randomized the ciphertexts multiplying them by a hidden matrix $\mathbf{K} \in \mathbb{Z}_{x_0}^{m\times m}$, then we could reduce the size of the parameters, in particular, we would have a smaller γ, approximately equal to the original γ divided by m, and the ciphertext expansion would be foreshortened even further, namely, instead of having ciphertext expansion equal to γ^2, we would have $\gamma^2/(m^2 \log b \log B)$.

Hence, our scheme applies those changes in order to be more practical and other ones to maintain the homomorphic properties. We present it in detail in the next section.

5.3.2 The procedures

In which follows, λ is the security parameter and k is the maximum multiplicative degree of the functions to be evaluated homomorphically. The plaintext space is the set of m-dimensional integer vectors and $m \times m$ matrices with norm bounded by B, that is, $\mathcal{M} := [\![-B, B]\!]^m \cup [\![-B, B]\!]^{m\times m}$. The value B must satisfy $1 \leq B \leq 2^{\eta-4}$, where η is the bit-length of the secret prime p.

- $\mathsf{HE.KeyGen}(1^\lambda, m, k, B)$: Choose the parameters η, ρ, ρ_0, and γ. Sample an η-bit prime p. Sample x_0 from $\mathcal{D}_{\gamma,\rho_0}(p)$ until $x_0 > 2^{\gamma-1}$. Then, sample $\mathbf{K}$ uniformly from $\mathbb{Z}_{x_0}^{m\times m}$ until

$\mathbf{K}^{-1}$ exists over $\mathbb{Z}_{x_0}$. Define $\alpha := \lfloor 2^{\eta-1}/(2B+1) \rfloor$. The secret key is then $\mathsf{sk} := (p, \mathbf{K})$ and the public parameters are $\{m, k, B, \eta, \gamma, \rho, \alpha, x_0\}$.

- $\mathsf{HE.EncMat}(\mathsf{sk}, \mathbf{M})$: Given a $\mathbf{M} \in \mathcal{M}$, sample $\mathbf{X} := p\mathbf{Q} + \mathbf{R} \leftarrow \mathcal{D}^{m\ell \times n}_{<x_0}$ then compute $\mathbf{C} := (\mathbf{X} + \mathbf{GKM})\mathbf{K}^{-1} \bmod x_0$. Output $\mathbf{C}$.

- $\mathsf{HE.DecMat}(\mathsf{sk}, \mathbf{C})$: Given a ciphertext $\mathbf{C} \in \mathbb{Z}^{m\ell \times m}$, multiply it over $\mathbb{Z}_{x_0}$ on the left by $G^{-1}(\alpha \mathbf{K}^{-1})$ and on the right by $\mathbf{K}$, i.e., $\mathbf{C}' := G^{-1}(\alpha \mathbf{K}^{-1})\mathbf{CK} \mod x_0$, then reduce it modulo the secret prime p, that is, $\mathbf{C}^\star := [\mathbf{C}']_p$, and output

$$\left\lfloor \frac{\mathbf{C}^\star}{\alpha} \right\rceil .$$

- $\mathsf{HE.EncVec}(\mathsf{sk}, \mathbf{m})$: Given a plaintext $\mathbf{m} \in \mathcal{M}$, sample $\mathbf{x} := p\mathbf{q} + \mathbf{r} \leftarrow \mathcal{D}^{m}_{<x_0}$, then output the following m-dimensional vector

$$\mathbf{c} := (\mathbf{x} + \alpha \mathbf{m})\mathbf{K}^{-1} \mod x_0.$$

- $\mathsf{HE.DecVec}(\mathsf{sk}, \mathbf{c})$: Given a ciphertext $\mathbf{c} \in \mathbb{Z}^m$, multiply it over $\mathbb{Z}_{x_0}$ on the right by $\mathbf{K}$, that is, $\mathbf{c}' := \mathbf{cK} \mod x_0$, then do $\mathbf{c}^\star := [\mathbf{c}']_p$ and output

$$\left\lfloor \frac{\mathbf{c}^\star}{\alpha} \right\rceil .$$

5.3.3 Correctness of decryption

In this section, we provide sufficient conditions for the decryption procedures to work. For this, we will use that $G^{-1}(\alpha \mathbf{K}^{-1})\mathbf{G} = \alpha \mathbf{K}^{-1}$ over $\mathbb{Z}_{x_0}$. In this analysis, we have to be careful with the contribution of x_0 to the noise. Basically, each vector ciphertext has the form $\mathbf{c} = (p\mathbf{q} + \mathbf{r} + \alpha \mathbf{m})\mathbf{K}^{-1} - \mathbf{v}x_0$ and during the decryption, when we do the modular reduction by x_0, we remove $x_0\mathbf{v}$ but add another multiple of x_0, obtaining

$$\mathbf{c}' = \mathbf{cK} \mod x_0 = p\mathbf{q} + \mathbf{r} + \alpha \mathbf{m} - \mathbf{u}x_0 = p(\mathbf{q} - \mathbf{u}q_0) + (\mathbf{r} - \mathbf{u}r_0) + \alpha \mathbf{m}.$$

Therefore, instead of having the noise given simply by $\mathbf{r}$, we have the extra term $\mathbf{u}r_0$, which is the contribution of x_0, and thus, the noise in a ciphertext is approximately

$\|\mathbf{r}\|+2^{\rho_0}\|\mathbf{u}\|$. But the norm of $\mathbf{u}$ is easy to estimate. First, we know that $\|p\mathbf{q}+\mathbf{r}+\alpha\mathbf{m}\| \approx p\|\mathbf{q}\|$. Second, we have $\mathbf{u} = \lfloor(p\mathbf{q}+\mathbf{r}+\alpha\mathbf{m})/x_0\rfloor$. Thus, $\|\mathbf{u}\| \approx p\|\mathbf{q}\|/x_0$, and the contribution of x_0 to the noise is then $r_0\|\mathbf{u}\| \approx 2^{\rho_0}p\|\mathbf{q}\|/x_0$. Consequently, x_0 contributes little to the noise of fresh ciphertexts, since $p\mathbf{q}$ has small norm in this case. But as we perform homomorphic operations, the norm of $\mathbf{q}$ grows and the additional term $\mathbf{u}r_0$ starts to be relevant.

Basically the same reasoning applies to matrix ciphertexts, i.e., there is an additional noise term $\mathbf{U}r_0$ that we get from x_0 and that depends on the term $p\mathbf{Q}$, whose norm is small at the beginning and grows with the number of homomorphic operations. We present these arguments formally in the following definitions and lemmas. [2]

Definition 5.3.2 (Noise of vector ciphertext)**.** *Let* $\mathbf{c}$ *be a ciphertext encrypting a message* $\mathbf{m}$*. We define the noise of* $\mathbf{c}$ *as* $\mathsf{err}(\mathbf{c}) := ((\mathbf{cK} \mod x_0) - \alpha\mathbf{m}) \bmod p$.

Definition 5.3.3 (Noise of matrix ciphertext)**.** *Let* $\mathbf{C}$ *be an encryption of* $\mathbf{M}$*. We define the noise of* $\mathbf{C}$ *as* $\mathsf{err}(\mathbf{C}) := (G^{-1}(\alpha\mathbf{K}^{-1})\mathbf{CK} \mod x_0) - \alpha\mathbf{M} \mod p$.

Lemma 5.3.4 (A bound to the noise of vector ciphertext)**.** *For* $\mathbf{c} = (p\mathbf{q}+\mathbf{r}+\alpha\mathbf{m})\mathbf{K}^{-1} \bmod x_0$*, assuming that* $\|\mathsf{err}(\mathbf{c})\| < p$*, there exists* $\mathbf{u} \in \mathbb{Z}^n$ *such that* $\mathsf{err}(\mathbf{c}) := \mathbf{r} - r_0\mathbf{u}$ *and* $\|\mathbf{u}\| \le \lceil\|p\mathbf{q}\|/x_0\rceil$*. As a consequence,* $\|\mathsf{err}(\mathbf{c}))\| < \|\mathbf{r}\| + 2^{\rho_0}\lceil\|p\mathbf{q}\|/x_0\rceil$.

Proof. Let $\mathbf{c}' := (\mathbf{cK} \mod x_0)$, then $\mathbf{c}' = p\mathbf{q}+\mathbf{r}+\alpha\mathbf{m} \mod x_0$, which means that $\mathbf{c}' = p\mathbf{q}+\mathbf{r}+\alpha\mathbf{m}-x_0\mathbf{u}$ for $\mathbf{u} = \lfloor(p\mathbf{q}+\mathbf{r}+\alpha\mathbf{m})/x_0\rfloor$.

Therefore, $\mathsf{err}(\mathbf{c}) = \mathbf{c}' - \alpha\mathbf{m} \mod p = p\mathbf{q}+\mathbf{r}-x_0\mathbf{u} \mod p = \mathbf{r}-r_0\mathbf{u} \mod p$. And since $\|\mathsf{err}(\mathbf{c})\| < p$, we have the equality $\mathsf{err}(\mathbf{c}) = \mathbf{r}-r_0\mathbf{u}$ over $\mathbb{Z}$.

Now, to bound the norm of $\mathbf{u}$, notice that for each entry u_i, we have $u_i = \left\lfloor \frac{pq_i}{x_0} + \frac{r_i+\alpha m_i}{x_0} \right\rfloor$ and $|\frac{r_i+\alpha m_i}{x_0}| < 1$, thus, if $\frac{pq_i}{x_0}$ is integer, than $u_i = \frac{pq_i}{x_0}$, otherwise, $-1 < u_i \le \frac{pq_i}{x_0} + 1$. So, in both cases, $|u_i| \le \left\lceil \frac{pq_i}{x_0} \right\rceil$. Therefore, $\|\mathbf{u}\| \le \lceil\|p\mathbf{q}\|/x_0\rceil$.

Finally, because $|r_0| < 2^{\rho_0}$, it holds that $\|\mathsf{err}(\mathbf{c})\| < \|\mathbf{r}\| + 2^{\rho_0}\left\lceil \frac{\|p\mathbf{q}\|}{x_0} \right\rceil$. □

[2] Notice that everything would be simplified if x_0 were noiseless, since the noise of the ciphertexts would be simply $\mathbf{r}$ or $\mathbf{R}$.

Lemma 5.3.5 (A bound to the noise of matrix ciphertext)**.** *For* $\mathbf{C} = (p\mathbf{Q} + \mathbf{R} + \mathbf{GKM})\mathbf{K}^{-1} \mod x_0$*, assuming that* $\|\mathsf{err}(\mathbf{C})\| < p$*, there exists* $\mathbf{U} \in \mathbb{Z}^{m\ell \times m}$ *such that*

$$\mathsf{err}(\mathbf{C}) := G^{-1}(\alpha\mathbf{K})\mathbf{R} - r_0\mathbf{U}$$

and $\|\mathbf{U}\| \leq m\ell b \left\lceil \frac{\|p\mathbf{Q}\|}{x_0} \right\rceil$*. As a consequence,* $\|\mathsf{err}(\mathbf{C})\| < m\ell b \left(\|\mathbf{R}\| + 2^{\rho_0} \left\lceil \frac{\|p\mathbf{Q}\|}{x_0} \right\rceil \right)$.

Proof. Write $\mathbf{U} = \left\lfloor (pG^{-1}(\alpha\mathbf{K})\mathbf{Q} + G^{-1}(\alpha\mathbf{K})\mathbf{R} + \alpha\mathbf{M})/x_0 \right\rfloor$ and proceed in the same way as in the proof of Lemma 5.3.4. The extra term $m\ell b$ comes from the fact that $\left\|G^{-1}(\alpha\mathbf{K})\mathbf{R}\right\| \leq m\ell b \|\mathbf{R}\|$. □

Corollary 5.3.6 (Noise of fresh ciphertexts)**.** *Let* $\mathbf{m}$ *and* $\mathbf{M}$ *be two plaintexts and* $\mathbf{c} = \mathsf{HE.EncVec}(\mathbf{m})$ *and* $\mathbf{C} = \mathsf{HE.EncMat}(\mathbf{M})$ *to fresh ciphertexts. Then,* $\|\mathsf{err}(\mathbf{c})\| < 2^{\rho} + 2^{\rho_0}$ *and* $\|\mathsf{err}(\mathbf{C})\| < m\ell b(2^{\rho} + 2^{\rho_0})$.

Proof. Because of the sampling procedure used in HE.EncVec, we have $0 \leq pq_i < x_0$ and $|r_i| < 2^{\rho}$, therefore, $\lceil \|p\mathbf{q}\| / x_0 \rceil \leq 1$ and $\|\mathbf{r}\| \leq 2^{\rho}$. Similarly $\lceil \|p\mathbf{Q}\| / x_0 \rceil \leq 1$ and $\|\mathbf{R}\| \leq 2^{\rho}$. Hence, the result follows from lemmas 5.3.4 and 5.3.5. □

For the correctness of decryption, we need the noise to be less than a fraction of p defined by B, i.e., $\alpha/2 \approx p/(4B+2)$. We prove that in the following lemmas.

Lemma 5.3.7 (Sufficient conditions for correctness of vector decryption)**.** *Let* $\mathbf{c}$ *be an encryption of* $\mathbf{m}$ *and* $\|\mathbf{m}\| \leq B$*. If* $\|\mathsf{err}(\mathbf{c})\| < \frac{\alpha}{2}$*, then* $\mathsf{HE.DecVec}(\mathsf{sk}, \mathbf{c})$ *outputs* $\mathbf{m}$.

Proof. Considering the vector $\mathbf{c}'$ defined in HE.DecVec, there is a $\mathbf{u}$ such that

$$\mathbf{c}' = (p\mathbf{q} + \mathbf{r} + \mathbf{m})\mathbf{K}^{-1}\mathbf{K} \mod x_0 = p\mathbf{q} + \mathbf{r} + \alpha\mathbf{m} \mod x_0 = p\mathbf{q} + \mathbf{r} + \alpha\mathbf{m} - x_0\mathbf{u}.$$

Then, reducing $\mathbf{c}'$ modulo p gives us $\mathbf{c}^{\star} = [\mathbf{r} + \alpha\mathbf{m} - r_0\mathbf{u}]_p = [\alpha\mathbf{m} + \mathsf{err}(\mathbf{c})]_p$.

But the last inequality holds over the integers because the norm of $\alpha\mathbf{m} + \mathsf{err}(\mathbf{c})$ is bounded by $p/2$, namely, since $\alpha < p/(2B+1)$, we have

$$\|\alpha\mathbf{m}\| + \|\mathsf{err}(\mathbf{c})\| < \alpha \left(\|\mathbf{m}\| + \frac{1}{2} \right) \leq \alpha \left(B + \frac{1}{2} \right) = \alpha \left(\frac{2B+1}{2} \right) < \frac{p}{2}.$$

Therefore, the output of HE.DecVec is

$$\lfloor \mathbf{c}^{\star}/\alpha \rceil = \lfloor \alpha\mathbf{m} + \mathsf{err}(\mathbf{c})/\alpha \rceil = \mathbf{m} + \lfloor \mathsf{err}(\mathbf{c})/\alpha \rceil = \mathbf{m}$$

where the last equality holds because $\alpha > 2\,\|\mathsf{err}(\mathbf{c})\|$. □

Lemma 5.3.8 (Sufficient conditions for correctness of matrix decryption)**.** *Let* $\mathbf{C}$ *be an encryption of* $\mathbf{M}$ *such that* $\|\mathbf{M}\| \leq B$*. If* $\|\mathsf{err}(\mathbf{C})\| < \frac{\alpha}{2}$*, then* $\mathsf{HE.DecVec}(\mathsf{sk}, \mathbf{c})$ *outputs* $\mathbf{m}$*.*

Proof. To simplify the notation, denote $G^{-1}(\alpha\mathbf{K}^{-1})$ by $\mathbf{A}$. Remember that $\mathbf{AGK} = \alpha\mathbf{I} \mod x_0$. Considering the matrix $\mathbf{C}'$ defined in $\mathsf{HE.DecMat}$, we have

$$\begin{aligned}\mathbf{C}' &= \mathbf{ACK} \mod x_0 \\ &= \mathbf{A}((p\mathbf{Q} + \mathbf{R} + \mathbf{GKM})\mathbf{K}^{-1})\mathbf{K} \mod x_0 \\ &= p\mathbf{AQ} + \mathbf{AR} + \alpha\mathbf{M} \mod x_0.\end{aligned}$$

Thus, there exists a $\mathbf{U} \in \mathbb{Z}^{m\ell \times n}$ such that $\mathbf{C}' = p\mathbf{AQ} + \mathbf{AR} + \alpha\mathbf{M} - x_0\mathbf{U}$.

Then, when we perform the reduction modulo p, we obtain

$$\mathbf{C}^\star = [\mathbf{C}']_p = [\mathbf{AR} + \alpha\mathbf{M} - r_0\mathbf{U}]_p = [\alpha\mathbf{M} + \mathsf{err}(\mathbf{C})]_p$$

where the last equality holds also over $\mathbb{Z}$ because $\|\alpha\mathbf{M} + \mathsf{err}(\mathbf{C})\|$ is bounded by

$$\alpha\,\|\mathbf{M}\| + \|\mathsf{err}(\mathbf{C})\| < \alpha\left(\|\mathbf{M}\| + \frac{1}{2}\right) \leq \alpha\left(B + \frac{1}{2}\right) = \alpha\left(\frac{2B+1}{2}\right) < \frac{p}{2}.$$

Finally, because $\alpha > 2\,\|\mathsf{err}(\mathbf{C})\|$, the output of $\mathsf{HE.DecMat}$ is

$$\left\lfloor \frac{\mathbf{C}^\star}{\alpha} \right\rceil = \mathbf{M} + \left\lfloor \frac{\mathsf{err}(\mathbf{C})}{\alpha} \right\rceil = \mathbf{M}.$$

□

5.3.4 Homomorphic properties

Now we show how to perform homomorphic operations.

- **Additions**: One just has to add the corresponding ciphertexts over $\mathbb{Z}_{x_0}$, since

$$\mathbf{c}_0 + \mathbf{c}_1 = (p(\mathbf{q}_0 + \mathbf{q}_1) + (\mathbf{r}_0 + \mathbf{r}_1) + \alpha(\mathbf{m}_0 + \mathbf{m}_1))\mathbf{K}^{-1} \mod x_0$$

and

$$\mathbf{C}_0 + \mathbf{C}_1 = (p(\mathbf{Q}_0 + \mathbf{Q}_1) + (\mathbf{R}_0 + \mathbf{R}_1) + \mathbf{GK}(\mathbf{M}_0 + \mathbf{M}_1))\mathbf{K}^{-1} \mod x_0$$

are valid encryptions of the corresponding sums.

- **Matrix-matrix product**: Given ciphertexts $\mathbf{C}_0$ and $\mathbf{C}_1$, we apply G^{-1} to $\mathbf{C}_0$ and multiply it by $\mathbf{C}_1$ over $\mathbb{Z}_{x_0}$, that is, $\mathbf{C}_{mult} := G^{-1}(\mathbf{C}_0)\mathbf{C}_1 \mod x_0$. Notice that the following holds modulo x_0:

$$\begin{aligned}
\mathbf{C}_{mult} &= (pG^{-1}(\mathbf{C}_0)\mathbf{Q}_1 + G^{-1}(\mathbf{C}_0)\mathbf{R}_1 + G^{-1}(\mathbf{C}_0)\mathbf{GKM}_1)\mathbf{K}^{-1} \\
&= (pG^{-1}(\mathbf{C}_0)\mathbf{Q}_1 + G^{-1}(\mathbf{C}_0)\mathbf{R}_1 + \mathbf{C}_0\mathbf{KM}_1)\mathbf{K}^{-1} \\
&= (pG^{-1}(\mathbf{C}_0)\mathbf{Q}_1 + G^{-1}(\mathbf{C}_0)\mathbf{R}_1 + (p\mathbf{Q}_0 + \mathbf{R}_0 + \mathbf{GKM}_0)\mathbf{K}^{-1}\mathbf{KM}_1)\mathbf{K}^{-1} \\
&= (pG^{-1}(\mathbf{C}_0)\mathbf{Q}_1 + G^{-1}(\mathbf{C}_0)\mathbf{R}_1 + (p\mathbf{Q}_0\mathbf{M}_1 + \mathbf{R}_0\mathbf{M}_1 + \mathbf{GKM}_0\mathbf{M}_1))\mathbf{K}^{-1} \\
&= (p\underbrace{(G^{-1}(\mathbf{C}_0)\mathbf{Q}_1 + \mathbf{Q}_0\mathbf{M}_1)}_{\mathbf{Q}_{mult}} + \underbrace{(G^{-1}(\mathbf{C}_0)\mathbf{R}_1 + \mathbf{R}_0\mathbf{M}_1)}_{\mathbf{R}_{mult}} + \mathbf{GKM}_0\mathbf{M}_1))\mathbf{K}^{-1}
\end{aligned}$$

  which is a valid encryption of the matrix $\mathbf{M}_0 \cdot \mathbf{M}_1$.

- **Vector-Matrix product:** We can multiply $\mathbf{c}_i$ and $\mathbf{C}_i$ homomorphically by doing $\mathbf{c}_{i+1} := G^{-1}(\mathbf{c}_i)\mathbf{C}_i \mod x_0$. Like the matrix-matrix product, we have the following over $\mathbb{Z}_{x_0}$:

$$\begin{aligned}
\mathbf{c}_{i+1} &= (pG^{-1}(\mathbf{c}_i)\mathbf{Q}_i + G^{-1}(\mathbf{c}_i)\mathbf{R}_i + G^{-1}(\mathbf{c}_i)\mathbf{GKM}_i)\mathbf{K}^{-1} \\
&= (pG^{-1}(\mathbf{c}_i)\mathbf{Q}_i + G^{-1}(\mathbf{c}_i)\mathbf{R}_i + (p\mathbf{q}_i + \mathbf{r}_i + \alpha\mathbf{m}_i)\mathbf{M}_i)\mathbf{K}^{-1} \\
&= (p\underbrace{(G^{-1}(\mathbf{c}_i)\mathbf{Q}_i + \mathbf{q}_i\mathbf{M}_i)}_{\mathbf{q}_{i+1}} + \underbrace{(G^{-1}(\mathbf{c}_i)\mathbf{R}_i + \mathbf{r}_i\mathbf{M}_i)}_{\mathbf{r}_{i+1}} + \alpha\mathbf{m}_i\mathbf{M}_i)\mathbf{K}^{-1}
\end{aligned}$$

  which is a valid encryption of the vector $\mathbf{m}_i \cdot \mathbf{M}_i$.

5.3.5 Analysis of the accumulated error

In this section, the analysis done in Section 5.3.4 is used to derive upper bounds to the noise accumulated by the homomorphic operations. To simplify the notation, we denote the infinity norm and the maximum norm simply by $\| \cdot \|$.

Lemma 5.3.9 (Sum of vectors)**.** *Let $k \in \mathbb{Z}_{\geq 2}$. For $i \in [\![1, k]\!]$, let $\mathbf{c}_i$ be an encryption of $\mathbf{m}_i$ with noise term $\mathsf{err}(\mathbf{c}_i)$. Define $\mathbf{c}$ as the homomorphic sum of those ciphertexts,*

i.e., $\mathbf{c} := \sum_{i=1}^{k} \mathbf{c}_i \mod x_0$. *Then,* $\mathsf{err}(\mathbf{c}) = \sum_{i=1}^{k} \mathsf{err}(\mathbf{c}_i)$. *In particular, if all* $\mathbf{c}_i$*'s are fresh ciphertexts, we have*

$$\|\mathsf{err}(\mathbf{c})\| \leq k(2^{\rho} + 2^{\rho_0}).$$

Proof. Notice that $\mathsf{err}(c_i) = \mathbf{r}_i - r_0\mathbf{u}_i$ where $x_0\mathbf{u}_i = p\mathbf{q}_i - \mathbf{c}_i + \mathbf{r}_i + \alpha\mathbf{m}_i$. Moreover, given the analysis of Section 5.3.4, it is trivial that $\mathbf{c} = \sum_{i=1}^{k}(p\mathbf{q}_i + \mathbf{r}_i + \alpha\mathbf{m}_i)\mathbf{K}^{-1} \mod x_0$, thus, we have $\mathbf{c}' := \mathbf{cK} \mod x_0 = \sum_{i=1}^{k}(p\mathbf{q}_i + \mathbf{r}_i + \alpha\mathbf{m}_i) - x_0\mathbf{u}$ and $\mathbf{u}$ must be equal to $\sum_{i=1}^{k}\mathbf{u}_i$, which means that

$$\mathsf{err}(\mathbf{c}) = \mathbf{c}' - \sum_{i=0}^{k} \alpha\mathbf{m}_i \mod p = \sum_{i=1}^{k}(\mathbf{r}_i - r_0\mathbf{u}_i) = \sum_{i=1}^{k} \mathsf{err}(\mathbf{c}_i).$$

If all $\mathbf{c}_i$'s are fresh ciphertexts, then $\|\mathsf{err}(\mathbf{c}_i)\| \leq 2^{\rho} + 2^{\rho_0}$ and the particular case holds. □

Lemma 5.3.10 (Sum of matrices)**.** *Let* $k \in \mathbb{Z}_{\geq 2}$. *For* $i \in [\![1, k]\!]$, *let* $\mathbf{C}_i$ *be an encryption of* $\mathbf{M}_i$. *Define* $\mathbf{C}$ *as the homomorphic sum* $\mathbf{C} := \sum_{i=1}^{k} \mathbf{C}_i \mod x_0$. *Then,* $\mathsf{err}(\mathbf{C}) = \sum_{i=1}^{k} \mathsf{err}(\mathbf{C}_i)$. *In particular, if all* $\mathbf{C}_i$*'s are fresh ciphertexts, then*

$$\|\mathsf{err}(\mathbf{C})\| \leq km\ell b(2^{\rho} + 2^{\rho_0}).$$

Proof. We can prove this Lemma using essentially the same arguments used in the proof of Lemma 5.3.9.

To simplify the notation, let $\mathbf{A} = G^{-1}(\alpha\mathbf{K}^{-1})$. Let $\mathbf{C}'_i := \mathbf{AC_iK} \mod x_0 = p\mathbf{AQ}_i + \mathbf{AR}_i + \alpha\mathbf{M}_i - x_0\mathbf{U}_i$. Then, the noise of $\mathbf{C}_i$ is $\mathsf{err}(\mathbf{C}_i) = \mathbf{AR}_i - r_0\mathbf{U}_i$.

Doing the same for $\mathbf{C}$, we have

$$\mathbf{C}' := \mathbf{ACK} \mod x_0 = p\mathbf{A}\sum_{i=1}^{k}\mathbf{Q}_i + \mathbf{A}\sum_{i=1}^{k}\mathbf{R}_i + \alpha\sum_{i=1}^{k}\mathbf{M}_i - x_0\sum_{i=1}^{k}\mathbf{U}_i$$

and it is easy to see that $\mathsf{err}(\mathbf{C}) = \mathbf{A}\sum_{i=1}^{k}\mathbf{R}_i - r_0\sum_{i=1}^{k}\mathbf{U}_i = \sum_{i=1}^{k}\mathsf{err}(\mathbf{C}_i)$.

If all $\mathbf{C}_i$ are fresh ciphertexts, then $\|\mathsf{err}(\mathbf{C}_i)\| \leq m\ell b(2^{\rho} + 2^{\rho_0})$ for $1 \leq i \leq k$ and the particular case holds.

□

Let's analyze the noise growth after a sequence of k vector-matrix products and show that computing homomorphically a ciphertext $\mathbf{c}_k$ that encrypts a product of the form $\mathbf{m}\left(\prod_{i=0}^{k-1}\mathbf{M}_i\right)$ increases the noise just linearly in k. Namely, using the bounds of lemmas 5.3.4 and 5.3.5 to approximate the noise of the vector ciphertext as $\|\mathsf{err}(\mathbf{c}_0)\| \approx \|\mathbf{r}_0\| + 2^{\rho_0}\|p\mathbf{q}_0\|/x_0$ and the noise of the ciphertexts encrypting the matrices as $\|\mathsf{err}(\mathbf{C}_i)\| \approx m\ell b(\|\mathbf{R}_i\| + 2^{\rho_0}\|p\mathbf{Q}_i\|/x_0)$, we see that the noise of the final ciphertext is $\|\mathsf{err}(\mathbf{c}_k)\| \approx mB(\|\mathsf{err}(\mathbf{c}_0)\| + \sum_{i=0}^{k-1}\|\mathsf{err}(\mathbf{C}_i)\|)$. Notice that the noise growth is similar to the one of [GGH+19].

Lemma 5.3.11 (Products of vectors and matrices)**.** *Let $k \in \mathbb{Z}_{\geq 2}$. For all $i \in [\![1,k]\!]$, let $\mathbf{C}_i$ be an encryption of $\mathbf{M}_i$. Let also $\mathbf{c}_0$ be an encryption of $\mathbf{m}_0$. Assume that B is an upper bound to the entries of the product of plaintext matrices, i.e., $\left\|\prod_{i=j}^{k-1}\mathbf{M}_i\right\| \leq B$ for $0 \leq j \leq k-1$. Finally, for $1 \leq i \leq k-1$, define $\mathbf{c}_{i+1} := G^{-1}(\mathbf{c}_i)\cdot\mathbf{C}_i \mod x_0$. Then,*

$$\|\mathsf{err}(\mathbf{c}_k)\| < mB\cdot(\underbrace{\|\mathbf{r}_0\| + 2^{\rho_0}\|p\mathbf{q}_0\|/x_0}_{\approx\|\mathsf{err}(\mathbf{c}_0)\|} + \sum_{i=0}^{k-1}\underbrace{m\ell b\,(\|\mathbf{R}_i\| + 2^{\rho_0}\|p\mathbf{Q}_i\|/x_0)}_{\approx\|\mathsf{err}(\mathbf{C}_i)\|}) + 2^{\rho_0}. \quad (5.1)$$

In particular, if $\mathbf{c}_0$ and all the $\mathbf{C}_i$'s are fresh ciphertexts, then

$$\|\mathsf{err}(\mathbf{c}_k)\| < mB(2^{\rho} + 2^{\rho_0} + km\ell b(2^{\rho}+2^{\rho_0})) + 2^{\rho_0}. \quad (5.2)$$

Proof. By the analysis done in Section 5.3.4, we know that the term $\mathbf{r}_{i+1}$ of $\mathbf{c}_{i+1}$ is $G^{-1}(\mathbf{c}_i)\mathbf{R}_i + \mathbf{r}_i\mathbf{M}_i$. Therefore, the term $\mathbf{r}_k$ after k homomorphic products is

$$\mathbf{r}_k = \mathbf{r}_0\prod_{i=0}^{k-1}\mathbf{M}_i + \sum_{i=0}^{k-1}G^{-1}(\mathbf{c}_i)\mathbf{R}_i\left(\prod_{j=i+1}^{k-1}\mathbf{M}_j\right).$$

Thus,

$$\begin{aligned}\|\mathbf{r}_k\| &\leq \left\|\mathbf{r}_0\prod_{i=0}^{k-1}\mathbf{M}_i\right\| + \left\|\sum_{i=0}^{k-1}G^{-1}(\mathbf{c}_i)\mathbf{R}_i\left(\prod_{j=i+1}^{k-1}\mathbf{M}_j\right)\right\| \\ &\leq m\|\mathbf{r}_0\|\left\|\prod_{i=0}^{k-1}\mathbf{M}_i\right\| + \sum_{i=0}^{k-1}m\ell\left\|G^{-1}(\mathbf{c}_i)\right\|\left\|\mathbf{R}_i\left(\prod_{j=i+1}^{k-1}\mathbf{M}_j\right)\right\| \\ &\leq mB\|\mathbf{r}_0\| + \sum_{i=0}^{k-1}m^2\ell bB\|\mathbf{R}_i\|\end{aligned}$$

Similarly,

$$\|\mathbf{q}_k\| \leq mB\,\|\mathbf{q}_0\| + \sum_{i=0}^{k-1} m^2\ell bB\,\|\mathbf{Q}_i\|\,.$$

Thus, we obtain inequality (5.1) from Lemma 5.3.4, because

$$\|\mathsf{err}(\mathbf{c}_k)\| < \|\mathbf{r}_k\| + 2^{\rho_0}\left\lceil \frac{\|p\mathbf{q_k}\|}{x_0}\right\rfloor \leq \|\mathbf{r}_k\| + \frac{2^{\rho_0}}{x_0}\,\|p\mathbf{q_k}\| + 2^{\rho_0}.$$

If all the operands are fresh ciphertexts, then both $\|\mathbf{r}_0\|$ and $\|\mathbf{R}_i\|$ are bounded by 2^ρ and both $\|p\mathbf{q}_0\|$ and $\|\mathbf{pQ}_i\|$ are bounded by x_0, therefore, the particular case also holds. □

When we compute a sequence of k homomorphic products like $\prod_{i=0}^{k}\mathbf{M}_i$, the noise growth is basically the same as the one described in lemma 5.3.11, that is, approximately from $\beta := m\ell b(2^\rho + 2^{\rho_0})$ to $kmB\beta$.

Lemma 5.3.12 (Products of matrices)**.** *Let k be an integer bigger than 1. For $i \in [\![0,k]\!]$, let $\mathbf{C}_i$ be an encryption of $\mathbf{M}_i$. Let also $\mathbf{C}'_0 := \mathbf{C}_0$, $\mathbf{C}'_i := G^{-1}(\mathbf{C}'_{i-1})\mathbf{C}_i \mod x_0$ for $i > 0$. (Notice that $\mathbf{C}'_i$ is an encryption of $\prod_{j=0}^{i}\mathbf{M}_j$). Assume that B is an upper bound to the entries of the product of plaintext matrices, i.e., $\left\|\prod_{i=j}^{k}\mathbf{M}_i\right\| \leq B$ for $1 \leq j \leq k$. Then,*

$$\left\|\mathsf{err}(\mathbf{C'}_k)\right\| < mB\cdot(\underbrace{\|\mathbf{R}_0\| + 2^{\rho_0}\,\|p\mathbf{Q}_0\|\,/x_0}_{\approx\|\mathsf{err}(\mathbf{C}_0)\|} + \sum_{i=1}^{k}\underbrace{m\ell b\,(\|\mathbf{R}_i\| + 2^{\rho_0}\,\|p\mathbf{Q}_i\|\,/x_0))}_{\approx\|\mathsf{err}(\mathbf{C}_i)\|} + 2^{\rho_0}.$$

In particular, if all the products only involve fresh ciphertexts, then

$$\left\|\mathsf{err}(\mathbf{C}'_k)\right\| < mB(2^\rho + 2^{\rho_0} + km\ell b(2^\rho + 2^{\rho_0})) + 2^{\rho_0}.$$

Proof. Similar to the proof of Lemma 5.3.11. □

5.4 Security analysis

5.4.1 Hardness of the AGCD problem implies semantic security

In this section we prove that our scheme is CPA secure under the assumption that decisional AGCD problem is computationally hard. We first prove that for $x_0 > 2^{\gamma-1}$ the

hardness of the AGCD problem implies that the samples of $\mathcal{D}_{<x_0}$ are indistinguishable of values sampled uniformly from $\mathbb{Z}_{x_0}$ (recall that $\mathcal{D}_{<x_0}$ is the distribution $\mathcal{D}_{\gamma,\rho}(p)$ with samples bigger than or equal to x_0 being rejected). Then, using a sequence of hybrids, we prove the indistinguishably of encrypted matrices. With essentially the same proof, we can show that encryptions of any pair of vectors are also indistinguishable. Finally, those two results imply CPA security.

Lemma 5.4.1. *Let $x_0 > 2^{\gamma-1}$. Under the decisional AGCD assumption, the distributions $\mathcal{D}_{<x_0}$ and $\mathcal{U}(\mathbb{Z}_{x_0})$ are computational indistinguishable.*

Proof. The proof is adapted from Lemma 2.3 of [BBL17].

We can sample efficiently from $\mathcal{U}(\mathbb{Z}_{x_0})$ by sampling from $\mathcal{U}(\llbracket 0, 2^\gamma \rrbracket)$ and rejecting samples that are larger than or equal to x_0. Since $x_0 > 2^{\gamma-1}$, the probability that a sample is rejected is less than $1/2$, therefore, sampling from $\mathcal{U}(\mathbb{Z}_{x_0})$ uses only a constant number of calls to $\mathcal{U}(\llbracket 0, 2^\gamma \rrbracket)$. Replacing $\mathcal{U}(\llbracket 0, 2^\gamma \rrbracket)$ by $\mathcal{D}_{\gamma,\rho}(p)$ gives us an efficient sampling procedure to $\mathcal{D}_{<x_0}$.

Moreover, if one can distinguish between $\mathcal{D}_{<x_0}$ and $\mathcal{U}(\mathbb{Z}_{x_0})$, then it is also possible to distinguish between $\mathcal{D}_{\gamma,\rho}(p)$ and $\mathcal{U}(\llbracket 0, 2^\gamma \rrbracket)$ simply by using this sampling procedure and deciding whether the produced distribution is $\mathcal{D}_{<x_0}$ or $\mathcal{U}(\mathbb{Z}_{x_0})$. Therefore, the decisional AGCD assumption implies that $\mathcal{D}_{<x_0}$ is computational indistinguishable from $\mathcal{U}(\mathbb{Z}_{x_0})$. □

Lemma 5.4.2. *Under the decisional AGCD assumption, encryptions of any pair of matrices are computationally indistinguishable.*

Proof. Via a sequence of hybrids we prove that no PPT adversary $\mathcal{A}$ can distinguish between encryptions of two matrices $\mathbf{M}_0$ and $\mathbf{M}_1$ of their choice.

Hybrid H_0: Use the key generation function to get $\mathsf{sk} = (p, \mathbf{K})$ and the public parameters params. Given $\mathbf{M}_0$ and $\mathbf{M}_1$ chosen by $\mathcal{A}$, we always encrypt $\mathbf{M}_0$, that is, we let $\mathbf{C}_0 := \mathsf{HE.EncMat}(\mathsf{sk}, \mathbf{M}_0)$ and return $\mathbf{C}_0$ to $\mathcal{A}$. ◇

Hybrid H_1: The only difference between this hybrid and H_0 is that we use $\mathcal{U}(\mathbb{Z}_{x_0})$ instead of $\mathcal{D}_{<x_0}$ to encrypt $\mathbf{M}_0$, i.e., we sample $\mathbf{X}_1 \leftarrow \mathcal{U}(\mathbb{Z}_{x_0})^{m\ell \times m}$ and define $\mathbf{C}_1 := (\mathbf{X}_1 + \mathbf{GKM}_0)\mathbf{K}^{-1} \mod x_0$. ◇

Since in H_0 we have $\mathbf{C}_0 := (\mathbf{X}_0 + \mathbf{GKM})\mathbf{K}^{-1} \mod x_0$ for some $\mathbf{X}_0 \leftarrow (\mathcal{D}_{<x_0})^{m\ell \times m}$, then by Lemma 5.4.1, it holds that

$$\left|\Pr_{H_0}[\mathcal{A}(1^\lambda, \mathsf{params}, \mathbf{C}_0)] - \Pr_{H_1}[\mathcal{A}(1^\lambda, \mathsf{params}, \mathbf{C}_1)]\right| \leq \mathrm{negl}(\lambda)\,.$$

Hybrid H_2: In this hybrid, we ignore the two plaintext matrices, we sample $\mathbf{X}_2 \leftarrow \mathcal{U}(\mathbb{Z}_{x_0})^{m\ell \times m}$, and define $\mathbf{C}_2 := \mathbf{X}_2$. ◇

We know that for any $t \in \mathbb{Z}_{x_0}$, $\mathcal{U}(\mathbb{Z}_{x_0})$ and $\mathcal{U}(\mathbb{Z}_{x_0}) + t \mod x_0$ are the same distribution. Consequently, $\mathbf{X}_1 + \mathbf{GKM}_0$ follows $\mathcal{U}(\mathbb{Z}_{x_0})^{m\ell \times m}$. Additionally, multiplying by an invertible element also does not change the distribution, thus, $\mathbf{C}_1 = (\mathbf{X}_1 + \mathbf{GKM}_0)\mathbf{K}^{-1} \mod x_0$ follows $\mathcal{U}(\mathbb{Z}_{x_0})^{m\ell \times m}$ as well. Therefore, $\mathbf{C}_1$ and $\mathbf{C}_2$ are indistinguishable.

Hybrid H_3: In this hybrid, we encrypt $\mathbf{M}_1$ using $\mathcal{U}(\mathbb{Z}_{x_0})$, i.e., we sample $\mathbf{X}_3 \leftarrow \mathcal{U}(\mathbb{Z}_{x_0})^{m\ell \times m}$ and define $\mathbf{C}_3 := (\mathbf{X}_3 + \mathbf{GKM}_1)\mathbf{K}^{-1} \mod x_0$. ◇

By the same argument used in the transition from H_1 to H_2, we see that $\mathbf{C}_2$ and $\mathbf{C}_3$ are indistinguishable, therefore, $\mathcal{A}$'s advantage in distinguishing H_2 from H_3 is negligible.

Hybrid H_4: In this hybrid, we replace $\mathcal{U}(\mathbb{Z}_{x_0})$ with $\mathcal{D}_{<x_0}$ to get a valid encrypt of $\mathbf{M}_1$, that is, we define $\mathbf{C}_4 := \mathsf{HE.EncMat}(\mathsf{sk}, \mathbf{M}_1)$ and return $\mathbf{C}_4$ to $\mathcal{A}$. ◇

Using Lemma 5.4.1 again, we conclude that $\mathcal{A}$'s advantage in distinguishing between hybrids 3 and 4 is also negligible. Since $\mathcal{A}$'s advantage in each transition is negligible, it holds that

$$\left|\Pr_{H_0}[\mathcal{A}(1^\lambda, \mathsf{params}, \mathbf{C}_0)] - \Pr_{H_4}[\mathcal{A}(1^\lambda, \mathsf{params}, \mathbf{C}_4)]\right| \leq \mathrm{negl}(\lambda)\,.$$

But this is exactly the definition of $\mathcal{A}$'s advantage in distinguishing encryptions of $\mathbf{M}_0$ from encryptions of $\mathbf{M}_1$. □

Lemma 5.4.3. *Under the decisional AGCD assumption, encryptions of any pair of vectors are computationally indistinguishable.*

Proof. We can use basically the same sequence of hybrids used in the proof of Lemma 5.4.2, but replacing matrices by vectors and HE.EncMat by HE.EncVec. □

Theorem 5.4.4. *The scheme is CPA-secure under the decisional-AGCD assumption.*

Proof. This follows directly from Lemma 5.4.2 and Lemma 5.4.3. □

5.4.2 Parameter selection

In this section, we briefly recall the attacks on the vector AGCD problem presented in Section 3.5 and find the constraints that they impose to the parameters.

Firstly, let's recall the role of the main parameters:

- η: it is the bit-length of the secret prime p;
- ρ: the noise of fresh ciphertexts is bounded by 2^ρ;
- ρ_0: the noise r_0 of x_0 satisfies $-2^{\rho_0} < r_0 < 2^{\rho_0}$;
- γ: the entries of the vectors and matrices ciphertexts are bounded by 2^γ;
- B: we must have $\|\mathbf{m}\| \leq B$ and $\|\mathbf{M}\| \leq B$ for any plaintext $\mathbf{m}$ or $\mathbf{M}$;
- m: it is the dimension of the vectors and matrices to be encrypted;
- b: it is the base in which the decomposition G^{-1} is performed;
- ℓ: it is defined as $\lceil \log_b(2^\gamma) \rceil$, thus, it is the number of words used in G^{-1}. The matrix ciphertexts have dimension $m\ell \times m$.

Taking into account the analysis of the orthogonal lattice attack, we see that we can choose $\gamma = \lceil \lambda(\eta - \rho)^2/(m \log \lambda) \rceil$. But when m is close to λ, we can have $\gamma < 2\eta$, and in this case we simply choose $\gamma = 2\eta$. Those two scenarios are very distinct, thus, let's first analyze the case $\gamma > 2\eta$.

For the correctness, we just have to guarantee that the inequality (5.2) is satisfied. It basically means that we can choose ρ, ρ_0 and b such that

$$\eta - 2\log m - \log k - \log \ell - \log B = \max(\rho, \rho_0) + \log b. \tag{5.3}$$

Typically, we will have $\rho \geq \rho_0$, thus, if B is somehow small, we are free to choose $\rho + \log b \approx \eta$, say $\rho + \log b = (1-\epsilon)\eta$ for some $\epsilon \in \ ]0,1[$. Using $\eta - \rho = \epsilon\eta + \log b$ we can express the size of encrypted matrices as

$$m^2\ell\gamma \approx \frac{m^2\gamma^2}{\log b} \approx \frac{\lambda^2}{\log^2 \lambda}\frac{(\eta-\rho)^4}{\log b} = \frac{\lambda^2}{\log^2 \lambda}\frac{(\epsilon\eta + \log b)^4}{\log b}$$

which is minimized when $\log b = \epsilon\eta/3$.

The cost of a homomorphic evaluation of a product like $\mathbf{m}\prod_{i=1}^{k}\mathbf{M}_i$ is dominated by the k vector-matrix multiplications, which cost approximately $km^2\ell\gamma$, therefore, it is also minimized when $\log b = \epsilon\eta/3$. The encryption cost is dominated by the matrices multiplications, which cost $m^3\ell\gamma$. Thus, it can be approximated by

$$m^3\ell\gamma \approx m^3\frac{\gamma^2}{\log b} = \frac{m}{\log b}\left(\frac{(\eta-\rho)^2\lambda}{\log \lambda}\right)^2 = \frac{m\lambda^2}{\log^2 \lambda}\frac{(\eta-\rho)^4}{\log b}$$

which is also minimized when $\log b = \epsilon\eta/3$.

Therefore, in order to choose the parameters, we first set the desired security level λ. For usual applications, the noise factor $2\log m + \log k + \log B$ in equation (5.3) is small and it is sufficient to take $\eta = \lambda$. If it is not the case, we can choose $\eta = \lambda + c$ for some positive constant c. Once we have defined η, we use equation (5.3) to estimate ϵ, for instance, taking $\epsilon = 2(\log k + \log B + \log m)/\eta$. Then, we set $\rho = \lfloor 2\epsilon\eta/3 \rfloor$ and $\log b = \lfloor \epsilon\eta/3 \rfloor$.

For security reasons, we must ensure that $T_{GCD,x_0}(\eta, \rho, \rho_0, \gamma, n) \geq 2^\lambda$ and $T_{FAC}(\eta, \rho_0, \gamma) \geq 2^\lambda$. In general, we can find a $\rho_0 \leq \rho$ such that these two constraints are satisfied. If there is no such ρ_0, then we can increase η and choose all the parameters again. Notice that we choose ρ close to η, generally bigger than what we would need to guarantee the security, because it decreases the size of γ, which makes the operations cheaper. However if m is big enough for us to choose $\gamma = 2\eta$, then there is no advantage in choosing a big ρ. In this case, we simple choose the minimum ρ and ρ_0 such that

Table 5.1: Proposed sets of parameters for two levels of security, considering that x_0 is public. The value of ℓ is always $\lceil \log_b(2^\gamma) \rceil$ and α must be set to $\lfloor 2^{\eta-1}/(2B+1) \rfloor$ where B defines the plaintext space.

λ	η	m	ρ	ρ_0	γ	$\log b$
80	80	$8 \leq m \leq 64$	52	38	$\lceil 80 \cdot 28^2/m \log(80) \rceil$	7
		128	40	40	2η	13
		256	23	40	2η	14
		512	2	40	2η	14
		1024	2	40	2η	15
100	100	$8 \leq m \leq 52$	73	58	$\lceil 100 \cdot 27^2/m \log(100) \rceil$	7
		64	71	59	2η	11
		128	59	59	2η	17
		256	43	59	2η	17
		512	19	59	2η	17
		1024	2	59	2η	16

$T_{FAC}(\eta, \rho_0, \gamma) \geq 2^\lambda$, $T_{GCD,x_0}(\eta, \rho, \rho_0, \gamma, m) \geq 2^\lambda$, and $\lceil \lambda(\eta-\rho)^2/(m \log \lambda) \rceil < 2\eta$, then we choose $\log b = (1-\epsilon)\eta - \max(\rho, \rho_0)$, i.e., we decrease ρ and ρ_0 as much as the security allows us, and we increase $\log b$ respecting the correctness condition.

In Table 5.1, we propose some sets of parameters for two security levels ($\lambda = 80$ and $\lambda = 100$) and several values of m. We see that increasing m allows us to choose smaller ρ maintaining the same security, but ρ_0 does not depend on m and it is basically fixed. As a consequence, we cannot increase $\log b$ too much as we decrease ρ in the regime $\gamma = 2\eta$, because when ρ becomes smaller than ρ_0, the final noise begins to be dominated by ρ_0 and we must then respect the constraint $\log b + \rho_0 = (1-\epsilon)\eta$. In Section 5.7, we attempt to resolve this problem by proposing a simple variant of the scheme that has better parameters for large m.

5.5 Implementation and general performance

In this section, we show basic practical results, like running times of encryption functions and bit-length of ciphertexts. We implemented a proof of concept of our scheme in C++ using the NTL library, version 11.3.2, and made it publicly available in the GitHub repository [Per20c]. All the experiments were run on a machine with the GNU/Linux operating system Ubuntu 18.04.2 LTS, 32GB of RAM memory, and processor Intel Core i5-8600K 3.60 GHz. One single core was used. We ran the experiments using parameters for the two different levels of security $\lambda = 80$ and $\lambda = 100$ described in Table 5.1.

The running times and the size of the encrypted matrices are shown in figures 5.1 and 5.2. Since for small values of m we have both γ and ℓ proportional to $1/m$, and the bit-length of a matrix ciphertext is $m^2\ell\gamma$, the size of the encrypted matrices and also the encryption and decryption times are approximately constant as we increase m, until we switch to the regime of parameters that uses $\gamma = 2\eta$. From this point, the efficiency of the scheme starts to deteriorate, but it is still very good even for moderate values of m. For instance, for $\lambda = 80$, it takes less than 2.5 seconds to encrypt a 150×150 matrix and we need less than 6 MB to represent the corresponding ciphertext. Even considering that the plaintext matrix is binary, we are encrypting 150^2 bits into 6 MB, which corresponds to a ciphertext expansion of 0.266 KB per encrypted bit. As a comparison, for 80 bits of security, the basic scheme of [BBL17] encrypts a single bit into a 19 MB ciphertext, and the batched version, that uses the Chinese Remainder Theorem to encrypt several bits into a single ciphertext, encrypts roughly 70 bits into the same 19 MB, which represents a ciphertext expansion of 217 KB per encrypted bit.

5.6 Some applications

5.6.1 Nondeterministic finite-state automaton evaluation

In this section we show how to homomorphically evaluate finite state automaton using our scheme. We represent an m-state automaton A over an alphabet Σ by $m \times m$ transition

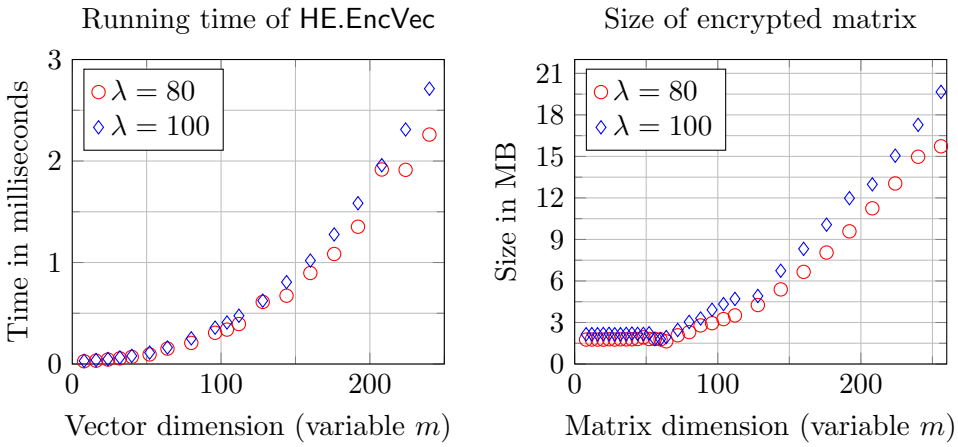

Figure 5.1: Running times to encrypt an m-dimensional vector and size of encrypted matrix.

matrices $\mathbf{M}_a$ for each $a \in \Sigma$. They are defined as follows: each entry (i, j) of $\mathbf{M}_a$ is equal to 1 if A has a transition from state i to state j using the letter a, and it is equal to 0 otherwise. Additionally, we need an m-dimensional vector $\mathbf{m}$ to represent the current states. At any point of the evaluation, $m_i = 0$ if we are not in state i, and $m_i \geq 1$ if we are in state i.

If A is deterministic, we are always at one single state, then $\mathbf{m} \in \{0, 1\}^m$ and there is a unique position i such that $m_i = 1$. However, if A is nondeterministic, then we can be in several states at the same time and $\mathbf{m}$ may have multiples non-zero entries, which can be larger than 1. In particular, it implies that products of transition matrices are always binary for deterministic automata, hence, the noise accumulated on the homomorphic evaluation is typically smaller, i.e., we can set the parameter B to be one, while with nondeterministic automata, we may need $B > 1$. However, in general we need fewer states to represent a language using non-deterministic automata.

Evaluating an automaton represented in this way is done as follows: we start with a state vector $\mathbf{m}_0$ that has ones in the positions corresponding to initial states and zeros elsewhere. Then, given a length-k input string $\mathbf{s} \in \Sigma^k$, at each step i (from 1 to k), we look at the i-th letter s_i and update the state vector as $\mathbf{m}_i = \mathbf{m}_{i-1}\mathbf{M}_{s_i}$. If $\mathbf{m}_k$ has a non-zero entry in some position i corresponding to an accepting state of A, then the input

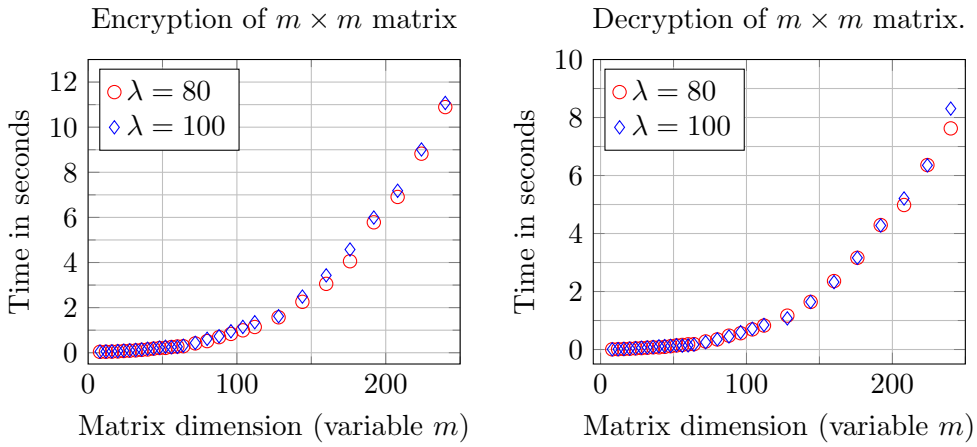

Figure 5.2: Running times of HE.EncMat and HE.DecMat.

string is said to be accepted by A. Hence, to evaluate an NFA homomorphically, it is sufficient to perform homomorphic vector-matrix products.

As a possible application of homomorphic evaluation of NFA, we can imagine a server that holds input strings, for instance, text files, and a user that wants to retrieve the files that contain strings respecting some regular expression R, but without revealing R to the server. For example, to get files that contain an e-mail address of someone from the University of Luxembourg, the user could use $R =$ [a-z][a-z0-9][a-z0-9]*@uni.lu, for which we can construct the automaton with 10 states represented in Figure 5.3.

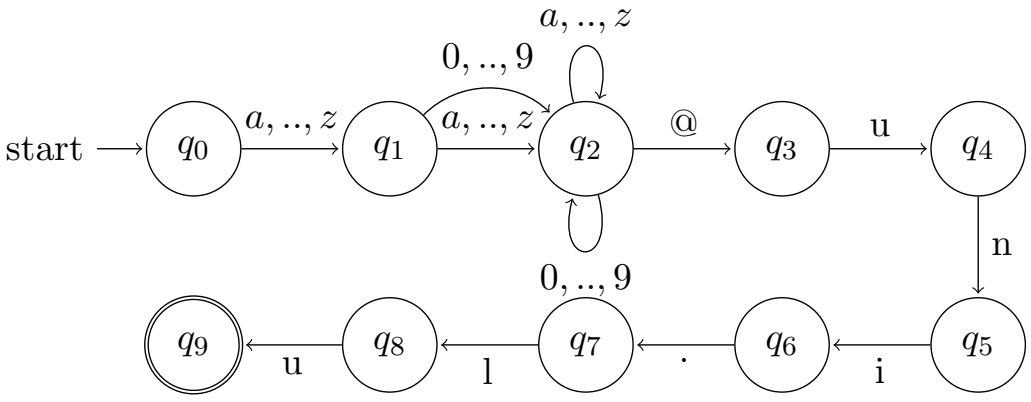

Figure 5.3: An NFA corresponding to the regular expression [a-z][a-z0-9][a-z0-9]*@uni.lu.

Then, the user would encrypt the 10×10 transition matrices and send them to the server, that would evaluate the NFA homomorphically on each file f_i, generating an encrypted state vector $\mathbf{c}_i$, and return each $\mathbf{c}_i$ to the user. Finally, the user could decrypt each $\mathbf{c}_i$ to check if the file f_i matches R. Notice that the scenarios considered in papers about homomorphic encryption generally assume that the server has a program and the user will encrypt the input, which is the converse of what we just proposed, since the program, i.e, the automaton, is encrypted by the client and the input is known in clear by the server.

In the article [GGH+19], the authors construct a homomorphic scheme for NFA evaluation. In order to compare the results of this section with their results, we use the same family of automata and the same security level used there (namely, $\lambda = 100$). Thus, let's consider the regular language $L_m := (a+b)^* a (a+b)^{m-2}$. It is known that one needs at least 2^{m-1} states to represent L_m with a deterministic automaton, however, we can represent it with a nondeterministic automaton with m states [MF71].

We evaluated L_m homomorphically for various values of m and k, using always random input strings sampled from $\{a, b\}^k$. The experiments of [GGH+19] were executed on an Intel i7-2600 3.4 GHz CPU while ours were run on an Intel Core i5-8600K 3.60 GHz machine. The practical results are summarized in Table 5.2. For m up to 100, our scheme is more efficient than [GGH+19] in terms of running time and of memory requirements. For $m = 128$, our ciphertext size and encryption times are better, but the evaluation times start to be worse then theirs. Then, for bigger values of m, our scheme is less efficient. Notice that in their scheme, the variable m has a double role, acting as the security parameter and as the number of states at the same time. Moreover, to achieve a security level of 100 bits, they set $m = 1024$. Hence, to evaluate automata with less than 1024 states, they must embed the low-dimensional transition matrices into 1024×1024 matrices. In particular, it means that for all the values of m presented in Table 5.2, their scheme would use 33 MB per encrypted matrix and around 16.5 seconds to encrypt L_m. It is also worth to stress that they use an *ad hoc* hardness assumption while our scheme is based on the AGCD problem, which is more standard.

Table 5.2: Practical results of the homomorphic evaluation of L_n on input strings with k letters. All running times are presented in seconds. The second column shows the size of each encrypted matrix. The third column shows the time needed to encrypt the entire automaton (two transition matrices and the state vector). All the last six columns show the running time to perform a match on a string of length $k = 2^i$, for $5 \leq i \leq 10$. Parameters used: setting $\lambda = 100$ from Table 5.1. The last row shows the corresponding practical results presented in [GGH+19]. For all n up to 1024, their scheme have the same encryption and evaluation times, and also ciphertext size.

n	Encrypted matrix	Encryption time	Evaluation time on inputs of length k					
			32	64	128	256	512	1024
8	2.15 MB	0.10	0.028	0.06	0.12	0.24	0.47	0.96
16	2.15 MB	0.12	0.041	0.08	0.17	0.34	0.67	1.34
32	2.15 MB	0.20	0.065	0.13	0.27	0.53	1.08	2.15
64	1.94 MB	0.44	0.083	0.17	0.33	0.67	1.33	2.67
128	4.91 MB	2.20	0.240	0.49	0.98	1.97	3.9	7.87
256	19.66 MB	19.15	1.138	2.27	4.55	9.12	18.35	36.88
512	78.64 MB	202.60	5.235	10.4	20.8	41.7	83.6	167.8
1024	340.78 MB	2211.96	44.061	86.3	174.0	352.3	704.4	1414
≤ 1024	33 MB	16.5	-	-	-	1.53	3.34	6.63

5.6.2 Naïve Bayes Classification

As a second application, we implemented a homomorphic classifier. In this scenario, the server has a trained model and the client has some data (instances) to be classified. The client then encrypt each instance and send it to the server, which evaluates the model homomorphically and returns to the client an encryption of the assigned class.

Each instance has a fixed number of attributes, say k, and is represented by a vector $\mathbf{y} = (y_1, ..., y_k)$. For example, we could have a data set about patients containing medical information as the three following attributes: x_1 = "blood type", x_2 = "age", and x_3 = "body mass index". And an instance could be $\mathbf{y} = (O+, 47, 22)$. The number of possible classes is fixed and typically small (say, between two and fifty).

In the Naïve Bayes Classification, we use an already classified data set, called training set, to estimate the probabilities that each attribute x_i is equal to y_i given that the class is c, that is, $\Pr[x_i = y_i | \text{ class} = c]$, and also the probabilities $\Pr[\text{class} = c]$. Then, to classify $\mathbf{y}$, we just compute each $\Pr[\text{class}(\mathbf{y}) = c]$ and assign the class c for which this

probability is the biggest. Notice that for each class c, the probability $\Pr[\text{class}(\mathbf{y}) = c]$ is equal to

$$\Pr[\text{class} = c | x_1 = y_1, ..., x_k = y_k] = \frac{\Pr[x_1 = y_1, ..., x_k = y_k | \text{class} = c] \Pr[class = c]}{\Pr[x_1 = y_1, ..., x_k = y_k]}.$$

Moreover, because we are only interested in relative values, that is, which probability is the biggest, not on their actual values, we can ignore the denominator, since it is a constant that does not depend on c. Furthermore, using the "naïve" hypotheses of independence of attributes, we can replace $\Pr[x_1 = y_1, ..., x_k = y_k | \text{class} = c]$ by $\prod_{i=1}^{k} \Pr[x_i = y_i | \text{class} = c]$. Finally, we apply the logarithm function so that we can replace products by additions. This does not change the output of the classifier because log is an increasing function, thus, if a probability p_{c_1} is bigger than another probability p_{c_2}, it holds that $\log(p_{c_1}) > \log(p_{c_2})$, and the classifier will still assign the class c_1.

Hence, the instances are classified based on the following formula:

$$\beta_{\mathbf{y},c} := \log(\Pr[\text{class} = c]) + \sum_{i=1}^{k} \log(\Pr[x_i = y_i | \text{class} = c]).$$

To efficiently evaluate this classifier with our scheme, we assume that each attribute x_i can be represented by the set $[\![1, m_i]\!]$, for some m_i, and we define $m := \max\{m_1, ..., m_k\}$. If we try to encrypt and classify one instance per time, we end up with a solution that is not optimized in terms of space. For instance, at the end of the classification, the server has to send to the client a ciphertext encrypting $(\beta_{\mathbf{y},c}, 0, ..., 0)$, thus, $m - 1$ entries are not used. Therefore, we propose to classify m instances simultaneously. To represent the instances, we use "indicator vectors", that is, we define $\psi(i) := \mathbf{e}_i \in \{0, 1\}^m$ to be vector with a single 1 in the i-th entry. Then, given an instance $\mathbf{y} = (y_1, ..., y_k)$, notice that $\psi(y_i)$ is a vector whose non-zero entry indicates the value of the i-th attribute of $\mathbf{y}$. Also, notice that by defining the vector $\mathbf{v} := (\log(\Pr[x_i = 1 | \text{class} = c]), ..., \log(\Pr[x_i = m | \text{class} = c]))$, it holds that $\mathbf{v} \cdot \psi(y_i) = \log(\Pr[x_i = y_i | \text{class} = c])$.

Considering that, the protocol to perform homomorphic classification is the following:

- Client's setup: Compute $\mathsf{sk} = \mathsf{HE.KeyGen}$ and $\tilde{\mathbf{e}}_i := \mathsf{HE.EncVec}(\mathbf{e}_i)$ for $1 \leq i \leq m$, then send $(\tilde{\mathbf{e}}_1, ..., \tilde{\mathbf{e}}_m)$ to the server.

- Server's setup: Assume that all the logarithms are scaled to integers. For each class c, compute $\tilde{\mathbf{p}}_c = \sum_{i=1}^{m} \log(\Pr[\text{class} = c]) \cdot \tilde{\mathbf{e}}_i$ and $\tilde{\mathbf{p}}_{i,c} = \sum_{j=1}^{m} \log(\Pr[x_i = j | \text{class} = c]) \cdot \tilde{\mathbf{e}}_j$. Precompute each decomposition $G^{-1}(\tilde{\mathbf{p}}_{i,c})$.

- Client's query: Consider m instances of the form $\mathbf{y}^{(j)} = (y_1^{(j)}, ..., y_k^{(j)})$. Construct one matrix $\mathbf{Y}_s$ for each attribute x_s:

$$\mathbf{Y}_s = \left(\phi\left(y_s^{(1)}\right) \quad \dots \quad \phi\left(y_s^{(m)}\right)\right) \in \{0,1\}^{m \times m}.$$

  For $1 \leq s \leq k$, do $\tilde{\mathbf{Y}}_s := \mathsf{HE.EncMat}(\mathbf{Y}_s)$. Finally, send $\tilde{\mathbf{Y}}_1, ..., \tilde{\mathbf{Y}}_k$ to the server.

- Server's classification: For each c, do $\tilde{\mathbf{b}}_c := \tilde{\mathbf{p}}_c + \sum_{s=1}^{k} G^{-1}(\tilde{\mathbf{p}}_{s,c})\tilde{\mathbf{Y}}_s \mod x_0$. Return all the $\tilde{\mathbf{b}}_c$'s to the client. If there are only two possible classes, then return $\tilde{\mathbf{b}} := \tilde{\mathbf{b}}_0 - \tilde{\mathbf{b}}_1$ instead.

- Client's decoding: If there are more than two possible classes, then for each c, do $\mathbf{b}_c := \mathsf{HE.DecVec}(\tilde{\mathbf{b}}_c)$, and for each j, select the maximum among the j-th entry of each $\mathbf{b}_c$ and assign to $\mathbf{y}^{(j)}$ the class c corresponding to that maximum. If there are only two possible classes, do $\mathbf{b} := \mathsf{HE.DecVec}(\tilde{\mathbf{b}})$, assign class 0 to $\mathbf{y}^{(j)}$ if the j-th entry of $\mathbf{b}$ is positive and class 1 otherwise.

Notice that in the server's setup, each $\tilde{\mathbf{p}}_{i,c}$ encrypts $(\log(\Pr[x_i = 1|\text{class} = c]), ..., \log(\Pr[x_i = m|\text{class} = c]) \in \mathbb{Z}^m$, thus, when we multiply it homomorphically with $\mathbf{Y}_i$, we obtain an encryption of $(\log(\Pr[x_i = y_i^{(1)}|\text{class} = c]), ..., \log(\Pr[x_i = y_i^{(m)}|\text{class} = c])$. Thus, doing the k products and executing the sum produces an encryption of

$$\left(\sum_{s=1}^{k} \log(\Pr[x_s = y_s^{(1)}|\text{class} = c]), ..., \sum_{s=1}^{k} \log(\Pr[x_s = y_s^{(m)}|\text{class} = c]\right) \in \mathbb{Z}^m.$$

Finally, because each vector $\tilde{\mathbf{p}}_c$ encrypts $(\log(\Pr[\text{class} = c], ..., \log(\Pr[\text{class} = c]) \in \mathbb{Z}^m$, it is easy to see that each $\tilde{\mathbf{b}}_c$ is an encryption of $\mathbf{b}_c = (\beta_{\mathbf{y}^{(1)},c}, ..., \beta_{\mathbf{y}^{(m)},c})$, therefore, we are assigning the classes based on the correct formula.

We have implemented this protocol and executed it using the Breast Cancer Wisconsin (Diagnostic) Data Set[3], which is a data set with two classes, benign and malignant, and

[3]From UCI's Machine Learning Data Sets Repository: archive.ics.uci.edu/ml

Table 5.3: Homomorphic evaluation of Naïve Bayes Classifier on Breast Cancer Wisconsin Data Set for two security levels. Columns Classification, Upload, and Download show values per instance.

λ	Client				Server	
	Setup	Classification	Upload	Download	Setup	Classification
80	1 ms	34.3 ms	46 KB	0.13 KB	5 ms	1.44 ms
100	1 ms	45.36 ms	49 KB	0.14 KB	5 ms	1.66 ms

nine attributes about tumors (like "Clump Thickness" and "Uniformity of Cell Shape"), each one with ten possible values, therefore, we used $k = 9$ and $m = 10$. We randomly partitioned the data set several times into a training set containing approximately two thirds of the data and a testing set with the remaining one third. The logarithms of the probabilities were computed and multiplied by 10^5 to scale to integers. Then we executed the homomorphic classification for the two parameter sets proposed in Table 5.1 (with $m = 10$ and $B = 2^{19}$). We also executed a normal Naïve Bayes Classifier over the plaintext, obtaining always the same accuracy for the clear text and the homomorphic versions, around 90%, which means that we did not lose precision by evaluating the classifier homomorphically. We summarize the results in Table 5.3. The protocol is very efficient, as the amount of data that each party needs to send over the network is just few kilobytes per classified instance and the running times are just few milliseconds. When compared with other papers that propose Naïve Bayes classification over encrypted data, our solution seems to be more straightforward and to run faster, although the comparisons are not trivial, since there are always some differences in the models.

For example, in [BPTG15], the client has an instance $\mathbf{y}$, the server encrypts a table T with all the probabilities and sends it to the client, then the client runs the classification homomorphically and locally, obtaining ciphertexts encrypting the scores of each class. Finally, the client and the server run an interactive algorithm to reveal the class with the largest score, which is then assigned to $\mathbf{y}$. But downloading the entire model may represent a huge overhead for the client and the interactive step is surely a drawback. Furthermore, when they ran their protocol on the same dataset we used, the total time to classify a single instance was 419 milliseconds using 4 cores at 2.66 GHz each, for 80

bits of security, while our protocol takes about 42 milliseconds to classify one instance on a single core at 3.6 GHz, also for $\lambda = 80$, and with no interactive step.

In [PKK+18], the protocol is non-interactive and closer to ours. Namely, the client just encrypts $\mathbf{y}$ and send it to the server, which then uses the client's public key to encrypt the model and to run the classification homomorphically, sending the encrypted answer to the client at the end. However, all the functions evaluated homomorphically are quite complicated, because they are described as binary circuits, thus, in low level. As for the running times, they are much worse: the authors report that the server took about 60 seconds to classify one instance of the same dataset using 4 cores at 3.4 GHz each, for 80 bits of security.

5.7 A variant with private x_0

In this section we propose keeping the modulus x_0 secret and show that for some applications, like the homomorphic evaluation of NFA, we can obtain better running times by doing so. The main motivation is to improve the efficiency for large values of m.

As observed in Section 5.4.2, when m is big, we choose $\gamma = 2\eta$ and there is no reason to use a big value of ρ, thus, we can decrease it. Ideally, we would like to decrease both ρ and ρ_0 and increase b, which would give us a smaller $\ell = \lceil \log_b(2^\gamma) \rceil$, reducing thus the size of the ciphertexts and improving the efficiency. But we cannot set ρ_0 to be a small value, because of the factorization attacks that are applicable on x_0, consequently, we cannot increase b as we wish, because the final noise imposes the constraint $\log b \leq (1-\epsilon)\eta - \max(\rho, \rho_0)$.

Thus, if we keep x_0 private, the factorization attack is no longer possible, then we can set $\rho_0 = 0$ and increase b. However, we cannot perform the reductions modulo x_0 during the homomorphic evaluations and the norm of the ciphertexts are not bounded by 2^γ anymore. Actually, if we do not work over $\mathbb{Z}_{x_0}$, the bit length of the ciphertexts may increase indeterminately.

For instance, consider that we have a fresh ciphertext $\mathbf{C}_1$ encrypting $\mathbf{M}$. The entries

of $\mathbf{C}_1$ are then bounded by 2^γ. When we perform the homomorphic product $\mathbf{M} \cdot \mathbf{M}$ over $\mathbb{Z}$, we obtain $\mathbf{C}_2 = G^{-1}(\mathbf{C}_1)\mathbf{C}_1$ and we expect that $\|\mathbf{C}_2\| \approx m\ell b 2^\gamma$. Now, if we do $\mathbf{C}_3 = G^{-1}(\mathbf{C}_2)\mathbf{C}_2$, then we expect that $\|\mathbf{C}_3\| \approx m\ell \left\|G^{-1}(\mathbf{C}_2)\right\| \|\mathbf{C}_2\| \approx (m\ell b)^2 2^\gamma$. Thus, if we proceed like this, we obtain $\mathbf{C}_k$ encrypting $\mathbf{M}^{2^{k-1}}$ and the entries of $\mathbf{C}_k$ can be as big as $(m\ell b)^{k-1}2^\gamma$.

But if in each homomorphic multiplication one of the operands is a fresh ciphertext (or a ciphertext that has passed for only a constant number of operations), then we are always operating with a ciphertext that is potentially large and one that is for sure short, thus, we can decompose the larger one, and the ciphertext produced by the homomorphic product has norm approximately $m\ell b$ times the norm of the short ciphertext. Notice that this basically eliminates the accumulation of the factor $m\ell b$.

To analyze this more formally, define $\ell_0 := \log_b(2^\gamma)+1$, which is an upper bound to the number of words that we need to use to perform the decomposition of fresh ciphertexts. Let all $\mathbf{C}_i$ be fresh ciphertexts, thus, $\ell m \times m$ matrices with entries bounded by $s_0 := 2^\gamma$ and with ℓ being some integer bigger than ℓ_0 (we will define later how bigger ℓ must be). Let also $\mathbf{c}_0$ be a fresh ciphertext, thus, $\|\mathbf{c}_0\| \leq s_0$. Hence, the most significant words of the decomposition $G^{-1}(\mathbf{c}_0)$ will be zero. Actually, only $m\ell_0$ words of $G^{-1}(\mathbf{c}_0)$ will be non zero. Therefore, we see that the entries of $\mathbf{c}_1 := G^{-1}(\mathbf{c}_0)\mathbf{C}_0$ are bounded by $s_1 := \ell_0 m b 2^\gamma$. Now, letting $\ell_1 := \log_b(s_1)+1$, we see that applying the decomposition G^{-1} to $\mathbf{c}_1$ produces at most $m\ell_1$ nonzero words. Therefore, the entries of $\mathbf{c}_2 := G^{-1}(\mathbf{c}_1)\mathbf{C}_1$ are bounded by $s_2 := \ell_1 m b 2^\gamma$.

In general, after k such products, the entries of $\mathbf{c}_k$ are bounded by $s_k := \ell_{k-1} m b 2^\gamma$ and the number of words we need to decompose $\mathbf{c}_k$ is $\ell_k := \log_b(s_k) + 1 = \log_b(mb2^\gamma) + \log_b(\ell_{k-1}) + 1$. Thus, letting $y := \log_b(mb2^\gamma) + 1$, we see that

$$\ell_k = y + \log_b(\ell_{k-1}) = y + \underbrace{\log_b(y + \log_b(y + \ldots + \log_b(y + \ell_0)\ldots))}_{\log_b \text{ applied } k \text{ times.}}.$$

But the growth rate of the iterated logarithm is actually very slow and as k increases, ℓ_k is bounded by $y + \log_b y + \delta$ where $\delta := \log_b(3) < 2$. Thus, noticing that $y = \ell_0 + \log_b(n) + 1$

we can set

$$\ell = y + \log_b y + \delta = \ell_0 + \log_b(n) + 1 + \log_b(\ell_0 + \log_b(n) + 1) + \delta$$

which in is just slightly bigger than the dimension ℓ_0 that we would need if the operations were done over $\mathbb{Z}_{x_0}$, considering that we choose the same value of b. But since we will be able to use bigger values of b, this ℓ can even be smaller than the one we have to use when x_0 is public. Consequently, the size of the entries of the ciphertexts, that is, s_k, converges to $\ell mb2^\gamma$ after k homomorphic products, while they would be bounded by 2^γ if we worked modulo x_0.

We now prove that the value ℓ_k is bounded by the expression above.

Lemma 5.7.1. *Let $y, b \in \mathbb{Z}_{\geq 2}$ and $\delta := \log_b(3)$. Then, $\log_b(y + \log_b(y) + \delta) \leq \log_b(y) + \delta$.*

Proof. Because $b \geq 2$, we have $\delta < 2$, thus, $\delta/y < 1$. It follows then that

$$b^\delta = 3 > y/y + y/y + \delta/y \geq \frac{y + \log_b(y) + \delta}{y}.$$

Thus, applying logarithm in base b, we have

$$\delta > \log_b(y + \log_b(y) + \delta) - \log_b(y).$$

□

Lemma 5.7.2. *Let $\delta = \log_b(3)$. For all $k \in \mathbb{N}$, $\ell_k \leq \ell_0 + \log_b(m) + 1 + \log_b(\ell_0 + \log_b(m) + 1) + \delta$.*

Proof. Let's do a proof by induction. To simplify the notation, consider again $y = \ell_0 + \log_b(m) + 1$. The base case $k = 0$ is obviously true. Now, suppose that $\ell_k \leq y + \log_b(y) + \delta$ for some k. Then,

$$\begin{aligned}
\ell_{k+1} &= y + \log_b(\ell_k) && \text{(Definition of } \ell_{k+1}) \\
&\leq y + \log_b(y + \log_b(y) + \delta) && \text{(Inductive hypothesis)} \\
&\leq y + \log_b(y) + \delta && \text{(Lemma 5.7.1)}
\end{aligned}$$

□

Table 5.4: Proposed sets of parameters for 100 bits of security, considering that x_0 is private. Set $\ell = \lceil \log_b(2^\gamma) + \log_b(m) + \log_b(\log_b(2^\gamma) + \log_b(m) + 1) \rceil + 1$, $\rho_0 = 0$, and $\lambda = \eta = 100$. Now b is bigger than in Table 5.1.

	$8 \leq m \leq 52$	$m = 64$	$m = 128$	$m = 256$	$m = 512$	$m = 1024$
γ	$\lceil 100 \cdot 27^2/m \log(100) \rceil$	2η	2η	2η	2η	2η
ρ	73	72	59	42	18	2
$\log b$	7	11	19	36	60	76

Table 5.5: Practical results of the homomorphic evaluation of L_n on input strings with k letters when x_0 is kept secret. The security level is $\lambda = 100$ and the parameters used are presented in Table 5.4. The last row shows again the corresponding data for the NFA evaluation presented in [GGH+19]. Compare with Table 5.2.

n	Encrypted matrix	Encryption time	Evaluation time on inputs of length k					
			32	64	128	256	512	1024
128	5.73 MB	3.06	0.27	0.55	1.11	2.21	4.42	9.06
256	14.74 MB	14.11	0.704	1.41	2.81	5.68	11.32	22.66
512	45.87 MB	117.64	2.552	5.16	10.28	20.56	41.13	82.53
1024	157.28 MB	1020.70	10.768	21.54	43.49	87.36	175.48	350.24
≤ 1024	33 MB	16.5	-	-	-	1.53	3.34	6.63

Thus, we propose the parameters in Table 5.4. Now, ρ_0 is considered to be zero. The other parameters are basically the same when m is small, but when we turn to the regime $\gamma = 2\eta$, we can use much bigger values of b, since ρ_0 has no longer impact on the noise. We then used these parameters to evaluate NFA again and obtained the results presented in Table 5.5 (we omit the results for small m, since they are roughly the same as when x_0 is public). Comparing it with Table 5.2, we see that for large m the running times and the memory requirements become much better: the big values of b gives us ℓ smaller than before, and since the dimensions of the ciphertext matrices are $m\ell \times m$, their sizes decrease and the running times are improved. For example, for $\lambda = 100$ and $m = 1024$, when x_0 is public, we have $\gamma = 200$ and $b = 2^{15}$, which gives us $\ell = \lceil \log_b(2^\gamma) \rceil = 14$, while with private x_0, we have $\gamma = 200$, $b = 2^{76}$ and $\ell = \lceil \log_b(2^\gamma) + \log_b(n) + \log_b(\log_b(2^\gamma) + \log_b(n) + 1) \rceil + 1 = 4$. When compared to [GGH+19], the difference between the running times decreased for big m. For example, for $m = 1024$,

the time we need to evaluate a string with $k = 1024$ letters is now around 52 times slower than the time spent by [GGH+19], but with x_0 public, our evaluation time for $m = k = 1024$ is about 213 times slower. Notice that for m up to 100 our running times and ciphertext sizes are still better than the ones of [GGH+19].

5.8 An asymmetric variant

In several situations, it is more convenient to have a public key scheme than a symmetric one. For instance, when one evaluates a function homomorphically, it is common to need not only encryptions of the actual data, but also ciphertexts encrypting extra information. For example, in our homomorphic naïve Bayes classifier presented in Section 5.6.2, the evaluator (the server) needs not only the instances, but also encryption of the precomputed and scaled probabilities. In this type of scenario, the data owner must encrypt the extra information and send it to the evaluator before the homomorphic computation takes place, but this restricts the power of the evaluator, since it cannot evaluate other functions that may require other additional encrypted values not sent by the data owner. If the homomorphic scheme were asymmetric, the data owner could send the encrypted data and the public key to the evaluator, which could then evaluate any function homomorphically, regardless of the additional encrypted information required by this function, since the evaluator could just encrypt this information by themself.

One way of transforming a symmetric homomorphic encryption scheme into an asymmetric one is by publishing encryptions of "elementary values" that can generate all the possible plaintexts. For instance, by defining the public key as two ciphertexts $\mathbf{c}_1$ and $\mathbf{c}_2$ encrypting $(0, 1)$ and $(1, 0)$, respectively, one could generate a ciphertext $\mathbf{c}$ encrypting any plaintext vector $\mathbf{m} = (m_1, m_2)$ by computing $\mathbf{c} := m_1\mathbf{c}_1 + m_2\mathbf{c}_2 \mod x_0$. Of course, this would be insecure, because the ciphertexts produced in this way would not be "random enough". For instance, an encryption of $(0, 1)$ would always be simply $\mathbf{c}_1$. In general, anyone could guess the message encrypted by $\mathbf{c}$ by solving the system $\mathbf{c} = m_1\mathbf{c}_1 + m_2\mathbf{c}_2$ and finding m_1 and m_2. This problem is addressed by homomorphically adding a random

linear combination of encryptions of zero, so that the encrypted value does not change, as we are adding zeros, but the produced ciphertext is random. This technique is standard and has already been used, with some variations, in several important homomorphic schemes, e.g. [DGHV10, BGV12, BBL17].

In particular, when applying this technique to our scheme, we would proceed as follows: Let $\mathbf{e}_i \in \{0,1\}^m$ be a vector with 1 in the i-th entry and 0 elsewhere. Likewise, let $\mathbf{E}_{i,j} \in \{0,1\}^{m\times m}$ be a matrix with a single 1 in the entry (i,j). Let also v and M be two additional integer parameters of the scheme. In the procedure $\mathsf{HE.KeyGen}$, we compute $\tilde{\mathbf{e}}_i := \mathsf{HE.EncVec}(\mathsf{sk}, \mathbf{e}_i)$ and $\tilde{\mathbf{E}}_{i,j} := \mathsf{HE.EncMat}(\mathsf{sk}, \mathbf{E}_{i,j})$, for $1 \le i,j \le m$. Besides that, we compute the encryptions of zero, i.e., $\mathbf{z}_i := \mathsf{HE.EncVec}(\mathsf{sk}, \mathbf{0})$ for $1 \le i \le v$ and $\mathbf{Z}_j := \mathsf{HE.EncMat}(\mathsf{sk}, \mathbf{0})$ for $1 \le j \le M$. Finally, the public key is defined as

$$\mathsf{pk} := (\tilde{\mathbf{e}}_1, \ldots, \tilde{\mathbf{e}}_m, \tilde{\mathbf{E}}_{1,1}, \ldots, \tilde{\mathbf{E}}_{m,m}, \mathbf{z}_1, \ldots, \mathbf{z}_v, \mathbf{Z}_1, \ldots, \mathbf{Z}_M).$$

Hence, to encrypt a vector $\mathbf{u} = (u_1, \ldots, u_m) \in \mathbb{Z}^m$ using pk, we just sample v random bits b_i's and do

$$\mathbf{c} := \sum_{i=1}^{m} u_i \tilde{\mathbf{e}}_i + \sum_{i=1}^{v} b_i \mathbf{z}_i \mod x_0.$$

Notice that $\sum_{i=1}^{m} u_i \hat{\mathbf{e}}_i$ yields an encryption of $\mathbf{u}$ and that $\sum_{i=1}^{v} b_i \mathbf{z}_i$ is a random encryption of zero. To encrypt a matrix $\mathbf{U} \in \mathbb{Z}^{m\times m}$ whose entries are $u_{i,j}$, we proceed in a similar way, namely, just sample M random bits b_i's and do

$$\mathbf{C} := \sum_{i=1}^{m}\sum_{j=1}^{m} u_{i,j} \tilde{\mathbf{E}}_{i,j} + \sum_{i=1}^{M} b_i \mathbf{Z}_i \mod x_0.$$

The decryption methods are the same and even the analysis of the error growth is equal, with the exception that now instead of having 2^ρ as the magnitude of the initial noise, we have approximately $(m+v)2^\rho$ for vectors ciphertexts and $(m^2+M)2^\rho$ for matrix ciphertext.

To prove that the ciphertexts generated with the public key are indistinguishable, we just have to set large enough values for the parameters v and M and use th Left Over Hash lemma, as follows:

Lemma 5.8.1. *Let $v = \gamma \cdot m + \Omega(\lambda)$. Then, under the decisional AGCD assumption, any PPT adversary has only negligible advantage in distinguishing encryption of two vectors.*

Proof. Let $\mathbf{V}$ be a matrix sampled from $\mathcal{U}(\mathbb{Z}_{x_0}^{v \times m})$. Then, the function $h : \{0,1\}^v \to \mathbb{Z}_{x_0}^m$ defined as $h(\mathbf{b}) = \mathbf{b} \cdot \mathbf{V} \mod x_0$ is a 2-universal hash function. Thus, by the Leftover Hash Lemma, the statistical distance between $\mathbf{b} \cdot \mathbf{V} \mod x_0$ and $\mathcal{U}(\mathbb{Z}_{x_0}^m)$ is upper bounded by

$$\frac{1}{2}\sqrt{\frac{|\mathbb{Z}_{x_0}^m|}{|\{0,1\}^v|}} = 2^{(m\log(x_0)-v)/2-1} < 2^{(m\gamma-v)/2-1} = 2^{-\Omega(\lambda)-1}$$

therefore, it is negligible in λ and so is the advantage of any PPT in distinguishing these distributions.

Let $\mathbf{Z} \in \mathbb{Z}_{x_0}^{v \times m}$ such that $\mathsf{row}_i(\mathbf{Z}) = \mathbf{z}_i$. Considering that $x_0 > 2^{\gamma-1}$, Lemma 5.4.1 implies that $\mathbf{Z}$ is computationally indistinguishable from a matrix sampled from $\mathcal{U}(\mathbb{Z}_{x_0}^{v \times m})$, therefore, replacing $\mathbf{V}$ in the definition of h by $\mathbf{Z}$ implies that $\mathbf{b} \cdot \mathbf{Z} \mod x_0$ is indistinguishable from $\mathcal{U}(\mathbb{Z}_{x_0}^m)$.

Finally, because $\mathcal{U}(\mathbb{Z}_{x_0})$ and $\mathcal{U}(\mathbb{Z}_{x_0}) + y \mod x_0$ are the same distribution regardless of the value of y, we have that a ciphertext defined as $\mathbf{c} := \sum_{i=1}^m m_i \tilde{\mathbf{e}}_i + \mathbf{b} \cdot \mathbf{Z} \mod x_0$ is also computationally indistinguishable from $\mathcal{U}(\mathbb{Z}_{x_0}^m)$. □

Lemma 5.8.2. *Let $M = \gamma \cdot m^2 \cdot \ell + \Omega(\lambda)$. Then, under the decisional AGCD assumption, any PPT adversary has only negligible advantage in distinguishing encryption of two matrices.*

Proof. Essentially the same proof as the one of Lemma 5.8, but replacing the function h by $h' : \{0,1\}^M \to \mathbb{Z}_{x_0}^{m\ell \times m}$ defined as $h'(b_1, ..., b_M) := \sum_{i=1}^M b_i \cdot \mathbf{V}_i \mod x_0$ for $\mathbf{V}_i$'s sampled uniformly from $\mathbb{Z}_{x_0}^{m\ell \times m}$. Using the Leftover Hash Lemma, the advantage of any PPT adversary in distinguishing $h'(\mathbf{b})$ from $\mathcal{U}(\mathbb{Z}_{x_0}^{m\ell \times m})$ is bounded by

$$\frac{1}{2}\sqrt{\frac{|\mathbb{Z}_{x_0}^{m\ell \times m}|}{|\{0,1\}^v|}} = 2^{(m^2\ell\log(x_0)-v)/2-1} < 2^{(m^2\ell\gamma-v)/2-1} = 2^{-\Omega(\lambda)-1}.$$

Using Lemma 5.4.1 again, we can replace the matrices $\mathbf{V}_i$'s in the definition of h' by the encryptions of zero $\mathbf{Z}_i$'s present in the public key, which implies that $\sum_{i=1}^M b_i \mathbf{Z}_i \mod x_0$

is computationally indistinguishable from uniform matrices of $\mathbb{Z}_{x_0}^{m\ell \times m}$, and, therefore, any ciphertext $\mathbf{C} := \sum_{i=1}^{m}\sum_{j=1}^{m} m_{i,j}\tilde{\mathbf{E}}_{i,j} + \sum_{i=1}^{M} b_i\mathbf{Z}_i \mod x_0$ is also so, independently of the message being encrypted. □

Thus, the public key is composed by $m + v = (\gamma + 1)m + \Omega(\lambda)$ vector ciphertexts plus $m^2 + M = (\gamma\ell + 1)m^2 + \Omega(\lambda)$ matrices ciphertexts. Considering that the bit length of each vector ciphertext is $m\gamma$ and of each matrix ciphertext is $m^2\ell\gamma$, the size in bits of the public key is

$$m\gamma \cdot ((\gamma + 1)m + \Omega(\lambda)) + m^2\ell\gamma \cdot (\gamma\ell m^2 + \Omega(\lambda)) = \Omega(m^4\ell^2\gamma^2 + m^2\ell\gamma\lambda).$$

For $m \approx \lambda$, ℓ is a small constant, γ is approximately equal to λ, and, hence, the size of the public key is $\Omega(\lambda^6)$, which may be too big to be practical. However, there are several ways of reducing the size of public key. For instance, one could adapt the key compression techniques of [CNT12] and randomize the matrix ciphertexts with $\sum_{i=1}^{M}\sum_{j=1}^{M} b_{i,j}G^{-1}(\mathbf{Z}_i)\cdot \mathbf{Z}_j \mod x_0$ instead of $\sum_{i=1}^{M} b_i\mathbf{Z}_i \mod x_0$. With this, we could set $M = \tilde{O}(m \cdot \sqrt{\gamma\ell})$ in lieu of $M = \gamma \cdot m^2 \cdot \ell + \Omega(\lambda)$, thus, because we would have $2M$ matrices $\mathbf{Z}_i$'s, the size of the public key would be approximately $2Mm^2\ell\gamma = \tilde{O}(m^3 \cdot (\gamma\ell)^{1.5})$. Considering $m \approx \lambda$, this gives $\tilde{O}(\lambda^{4.5})$, which is already much smaller than $\Omega(\lambda^6)$. The size of the public key could be further reduced if cubic expressions, like $(G^{-1}(\mathbf{Z}_i) \cdot (G^{-1}(\mathbf{Z}_j) \cdot \mathbf{Z}_k)) \mod x_0$, or even higher degree expressions were used instead of quadratic ones to randomize the ciphertexts.

5.9 Conclusion

In this chapter, we proposed a new homomorphic encryption scheme based whose security relies on the AGCD problem and that can operate natively with vectors and matrices. When compared to other AGCD-based schemes, the running times, the ciphertext expansion, and the cost of homomorphic evaluations are good. In particular, we can efficiently evaluate long sequences of vector-matrix products. Then, we presented two applications of this scheme: homomorphic evaluation of encrypted nondeterministic finite state au-

tomata on plain text input and a homomorphic Naïve Bayes Classifier. As the dimension of the encrypted vectors increases, the efficiency of the scheme deteriorates, thus, trying to solve this problem, we also proposed a variant of the scheme in which the modulus x_0 is not published and we showed that the parameters of the scheme can be then improved.

Chapter 6

Randomized polynomial AGCD problem

6.1 Motivation

By representing polynomial rings with matrices and vectors, as described in Section 2.5 of Chapter 2, we can operate homomorphically over polynomials using our homomorphic scheme for vectors and matrices (HEVAM) proposed in Chapter 5. However, the efficiency is not satisfactory, since a polynomial of degree m would be represented by an $m \times m$ matrix, which would then be encrypted into an $m\ell \times m$ matrix whose entries are γ bits long. In summary, we would need $m^2\ell\gamma$ bits to encrypt a degree-m polynomial, which can be prohibitive for large values of m. Thus, in this chapter, we investigate an alternative randomization of the AGCD problem using polynomials instead of matrices, which gives rise to a new cryptographic problem, that we call the randomized polynomial AGCD problem (RAGCD). We show that this problem cannot be easier than the AGCD problem, then, we show how to cryptanalyze the the RAGCD problem by the VAGCD problem defined in Chapter 3, concluding, thus, that we can select parameters in similar ways for both the RAGCD and the VAGCD problem. Finally, we demonstrate the usefulness of the RAGCD problem by using it as the underlying problem of a new leveled homomorphic encryption scheme that can operate natively and efficiently over polynomials. In Chapter 7, this scheme is used to perform new fast bootstrapping procedures for FHE schemes based on the AGCD problem.

6.2 Randomized polynomial AGCD problem

We start by extending the AGCD problem to a problem that is strictly equivalent, but that works on polynomials. Then, we propose to randomize this problem with a hidden polynomial $k(x)$, obtaining thus the underlying problem that will be used in our scheme.

Definition 6.2.1 (Underlying distribution of PAGCD)**.** *Let N, ρ, η, γ, and p be integers such that $\gamma > \eta > \rho > 0$ and p is an η-bit integer. The distribution $\mathcal{P}_{N,\gamma,\rho}(p)$, whose support is $[\![0, 2^\gamma - 1]\!]^N$, is defined as*

$$\mathcal{P}_{N,\gamma,\rho}(p) := \left\{ \textit{Sample } c_0, ..., c_{N-1} \leftarrow \mathcal{D}_{\gamma,\rho}(p) : \ \textit{Output } c := \sum_{i=0}^{N-1} c_i x^i \right\}.$$

Definition 6.2.2 (PAGCD)**.** *The (N, ρ, η, γ)-polynomial-approximate-GCD problem is the problem of finding p, given arbitrarily many samples from $\mathcal{P}_{N,\gamma,\rho}(p)$.*

The (N, ρ, η, γ)-decisional-PAGCD problem is the problem of distinguishing between $\mathcal{P}_{N,\gamma,\rho}(p)$ and $\mathcal{U}([\![0, 2^\gamma[\![^N)$.

Because each coefficient of each polynomial output by $\mathcal{P}_{N,\gamma,\rho}(p)$ is an independent sample of $\mathcal{D}_{\gamma,\rho}(p)$, having N samples of the AGCD problem is the same as having one sample of the PAGCD problem, hence, it is clear that the PAGCD and the original AGCD problem are equivalent.

Now, aiming to choose smaller parameters, we propose a randomized version of this problem, but instead of randomizing a vector of AGCD samples with a hidden matrix $\mathbf{K}$, as it was done in Section 3.4, we randomize a sample of $\mathcal{P}_{N,\gamma,\rho}(p)$ with a hidden polynomial k, performing the operations on the polynomial ring $R := \mathbb{Z}[x]/\langle f \rangle$, for some f of degree N.

Definition 6.2.3 (Underlying distribution of RAGCD)**.** *Let N, ρ, η, γ, and p be integers such that $\gamma > \eta > \rho > 0$ and p has η bits. Let f be a degree-N integral polynomial, $R := \mathbb{Z}[x]/\langle f \rangle$, x_0 be a sample from $\mathcal{D}_{\gamma,\rho}(p)$, and k be an random invertible polynomial of R/x_0R. The distribution $\mathcal{R}_{N,\gamma,\rho,x_0}(p, k)$, whose support is R/x_0R is defined as*

$$\mathcal{R}_{N,\gamma,\rho,x_0}(p, k) := \{ \textit{Sample } c \leftarrow \mathcal{P}_{N,\gamma,\rho}(p) : \ \textit{Output } \tilde{c} := c \cdot k \in R/x_0R \}.$$

Definition 6.2.4 (RAGCD)**.** *The $(x_0, N, \rho, \eta, \gamma)$-RAGCD problem is the problem of finding p and k, given arbitrarily many samples from $\mathcal{R}_{N,\gamma,\rho,x_0}(p,k)$.*

The $(x_0, N, \rho, \eta, \gamma)$-decisional-RAGCD problem is the problem of distinguishing between $\mathcal{R}_{N,\gamma,\rho,x_0}(p,k)$ and $\mathcal{U}(R/x_0R)$.

Notice that we can instantiate this problem using any polynomial ring $\mathbb{Z}[x]/\langle f\rangle$, however, one has to carefully choose the polynomial used as the modulus, in particular, if f is not irreducible in $\mathbb{Z}[x]$, then its factors can lead to attacks on easier instances of the RAGCD problem. In Section 6.2.1, a detailed discussion about the choice of f is presented.

It is easy to see that any PPT adversary that solves the RAGCD problem can be converted in an algorithm to solve the original AGCD problem, which means that the RAGCD problem is at least as hard as the AGCD problem. But, for completeness, we present a formal reduction to prove this statement.

Lemma 6.2.5 (AGCD $\leq$ RAGCD)**.** *The RAGCD problem is not easier then the AGCD problem.*

Proof. Let $\mathcal{A}$ be a PPT adversary that solves the RAGCD problem with probability ϵ. Let's construct an algorithm $\mathcal{B}$ to solve the PAGCD problem using $\mathcal{A}$ as a black-box.

Let $c_0, c_1, ..., c_t \leftarrow \mathcal{P}_{N,\gamma,\rho}(p)$ be an instance of the (N, ρ, η, γ)-PAGCD problem. The algorithm $\mathcal{B}$ consists of the following steps: Select any coefficient of c_0 to be x_0 and sample a degree-N polynomial k invertible on R/x_0R. Randomize the polynomials c_i's, that is, compute $\tilde{c}_i := c_i \cdot k \in R/x_0R$ for $1 \leq i \leq t$ (reducing modulo f and modulo x_0). Output $\mathcal{A}(\tilde{c}_1, ..., \tilde{c}_t)$.

Notice that $(\tilde{c}_1, ..., \tilde{c}_t)$ is a valid instance of the RAGCD problem, thus, $\mathcal{A}$ finds p with probability ϵ, therefore, $\mathcal{B}$ solves the PAGCD problem with same probability.

Because the PAGCD problem and the AGCD problem are equivalent, $\mathcal{B}$ also solves the AGCD problem. □

In the next lemma, we prove that if we fix $x_0 \geq 2^{\gamma-1}$ and restrict the distribution $\mathcal{R}_{N,\gamma,\rho,x_0}(p,k)$ so that it only randomizes polynomials with coefficients smaller than x_0,

then we obtain a distribution that is indistinguishable from $\mathcal{U}(R/x_0R)$ under the hardness of the decisional AGCD problem, that is, the assuming that the dAGCD problem is hard, we have that the decisional RAGCD problem is also hard for $x_0 > 2^{\gamma-1}$.

Lemma 6.2.6. *Let x_0 be a sample of $\mathcal{D}_{\gamma,\rho}(p)$ such that $x_0 > 2^{\gamma-1}$. Let $\mathcal{D}_{<x_0}$ be the distribution obtained by rejecting samples of $\mathcal{D}_{\gamma,\rho}(p)$ that are bigger than or equal to x_0. Let $\mathcal{R}_{<x_0}$ be defined as $\mathcal{R}_{N,\gamma,\rho,x_0}(p,k)$, but randomizing polynomials with coefficient smaller than x_0, that is:*

$$\mathcal{R}_{<x_0} := \left\{ \textit{Sample } c \leftarrow \mathcal{D}^N_{<x_0} : \ \textit{Output the polynomial } \tilde{c} := c \cdot k \in R/x_0R \right\}.$$

Then, distinguishing between $\mathcal{R}_{<x_0}$ and $\mathcal{U}(R/x_0R)$ is not easier than the decisional AGCD problem.

Proof. Because $x_0 > 2^{\gamma-1}$, Lemma 5.4.1 implies that the distributions $\mathcal{D}_{<x_0}$ and $\mathcal{U}(\mathbb{Z}_{x_0})$ are computationally indistinguishable under the decisional AGCD assumption. Therefore, polynomials from R with coefficients sampled from $\mathcal{D}_{<x_0}$ or from $\mathcal{U}(\mathbb{Z}_{x_0})$ are also computationally indistinguishable. But because k is invertible in R modulo x_0, multiplying by k does not change the distributions, that is,

$$\mathcal{D}^N_{<x_0} = \mathcal{D}^N_{<x_0} \bmod x_0 = \mathcal{D}^N_{<x_0} \cdot k \bmod x_0 = \mathcal{R}_{<x_0}$$

and

$$\mathcal{U}(\mathbb{Z}^N_{x_0}) = \ \mathcal{U}(\mathbb{Z}^N_{x_0}) \bmod x_0 = \ \mathcal{U}(\mathbb{Z}^N_{x_0}) \cdot k \bmod x_0 = \ \mathcal{U}(R/x_0R).$$

Therefore, $\mathcal{R}_{<x_0}$ and $\mathcal{U}(R/x_0R)$ are computationally indistinguishable. □

6.2.1 Cryptanalysis of the RAGCD problem

As explained in Section 2.5, we can represent the elements of R with matrices and vectors using the maps $\mathbf{\Phi}$, which is the circulant matrix, and $\boldsymbol{\phi}$, the vector of coefficients. Thus, given an RAGCD instance $\tilde{c} := c \cdot k = (pq + r)k \in R/x_0R$, we can represent it as

$$\boldsymbol{\phi}(\tilde{c}) = \boldsymbol{\phi}(c) \cdot \mathbf{\Phi}(k) = \boldsymbol{\phi}(pq + r) \cdot \mathbf{\Phi}(k) = (p\boldsymbol{\phi}(q) + \boldsymbol{\phi}(r)) \cdot \mathbf{\Phi}(k) \in \mathbb{Z}^N_{x_0}.$$

Notice that denoting $\tilde{\mathbf{c}} := \boldsymbol{\phi}(\tilde{c})$, $\mathbf{q} := \boldsymbol{\phi}(q)$, $\mathbf{r} := \boldsymbol{\phi}(r)$, and $\mathbf{K} := \mathbf{\Phi}(k)$, we have an

N-dimensional instance $\tilde{\mathbf{c}} = (p\mathbf{q}+\mathbf{r})\mathbf{K}$ of the vector AGCD problem, except that now the matrix $\mathbf{K}$ is not completely random, but instead, its first row is random and the remaining rows are defined by the first one. Thus, in order to solve the RAGCD problem, we can represent it as the the vector AGCD problem, ignore the structure of the matrix $\mathbf{K}$, and run the attacks on the vector AGCD problem. In other words, we can cryptanalyze the RAGCD problem by the VAGCD problem.

The parallel with the RLWE problem is worthy of note: the attacks against the RLWE problem are adapted from the cryptanalysis of the LWE problem, since the RLWE can be seen as a structured version of the LWE. Also, in practice, such structure is ignored because there is no known way of exploiting it.

Thus, to guarantee the security, we must set $\gamma \in \Omega\left(\frac{\lambda(\eta-\rho)^2}{N\log\lambda}\right)$ to rule out orthogonal lattice attacks and $\rho \in \Omega(\lambda/N)$ to avoid GCD attacks.

A simple choice for the modulus polynomial f when instantiating the RAGCD problem is $f = x^N + 1$ with N being a power of two, because in this case, f is a cyclotomic polynomial, therefore, irreducible on $\mathbb{Z}[x]$. However, other choices of f are possible, but we suggest that f must be irreducible, since using a reducible polynomial f make other attacks possible, as we discuss now.

Polynomial evaluation:

Consider that we define the ring R as $\mathbb{Z}[x]/\langle f\rangle$ for some degree-N polynomial f. Given an instance RAGCD $c := (pq + r)k \in R$, there is a polynomial u such that the following holds over $\mathbb{Z}[x]$: $c = (pq + r)k - uf$. Therefore, when we evaluate c at some integer z, we obtain $c(z) = (pq(z) + r(z))k(z) - u(z)f(z) \in \mathbb{Z}$. Now, if z is a root of f, we have $c(z) = (pq(z) + r(z))k(z)$, which can be viewed as an instance of the original AGCD problem, but masked by an integer $k(z)$. Even for small values of N, we expect $r(z)$ to be bigger than p in general. For instance, setting $\lambda = \eta = 100$ and $N = 32$, the power z^{N-1} alone would already be bigger than p for any z such that $|z| > 10$, thus, it is very likely that $|r(z)| = |\sum_{i=0}^{N-1} r_i \cdot z^i|$ is also bigger than p. In this case, $c(z)$ is an ill formed AGCD sample and we can not recover p from it.

However, if $r(z)$ happens to be small, then $c(z)$ is an AGCD sample with noise term $r(z)$ and we could use the attacks against the original AGCD problem instead of the attacks against the vector AGCD problem. This could be problematic because we would be attacking an AGCD instance with parameters much smaller than what it is needed to guarantee the security, since parameters of the vector AGCD problem (and thus, the ones of the RAGCD problem) are usually equal to the parameters of the AGCD problem divided by the dimension of the vectors, in this case, the value N.

As a concrete example, consider that one tries to instantiate the RAGCD problem using the ring $R = \mathbb{Z}[x]/\langle x^N - 1\rangle$. To achieve security of λ bits, one could set, say, $\rho = \lambda/N$. But then, the noise term $r = \sum_{i=0}^{N-1} r_i \cdot x^i \in \mathbb{Z}[x]$ satisfies $|r_i| < 2^{\lambda/N}$. Now, the problem is that $x^N - 1$ has a very small root, namely, the value 1, thus, it holds that $c(1) = (pq(1) + r(1))k(1)$, and $|r(1)| \leq \sum_{i=0}^{N-1} |r_i| \leq N \cdot 2^{\lambda/N}$. In other words, evaluating any RAGCD instance c at 1 produces a "masked" AGCD instance with small noise term, around $N \cdot 2^{\lambda/N}$. One could, for instance, run Lee-Seo's GCD attack [LS14], which would recover the secret p in time and memory $\tilde{O}(2^{\lambda/N})$, that is, in much less time than the $\Omega(2^\lambda)$ that the chosen security level is supposed to guarantee.

Dimension reduction:

Actually, evaluating a polynomial c at a point z is equivalent to reducing c modulo $x - z$. And assuming that z is an integer and a root of f is the same as assuming that $x - z$ is a factor of f. Thus, we can generalize the previous attack as follows: Consider that one instantiates the RAGCD problem using the ring $\mathbb{Z}[x]/\langle f\rangle$ for a polynomial f that has a non-trivial factor g, that is, the degree of g is at least one and g divides f on $\mathbb{Z}[x]$.

Then, because an RAGCD instance c can be written as $c = (pq + r)k - uf$, for some $u \in \mathbb{Z}[x]$, we can reduce c modulo g obtaining $c' = (p[q]_g + [r]_g) \cdot [k]_g - vg$, for some $v \in \mathbb{Z}[x]$. But this c' is a new RAGCD instance over the "smaller" ring $\mathbb{Z}[x]/\langle g\rangle$, because the degree of g is less than the degree of f. For some polynomials g, the norm of $[r]_g$ will be bigger than p and c' will be an ill defined RAGCD instance from which we cannot recover p. However, depending on the degree and on the coefficients of g, the infinity

norm of $[r]_g$ can be just slightly larger than the norm of r, which means that c' can be effectively used in an attack in lower dimension (thus, in an easier RAGCD instance).

For example, let N be even and consider that the $g(x) = x^{N/2} + 1$ is a factor of f. Then, reducing c modulo g yields $c' = (p[q]_g + [r]_g) \cdot [k]_g - vg$, for some $v \in \mathbb{Z}[x]$. But c' is a polynomial with half the degree of the original c. Moreover, defining $r := \sum_{i=0}^{N-1} r_i \cdot x^i$, we have $[r]_g = \sum_{i=0}^{N/2-1}(r_i + r_{i+N/2}) \cdot x^i$, hence, $\|[r]_g\| \leq 2\|r\|$. The same holds for q, i.e., $\|[q]_g\| \leq 2\|q\|$. Thus, we essentially reduce an RAGCD instance with parameters N, γ, η, and ρ, to a much easier RAGCD instance with parameters $N/2$, $\gamma + 1$, η, and $\rho + 1$. To illustrate that: the time complexity of the GCD attacks would decrease from $\tilde{O}(2^{N\rho})$ to $\tilde{O}(2^{N(\rho+1)/2})$ and orthogonal lattice attacks would have their costs reduced from $2^{\Omega(\gamma N/(\eta-\rho)^2)}$ to $2^{\Omega\left(\frac{N}{2}\cdot(\gamma+1)/(\eta-\rho-1)^2\right)}$, thus, attacking this new RAGCD instance would take roughly the square root of the time needed to attack the original instance.

6.3 GAHE: GSW-like AGCD-based homomorphic encryption scheme

In this section, we use the RAGCD problem to propose a GSW-like AGCD-based Homomorphic Encryption (GAHE) scheme that can operate homomorphically and natively with polynomials. We start with a basic scheme that can encrypt a polynomial $f \in R$ into a vector $\mathbf{c} \in R^\ell$. Then, by assuming circular security, we extend the definition of the scheme so that we also have scalar ciphertexts. Thus, in the end, we can encrypt a polynomial $m \in R$ in two formats:

- Scalar format: $(pq + r + \alpha m) \cdot k \in R$.
- Vector format: $(p\mathbf{q} + \mathbf{r}) \cdot k + \mathbf{g}m \in R^\ell$, where $\mathbf{g} = (b^0, ..., b^{\ell-1})$ for some $b \in \mathbb{Z}$.

Therewith we can define an efficient mixed homomorphic multiplication, from $R \times R^\ell$ to R, i.e., a product involving a scalar ciphertext and a vector ciphertext. This is somehow equivalent to the external product used in [CGGI16a]. To perform the homomorphic products, we decompose the polynomials in base b, thus, for any integer a with absolute

value smaller than b^ℓ, we denote by $g^{-1}(a)$ the signed decomposition of a in base b, that is, $g^{-1}(a) \in [\![-b+1, b-1]\!]^\ell$ and satisfies $g^{-1}(a)\cdot\mathbf{g} = a$. Then, we extend g^{-1} as follows: Given a polynomial $a := \sum_{i=0}^{N} a_i x^i$ such that $\|a\|_\infty < b^\ell$, we define $g^{-1}(a) := \sum_{i=0}^{N} g^{-1}(a_i)x^i$. Notice that $\left\|g^{-1}(a)\right\|_\infty \leq b-1$ and $g^{-1}(a)\cdot\mathbf{g} = \sum_{i=0}^{N}(g^{-1}(a_i)\cdot\mathbf{g})x^i = \sum_{i=0}^{N} a_i x^i = a$.

To ease the presentation, we use a noiseless x_0, but we keep it private. This allows us to simplify the noise-growth analysis, because a noiseless x_0 adds no extra error term to the ciphertexts. Thus, the homomorphic operations are performed on R instead of R/x_0R, which means that the bit length of the ciphertext grows. However, in several applications, like our bootstrapping procedures presented in Chapter 7, they grow only by a small factor that is independent of the multiplicative depth of the homomorphic evaluation. The tradeoffs between public and private x_0 were already discussed in Section 5.7, where we presented a variant of the homomorphic scheme for vectors and matrices in which x_0 was private, and they are basically the same here.

Finally, we use the cyclotomic polynomial $x^N + 1$, where N is a power of two, as the modulus, thus, we are fixing the polynomial ring as $R := \mathbb{Z}[x]/\langle x^N + 1\rangle$.

6.3.1 The basic procedures

- $\mathsf{GAHE.KeyGen}(1^\lambda, N, t, b)$: Choose the parameters η, ρ, and γ. Sample an η-bit random prime p. Sample x_0 from $p \cdot \mathcal{U}([\![1, 2^\gamma/p]\!])$, until $x_0 \geq 2^{\gamma-1}$. Then, sample k uniformly from R/x_0R until k^{-1} exists over R/x_0R. Define $\ell_0 := \lceil \log_b(2^\gamma)\rceil$ and $\ell := \lceil \ell_0 + \log_b(N) + 1 + \log_b(\ell_0 + \log_b(N) + 1)\rceil$[1]. The public parameters are $\mathsf{params} := \{N, t, \ell, b, \eta, \gamma, \rho\}$ and secret key is $\mathsf{sk} := (p, k, x_0)$.

- $\mathsf{GAHE.EncVec}(\mathsf{sk}, m)$: Given a polynomial $m \in R/tR$, construct a vector $\mathbf{x} := (p\mathbf{q} + \mathbf{r})k \in R^\ell$ by sampling each entry x_i independently from $\mathcal{R}_{N,\gamma,\rho,x_0}(p, k)$, then output the following vector $\mathbf{c}$:

$$\mathbf{c} := [\mathbf{x} + \mathbf{g}\cdot m]_{x_0} \in R^\ell.$$

[1] If we were publishing x_0, then the homomorphic operations could be done modulo x_0 and we could set $\ell = \ell_0$, without adding these extra logarithmic terms.

- $\mathsf{GAHE.DecVec}(\mathsf{sk}, \mathbf{c})$: Let $\alpha := \lfloor p/t \rceil$. Compute the inner product $c := g^{-1}([\alpha k]_{x_0}) \cdot \mathbf{c}$ over R/x_0R. Then do $c' := c \cdot k^{-1} \in R/x_0R$ and output

$$\left\lfloor \frac{t \cdot [c']_p}{p} \right\rceil \bmod t.$$

6.3.2 Assuming circular security to extend the scheme

In this section we show that by assuming circular security we can encrypt an element of R/tR into a single element of R instead of into a vector. We call a ciphertext produced by this new encryption method a scalar ciphertext and the ones produced by the encryption function defined before are vector ciphertexts. Moreover, we define the mixed homomorphic product between a vector and a scalar ciphertext. It is worth noticing that circular security is regarded as a weak assumption and has been used extensively in all types of homomorphic encryption schemes.

Thus, notice that by assuming circular security, we can include the secret key in the message and use $\mathsf{GAHE.EncVec}$ to encrypt $m \cdot k \cdot \lfloor p/t \rceil$, obtaining then a vector ciphertext $\mathbf{c} = (p\mathbf{q}+\mathbf{r})k+(m \cdot k \cdot \lfloor p/t \rceil)\mathbf{g} = (p\mathbf{q}+\mathbf{r}+m \cdot \lfloor p/t \rceil \cdot \mathbf{g})k$. But then, because the first entry of $\mathbf{g}$ is 1, we see that the first entry of $\mathbf{c}$ has the following format: $c_1 = (pq_1+r_1+m \cdot \lfloor p/t \rceil)k \in R$. Thus, we can extend our scheme with the following procedures:

- $\mathsf{GAHE.EncScalar}(\mathsf{sk}, m)$: Given a polynomial $m \in R/tR$, sample $x := (pq + r)k \leftarrow \mathcal{R}_{N,\gamma,\rho,x_0}(p, k)$ and output

$$c := [x + m \cdot \lfloor p/t \rceil \cdot k]_{x_0} \in R.$$

- $\mathsf{GAHE.DecScalar}(\mathsf{sk}, c)$: Output $\left\lfloor \frac{t \cdot [c']_p}{p} \right\rceil \bmod t$ where $c' := c \cdot k^{-1} \in R/x_0R$.

6.3.3 Homomorphic operations

- $\mathsf{GAHE.AddVec}(\mathbf{c}_1, \mathbf{c}_2)$: to homomorphically add two vector ciphertexts, just add them entry-wise: $\mathbf{c}_{add} := \mathbf{c}_1 + \mathbf{c}_2 \in R^\ell$.

- $\mathsf{GAHE.MultVec}(\mathbf{c}_1, \mathbf{c}_2)$: to homomorphically multiply two vector ciphertexts, apply g^{-1}

to each entry of $\mathbf{c}_1$ obtaining a matrix of polynomials $\mathbf{A} := \left(g^{-1}(c_{1,1}) \ldots g^{-1}(c_{1,\ell})\right) \in R^{\ell\times\ell}$, then perform a vector-matrix product over R: $\mathbf{c}_{mult} := \mathbf{c}_2 \cdot \mathbf{A} \in R^{\ell}$.

- $\mathsf{GAHE.AddScalar}(c_1, c_2)$: to perform a homomorphic addition, just add the ciphertexts: $c_{add} := c_1 + c_2 \in R$.

- $\mathsf{GAHE.AddPlaintext}(\mathbf{c}_1, h)$ and $\mathsf{GAHE.MultPlaintext}(\mathbf{c}_1, h)$: to add a plaintext h, output $\mathbf{c}_1 + \mathbf{g}\cdot h$. To multiply, simply multiply each entry of $\mathbf{c}_1$ by h in R, i.e., output $h\cdot\mathbf{c}_1 \in R^{\ell}$.

- $\mathsf{GAHE.MultMix}(c, \mathbf{c})$: to perform a homomorphic mixed product, we decompose and multiply the scalar ciphertext c by the vector ciphertext $\mathbf{c}$, outputting the following inner product over R: $c_{mult} := g^{-1}(c) \cdot \mathbf{c} \in R$.

6.3.4 Correctness of decryption

In this section we define the noise of a ciphertext and show the necessary conditions for the decryption functions to work.

Definition 6.3.1 (Noise of scalar ciphertext)**.** *Let $c = (pq + r + \lfloor p/t \rceil\, m)k$ be a scalar ciphertext encrypting a message $m \in R/tR$. We define the noise of c as $\mathsf{err}(c) := [(c \cdot k^{-1} - \lfloor p/t \rceil\, m) \bmod x_0]_p$. Notice that $\mathsf{err}(c)$ is exactly r if $\|r\| < p/2$.*

Definition 6.3.2 (Noise of vector ciphertext)**.** *Let $\mathbf{c} = (p\mathbf{q} + \mathbf{r})k + \mathbf{g}m$ be a vector encryption of $m \in R/tR$. We define the noise of $\mathbf{c}$ as $\mathsf{err}(\mathbf{c}) := [(\mathbf{c} - \mathbf{g}m) \cdot k^{-1} \bmod x_0)]_p$. Notice that $\mathsf{err}(\mathbf{c})$ is $\mathbf{r}$ if $\|\mathbf{r}\| < p/2$.*

Lemma 6.3.3 (Upper bound on the noises)**.** *Let $c = (pq + r + \alpha m_1)k \in R$ be a scalar ciphertext and $\mathbf{c} = (p\mathbf{q} + \mathbf{r})k + \mathbf{g}m_2 \in R^{\ell}$ be a vector ciphertext. Assuming that $\|\mathsf{err}(c)\|$ and $\|\mathsf{err}(\mathbf{c})\|$ are both smaller than $p/2$, it holds that $\|\mathsf{err}(c)\| = \|r\|$ and $\|\mathsf{err}(\mathbf{c})\| = \|\mathbf{r}\|$. In particular, if c and $\mathbf{c}$ are fresh ciphertexts, then $\|\mathsf{err}(c)\| < 2^{\rho}$ and $\|\mathsf{err}(\mathbf{c})\| < 2^{\rho}$.*

Proof. It is clear that $\mathsf{err}(c) = r$, therefore, $\|\mathsf{err}(c)\| = \|r\|$. Moreover, for fresh ciphertexts, we have $\|r\| < 2^{\rho}$ because each coefficient of r is sampled uniformly from $\rrbracket - 2^{\rho}, 2^{\rho}\llbracket$. The same argument applies to $\mathsf{err}(\mathbf{c})$.

□

Let's first analyze GAHE.DecScalar. Then, the correctness of GAHE.DecVec follows basically by the same argument.

Lemma 6.3.4 (Correctness of scalar decryption)**.** *Let c be a scalar encryption of $m \in R/tR$. If $\|\mathsf{err}(c)\| < \frac{p}{3t}$, then* GAHE.DecScalar$(\mathsf{sk}, c)$ *outputs m.*

Proof. Let $c = (pq + r + \lfloor p/t \rceil\, m)k$. Consider the polynomial $c' = c \cdot k^{-1} \in R/x_0R$ defined in GAHE.DecScalar. We can write it as $c' = pq' + r + \lfloor p/t \rceil\, m \in R$. Then, when we perform the reduction modulo p, we obtain $\bar{c} = [r + \lfloor p/t \rceil\, m]_p = [\mathsf{err}(c) + \lfloor p/t \rceil\, m]_p = \mathsf{err}(c) + \epsilon + mp/t - pu$ for some $\epsilon, u \in R$ with $\|\epsilon\| \leq 1/2$.

Thus, in the next step of the decryption function, we have

$$\frac{t\bar{c}}{p} = \frac{t(\mathsf{err}(c) + \epsilon)}{p} + m - ut.$$

But because $\|\mathsf{err}(c)\| < \frac{p}{3t}$, we have $\|t(\mathsf{err}(c) + \epsilon)/p\| < 1/3 + \|t\epsilon/p\| < 1/2$. Hence, since $m - ut$ has integer coefficients, the rounding function outputs

$$\left\lfloor \frac{t\bar{c}}{p} \right\rceil = \left\lfloor \frac{t(\mathsf{err}(c) + \epsilon)}{p} \right\rceil + m - ut = m - ut.$$

Therefore, the reduction modulo t indeed gives us m. □

Lemma 6.3.5 (Sufficient conditions for correctness of vector decryption)**.** *Let $\mathbf{c}$ be a vector encryption of $m \in R/tR$. If $\|\mathsf{err}(\mathbf{c})\| < \frac{p}{3N\ell bt}$, then* GAHE.DecVec$(\mathsf{sk}, \mathbf{c})$ *outputs m.*

Proof. Let $\alpha := \lfloor p/t \rceil$. Notice that GAHE.DecVec$(\mathsf{sk}, \mathbf{c})$ can be rewritten as

1. Compute a scalar encryption of the same message m, i.e., $c := g^{-1}([\alpha k]_{x_0}) \cdot \mathbf{c}$.
2. Output GAHE.DecScalar(sk, c).

But by definitions 6.3.1 and 6.3.2, we have

$$\mathsf{err}(c) = [(c \cdot k^{-1} - \alpha m \bmod x_0)]_p = [g^{-1}([\alpha k]_{x_0}) \cdot \mathsf{err}(\mathbf{c})]_p.$$

But $\left\|g^{-1}([\alpha k]_{x_0}) \cdot \mathsf{err}(\mathbf{c})\right\| \leq N\ell b \,\|\mathsf{err}(\mathbf{c})\| < p/(3t)$. Therefore, GAHE.DecScalar(sk, c) outputs m by Lemma 6.3.4. □

6.3.5 Correctness of homomorphic operations

Let's analyze the homomorphic operations and show that they produce ciphertexts encrypting the correct plaintext corresponding to the performed operation. For $i \in \{1, 2\}$, let $\mathbf{c}_i$ be a vector encryption of $v_i \in R$ and c_i be a scalar encryption of $s_i \in R$. Thus, we have $\mathbf{c}_i = (p\mathbf{q}_i + \mathbf{r}_i)k + \mathbf{g}v_i \in R^\ell$ and $c_i = (pq_i + r_i + \alpha s_i)k \in R$.

Hence, it is trivial that the homomorphic additions produce valid ciphertexts, i.e.,

- $c_1 + c_2 = (p(q_1 + q_2) + (r_1 + r_2) + \alpha(s_1 + s_2))k \in R$.

- $\mathbf{c}_1 + \mathbf{c}_2 = (p(\mathbf{q}_1 + \mathbf{q}_2) + (\mathbf{r}_1 + \mathbf{r}_2))k + \mathbf{g}(v_1 + v_2) \in R^\ell$.

To see that the homomorphic product of two vector ciphertexts is correct, notice that we decompose one of the operands, say, $\mathbf{c}_1$, as $\mathbf{A} = \left(g^{-1}(c_{1,1}) \ \ldots \ g^{-1}(c_{1,\ell})\right) \in R^{\ell \times \ell}$, and when we multiply $\mathbf{A}$ by $\mathbf{g}$, we obtain again $\mathbf{c}_1$, i.e., $\mathbf{g} \cdot \mathbf{A} = \mathbf{c}_1$. Hence, we have the following:

$$\begin{aligned}
\mathbf{c}_{mult} &= \mathbf{c}_2 \cdot \mathbf{A} \\
&= ((p\mathbf{q}_2 + \mathbf{r}_2)k + \mathbf{g}v_2) \cdot \mathbf{A} \\
&= (p\mathbf{q}_2\mathbf{A} + \mathbf{r}_2\mathbf{A})k + \mathbf{g}\mathbf{A}v_2 \\
&= (p\mathbf{q}_2\mathbf{A} + \mathbf{r}_2\mathbf{A})k + ((p\mathbf{q}_1 + \mathbf{r}_1)k + \mathbf{g}v_1)v_2 \\
&= (p\underbrace{(\mathbf{q}_2\mathbf{A} + \mathbf{q}_1 v_2)}_{\mathbf{q}_{mult}} + \underbrace{(\mathbf{r}_2\mathbf{A} + \mathbf{r}_1 v_2)}_{\mathbf{r}_{mult}})k + \mathbf{g}v_1v_2
\end{aligned}$$

Therefore, the homomorphic multiplication yields a valid encryption of the product of the messages.

Let's now analyze the mixed homomorphic product. To simplify the notation, define $\mathbf{y} := g^{-1}(c_1) \in R^\ell$. Thus, $\mathsf{GAHE.MultMix}(c_1, \mathbf{c}_1)$ outputs $c_{mult} := \mathbf{y} \cdot \mathbf{c}_1$ and the following

holds:

$$\begin{aligned}
c_{mult} &= (p\mathbf{y}\mathbf{q}_1 + \mathbf{y}\mathbf{r}_1)k + \mathbf{y}\mathbf{g}v_1 && \text{(By definition of } \mathbf{c}_1\text{)}\\
&= (p\mathbf{y}\mathbf{q}_1 + \mathbf{y}\mathbf{r}_1)k + c_1 v_1 && \text{(Because } \mathbf{y}\mathbf{g} = c_1\text{)}\\
&= (p\mathbf{y}\mathbf{q}_1 + \mathbf{y}\mathbf{r}_1)k + (pq_1 + r_1 + \lfloor p/t \rceil\, s_1 k)v_1 && \text{(By definition of } c_1\text{)}\\
&= (p\underbrace{(\mathbf{y}\mathbf{q}_1 + q_1 v_1)}_{q_{mult}} + \underbrace{(\mathbf{y}\mathbf{r}_1 + r_1 v_1)}_{r_{mult}} + \lfloor p/t \rceil\, s_1 v_1)k && \text{(Grouping the terms)}
\end{aligned}$$

Therefore, the mixed homomorphic product takes encryptions of s_1 and v_1 and produces $c_{mult} = (pq_{mult} + r_{mult} + \lfloor p/t \rceil\, s_1 v_1)k \in R$, which is a valid scalar encryption of the product of the messages, as expected.

6.3.6 Noise growth of homomorphic operations

In this section we show that the noise in the ciphertexts grows basically additively when we perform any homomorphic operation, including products. Using the analysis done in Section 6.3.5, it is easy to derive upper bounds to the noise accumulated by the homomorphic operations.

Lemma 6.3.6 (Noise of homomorphic additions)**.** *Let n be an integer larger than or equal to 2. For $i \in [\![1, n]\!]$, let c_i be a scalar encryption of s_i and $\mathbf{c}_i$ be a vector encryption of v_i. Compute the homomorphic sum of these ciphertexts as follows: $c := \sum_{i=1}^{n} c_i \in R$ and $\mathbf{c} := \sum_{i=1}^{n} \mathbf{c}_i \in R^\ell$. Then, $\mathsf{err}(c) = \sum_{i=1}^{n} \mathsf{err}(c_i)$ and $\mathsf{err}(\mathbf{c}) = \sum_{i=1}^{n} \mathsf{err}(\mathbf{c}_i)$. In particular, if all c_i's and $\mathbf{c}_i$'s are fresh ciphertexts, we have*

$$\|\mathsf{err}(c)\| < n2^\rho \text{ and } \|\mathsf{err}(\mathbf{c})\| < n2^\rho.$$

Proof. Because each c_i is of the form $(pq_i + r_i + \lfloor p/t \rceil\, s_i)k$, it is clear that $\mathsf{err}(c) = \sum_{i=1}^{n} r_i = \sum_{i=1}^{n} \mathsf{err}(c_i)$. By Lemma 6.3.3, if all c_i's are fresh ciphertexts, we have $\|\mathsf{err}(c)\| \leq \sum_{i=1}^{n} \|\mathsf{err}(c_i)\| < n2^\rho$ and the particular case holds.

The same holds for vector ciphertexts. □

Lemma 6.3.7 (Noise growth of mixed products)**.** *Let $n \in \mathbb{N}^*$. For all $i \in [\![1, n]\!]$, let $\mathbf{c}_i$ be a vector encryption of m_i. Let also c_0 be a scalar encryption of m_0. Assume that B is an*

upper bound to the norm of the products of plaintexts, i.e., $\left\|\prod_{i=j}^{n} m_i\right\| \leq B$ *for* $0 \leq j \leq n$. *Finally, for* $1 \leq i \leq n$, *define* $c_i := \mathsf{GAHE.MultMix}(c_{i-1}, \mathbf{c}_i) \in R$ *(notice that* c_i *is a scalar encryption of* $\prod_{j=0}^{i} m_j$*). Then,*

$$\|\mathsf{err}(c_n)\| < NB \|\mathsf{err}(c_0)\| + \sum_{i=1}^{n} N^2 B\ell b \|\mathsf{err}(\mathbf{c}_i)\| . \tag{6.1}$$

In particular, if c_0 *and all the* $\mathbf{c}_i$*'s are fresh ciphertexts, then*

$$\|\mathsf{err}(c_n)\| < 2N^2 B\ell bn 2^{\rho}. \tag{6.2}$$

Proof. By the analysis done in Section 6.3.5, we know that the noise term r_i of c_i is $g^{-1}(c_{i-1})\mathbf{r}_i + r_{i-1}m_i$. Hence, the term r_n after n homomorphic products is

$$r_n = r_0 \prod_{i=1}^{n} m_i + \sum_{i=1}^{n} g^{-1}(\mathbf{c}_{i-1})\mathbf{r}_i \left(\prod_{j=i+1}^{n} m_j\right) \in R.$$

Thus,

$$\begin{aligned}
\|r_n\| &\leq \left\| r_0 \prod_{i=1}^{n} m_i \right\| + \left\| \sum_{i=1}^{n} g^{-1}(c_{i-1})\mathbf{r}_i \left(\prod_{j=i+1}^{n} m_j\right) \right\| \\
&\leq N \|r_0\| \left\| \prod_{i=1}^{n} m_i \right\| + \sum_{i=1}^{n} N\ell \left\| g^{-1}(\mathbf{c}_i) \right\| \left\| \mathbf{r}_i \left(\prod_{j=i+1}^{n} m_j\right) \right\| \\
&\leq NB \|r_0\| + \sum_{i=1}^{n} N^2 \ell b B \|\mathbf{r}_i\| .
\end{aligned}$$

Therefore, Inequality 6.1 holds. By Lemma 6.3.3, if all the operands are fresh ciphertexts, then, $\|r_0\| < 2^{\rho}$ and $\|\mathbf{r}_i\| < 2^{\rho}$, and the particular case also holds. □

The noise growth of a sequence of homomorphic products involving only vector ciphertexts is essentially equal to the one of mixed products.

Lemma 6.3.8 (Noise growth of products of vector ciphertexts)**.** *Let* n *be an integer larger than or equal to* 1. *For* $i \in [\![0, n]\!]$, *let* $\mathbf{c}_i$ *be an encryption of* m_i. *Let also* $\mathbf{c}'_0 := \mathbf{c}_0$ *and* $\mathbf{c}'_i := \mathsf{GAHE.MultVec}(\mathbf{c}'_{i-1}, \mathbf{c}_i)$ *for* $i > 0$. *(Notice that* $\mathbf{c}'_i$ *is an encryption of* $\prod_{j=0}^{i} m_j$*). Assume that* B *is an upper bound to the product of the plaintexts, i.e.,* $\left\|\prod_{i=j}^{n} m_i\right\| \leq B$

for $0 \leq j \leq n$. *Then,*

$$\left\|\mathsf{err}(\mathbf{c}'_n)\right\| < NB \left\|\mathsf{err}(\mathbf{c}_0)\right\| + \sum_{i=1}^{n} N^2 B\ell b \left\|\mathsf{err}(\mathbf{c}_i)\right\|.$$

In particular, if all the products only involve fresh ciphertexts, then

$$\left\|\mathsf{err}(\mathbf{c}'_n)\right\| < 2N^2 B\ell bn 2^{\rho}.$$

Proof. This proof is essentially the same as the one of Lemma 6.3.7, therefore, we omit it. □

6.3.7 Semantic security

In this section, we prove that assuming the hardness of the RAGCD problem and the circular security, our scheme is semantic secure, i.e., no PPT adversary can distinguish encryptions of any pair of messages. As in Section 5.4.1, we first show that vector ciphertexts are computationally indistinguishable, then we prove that scalar ciphertexts are also so, and these two results imply the semantic security of the scheme.

Lemma 6.3.9. *Under the decisional-RAGCD assumption, vector encryptions of any pair of polynomials are computationally indistinguishable.*

Proof. Via a sequence of hybrids we prove that no PPT adversary $\mathcal{A}$ can distinguish between vector encryptions of two polynomials m_0 and m_1 of their choice.

Hybrid H_0: Use the key generation function to get sk and the public parameters params. Given m_0 and m_1 chosen by $\mathcal{A}$, we always encrypt m_0, i.e, we let $\mathbf{c}_0 := \mathsf{GAHE.EncVec}(\mathsf{sk}, m_0)$ and return $\mathbf{c}_0$ to $\mathcal{A}$. ◇

Hybrid H_1: In this hybrid we use $\mathcal{U}(R/x_0R)$ instead of $\mathcal{R}_{N,\gamma,\rho,x_0}(p,k)$ to encrypt m_0, i.e., we sample $\mathbf{x}_1 \leftarrow \mathcal{U}(R/x_0R)^{\ell}$ and define $\mathbf{c}_1 := \mathbf{x}_1 + \mathbf{g} \cdot m_0 \bmod x_0$. ◇

In H_0 we have $\mathbf{c}_0 := \mathbf{x}_0 + \mathbf{g} \cdot m_0 \bmod x_0$ for some $\mathbf{x}_0 \leftarrow (\mathcal{R}_{N,\gamma,\rho,x_0}(p,k))^{\ell}$. By hypothesis, $\mathbf{x}_0$ and $\mathbf{x}_1$ are computationally indistinguishable, thus, $\mathbf{c}_0$ and $\mathbf{c}_1$ are also so.

Hybrid H_2: In this hybrid, we ignore the two plaintexts, we sample $\mathbf{x}_2 \leftarrow \mathcal{U}(R/x_0R)^\ell$, and define $\mathbf{c}_2 := \mathbf{x}_2$. $\diamond$

We know that for any $h \in R/x_0R$, the distributions $\mathcal{U}(R/x_0R)$ and $\mathcal{U}(R/x_0R) + h \bmod x_0$ are the same. Consequently, the vector $\mathbf{c}_1$ defined in hybrid H_1 follows a uniform over R/x_0R. Therefore, $\mathbf{c}_1$ is indistinguishable from $\mathbf{c}_2$.

Hybrid H_3: In this hybrid, we encrypt m_1 using $\mathcal{U}(R/x_0R)$, i.e., we sample $\mathbf{x}_3 \leftarrow \mathcal{U}(R/x_0R)^\ell$ and define $\mathbf{c}_3 := \mathbf{x}_3 + \mathbf{g} \cdot m_1 \bmod x_0$. $\diamond$

By the same argument used in the transition from H_1 to H_2, we see that $\mathbf{c}_2$ and $\mathbf{c}_3$ are indistinguishable, therefore, $\mathcal{A}$'s advantage in distinguishing H_2 from H_3 is negligible.

Hybrid H_4: In this hybrid, we replace $\mathcal{U}(R/x_0R)$ with $\mathcal{R}_{N,\gamma,\rho,x_0}(p,k)$ to get a valid encrypt of m_1, that is, we define $\mathbf{c}_4 := \mathsf{GAHE.EncVec}(\mathsf{sk}, m_1)$ and return $\mathbf{c}_4$ to $\mathcal{A}$. $\diamond$

By the same argument used in the transition from H_0 to H_1, we conclude that $\mathcal{A}$'s advantage in distinguishing between H_3 and H_4 is negligible. Since $\mathcal{A}$'s advantage in each transition is negligible, it holds that

$$\left|\Pr_{H_0}[\mathcal{A}(1^\lambda, \mathsf{params}, \mathbf{c}_0) = 1] - \Pr_{H_4}[\mathcal{A}(1^\lambda, \mathsf{params}, \mathbf{c}_4) = 1]\right| \leq \mathrm{negl}(\lambda)\,.$$

But this is exactly the definition of $\mathcal{A}$'s advantage in distinguishing vector encryptions of m_0 from vector encryptions of m_1. $\square$

Lemma 6.3.10. *Under the decisional-RAGCD assumption and the circular security assumption, scalar encryptions of any pair of polynomials are computationally indistinguishable.*

Proof. The procedure $\mathsf{GAHE.EncScalar}(\mathsf{sk}, m)$ can be implemented in terms of the vector encryption as follows: "let $\mathbf{c} := \mathsf{GAHE.EncVec}(\mathsf{sk}, m \cdot k \cdot \lfloor p/t \rceil)$ and output c_1, the first entry of $\mathbf{c}$". Moreover, assuming circular security means that $\mathsf{GAHE.EncVec}$ remains semantic secure even if the message to be encrypted depends on sk. Therefore, because the decisional-RAGCD assumption implies that outputs produced by $\mathsf{GAHE.EncVec}$ are

computationally indistinguishable, the ciphertexts produced by GAHE.EncVec are also so. □

Theorem 6.3.11. *The scheme is CPA-secure under the decisional-RAGCD assumption.*

Proof. This follows directly from Lemma 6.3.9 and Lemma 6.3.10. □

6.4 Practical results

In this section we show how to choose the parameters for our scheme, we present the running times and memory usage of basic procedures, and compare the scheme for vector and matrices presented in Chapter 5 with this polynomial scheme. We have implemented a proof-of-concept in C++ using the Number Theory Library[2] (NTL) and it is publicly available on [Per20e]. We ran the experiments on a single core of a processor Intel Core i5-8600K 3.60GHz, of a machine with 32GB of RAM memory.

6.4.1 Parameter selection of GAHE

Firstly we recall the definitions of the parameters used in our GAHE scheme:

- N: we work over the cyclotomic ring $R := \mathbb{Z}[x]/\langle x^N + 1\rangle$, where N is a power of two;
- η: we sample the secret prime p uniformly from $[\![2^{\eta-1}, 2^{\eta}]\!]$;
- ρ: during encryption, we sample the noise terms uniformly from $]\!] - 2^{\rho}, 2^{\rho}[\![$;
- γ: the private modulus x_0 satisfies $2^{\gamma-1} \leq x_0 < 2^{\gamma}$
- t: the message space is $R/tR = \mathbb{Z}_t[x]/\langle x^N + 1\rangle$;
- b: base in which we perform the decomposition g^{-1};
- ℓ: number of words used in g^{-1}. Vector ciphertexts belong to R^{ℓ}.

As explained in Section 6.2.1 to guarantee security, we must set $\gamma \in \Omega\left(\frac{\lambda(\eta-\rho)^2}{N \log \lambda}\right)$ to rule out lattice attacks and $\rho \in \Omega(\lambda/N)$ to avoid GCD attacks. If we let L be the maximum

[2] https://www.shoup.net/ntl/

multiplicative depth to be evaluated, then, the correctness of the decryption functions imposes the following constraint: By lemmas 6.3.4 and 6.3.5, decryption works if the final noise is smaller than $p/(3t)$. Thus, we can use $2^\eta/(6t) < p/(3t)$ as an acceptable bound to the noise. By Lemma 6.3.7, the final noise is upper bounded by $2N^2BL\ell b2^\rho$, thus, we need $2N^2BL\ell b2^\rho \leq 2^\eta/(6t)$, or, equivalently,

$$\eta \geq \rho + \log(tN^2BL\ell b) + \log(12).$$

Therefore, considering that B, t, L, and N are fixed, and that $\ell := \lceil \gamma/\log(b) \rceil$, i.e., ℓ is defined by b, we can choose ρ and b such that

$$\eta - 2\log N - \log(tBL) - \log(\ell) - \log(12) = \rho + \log(b),$$

which is very similar to Equation 5.3. Hence, as in Section 5.4.2, we can choose ρ and $\log b$ such that $\eta \approx \rho + \log b$, say $\eta(1-\epsilon) = \rho + \log b$ for some $0 \leq \epsilon < 1$. The bit length of the vector ciphertexts is $\gamma\ell N$ and the time complexity of the mixed homomorphic product is dominated by $\gamma\ell N \log N$ (ℓ products of degree-N polynomials whose coefficients have γ bits). Notice that $\gamma\ell \approx \frac{\gamma^2}{\log b} \approx \left(\frac{\lambda(\eta-\rho)^2}{N\log\lambda}\right)^2 \cdot \frac{1}{\log b}$ is minimized when $\log b = \epsilon\eta/3$. Thus, to choose the parameters for a security level of λ bits and to minimize the cost of the main procedures of GAHE, we can fix $\eta \geq \lambda$, then set a value for ϵ, e.g., $\epsilon := (\log(tN^2BL\ell b) + \log(12))/\eta$, and finally set $\rho = \lfloor 2\epsilon\eta/3 \rfloor$ and $\log b = \lfloor \epsilon\eta/3 \rfloor$.

6.4.2 Comparison: GAHE versus HEVAM

Consider that $R := \mathbb{Z}[x]/\langle f \rangle$ for some degree-N polynomial f. As described in Section 2.5, we can represent R using vectors and matrices, therefore, the scheme proposed in Chapter 5, which we name HEVAM, is already enough to perform homomorphic computations over R. In view of this, one could wonder if our GAHE scheme is really necessary. Hence, in this section, we compare these two schemes.

Firstly, notice that both the vector ciphertext of HEVAM and the scalar ciphertext of GAHE have bit length $N\gamma$. However, the vector ciphertext of GAHE has bit length $N\ell\gamma$ instead of the bit length $N^2\ell\gamma$ of the matrix ciphertext of HEVAM. Thus, by using

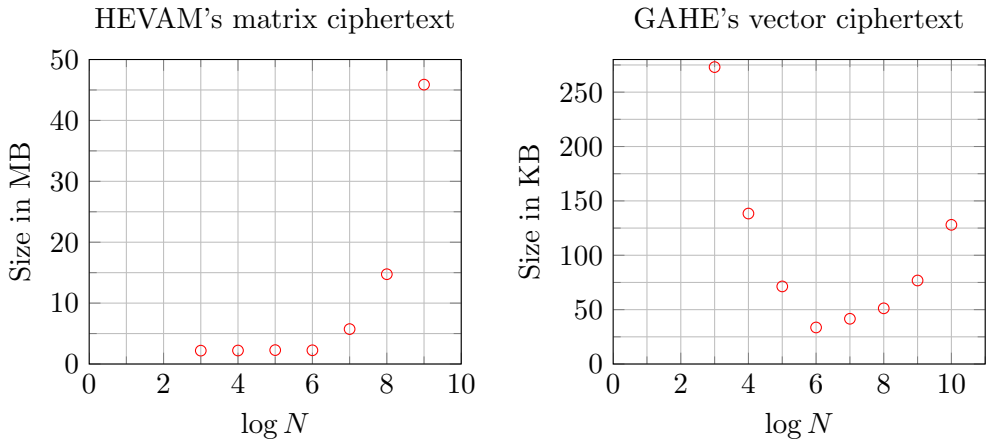

Figure 6.1: Size of ciphertext encrypting degree-N polynomial.

GAHE, we gain a factor of N when encrypting polynomials. Moreover, by using HEVAM to work homomorphically over R, the vector-matrix product corresponds to the GAHE's mixed product, but in the vector-matrix product, one has to multiply an $N\ell$-dimensional vector by an $N\ell \times N$ matrix of γ-bit integers, thus, the cost is $\Omega(N^2\ell\gamma)$, while in the mixed product, on needs to multiply ℓ polynomials of degree N with γ-bit coefficients, thus, the cost is $\Omega(N\log(N)\ell\gamma)$, i.e., we replace a factor N by $\log N$.

We instantiate GAHE and HEVAM using the parameters presented in Table 5.4 and ran GAHE's mixed product and HEVAM's vector-matrix product in a processor Intel Core i5-8600K 3.60GHz, using a single core. In Figure 6.1, we compare the size of the corresponding ciphertexts of GAHE and HEVAM. We can see that even for $N = 2^{10}$, a vector ciphertext produced by GAHE is only a few kilobytes long, while the matrix ciphertexts produced by HEVAM are already 2 MB bytes long for small values of N, like 2^3 and 2^4. In Figure 6.2, we compare the running times of the mixed homomorphic product with those of HEVAM's vector-matrix homomorphic multiplication. For small values of N, the running times are basically the same, but as N increases, HEVAM's homomorphic product becomes much more expensive, e.g., for $N = 2^9$, the mixed homomorphic product takes only 20 milliseconds while the vector-matrix homomorphic multiplication runs in approximately 130 milliseconds.

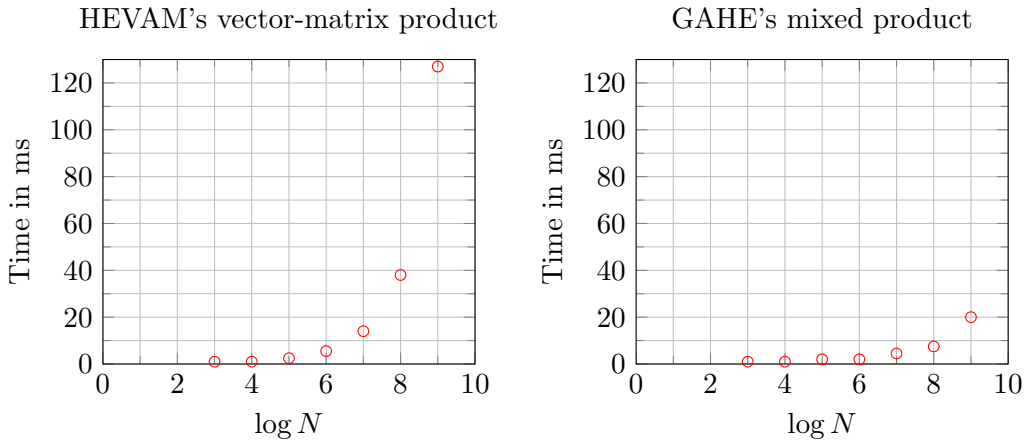

Figure 6.2: Running times of homomorphic product of degree-N polynomials.

6.5 Conclusion

In this chapter, we presented a new variant of the AGCD problem in which the approximate multiples of p are grouped as polynomials then randomized with a secret fixed polynomial. This variant, called RAGCD problem, is similar to the vector AGCD problem defined in Chapter 3 and also allows to reduce the size of the parameters of the original AGCD problem. Moreover, it is more suitable to work over polynomial rings. Thus, using the RAGCD problem, we proposed a new encryption scheme that can perform homomorphic operations on rings of the form $\mathbb{Z}[x]/\langle f \rangle$. This scheme can encrypt polynomials into vectors and into scalars, and its possible to efficiently multiply ciphertexts of these two types via mixed homomorphic products. The cryptanalysis is done by reducing the RAGCD problem to the vector AGCD problem. We also analyzed simple attacks exploiting non trivial factors of the modulus polynomial f. It would certainly be interesting to study if the ring structure can be used in other ways to yield more efficient attacks on the RAGCD problem.

Chapter 7

New bootstrapping techniques

7.1 Introduction

The two main families of fully homomorphic encryption (FHE) schemes are the ones based on lattices, mainly on the Ring Learning with Errors (RLWE) problem, and the schemes over the integers, based on the Approximate Greatest Common Divisor (AGCD) problem. Immediately after the first FHE scheme was proposed by Gentry [Gen09], a scheme over the integers was put forth as a simpler alternative [DGHV10]. Thereafter, several techniques were proposed to improve the efficiency of FHE, and one always found ways to apply those techniques to both families of homomorphic schemes. For example, a method to reduce the noise by scaling a ciphertext and switching the modulus of the ciphertext space, known as modulus switching, was proposed in [BV11] and was soon adapted for schemes over the integers [CNT12]. A technique known as batching, which consists in encrypting several messages into a single ciphertext so that each homomorphic operation acts in parallel on all the encrypted messages, has also been applied to RLWE schemes [BGV12, GHS12] and to schemes over the integers [CCK+13]. Finally, in 2013, Gentry, Sahai, and Waters introduced a FHE scheme that uses a decomposition technique to turn the noise growth of homomorphic products roughly additive [GSW13], i.e., the homomorphic product of two ciphertexts c and c' yields a ciphertext c_{mult} whose noise is approximately the noise of c plus the noise of c'. Even this technique was adapted to the schemes over the integers [BBL17].

However, new fast bootstrapping techniques, one of the last great achievements of FHE, has only been availed for (R)LWE schemes: In [ASP14], it was proposed to bootstrap a base scheme whose ciphertext space is $\mathbb{Z}_q$ by using a GSW-like scheme whose plaintext space contains $\mathbb{Z}_q$. Because of the slow noise growth of GSW-like schemes, the final noise accumulated in the refreshed ciphertext is only polynomial in the security parameter λ, therefore, it is not necessary to set large parameters for the base scheme as it was done in previous bootstrapping methods, where the parameters have to allow a scheme to evaluate its own decryption function. Then, in [DM15], the authors found an efficient way to represent $\mathbb{Z}_q$, removed the expensive final step of the method proposed in [ASP14], and implemented a boostrapping that runs in less than one second in a common laptop using a GSW-like scheme based on the RLWE. The running times of [DM15] were further improved in [CGGI16a] and a base scheme based on LWE was bootstrapped in less than 0.1 second also using a RLWE-based GSW-like scheme. Nevertheless, none of those techniques has been adapted to FHE over the integers.

The main difficulties one has to deal with when trying to create similar bootstrapping methods for FHE over the integers are:

1. One needs an efficient GSW-like scheme based on the AGCD problem. For instance, the GSW-like scheme proposed in [BBL17] is far from practical.

2. The modulus p is secret: the decryption function of (R)LWE-based schemes is defined modulo a *public* integer q, thus, all the homomorphic operations performed during the bootstrapping can safely disclose q, but for AGCD-based schemes, we have an integer p which is at the same time the modulus and the secret key, hence, the bootstrapping must hide p.

3. The modulus p is exponentially large in λ: in (R)LWE-based schemes, the modulus q is just polynomially large in the security parameter, while in FHE over the integers we have $p \in \Omega(2^\lambda)$, thus, the techniques to perform fast bootstrapping require that the message space of the GSW-like scheme contain a set that is exponentially large, i.e., the set $\mathbb{Z}_p$.

Thus, in this chapter, we aim to close the gap between (R)LWE- and AGCD-based schemes by proposing fast bootstrapping methods for FHE over the integers. Namely, using the RAGCD problem and the GSW-like scheme proposed in Chapter 6, we show how to perform gate bootstrapping, as in [DM15, CGGI16a], and then a more general multi-value bootstrapping, as in [CIM19]. We implemented a proof of concept in C++ and refreshed ciphertexts of FHE schemes over the integers in less than one second.

7.1.1 Overview of our techniques and results

Fast bootstrapping for FHE over the integers: Firstly, we notice that simply trying to implement the bootstrapping procedures of [DM15] or [CGGI16a] with our GSW-like scheme would not work, since it would require $N > p \in \Omega(2^\lambda)$, which would not be efficient and would also leak p. Therefore, to solve these issues related to the size and the privacy of the modulus used in the decryption of AGCD-based schemes, we propose to perform a "hidden approximate modulus switching". Namely, given a ciphertext $c = pq + r + mp/t$ to be bootstrapped, multiplying it by N/p would switch the modulus, resulting in $c' = Nq + r' + mN/t$, for a potentially small N that could be managed by our scheme. Of course, we cannot do it before refreshing because having access to N/p would leak p. Even if we could perform the modulus switching in a secure way, without revealing p, as in [CNT12], the resulting ciphertext c' would leak the message m, because N is known. Thus, we propose that the product $c \cdot N/p$ be performed as part of the refreshing procedure, so that the secret key p is encrypted in the bootstrapping keys and the resulting ciphertext c' is only produced in an encrypted form.

Essentially, since $y := x^2$ has order N in $R := \mathbb{Z}[x]/\langle x^N + 1\rangle$, we can use our GSW-like scheme, which we name GAHE, to work homomorphically over $\mathbb{Z}_N$. Thus, we would like to encrypt messages $y^{2^i N/p}$ for $0 \leq i < \gamma$, and then, to refresh a γ-bit ciphertext $c = pq + r + mp/t$ of the base scheme, we would decompose c obtaining $(c_0, ..., c_{\gamma-1})$ and use the homomorphic mixed product to compute a GAHE ciphertext $\tilde{c}$ encrypting $\prod_{i=0}^{\gamma-1} y^{c_i 2^i N/p \bmod N} = y^{r'+mN/t}$. The problem now is that $2^i N/p$ is not integer. Hence, we encrypt $y^{\lfloor 2^i N/p \rceil}$ instead. By noticing that $\lfloor 2^i N/p \rceil = 2^i N/p + \epsilon_i$ for some $\epsilon_i \in [-1/2, 1/2]$,

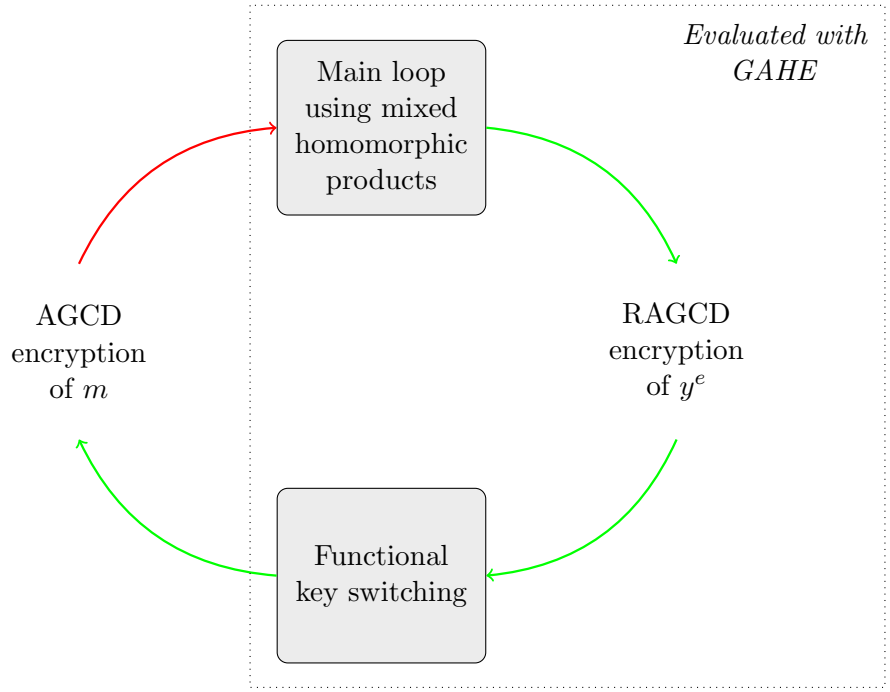

Figure 7.1: Two steps of our single-bit bootstrapping. Its input is an encryption of m under the AGCD problem with large noise and the output is an encryption of the same message with less noise.

we see that by computing the same sequence of homomorphic products, we have a GAHE encryption of y^e, with $e = r' + mN/t + \epsilon$ for some term ϵ that is not too big. Finally, we propose a functional key switching to transform this GAHE ciphertext into a base scheme (AGCD-based) encryption of m. By choosing the parameters carefully, the noise term of the final ciphertext is smaller than the initial noise. This procedure is illustrated in Figure 7.1.

Functional key switching: We propose a procedure to transform ciphertexts by switching the secret key under which they are encrypted and also applying some function to the message that is encrypted. Namely, given a ciphertext c encrypting a message m under key sk, our functional key-switching procedure produces a new ciphertext $\bar{c}$ that encrypts $\phi(m) \cdot \mathbf{u}$, where $\phi(m)$ is the vector of coefficients of m and $\mathbf{u}$ is an arbitrary vector. Depending on how the parameters are chosen, this procedure can be used to

switch the underlying problem from the RAGCD to the original AGCD problem and vice versa; or to reduce the noise of a ciphertext; or to change the message that is encrypted. In our bootstrapping methods proposed in this chapter, the functional key switching is used to transform a ciphertext of the GAHE scheme into a ciphertext of the base scheme (the scheme that is being bootstrapped). Furthermore, when compared to other key- or modulus-switching procedures for AGCD based schemes, as the one proposed in [CNT12], our procedure is more general and seems more straightforward.

More general multi-value bootstrapping: Finally, we adapt our bootstrapping procedure for a more general scenario in which a ciphertext of the base scheme has the form $(pq + r + \alpha m)k \in R$, that is, it is also a polynomial, and the message m belongs to $\mathbb{Z}_t$ instead of $\{0, 1\}$. We do so by using our functional key switching to securely transform polynomial ciphertexts encrypted under the RAGCD problem into integer ciphertexts based on the AGCD problem. From this point on, we can basically apply our original bootstrapping, but taking into account that the message space is now $\mathbb{Z}_t$.

Implementation and practical results: We implemented our bootstrapping procedures in C++ and executed experiments similar to those of [DM15], [CGGI16a], and [CIM19]. Although our implementation is not optimized, we obtained running times and memory consumption similar to [DM15], i.e., we could bootstrap the base scheme in less than one second. For the best of our knowledge, all the previous bootstrapping methods for FHE over the integers took several seconds (or even minutes). As for the multi-value bootstrapping, we could execute it in about the same time of [CIM19] and using approximately half of the memory. Our implementation is publicly available. All the details are shown in Section 7.6.

7.2 Fast bootstrapping for (R)LWE-based schemes

In this section, we recapitulate in chronological order the fast bootstrapping methods proposed to FHE schemes based on the LWE and the RLWE problem.

7.2.1 Using permutation groups

In [ASP14], Alperin-Sheriff and Peikert considered the problem of bootstrapping a scheme that uses the LWE problem to encrypts a message $m \in \{0,1\}$ as

$$\mathbf{c} := (\mathbf{a}, b := \mathbf{as} + 2e + m \bmod q) \in \mathbb{Z}_q^{n+1}.$$

Given the secret key $\mathbf{s}$, we can extend it as $\bar{\mathbf{s}} := (-\mathbf{s}, 1)$ and the decryption function of this base scheme is then

$$[\mathbf{c} \cdot \bar{\mathbf{s}}]_q \bmod 2 = [b - \mathbf{as}]_q \bmod 2 = 2e + m \bmod 2 = m.$$

At the time, the current way of performing bootstrapping was by setting large parameters for the scheme and using it to evaluate its own decryption function. However, Alperin-Sheriff and Peikert proposed to separate the scheme that will be bootstrapped from the scheme that will be used to evaluate the decryption function. Moreover, they suggested to instantiate the GSW scheme to bootstrap the base scheme. The first part of the bootstrapping consists in taking $\mathbf{c}$ and using the GSW to homomorphically compute $[\mathbf{c}\cdot\bar{\mathbf{s}}]_q$. To do so, we represent the ring $\mathbb{Z}_q$ as a direct product $\prod \mathbb{Z}_{q_i}$, for small factors q_i's, and use symmetric groups of order q_i to represent $\mathbb{Z}_q$. This boils down to using $q_i \times q_i$ permutation matrices to represent elements of $\mathbb{Z}_q$ and to multiplying such permutation matrices to perform additions modulo q. The reason to factor q like this is that a symmetric group of order q would be too big to be practical, but several symmetric groups of small order are manageable. Then, we use the GSW scheme to encrypt $\bar{\mathbf{s}}$ as elements of symmetric groups and use the GSW's homomorphic product to evaluate $[\mathbf{c} \cdot \bar{\mathbf{s}}]_q$, obtaining thus a GSW encryption of $2e + m$. The final step consists in evaluating homomorphically the function $f(x) = x \bmod 2$ and transforming the GSW encryption in a valid ciphertext of the base scheme. This is done with a smart equality test that is executed for all $x \in \mathbb{Z}_q$ such that $x \bmod 2 = 1$, and thus, it is an expensive step, costing $\Theta(q)$ homomorphic operations.

Notice that trying to adapt this technique to FHE over the integers means that we have to leak the factors of the secret p, or at least some upper bound P on these factors.

But doing so would probably break the security of the base scheme. A second problem is that the final step would take time $\Theta(p) = \Theta(2^\lambda)$.

7.2.2 Using polynomials

In [DM15], Ducas and Micciancio observed that in the polynomial ring $\mathbb{Z}[x]/\langle x^N + 1\rangle$ the element $y := x^{2N/q}$ has order q. Thus, the multiplicative group $\mathcal{G} := \langle y\rangle$ is isomorphic to $\mathbb{Z}_q$, in other words, we can map elements a_i from $\mathbb{Z}_q$ to $y^{a_i} \in \mathcal{G}$ and to add $a_i + a_j \bmod q$ we compute $y^{a_i} \cdot y^{a_j} \bmod x^N + 1 = y^{a_i + a_j \bmod q}$. Additionally, representing $\mathbb{Z}_q$ with $\mathcal{G}$ is more efficient than using symmetric groups, as it was proposed in [ASP14], since it allows us to instantiate a GSW-like scheme based in the RLWE instead of the LWE and to evaluate the decryption function of the base scheme by multiplying low-dimensional polynomial matrices instead of high-dimensional integral matrices.

Then, the authors of [DM15] propose a gate bootstrapping, i.e., they propose a simple base scheme that encrypts one bit and can evaluate one binary gate homomorphically, then it has to be bootstrapped. Thus, evaluating a binary circuit with this scheme requires that we perform the refreshing function after each gate. The binary gates are very efficient as they are performed with $\Theta(n)$ simple additions modulo q, hence, the expensive part is to refresh the resulting ciphertext. The base scheme uses the LWE problem to encrypt a message m as $\mathbf{c} := (\mathbf{a}, b := \mathbf{as} + e + mq/t \bmod q) \in \mathbb{Z}_q^{n+1}$. The bootstrapping keys are GSW encryptions of the secret key $\mathbf{s}$ essentially as follows: $\mathfrak{K}_{i,j} = \mathsf{GSW.Enc}(y^{-2^i \cdot s_j})$ for $0 \le i \le \ell := \log(q)$ and $1 \le j \le n$. Then, given a ciphertext $\mathbf{c} = (\mathbf{a}, b)$ to be refreshed, we write $\mathbf{a} = (a_1, ..., a_n)$, decompose each a_j in base 2, obtaining $(a_{0,j}, ..., a_{\ell-1,j})$, and the first step consists basically in using GSW's homomorphic product to compute $b - \mathbf{as} = e + mq/t$ as follows:

$$\mathsf{GSW.Enc}(y^b) \cdot \prod_{j=1}^{n} \prod_{0 \le i < \ell} \mathfrak{K}_{i,j} = \mathsf{GSW.Enc}(y^{b - \sum_{j=1}^{n} a_{i,j} s_j \bmod q}) = \mathsf{GSW.Enc}(y^{b - \mathbf{as} \bmod q}).$$

The second step consists in transforming a GSW encryption of $y^{e+mq/t}$ in a base scheme ciphertext encrypting m. Roughly speaking, this is done by taking the coefficient vector of one specific row of the GSW ciphertext and multiplying it by a fixed vector, then,

applying a modulus- and a key-switching. Therefore, the final step of this bootstrapping method is much cheaper than the one proposed in [ASP14].

Notice that in the context of AGCD-based schemes, q would be replaced by a secret $p \in \Omega(2^\lambda)$ and we would need $N \approx p$, thus, the degree of the polynomials encrypted by the GSW-like scheme would be exponentially large. Moreover, since N would be public and $2N \in p\mathbb{Z}$, it would be possible to recover p.

7.2.3 The TFHE bootstrapping

In [CGGI16a], the authors noticed that instead of simply using the GSW homomorphic product, which consists in multiplying matrices of polynomials, we can perform the bootstrapping using a mixed product in which one operand is an LWE ciphertext (thus, a vector) and the another one is a GSW ciphertext (thus, a matrix), resulting then in a LWE ciphertext (again a vector). The authors called it an external product. This speeds up the bootstrapping since it replaces matrix-matrix products by vector-matrix multiplications.

7.3 Functional Key-Switching

In this section we define a procedure that will play a main role in our bootstrapping methods, namely, a *functional* key-switching. Therewith we can change the keys and the dimension of the polynomial ring of a ciphertext and at the same time apply some function to the plaintext. That is to say, given two integers N_1 and N_2, we define two polynomial rings $R_i := \mathbb{Z}[x]/\langle x^{N_i} + 1\rangle$. Then, we can transform a scalar ciphertext $c_1 \in R_1$ that encrypts a message $m \in R_1/tR_1$ under key $\mathsf{sk}_1 := (p_1, k_1)$ into a ciphertext c_2 encrypting $\boldsymbol{\phi}(m) \cdot \mathbf{u}$ under another key $\mathsf{sk}_2 := (p_2, k_2)$ for any $\mathbf{u} \in R_2^{N_1}$, where $\boldsymbol{\phi}(m) \in \mathbb{Z}^{N_1}$ is the coefficient vector of m. For example, considering that $m = \sum_{i=0}^{N_1-1} m_i x^i$, by choosing $\mathbf{u} := (1, ..., 1) \in \mathbb{Z}^{N_1}$, we see that $\boldsymbol{\phi}(m) \cdot \mathbf{u} = \sum_{i=0}^{N_1-1} m_i$, thus, the functional key-switching produces an encryption of the sum of m's coefficients.

Our functional key-switching works as follows: Consider a scalar ciphertext c_1 en-

crypting m_1 under sk_1, thus, $c_1 = (p_1q_1 + r_1 + m_1 \cdot p_1/t) \cdot k_1 \in R_1$. Notice that multiplying c_1 by $(p_2/p_1) \cdot k^{-1}$ produces $p_2q_1 + (p_2/p_1)r_1 + m_1 \cdot p_2/t$, that is, we remove the key sk_1 and replace it by p_2. Thus, we would like to generate a key swk containing $(p_2/p_1) \cdot k^{-1}$ so that we could multiply c_1 by swk to generate a new ciphertext in R_2 that does not depend on sk_1. However, multiplying k by k^{-1}, which is the inverse in R_1/p_1R_1, while performing all the operations on R_2 is ill defined. Hence, we use the coefficient vector of c_1, that is, $\boldsymbol{\phi}(c_1)$, because it is an integral vector, thus the products performed on R_2 involve one integer and one element of R_2, which is then well defined. Remember that if $c = c' \cdot k \in R_1$, then $\boldsymbol{\phi}(c) = \boldsymbol{\phi}(c') \cdot \boldsymbol{\Phi}(k) \in \mathbb{Z}^{N_1}$, therefore,

$$\boldsymbol{\phi}(c_1) \cdot \boldsymbol{\Phi}(k^{-1}) = \phi(p_1q_1 + r_1 + m_1 \cdot p_1/t) = p_1\phi(q_1) + \phi(r_1) + \phi(m_1) \cdot p_1/t$$

and multiplying also by p_2/p_1, we have

$$\boldsymbol{\phi}(c_1) \cdot (p_2/p_1) \cdot \boldsymbol{\Phi}(k^{-1}) = p_2\phi(q_1) + (p_2/p_1) \cdot \phi(r_1) + \phi(m_1) \cdot p_2/t \ \in \mathbb{Z}^{N_1}.$$

Thus, in the first part of our functional key-switching, we use both private keys, sk_1 and sk_2, to generate a functional key-switching key swk that can be viewed as an encryption of $(p_2/p_1) \cdot \boldsymbol{\Phi}(k_1^{-1}) \cdot \mathbf{u}$ under the key sk_2. Then, swk can be used to homomorphically multiply a ciphertext $\boldsymbol{\phi}(c_1)$ by $(p_2/p_1) \cdot \boldsymbol{\Phi}(k^{-1})$ and by $\mathbf{u} \in R_2^{N_1}$, removing thus sk_1 and generating an encryption of $\boldsymbol{\phi}(m) \cdot \mathbf{u}$. Because swk is encrypted under sk_2, so is $\boldsymbol{\phi}(m) \cdot \mathbf{u}$. Therefore, we have a procedure to generate a functional key-switching key swk, and another one to apply swk and transform any given ciphertext. We define these two procedures as follows:

- $\mathsf{FuncKeySwtGen}(\mathsf{sk}_1, \mathsf{sk}_2, \mathsf{params}, \mathbf{u})$: given $\mathsf{params} = (N_1, N_2, \tilde{b}, \tilde{\ell}, \gamma_2, \rho_2)$, secret keys $\mathsf{sk}_i = (p_i, k_i) \in \mathbb{Z} \times R_i$, and a vector $\mathbf{u} \in R_2^{N_1}$, proceed as follows:

  1. Define $\mathbf{g}_{\tilde{b}} := (\tilde{b}^0, ..., \tilde{b}^{\tilde{\ell}-1}) \in \mathbb{Z}^{\tilde{\ell} \times 1}$ and $\mathbf{G} := \mathbf{I}_{N_1} \otimes \mathbf{g}_{\tilde{b}} \in \mathbb{Z}^{N_1\tilde{\ell} \times N_1}$.
  2. Let $\mathbf{v} := \left\lfloor \frac{p_2}{p_1} \mathbf{G}\boldsymbol{\Phi}(k_1^{-1})\mathbf{u} \right\rceil \in R_2^{N_1\tilde{\ell}}$, where p_2/p_1 must be interpreted as a fraction in $\mathbb{Q}$ and the inverse of k_1 is computed on R_1/p_1R_1.
  3. Sample M from $p_2 \cdot \mathcal{U}([0, 2^{\gamma_2}/p_2])$.
  4. Sample $\mathbf{y}$ from $(\mathcal{P}_{N_2,\gamma_2,\rho_2}(p_2))^{N_1\tilde{\ell}}$.

5. Output $\mathsf{swk} := [(\mathbf{y}+\mathbf{v})\cdot k_2]_M$. Notice that the output is of the form

$$\left(p_2\mathbf{q}+\mathbf{r}+\left\lfloor\frac{p_2}{p_1}\mathbf{G}\boldsymbol{\Phi}(k_1^{-1})\mathbf{u}\right\rceil\right)\cdot k_2 \in R_2^{N_1\tilde{\ell}}.$$

- $\mathsf{FuncKeySwt}(c_1, \mathsf{swk})$: Given a scalar ciphertext $c_1 \in R_1$ and a functional key-switching key $\mathsf{swk} \in R_2^{N_1\tilde{\ell}}$, define $\mathbf{z} := \boldsymbol{\phi}(c_1) \in \mathbb{Z}^{N_1}$, decompose each entry of $\mathbf{z}$ in base $\tilde{b}$ as $\mathbf{w} := (g^{-1}(z_1), ..., g^{-1}(z_{N_1})) \in \mathbb{Z}^{N_1\tilde{\ell}}$, and output $c_2 := \mathbf{w}\cdot\mathsf{swk} \in R_2$.

Lemma 7.3.1 (Length of output of functional key switching)**.** *Define* $\mathbf{u} \in R_2^{N_1}$, $\mathsf{sk}_1 := (p_1, k_1) \in \mathbb{Z}\times R_1$, $\mathsf{sk}_2 := (p_2, k_2) \in \mathbb{Z}\times R_2$, *and* $\mathsf{params} := (N_1, N_2, \tilde{b}, \tilde{\ell}, \gamma_2, \rho_2)$. *Let also* $\mathsf{swk} := \mathsf{FuncKeySwtGen}(\mathsf{sk}_1, \mathsf{sk}_2, \mathsf{params}, \mathbf{u})$. *Then, for any* $c_1 \in R_1$, *it holds that* $c_2 := \mathsf{FuncKeySwt}(c_1, \mathsf{swk})$ *is a polynomial in* R_2 *whose coefficients have bit length less than* $\gamma_2 + \log(\tilde{\ell}\cdot(\tilde{b}-1)\cdot N_1)$, *in other words,* $\|c_2\| < \tilde{\ell}N_1(\tilde{b}-1)2^{\gamma_2}$.

Proof. In the procedure $\mathsf{FuncKeySwtGen}$, we sample a γ_2-bit integer M and swk is defined modulo M, hence, we know that $\mathsf{swk} = (s_1, ..., s_{N_1\tilde{\ell}})$ for some s_i's such that $\|s_i\| < 2^{\gamma_2}$. It is also clear that $s_i \in R_2$. The vector $\mathbf{w} \in \mathbb{Z}^{N_1\tilde{\ell}}$ computed in $\mathsf{FuncKeySwt}(c_1, \mathsf{swk})$ is a decomposition of the coefficients of c_1 in base $\tilde{b}$, thus, $\mathbf{w} = (w_1, ..., w_{N_1\tilde{\ell}})$ with $|w_i| \le \tilde{b}-1$.

The output c_2 is equal to $\mathbf{w}\cdot\mathsf{swk}$, therefore,

$$\|c_2\| = \left\|\sum_{i=1}^{N_1\tilde{\ell}} w_i s_i\right\| \le \sum_{i=1}^{N_1\tilde{\ell}} \|w_i s_i\| \le \sum_{i=1}^{N_1\tilde{\ell}} |w_i|\,\|s_i\| < N_1\cdot\tilde{\ell}\cdot(\tilde{b}-1)\cdot 2^{\gamma_2}.$$

□

Lemma 7.3.2 (Correctness of functional key switching)**.** *Let* $\mathbf{u} \in R_2^{N_1}$, $\mathsf{sk}_i := (p_i, k_i) \in \mathbb{Z}\times R_i$, $\mathsf{params} := (N_1, N_2, \tilde{b}, \tilde{\ell}, \gamma_2, \rho_2)$, *and* $\mathsf{swk} := \mathsf{FuncKeySwtGen}(\mathsf{sk}_1, \mathsf{sk}_2, \mathsf{params}, \mathbf{u})$. *Then, for any* $c_1 \in R_1$ *encrypting* $m \in R_1/tR_1$ *under key* sk_1, *it holds that* $c_2 := \mathsf{FuncKeySwt}(c_1, \mathsf{swk})$ *is a valid encryption of* $\boldsymbol{\phi}(m)\cdot\mathbf{u} \in R_2$ *under key* sk_2, *if* $\|c_1\| < \tilde{b}^{\tilde{\ell}}$. *Moreover, the noise term of* c_2 *is bounded as follows:*

$$\|\mathsf{err}(c_2)\| \le \tilde{\ell}N_1\tilde{b}2^{\rho_2} + 2^{\eta_2-\eta_1+2}N_1\,\|\mathbf{u}\|\,\|\mathsf{err}(c_1)\|$$

where η_i *is the bit length of* p_i.

Proof. Let $c_1 = (p_1 q_1 + r_1 + \alpha_1 m)k_1 \in R_1$, where $\alpha_1 := \lfloor p_1/t \rceil$. Notice that $\mathbf{w}$ defined in $\mathsf{FuncKeySwt}$ satisfies $\mathbf{wG} = \boldsymbol{\phi}(c_1)$ because $\|c_1\| < \tilde{b}^{\tilde{\ell}}$. Moreover, $\boldsymbol{\phi}(c_1)\boldsymbol{\Phi}(k^{-1}) = p_1\mathbf{q}_1 + \boldsymbol{\phi}(r_1) + \alpha_1\boldsymbol{\phi}(m)$. Therefore, the output of $\mathsf{FuncKeySwt}$ is

$$
\begin{aligned}
c_2 &= \mathbf{w} \cdot \mathsf{swk} \\
&= \left(p_2\mathbf{wq} + \mathbf{wr} + \mathbf{w}\left\lfloor \frac{p_2}{p_1}\mathbf{G}\boldsymbol{\Phi}(k^{-1})\mathbf{u}\right\rceil\right) \cdot k_2 && \text{By definition of } \mathsf{swk} \\
&= (p_2\mathbf{wq} + \mathbf{wr} + \mathbf{w}\boldsymbol{\epsilon} + \frac{p_2}{p_1}\mathbf{wG}\boldsymbol{\Phi}(k^{-1})\mathbf{u}) \cdot k_2 && (\text{For some } \|\boldsymbol{\epsilon}\| \le 1/2) \\
&= (p_2\mathbf{wq} + \mathbf{w}(\mathbf{r}+\boldsymbol{\epsilon}) + \frac{p_2}{p_1}(p_1\mathbf{q}_1 + \boldsymbol{\phi}(r_1) + \alpha_1\boldsymbol{\phi}(m))\mathbf{u})k_2 \\
&= (p_2 q_2 + \mathbf{w}(\mathbf{r}+\boldsymbol{\epsilon}) + \frac{p_2}{p_1}(\boldsymbol{\phi}(r_1) + \alpha_1\boldsymbol{\phi}(m))\mathbf{u}) \cdot k_2 && (\text{For } q_2 := \mathbf{wq} + \mathbf{q}_1\mathbf{u}) \\
&= (p_2 q_2 + \mathbf{w}(\mathbf{r}+\boldsymbol{\epsilon}) + \frac{p_2}{p_1}(\boldsymbol{\phi}(r_1) + \epsilon\boldsymbol{\phi}(m))\mathbf{u} + \frac{p_2}{t}\boldsymbol{\phi}(m)\mathbf{u}) \cdot k_2 && (\text{For some } \epsilon \in R_2)
\end{aligned}
$$

Therefore, c_2 is indeed an encryption of $\boldsymbol{\phi}(m)\mathbf{u}$ with respect to the key sk_2, that is, $c_2 = (pq_2 + r_2 + p_2\boldsymbol{\phi}(m)\mathbf{u}/t)k_2 \in R_2$ with $\mathsf{err}(c_2) = r_2 = \mathbf{w}(\mathbf{r}+\boldsymbol{\epsilon}) + \frac{p_2}{p_1}(\boldsymbol{\phi}(r_1) + \epsilon\boldsymbol{\phi}(m))\mathbf{u}$. Furthermore,

$$
\begin{aligned}
\|\mathsf{err}(c_2)\| &\le \|\mathbf{wr}\| + \|\mathbf{w}\epsilon\| + \left\|\frac{p_2}{p_1}(\boldsymbol{\phi}(r_1) + \epsilon\boldsymbol{\phi}(m))\mathbf{u}\right\| \\
&\le \tilde{\ell}N_1\tilde{b}\,\|\mathbf{r}\| + \tilde{\ell}N_1\tilde{b}/2 + 2^{\eta_2-\eta_1+1}N_1\,\|\mathbf{u}\|\,(\|\mathsf{err}(c_1)\| + t/2) \\
&\le \tilde{\ell}N_1\tilde{b}2^{\rho_2} + 2^{\eta_2-\eta_1+1}N_1\,\|\mathbf{u}\|\,(\|\mathsf{err}(c_1)\| + t/2) \\
&\le \tilde{\ell}N_1\tilde{b}2^{\rho_2} + 2^{\eta_2-\eta_1+2}N_1\,\|\mathbf{u}\|\,\|\mathsf{err}(c_1)\|\,.
\end{aligned}
$$

□

It turns out that this procedure is very general. For example, if we set $\mathbf{u} = (1, x, ..., x^{N_1})$, then, $\boldsymbol{\phi}(m)\mathbf{u} = \sum_{i=0}^{N_1} m_i x^i = m$, therefore, by using such $\mathbf{u}$, our functional key-switching works as an ordinary key-switching, outputting an encryption of the same message m but in the ring R_2 and under the key sk_2. By setting $\mathbf{u} = (1, z, ..., z^{N_1})$ for any $z \in \mathbb{Z}$, we obtain an encryption of $\boldsymbol{\phi}(m)\mathbf{u} = m(z)$, i.e., the evaluation of m at the point z. Also notice that when $N_i = 1$, we have $R_i \simeq \mathbb{Z}$, thus, not only our procedure is well defined for $N_i = 1$, but it also switches the underlying problem from the AGCD to the RAGCD problem or vice versa. In Table 7.1, all the possible ways of using our functional key

Table 7.1: Possible usages of the functional key-switching procedure.

N_1	N_2	Underlying problems	Encrypted message
> 1	> 1	RAGCD $\longrightarrow$ RAGCD	$\sum_{i=0}^{N_1-1} m_i x^i \mapsto \sum_{i=0}^{N_1-1} m_i \cdot u_i$ with $u_i \in R_2$
> 1	$= 1$	RAGCD $\longrightarrow$ AGCD	$\sum_{i=0}^{N_1-1} m_i x^i \mapsto \sum_{i=0}^{N_1-1} m_i \cdot u_i$ with $u_i \in \mathbb{Z}$
$= 1$	> 1	AGCD $\longrightarrow$ RAGCD	$m \in \mathbb{Z} \mapsto m \cdot u$ with $u \in R_2$
$= 1$	$= 1$	AGCD $\longrightarrow$ AGCD	$m \in \mathbb{Z} \mapsto m \cdot u$ with $u \in \mathbb{Z}$

switching are shown. The third column shows the underlying problems used to encrypt the input and the output message depending on whether each N_i is bigger than one or not. For instance, if $N_1 = 1$ and $N_2 > 1$, then the vector $\mathbf{u} \in R_2^{N_1}$ collapses to a scalar, that is, a polynomial of R_2, thus we are taking a message $m \in \mathbb{Z}$ encrypted using the AGCD problem and we are producing a ciphertext that encrypts the polynomial $m \cdot u$ using the RAGCD problem.

As final observation, note that in Lemma 7.3.2 the noise term of the input ciphertext is multiplied by $2^{\eta_2 - \eta_1}$, thus, the noise of the ciphertext output by the functional key switching procedure can even be smaller the the noise of the input ciphertext if p_1 is sufficiently larger than p_2.

7.4 Single-gate bootstrapping for FHE over the integers

We start this section by showing the base scheme, which is FHE scheme based on the AGCD problem that will be bootstrapped, then we show how to generate the bootstrapping keys, and finally how the GAHE scheme is used to refresh a ciphertext.

7.4.1 Base scheme

In this section, a simple AGCD-based scheme that will be used as the base scheme, that is, as the scheme that will be bootstrapped, is presented. As it is done in [DM15, CGGI16a], this base scheme is a leveled scheme with two levels only, thus, fresh ciphertexts are at level-1, we can evaluate one homomorphic binary gate by performing some simple additions, obtaining a ciphertext at level-2, then we have to refresh the ciphertext to

reduce the noise and to go back to level 1. Since all the binary gates can be written as compositions of NAND gates, for simplicity, we just present this homomorphic gate (but it is trivial to change the scheme to make possible to perform other binary gates directly). Because both the base scheme and the GAHE scheme are based on the (R)AGCD problem, they have similar parameters (e.g. ρ is the bit length of the noise terms in both schemes), thus, to avoid confusion, we represent the parameters of the base scheme with an overscore. For instance, the secret key of the base scheme is a prime $\bar{p}$ of bit length $\bar{\eta}$, while GAHE's secret key has an η-bit prime p.

- $\mathsf{HE.ParamGen}(\lambda)$: Choose $\bar{\rho} = \lambda$, $\bar{\eta} = \bar{\rho} + \beta$ for some small constant β, and $\bar{\gamma} = \max(2\bar{\eta}, \lceil \beta^2 \lambda / \log(\lambda) \rceil)$. Output $\mathsf{params} := (\bar{\gamma}, \bar{\eta}, \bar{\rho}, \lambda)$.

- $\mathsf{HE.KeyGen}(\mathsf{params})$: Sample a random prime $\bar{p}$ from $[\![2^{\bar{\eta}-1}, 2^{\bar{\eta}}]\!]$ and $\bar{p}q_{\mathsf{ek}} + r_{\mathsf{ek}} \leftarrow \mathcal{D}_{\bar{\gamma},\bar{\rho}}(\bar{p})$. Define the evaluation key as $\bar{\mathsf{ek}} := \bar{p}q_{\mathsf{ek}} + r_{\mathsf{ek}} + \lfloor 5\bar{p}/8 \rceil$ and the secret key as $\bar{\mathsf{sk}} := \bar{p}$.

- $\mathsf{HE.Enc}(\bar{\mathsf{sk}}, m, L)$: To encrypt a bit m, sample $\bar{p}q + r \leftarrow \mathcal{D}_{\bar{\gamma},\bar{\rho}}(\bar{p})$ and output the level-L ciphertext $c = \bar{p}q + r + \lfloor L\bar{p}/4 \rceil\, m$.

- $\mathsf{HE.Dec}(\bar{\mathsf{sk}}, c)$: To decrypt a level-1 c, compute $c' := [c]_{\bar{p}}$, then output $\left[\left\lfloor \frac{4c'}{\bar{p}} \right\rceil\right]_2$.

- $\mathsf{HE.Nand}(c_1, c_2, \bar{\mathsf{ek}})$: Let c_0 and c_1 be level-1 ciphertexts encrypting m_1 and m_2, respectively. Output $c := \bar{\mathsf{ek}} - c_1 - c_2$.

The function $\mathsf{HE.ParamGen}$ chooses the parameters in a way that guarantees the correctness of $\mathsf{HE.Dec}$. Namely, because $|r| < 2^{\bar{\rho}}$, we have $|r + \lfloor \bar{p}/4 \rceil\, m| < \bar{p}/2$, therefore, $c' = [c]_{\bar{p}} = r + \lfloor \bar{p}/4 \rceil\, m$ in $\mathbb{Z}$. And since $\lfloor \bar{p}/4 \rceil > 2^{\bar{\rho}+1} > 2|r|$, we have $\lfloor 4r/\bar{p} \rceil = 0$, then, the output is $\lfloor 4c'/\bar{p} \rceil = \lfloor 4r/\bar{p} \rceil + m = m$.

Our NAND gate is the same of [DM15, CGGI16a] and one can see that it is correct as follows: Notice that because $m_1, m_2 \in \{0, 1\}$, it holds that $(m_1 - m_2)^2 = m_1^2 - 2m_1m_2 + m_2^2 = m_1 - 2m_1m_2 + m_2$. Moreover, $(m_1 - m_2)^2 = \pm 1$, thus, we obtain

$$-m_1 - m_2 = \pm 1 - 2m_1m_2.$$

Now, notice that for the output of the homomorphic NAND gate, the following holds

$$\begin{aligned}
c &= \bar{p}q + r_{\mathsf{ek}} - r_1 - r_2 + \frac{5\bar{p}}{8} - \frac{\bar{p}}{4}(m_1 + m_2) && \text{(For } q := q_{\mathsf{ek}} - q_1 - q_2\text{)} \\
&= \bar{p}q + r' + \frac{\bar{p}}{2}\left(\frac{5}{4} - \frac{m_1}{2} - \frac{m_2}{2}\right) && \text{(For } r' := r_{\mathsf{ek}} - r_1 - r_2\text{)} \\
&= \bar{p}q + r' + \frac{\bar{p}}{2}\left(\frac{5}{4} \pm \frac{1}{2} - m_1 m_2\right) && \text{(Since } -m_1 - m_2 = \pm 1 - 2m_1 m_2\text{)} \\
&= \bar{p}q + r' + \frac{\bar{p}}{2}\left(\frac{1}{4} \pm \frac{1}{2} + 1 - m_1 m_2\right) && \text{(Using } 5/4 = 1/4 + 1\text{)} \\
&= \bar{p}q + r' \pm \frac{\bar{p}}{8} + \frac{\bar{p}}{2}(1 - m_1 m_2) && \text{(Because } 1/4 \pm 1/2 = \pm 1/4\text{)}
\end{aligned}$$

Therefore, our homomorphic NAND gate produces a ciphertext of the form

$$c = \bar{p}\underbrace{(q_{\mathsf{ek}} - q_1 - q_2)}_{q_{nand}} + \underbrace{r_{\mathsf{ek}} - r_1 - r_2 \pm \frac{\bar{p}}{8}}_{r_{nand}} + \lfloor \bar{p}/2 \rceil\,(1 - m_1 m_2)$$

which is a level-2 encryption of $NAND(m_1, m_2)$ with noise $|r_{nand}| < 3 \cdot 2^{\bar{\rho}} + \bar{p}/8$.

By standard techniques [DGHV10, CS15, BBL17] one can prove that this base scheme is CPA-secure if the AGCD problem is computationally hard. Moreover, the parameters chosen in $\mathsf{HE.ParamGen}$ provide a security level of λ bits.

7.4.2 Generating the bootstrapping keys

To generate the key material used to bootstrap, we need to fix a base $B \geq 2$ in which we decompose the ciphertexts of the base scheme when they are refreshed. Then, we define $L := \lceil \log_B(2^{\bar{\gamma}}) \rceil$, which is the number of words needed to decompose the given ciphertexts. Moreover, the number of homomorphic mixed products that we perform during the refresh procedure is $\Theta(L)$, thus, there is a time-memory trade-off, as the amount of memory increases in general when we increase B, but at the same time, L decreases.

The bootstrapping procedure consists in two main steps: in the first one, we use the GAHE scheme to homomorphically multiply a given ciphertext $\bar{c}$ by $N/\bar{p}$, obtaining a GAHE's scalar ciphertext c; in the second step, we transform c in a valid base scheme

ciphertext c'. To perform the first step, we would like to encrypt values of the form $y^{gB^iN/\bar{p}}$, where $y := x^2$, but the exponent would not be an integer, thus, we encrypt $y^{\lfloor gB^iN/\bar{p} \rceil}$, that is, we define $\mathfrak{K}_{g,i} := \mathsf{GAHE.EncVec}\left(y^{\lfloor gB^iN/\bar{p} \rceil}\right)$ for $1 \leq g < B$ and $0 \leq i < L$. In addition, we also encrypt an integer δ that is added to the result obtained in the first step, so that the final result is contained in the interval $[\![0, N-1]\!]$. Thus, we define $\mathfrak{K}_\delta := \mathsf{GAHE.EncScalar}(y^\delta) \in R$.

Notice that $\mathfrak{K}_\delta$ is a scalar ciphertext, while $\mathfrak{K}_{g,i}$ are vector ciphertexts. During the refresh procedure, we use the mixed homomorphic product to multiply them. Hence, at the end of the first step of the refreshing algorithm, we have a scalar ciphertext $c = (pq + r + \alpha y^e)k$ for some $e \in [\![0, N-1]\!]$ whose value depends on the message m. Then, to extract m, we define a test vector $\mathbf{u} \in \{0,1\}^N$, such that $\boldsymbol{\phi}(y^e) \cdot \mathbf{u} = 1 - 2m$ and use our functional key-switching to transform c into a ciphertext that encrypts $\boldsymbol{\phi}(y^e) \cdot \mathbf{u}$ under the base scheme key $\bar{\mathsf{sk}}$. Thus, we also append the following key to the bootstrapping keys:

$$\mathsf{ek} := \mathsf{FuncKeySwtGen}(\mathsf{sk} := (p, k), \bar{\mathsf{sk}} := (\bar{p}, 1), \mathbf{u}).$$

In Algorithm 2, the procedure to generate the bootstrapping key bk is presented in detail.

7.4.3 Refreshing a ciphertext

The goal of the bootstrapping is to take a level-2 ciphertext $c = \bar{p}q + r + \lfloor \bar{p}/2 \rceil\, m \in \mathbb{Z}$ whose noise term satisfies $|r| < 3 \cdot 2^{\bar{\rho}} + \bar{p}/8$ and to output a level-1 ciphertext $c' = \bar{p}q' + r' + \lfloor \bar{p}/4 \rceil\, m$ with $|r'| < 2^{\bar{\rho}}$. The refreshing procedure is shown thoroughly in Algorithm 3 and it consists in two main steps: in the first one, we decompose c in the base B obtaining $(c_0, c_1, ..., c_{L-1})$, then we use the bootstrapping key and GAHE's mixed homomorphic multiplication to obtain a scalar encryption of y^e, where $y := x^2$ and

$$y^e = y^\delta \cdot \prod_{i=0}^{\ell-1} y^{\lfloor c_i b^i N/\bar{p} \rceil} = y^{\delta + cN/\bar{p} + \epsilon \bmod N} = y^{\delta + rN/\bar{p} + mN/2 + \epsilon}$$

for some small value ϵ.

Then, notice that $\boldsymbol{\phi}(y^e) = \boldsymbol{\phi}(x^{2e})$, thus, if $0 \leq e < N/2$, then, the only non-zero entry

Algorithm 2: GenBootstrapKeys

Input: Decomposition base B, secret key $\bar{p}$ of the base scheme
Output: Bootstrapping key bk

$y \leftarrow x^2$
$L \leftarrow \lceil \bar{\gamma} \cdot \log_B(2) \rceil$
for $g = 1$ until $B-1$ **do**
  for $i = 0$ until $L-1$ **do**
    $\mathfrak{K}_{g,i} \leftarrow \mathsf{GAHE.EncVec}\left(y^{\lfloor gB^i N/\bar{p} \rceil}\right)$
    $\epsilon_{g,i} \leftarrow gB^i N/\bar{p} - \lfloor gB^i N/\bar{p} \rceil$
Choose $\Delta \in \mathbb{N}$ such that $|\sum_{i=0}^{L-1} \epsilon_{g_i,i}| < \Delta$ for any $(g_0, ..., g_{L-1})$, e.g., $\Delta = L/2$
$\delta \leftarrow \lceil \Delta + (3 \cdot 2^{\bar{\rho}} + \bar{p}/8)N/\bar{p} \rceil$
$\mathfrak{K}_\delta \leftarrow \mathsf{GAHE.EncScalar}(y^\delta)$ using $\alpha := \lfloor p/8 \rceil$
$\gamma_{\mathsf{ek}} \leftarrow \lfloor \bar{\gamma} - \log(\ell N b) \rfloor$ and $\rho_{\mathsf{ek}} \leftarrow \lfloor \bar{\rho} - \log(\ell N b) - 2 \rfloor$
$\mathsf{params} \leftarrow (N, 1, b, \ell, \gamma_{\mathsf{ek}}, \rho_{\mathsf{ek}})$
$\mathbf{u} \leftarrow (1, 1, ..., 1) \in \mathbb{Z}^N$
$\mathsf{ek} \leftarrow \mathsf{FuncKeySwtGen}(\mathsf{sk}_1 := (\bar{p}, 1), \mathsf{sk}_2 := (p, k), \mathsf{params}, \mathbf{u}) \in \mathbb{Z}^{N\ell}$
$\mathfrak{K}_8 \leftarrow \mathcal{D}_{\bar{\gamma}, \bar{\rho}-1}(\bar{p}) + \lfloor \bar{p}/8 \rceil$
return $\mathsf{bk} := \left(\mathsf{ek}, \mathfrak{K}_\delta, \mathfrak{K}_8, \{\mathfrak{K}_{g,i}\}_{\substack{1 \le g < B \\ 0 \le i < L}}\right)$

of $\boldsymbol{\phi}(y^e)$ is 1, otherwise, there is a $s \in [\![0, N-1]\!]$ such that $x^{2e} = x^{N+s} = -x^s$ in R and $\boldsymbol{\phi}(x^{2e}) = \boldsymbol{\phi}(-x^s)$ has a single nonzero entry, which is -1. But as we show in Lemma 7.4.1, we have $0 \le e < N/2$ if $m = 0$ and $N/2 \le e \le N-1$ if $m = 1$, therefore, the vector $\mathbf{u} := (1, ..., 1) \in \mathbb{Z}^N$ satisfies $\boldsymbol{\phi}(y^e) \cdot \mathbf{u} = 1 - 2m$. Thus, because we used $\mathbf{u}$ to generate ek, when we apply the functional key-switching in the second step of Algorithm 3, we switch from GAHE to the base scheme and we obtain an encryption of $\boldsymbol{\phi}(x^e) \cdot \mathbf{u} = 1 - 2m$, that is, we obtain $\tilde{c} = \bar{p}\tilde{q} + \tilde{r} + (1-2m) \cdot \lfloor p/8 \rceil$. Therefore, subtracting $\tilde{c}$ from $\mathfrak{K}_8$ yields a valid level-one base scheme encryption of m, that is, $c' := \mathfrak{K}_8 - \tilde{c} = \bar{p}q' + r' + m \lfloor p/4 \rceil$.

In which follows, we prove the correctness of the refreshing procedure.

Lemma 7.4.1. *Let c be a level-2 encryption of $m \in \{0, 1\}$ with noise bounded by $3 \cdot 2^{\bar{\rho}} + \bar{p}/8$. Let $N \ge 4\delta$ where δ is the integer defined in Algorithm 2. Then, the ciphertext $z \in R$ obtained at the end of the main loop of Algorithm 3 is an encryption of x^{2e} where $e \in [\![0, N-1]\!]$. Moreover, $m = 0 \iff 0 \le e < N/2$ and $m = 1 \iff N/2 \le e < N$.*

Algorithm 3: REFRESH

Input: Level-2 ciphertext c of the base scheme, bootstrapping key bk
Output: Level-1 ciphertext c of the base scheme

Let $(c_0, c_1, ..., c_{L-1})$ be a decomposition of c in base B.
$z \leftarrow \mathfrak{K}_\delta$
for $i = 0$ until $L-1$ **do**
  if $c_i > 0$ **then**
    $z \leftarrow \mathsf{GAHE.MultMix}(z, \mathfrak{K}_{c_i,i})$
▷ `Second step: extract the message.`
$\tilde{c} \leftarrow \mathsf{FuncKeySwt}(z, \mathsf{ek})$
$c' \leftarrow \mathfrak{K}_8 - \tilde{c}$
return c'

Proof. Let $y := x^2$. We initialize z with a scalar encryption of y^δ and at each iteration i, we add $\lfloor c_i b^i N/\bar{p} \rceil$ to the exponent, thus, because y has order N in R, it is clear that at the end of the loop z encrypts y^e where $e = \delta + \sum_{i=0}^{L-1} \lfloor c_i B^i N/\bar{p} \rceil \bmod N$.

Now, let $\epsilon_i := \lfloor c_i B^i N/\bar{p} \rceil - c_i B^i N/\bar{p}$ and $\epsilon := \sum_{i=0}^{L-1} \epsilon_i$. Then,

$$e = \delta + \epsilon + (N/\bar{p}) \cdot \sum_{i=0}^{L-1} c_i B^i = \delta + \epsilon + cN/\bar{p} = \delta + \epsilon + rN/\bar{p} + mN/2 \bmod N$$

Notice that $|\epsilon + rN/\bar{p}| < \Delta + (3 \cdot 2^{\bar{\rho}} + \bar{p}/8)N/\bar{p} \leq \delta$. Also, because $N \geq 4\delta$ by hypothesis, we have $2\delta \leq N/2$. Thus, we can see that

$$\delta + \epsilon + rN/\bar{p} + mN/2 < 2\delta + \frac{N}{2} \leq N.$$

Similarly, $\delta + \epsilon + rN/\bar{p} + mN/2 \geq \delta - \Delta - \frac{(3 \cdot 2^{\bar{\rho}} + \bar{p}/8)N}{\bar{p}} \geq \delta - \delta = 0$. Therefore, we conclude that $0 \leq e < N$. But because e is integer, we have $0 \leq e \leq N-1$, as expected.

Thereby, $e = \delta + \epsilon + rN/\bar{p} + mN/2$ over $\mathbb{Z}$, without the reduction modulo N. Thus,

$$\begin{cases} m = 0 \implies e = \delta + \epsilon + rN/\bar{p} < 2\delta \leq N/2 \\ m = 1 \implies e = \delta + \epsilon + rN/\bar{p} + N/2 \geq N/2 \end{cases}$$

□

Lemma 7.4.2. *Let $m \in \{0,1\}$ and $\delta \in \mathbb{N}^*$ as defined in Algorithm 2. Let $e \in [\![0, N-1]\!]$ such that $m = 0 \iff 0 \leq e < N/2$. Let z be a scalar encryption of x^{2e} with $\alpha = \lfloor p/8 \rceil$ and noise bounded by some value B_z. Then, when given z as input, the second step*

of Algorithm 3 outputs a base scheme level-1 encryption of m with noise bounded by $2^{\bar{\rho}-1} + NB_z 2^{\bar{\eta}-\eta+2}$.

Proof. We know that $z = (pq_z + r_z + \lfloor p/8 \rceil\, x^{2e})k$ and that ek is a functional switching key from $\mathsf{sk} = (p, k)$ to $\bar{\mathsf{sk}} = (\bar{p}, 1)$ with respect to $\mathbf{u} = (1, ..., 1)$. Then, by Lemma 7.3.2, the output of $\mathsf{FuncKeySwt}(z, \mathsf{ek})$ is

$$\tilde{c} = \bar{p}\tilde{q} + \tilde{r} + \left\lfloor \frac{\bar{p}}{8} \right\rceil \boldsymbol{\phi}(x^{2e})\mathbf{u} \in \mathbb{Z},$$

where $|\mathsf{err}(\tilde{c})| \leq 2^{\bar{\rho}-2} + NB_z 2^{\bar{\eta}-\eta+2}$ and $\boldsymbol{\phi}(x^{2e})\mathbf{u} = 1 - 2m$.

Notice that $\lfloor p/8 \rceil - (1-2m)\lfloor p/8 \rceil = 2m \lfloor p/8 \rceil = m \lfloor p/4 \rceil + \epsilon$, therefore, the output $c' := \mathfrak{K}_8 - \tilde{c}$ is indeed of the form $\bar{p}q' + r' + m\lfloor p/4 \rceil$, which a valid base scheme level-1 encryption of m. Moreover, $\mathsf{err}(c') \leq \mathsf{err}(\mathfrak{K}_8) + \mathsf{err}(\tilde{r}) \leq 2^{\bar{\rho}-1} + NB_z 2^{\bar{\eta}-\eta+2}$ □

Theorem 7.4.3 (Correctness of bootstrapping)**.** *Let c be a level-2 encryption of $m \in \{0, 1\}$ with noise bounded by $3 \cdot 2^{\bar{\rho}} + \bar{p}/8$. Let $N \geq 4\delta$ where δ is the integer defined in Algorithm 2. Let $\bar{\rho} \geq \rho + \bar{\eta} - \eta + \log(N^3 \ell b L) + 4$. Then, the refresh procedure, Algorithm 3, outputs a valid base scheme level-1 encryption of m with noise smaller than $2^{\bar{\rho}}$.*

Proof. By Lemma 7.4.1, we know that the ciphertext z produced at the end of the first step of Refresh is of the form $(pq_z + r_z + \lfloor p/8 \rceil\, x^{2e})k$ for some $e \in [\![0, N-1]\!]$ such that $m = 0 \iff 0 \leq e < N/2$. Therefore, by Lemma 7.4.2, the output of Refresh is a base scheme level-1 ciphertext $c' = \bar{p}q' + r' + \lfloor \bar{p}/4 \rceil\, m$ with $|r'| < 2^{\bar{\rho}-1} + N \cdot \|\mathsf{err}(z)\| \cdot 2^{\bar{\eta}-\eta+2}$. But because z is computed via a sequence of L mixed homomorphic products, by Lemma 6.3.7, we have $\|\mathsf{err}(z)\| < 2N^2 \ell b L 2^{\rho}$, therefore,

$$|r'| < 2^{\bar{\rho}-1} + 2^{\rho+\bar{\eta}-\eta+\log(N^3\ell b L)+3} \leq 2^{\bar{\rho}-1} + 2^{\bar{\rho}-1} = 2^{\bar{\rho}}.$$

□

7.4.4 Truncating ciphertexts to speed up refreshing

When we encrypt a message m with the base scheme, we multiply it by a constant α that is bigger than the noise. This has the effect of shifting the message so that it is encrypted

"between" the noise and the key. It has already been noticed [Bra12, CS15] that when we use noisy encryption schemes that encrypt messages in this way, we can discard the least significant bits of the ciphertexts, at the expend of increasing the noise, to use less memory to represent encrypted messages. But in our case, because the main loop of Algorithm 3 ignores bits equal to zero, we can also save some homomorphic multiplications and speed up the bootstrapping. Moreover, we do not need to generate bootstrapping keys $\mathfrak{K}_{g,i}$'s for these truncated bits.

In detail, given a base scheme ciphertext $c = \bar{p}q + r + \bar{\alpha}m$, we can set the first μ bits to zero, for example, by subtracting $s := c \bmod 2^\mu$ from c. Notice that we obtain then $c' := c - s = \bar{p}q + (r - s) + \bar{\alpha}m$, that is a valid encryption m, but with a new noise term potentially bigger, satisfying $\mathsf{err}(c') \leq \mathsf{err}(c) + 2^\mu$. Then, when we decompose c' in base B in the refreshing procedure, the first $\mu_B := \lfloor \mu \cdot \log_B(2) \rfloor$ words are ignored and the keys $\mathfrak{K}_{g,i}$ for $0 \leq i < \mu_B$ are never used. Thus, we reduce the number of mixed homomorphic products from L to $L - \mu_B$ and the number of keys from $(B-1)L$ to $(B-1)(L-\mu_B)$.

7.5 Multi-value bootstrapping for FHE over the integers

In this section, we extend and adapt our single-gate bootstrapping to solve the more general problem of bootstrapping a scheme whose message space is $\mathbb{Z}_t$ instead of $\{0,1\}$ and that can perform more than one homomorphic operation before refreshing. Moreover, we show how to change the test vector so that instead of outputting a refreshed ciphertext encrypting the same message m, the bootstrapping outputs encryptions of $f_1(m), ..., f_n(m)$, for arbitrary functions $f_i : \mathbb{Z}_t \to \mathbb{Z}_t$. As before, the parameters of the base scheme are denoted with an overscore. In particular, we have now two polynomial rings: $R := \mathbb{Z}[x]\langle x^N + 1\rangle$, used by GAHE, and $\bar{R} := \mathbb{Z}[x]\langle x^{\bar{N}} + 1\rangle$, used by the base scheme. We assume that ciphertexts of the base scheme are of the form $(\bar{p}q + r + \bar{\alpha}m)\bar{k} \in \bar{R}$, for $\bar{\alpha} := \lfloor \bar{p}/(2t) \rceil$ and a degree-$\bar{N}$ polynomial $\bar{k}$. Some existing AGCD-based FHE schemes, as [CS15], encrypt a message m in a ciphertext of the form $\bar{p}q + r + \bar{\alpha}m \in \mathbb{Z}$, thus, they can be viewed as a particular case when $\bar{N} = 1$. Alternatively, one can adapt [CS15] to

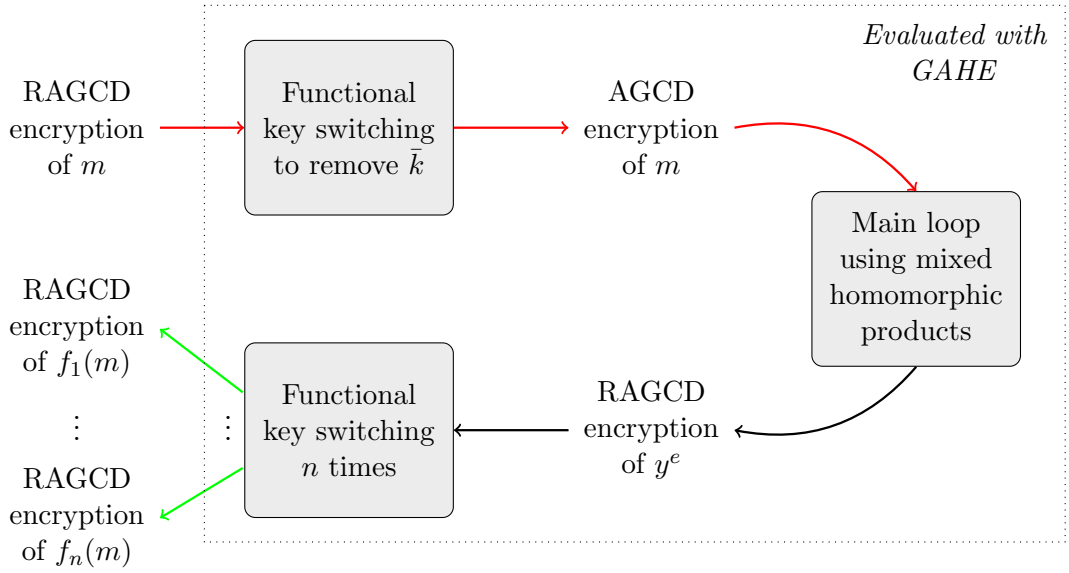

Figure 7.2: Our multi-bit bootstrapping. The input is an encryption of m under the RAGCD problem with large noise and the output is a set of n ciphertexts encrypting $f_1(m)$, ..., $f_n(m)$ with small noise.

use the polynomial ring $\bar{R}$ instead of $\mathbb{Z}$ and to randomize the ciphertexts so that they have the format we are proposing. Other schemes, like [DGHV10], use $c = pq + tr + m$ as the ciphertext format, but it is sufficient to apply a simple transformation on these ciphertexts to obtain our expected format, i.e., just homomorphically multiply c by the integer $(p - [p]_t)/t$. This simple transformation is discussed in Section 7.7.

Essentially, our multi-value bootstrapping consists in applying a functional key switching to the base scheme ciphertext to remove the polynomial $\bar{k}$ and obtain an encryption of the same message m, but under the AGCD problem. Then, we proceed as in the single-bit case, generating a GAHE encryption of y^e, for some e whose value depend on m. Finally, we apply n times the functional key switching, using extraction keys ek_i's with respect to functions f_i's of our choice, to obtain encryptions of $f_1(m)$, ..., $f_n(m)$. This procedure is illustrated in Figure 7.2.

7.5.1 Generating the multi-value bootstrapping keys

To perform the refreshing procedure we have to multiply the ciphertexts by $\bar{k}^{-1}$ and perform the reduction modulo $\bar{p}$. As in the single-gate case, instead of performing the bootstrapping on $\mathbb{Z}_{\bar{p}}$, we work on $\mathbb{Z}_N$. And rather than multiplying by $\bar{k}^{-1}$, we multiply by the first column of the circulant matrix of $\bar{k}^{-1}$, i.e., if we define $\mathbf{k}_1 := \mathsf{column}_1(\mathbf{\Phi}(\bar{k}^{-1}))$, then, $\mathbf{\Phi}(\bar{k})\mathbf{k}_1 = \mathbf{e}_1 := (1, 0, , ..., 0) \in \{1, 0\}^{\bar{N}}$, hence, given $c = (\bar{p}q + r + \bar{\alpha}m)\bar{k} \in \bar{R}$, it holds that

$$\boldsymbol{\phi}(c)\mathbf{k}_1 = \boldsymbol{\phi}(\bar{p}q + r + \bar{\alpha}m)\mathbf{\Phi}(\bar{k})\mathbf{k}_1 = \boldsymbol{\phi}(\bar{p}q + r + \bar{\alpha}m)\mathbf{e}_1 = \bar{p}q' + r_0 + \bar{\alpha}m \in \mathbb{Z}.$$

Therefore, one way of extending our single-gate bootstrapping is by multiplying the exponents encrypted in the bootstrapping keys by the entries of $\mathbf{k}_1$, that is, by defining $\mathfrak{K}_{g,i,j} := \mathsf{GAHE.EncVec}\left(x^{\lfloor gB^i k_{1,j} N/\bar{p} \rceil}\right)$ for $1 \leq g < B$, $1 \leq j \leq \bar{N}$ and $0 \leq i < L$. We would also need to add one extra loop in the refreshing procedure, because now each entry j of $\mathbf{a} := \boldsymbol{\phi}(c)$ would be decomposed in base B, and we would compute $\mathfrak{K}_\delta \prod_{j=1}^{\bar{N}} \prod_{i=0}^{L-1} \mathfrak{K}_{a_{i,j},i,j}$.

However the number of bootstrapping keys would be $\bar{N}(B-1)L$ in lieu of $(B-1)L$, which could result in huge memory requirements, and the number of homomorphic mixed multiplications in the first step would increase from $\Theta(L)$ to $\Theta(\bar{N}L)$, which could result in a considerable slowdown.

Thus, rather than inserting $\mathbf{k}_1$ into the keys $\mathfrak{K}_{g,i}$, we define a "remove $\bar{k}$ key", rk, which is a functional key-switching key from the RAGCD to the AGCD problem, that is, from the base scheme key $\bar{\mathsf{sk}} := (\bar{p}, \bar{k})$ to an intermediate key $\mathsf{sk}_{\mathsf{rk}} := (p_{\mathsf{rk}}, 1)$, with respect to the vector $\mathbf{e}_1 := (1, 0, ..., 0) \in \{0, 1\}^{\bar{N}}$. Because the base scheme ciphertext $c \in R$ encrypts an integer m, it is clear that $\boldsymbol{\phi}(m)\mathbf{e}_1 = m$, thus, applying rk to c produces an integer encryption of m of the form $c_{\mathsf{rk}} = p \cdot q_{\mathsf{rk}} + r_{\mathsf{rk}} + \alpha_{\mathsf{rk}} \cdot m \in \mathbb{Z}$, and we can then continue the bootstrapping as in the single-gate case, i.e., we decompose c_{rk} and execute a main loop using the GAHE's mixed homomorphic product to produce a scalar ciphertext encrypting x^{2e} where $e \approx c_{\mathsf{rk}} \cdot N/p_{\mathsf{rk}} \bmod N = r' + m \cdot N/(2t)$. As before, we can choose the parameters such that for each value of m, the exponent e is in a predetermined

interval. For instance, considering the parameter δ, when $m = 0$, we have $0 \leq e < 2\delta$, when $m = 1$, we have $N/(2t) \leq e < N/(2t) + 2\delta$, and so on. Therefore, the coefficient vector $\boldsymbol{\phi}(x^{2e}) \in \{1, 0\}^N$ has a single one which is some position between 0 and 4δ when $m = 0$, and it is in some position between $2 \cdot N/(2t)$ and $2 \cdot N/(2t) + 4\delta$ when $m = 1$, and so on. Thus, if we want the bootstrapping to output a ciphertext encrypting $f(m) \in \mathbb{Z}_t$ for some function f, we just have to use, in the second step of the refreshing function, a functional key-switching key with respect to a vector $\mathbf{u}$ that has the value $f(m)$ in the entries corresponding to interval defined by m.

That is to say, if the first 4δ entries of $\mathbf{u}$ are equal to $f(0)$, then, $m = 0$ implies $\boldsymbol{\phi}(x^{2e})\mathbf{u} = f(0)$ and the functional key-switching outputs the expected value. The same holds for other values of m, e.g., if the entries of $\mathbf{u}$ from $2 \cdot N/(2t)$ to $2 \cdot N/(2t) + 4\delta$ are equal to $f(1)$, then $m = 1$ implies $\boldsymbol{\phi}(x^{2e})\mathbf{u} = f(1)$. Therefore, we generate the bootstrapping keys as follows:

1. Choose $\eta_{\mathsf{rk}}, \rho_{\mathsf{rk}} = \Omega(\lambda)$ and $\gamma_{\mathsf{rk}} = \Omega(\lambda(\eta_{\mathsf{rk}} - \rho_{\mathsf{rk}})^2 / \log(\lambda))$. Sample a η_{rk}-bit prime p_{rk}. Let $\mathsf{params} := (\bar{N}, 1, 2, \bar{\gamma}, \gamma_{\mathsf{rk}}, \rho_{\mathsf{rk}})$, $\mathsf{sk}_{\mathsf{rk}} := (p_{\mathsf{rk}}, 1)$, and $\bar{\mathsf{sk}} := (\bar{p}, \bar{k})$. Then, define

$$\mathsf{rk} := \mathsf{FuncKeySwtGen}(\bar{\mathsf{sk}}, \mathsf{sk}_{\mathsf{rk}}, \mathsf{params}, \mathbf{e}_1).$$

2. Let $L := \lceil (\log(\bar{N}\bar{\gamma}) + \gamma_{\mathsf{rk}}) \log_B(2) \rceil$ instead of $L = \lceil \bar{\gamma} \log_B(2) \rceil$ and generate $\mathfrak{K}_{g,i}$'s as in Algorithm 2, but using p_{rk}, i.e., $\mathfrak{K}_{g,i} := \mathsf{GAHE.EncVec}\left(x^{2 \cdot \lfloor gB^i N/p_{\mathsf{rk}} \rceil}\right)$.

3. Let β be an upper bound to the noise of the ciphertexts that will be refreshed, define $\delta := \lceil \Delta + \beta N/p_{\mathsf{rk}} \rceil$, and $\mathfrak{K}_\delta := \mathsf{GAHE.DecScalar}(x^{2\delta})$, using $\alpha := \lfloor p/(2t) \rceil$.

4. Finally, instead of defining a single extraction key ek, we define one key for each chosen function. Namely, we assume that $N \geq 4t\delta$ and that the bootstrapping procedure also receives a list of functions $f_i : \mathbb{Z}_t \to \mathbb{Z}_t$, for $1 \leq i \leq n$. Then, we define the parameters γ_{ek} and ρ_{ek} as in Algorithm 2, and $\mathsf{params} := (N, \bar{N}, b, \ell, \gamma_{\mathsf{ek}}, \rho_{\mathsf{ek}})$. Finally,

$$\mathsf{ek}_i := \mathsf{FuncKeySwtGen}(\mathsf{sk}, \bar{\mathsf{sk}}, \mathsf{params}, \mathbf{u}_i)$$

where $\mathbf{u}_i \in \mathbb{Z}_t^N$ is a test vector defined as follows: for $0 \leq m \leq t - 1$, all the entries

$\mathbf{u}_i[2 \cdot \lfloor Nm/(2t) \rceil], ..., \mathbf{u}_i[4\delta + 2 \cdot \lfloor Nm/(2t) \rceil]$ are equal to $f_i(m)$. The remaining entries are zero. Expressly:

$$\mathbf{u}_i = (\underbrace{f_i(0), ..., f_i(0)}_{4\delta \text{ entries}}, 0, ..., 0, \underbrace{f_i(1), ..., f_i(1)}_{4\delta \text{ entries}}, 0, ..., 0, \underbrace{f_i(t-1), ..., f_i(t-1)}_{4\delta \text{ entries}}, 0, ..., 0)$$

Notice that if $Nm/(2t) < e < 2\delta + Nm/(2t)$, then $\boldsymbol{\phi}(x^{2e}) \cdot \mathbf{u}_i = f_i(m)$, thus, at the end of the bootstrapping, when we apply the functional key-switching on an encryption of $\boldsymbol{\phi}(x^{2e})$ using ek_i, we obtain an encryption of the desired value, i.e., $f_i(m)$. We also remark that, because the test vectors $\mathbf{u}_i$'s are encrypted, the functions f_i's are never disclosed, hence, we can dispatch the computation to a third party without revealing the functions that are being evaluated. The bootstrapping key is then defined as

$$\mathsf{bk} := \left(\mathsf{rk}, \{\mathsf{ek}_i\}_{1 \leq i \leq n}, \mathfrak{K}_\delta, \{\mathfrak{K}_{g,i}\}_{\substack{1 \leq g < B \\ 0 \leq i < L}}\right).$$

7.5.2 The multi-value refreshing procedure

Refreshing a ciphertext $\bar{c} = (\bar{p}\bar{q} + \bar{r} + \bar{\alpha}m)\bar{k} \in \bar{R}$ is similar to the single-gate case. Basically, we first transform $\bar{c}$ in an integer ciphertext c, then, we proceed as in Algorithm 3 and compute a scalar encryption of x^{2e}. Then, instead of extracting the message one time, we extract $f_i(m)$ for each f_i encoded in ek_i. We show it in detail in Algorithm 4.

By Lemma 7.3.1, we see that the bit length of the integer ciphertext c produced in the beginning of Algorithm 4 is bounded by $\log(\bar{N}\bar{\gamma}) + \gamma_{\mathsf{rk}}$. Thus, because we use $L = \lceil (\log(\bar{N}\bar{\gamma}) + \gamma_{\mathsf{rk}}) \log_B(2) \rceil$ in the bootstrapping key generation, we can decompose c in base B obtaining a vector $(c_0, ..., c_{L-1})$, then, we can proceed as in the single-gate case, using the mixed homomorphic product to multiply each bootstrapping key $\mathfrak{K}_{c_i,i}$ for $c_i \neq 0$, and obtaining a GAHE's scalar ciphertext $z = (pq_z + r_z + \alpha x^{2e})k \in R$, where

$$e = \delta + \epsilon + cN/p_{\mathsf{rk}} \bmod N = \delta + \epsilon + rN/p_{\mathsf{rk}} + mN/(2t).$$

Because we chose $\delta > |\epsilon + rN/p_{\mathsf{rk}}|$, we have, $mN/(2t) < e < 2\delta + mN/(2t)$. We want these intervals to be disjoint, thus, we need $2\delta + mN/(2t) \leq (m+1)N/(2t)$, which is satisfied if we choose $N \geq 4t\delta$. With this choice, we have then $\boldsymbol{\phi}(x^{2e})\mathbf{u}_i = f_i(m)$, which

Algorithm 4: MULTI-VALUE REFRESH

Input: Base scheme ciphertext $\bar{c} \in \bar{R}$, bootstrapping key bk
Output: Level-1 ciphertext c of the base scheme

▷ `Generate an intermediary encryption of` m `under the AGCD problem`
$c \leftarrow \mathsf{FuncKeySwt}(\bar{c}, \mathsf{rk}) \in \mathbb{Z}$
▷ `Now, the same main loop of Algorithm 3`
Let $(c_0, c_1, ..., c_{L-1})$ be a decomposition of c in base B.
$z \leftarrow \mathfrak{K}_\delta$
for $i = 0$ until $L - 1$ **do**
  if $c_i > 0$ **then**
    $z \leftarrow \mathsf{GAHE.MultMix}(z, \mathfrak{K}_{c_i,i})$
▷ `Then extract each` $f_i(m)$
for $i = 1$ until n **do**
  $c'_i \leftarrow \mathsf{FuncKeySwt}(z, \mathsf{ek}_i)$
return $c'_1, ..., c'_n$

means that the last step extracts the correct messages.

Similarly to Lemma 7.4.2, assuming that the noise of z is bounded by B_z, the final noise of each c'_i is $\|r'_i\| < 2^{\bar{\rho}-1} + B_z 2^{\log(Nt)+\bar{\eta}-\eta+1}$.

Then, as in Theorem 7.4.3, we have $B_z = 2N^2\ell bL2^\rho$, hence, if we choose $\bar{\rho} \geq \rho + \log(2N^3\ell bLt) + \bar{\eta} - \eta + 2$, we have

$$\left\|r'_i\right\| < 2^{\bar{\rho}-1} + 2^{\rho+\log(2N^3\ell bLt)+\bar{\eta}-\eta+1} \leq 2^{\bar{\rho}-1} + 2^{\bar{\rho}-1} = 2^{\bar{\rho}}$$

in other words, the final noise after bootstrapping is smaller than the initial noise.

7.6 Practical results

In this section we show how to choose the parameters of the GAHE scheme when it is used in our bootstrapping procedures, we also present the running times and memory usage of the single-gate and the multi-value bootstrapping, and we compare our results with previous works. Our proof-of-concept was implemented in C++ using the Number Theory Library[1] (NTL) and it is publicly available in [Per20e]. We ran the experiments

[1] https://www.shoup.net/ntl/

on a single core of a processor Intel Core i5-8600K 3.60GHz, of a machine with 32GB of RAM memory. We stress that our implementation is not optimized and the running times that we present below, although being enough to show that our techniques are practical, can certainly be improved.

7.6.1 Practical results of single-gate bootstrapping

In this section we present our practical results for the bootstrapping procedure described in Section 7.4. Our experiment consisted in encrypting two random bits m_1 and m_2 into c_1 and c_2, computing $c := \mathsf{HE.Nand}(c_1, c_2)$, then refreshing c. Because the homomorphic NAND gate is performed with three simple integer additions, its running time is negligible and we only measured the refreshing step. To show the time-memory trade-off explicitly, we used several values for the base B. For the base scheme, we fixed $\beta = 5$ and generated the other parameters as described in the function $\mathsf{HE.KeyGen}$ presented in Section 7.4.1. We also truncated $\mu = \bar{\rho} - 5$ bits of the ciphertexts, as explained in Section 7.4.4. We recall that when we generate bk, we must choose an upper-bound Δ for the sum of the rounding errors $\epsilon_{g,i}$'s. Because $|\epsilon_{g,i}| \leq 1/2$, it is clear that we can choose $\Delta = L/2$, however, this bound is not realistic, thus, we use $\Delta = (L - \lambda \log_B(2))/6$. Hence, we fixed $\lambda = \eta = 100$ and, for each B, we defined L and the parameters of the GAHE scheme as presented in Table 7.2.

Our running times and memory requirements are comparable with those of [DM15] and around 10 times larger than those of [CGGI16a]. A full comparison is presented in Table 7.2. We stress that this comparison is to be taken with care, because we used a 3.6 GHz processor while they used a 3.0 GHz one. On the other hand, while we used NTL's polynomial multiplication as a black box, they used very optimized Fast Fourier Transform (FFT) libraries to perform polynomial multiplications and they precomputed the FFTs of the bootstrapping keys, thus, in the main loop of the refreshing procedure, they just need to apply the FFT to one of the operands. If we had precomputed the FFTs, then, at each iteration of our bootstrapping we would need to compute only ℓ FFTs (on each entry of $g^{-1}(\mathfrak{K}_\delta) \in R^\ell$), thus, the total number of FFTs would decrease from around

Table 7.2: We show the practical results of our single-gate bootstrapping for several sets of parameters. The two last rows show [DM15] and [CGGI16a], which used only one fixed set of parameters. We show the running times they reported on a 3.0 GHz processor and also these timings multiplied by $\frac{3}{3.6}$ to make the comparison with our results more senseful. The security level is $\lambda = 100$ for the three schemes and we always used $\gamma = 200$ for our GAHE scheme.

	$\log B$	N	ρ	L	$\log b$	Size bk	Refreshing
Ours	5	256	51	76	26	115 MB	0.57 s
	7	128	65	54	14	283 MB	0.28 s
	9	128	65	42	14	889 MB	0.22 s
[DM15]	-					1.3 GB	0.69 s ($\times\frac{3}{3.6} = 0.57$ s)
[CGGI16a]	-					52 MB	0.05 s ($\times\frac{3}{3.6} = 0.04$ s)

Table 7.3: The first three rows show, with respect to B, the parameters used in GAHE and the practical results of our multi-value bootstrapping running on a 3.6 GHz processor. The last row shows the results reported in [CIM19] on a 3.5 GHz processor.

	$\log B$	L	N	ρ	$\log b$	ℓ	Size bk	Refreshing
Ours	9	27	2048	2	56	6	2.6 GB	7.74 s
	11	22	1024	2	57	6	4.4 GB	1.38 s
[CIM19]	-						$\approx$ 8 GB	1.57 s

$2L\ell$ to $L\ell$. Therefore, our running times can surely be improved.

7.6.2 Practical results of multi-value bootstrapping

In [CIM19] the authors propose some techniques to perform the fast bootstrapping procedures of [DM15] and [CGGI16a] on non-binary message spaces. Then, they implement a multi-value bootstrapping that takes an encryption of a 6-bit message m and outputs an encryption of $f(m)$ for an arbitrary function from $\{0,1\}^6$ to $\{0,1\}^6$. Thus, to compare our results with theirs, we use $t = 2^6$. In this experiment, we fix a random function f from $\mathbb{Z}_t$ to $\mathbb{Z}_t$ and we generate the bootstrapping keys. Then we sample a random message m from $\mathbb{Z}_t$, we encrypt it as $(\bar{p}q + r + \lfloor \bar{p}/(2t) \rceil\, m)\bar{k}$, and we executed the bootstrapping procedure obtaining a ciphertext encrypting $f(m)$. As in [CIM19], we can apply several additional functions to the same message in a single bootstrapping almost for free, that is,

outputting encryptions of $f_1(m), ..., f_n(m)$ for $n > 1$ costs almost the same as outputting one single ciphertext, since the main loop of the bootstrapping procedure is executed only one time and for each additional function we just have to compute one inner product $\mathbf{w} \cdot \mathsf{ek}_i$ over $\bar{R}$.

To achieve 100 bits of security, we fixed the following parameters:

- *Base scheme:* $\bar{N} = 8$, $\bar{\eta} = 100$, $\bar{\rho} = 85$, $\bar{\gamma} = 423$.
- *Key* rk*:* $\eta_{\mathsf{rk}} = 104$, $\rho_{\mathsf{rk}} = 88$, $\gamma_{\mathsf{rk}} = 231$.
- *Decomposition:* for each base B, we set $L = \lceil \log_B(241) \rceil$.
- *Key* $\mathfrak{K}_\delta$*:* $\Delta = (L - \log_B(2^{100}))/6$ and $\delta = \lceil 2\Delta \rceil$.

Moreover, after applying the key rk to obtain an integer ciphertext c, we set $\mu := \lambda - 7$ bits of c to zero, as discussed in Section 7.4.4, thus, we did not generate keys $\mathfrak{K}_{g,i}$'s for $0 \leq i < \mu_B := \lfloor \mu \log_B(2) \rfloor$. To instantiate GAHE, we used $\eta = 100$ and $\gamma = 200$. The other parameters depend on B and are shown in Table 7.3 along with the sizes of our bootstrapping keys and the running times. The parameter N was chosen as the smaller power of two that is bigger than $4t\delta$. We also included the practical results of [CIM19] for comparison. Using around 4 GB of key material, which is approximately half of the key size used in [CIM19], our running times are already faster than those of [CIM19]. We remark that truncating μ bits allowed us to save about 35 % of the memory requirements, as the sizes of bk would have been 4.24 GB and 6.91 GB, instead of the values presented in Table 7.3 if we had not ignored the least significant bits of the ciphertexts.

7.7 Applying multi-value bootstrapping to DGHV like schemes

In this section, we discuss briefly how our multi-value bootstrapping procedure can be applied to schemes like [DGHV10], which multiply the noise by t instead of shifting the message by some α. Hence, consider that $m \in \mathbb{Z}_t$ is encrypted as $c = (\bar{p}q + tr + m)\bar{k} \in \bar{R}$. For the original [DGHV10], just consider $\bar{N} = \bar{k} = 1$. We can transform c into a ciphertext $\bar{c} = (\bar{p}\bar{q} + \bar{r} + \bar{\alpha}m)\bar{k}$, which has the format expected by our refreshing procedure, as follows:

Let $e := -\bar{p} \bmod 2t$ and $\beta := \frac{(\bar{p}+e)}{t} \in \mathbb{Z}$. Notice then, that

$$g^{-1}(c)\mathbf{g} \cdot \beta = c\beta = (\bar{p}\beta q + t\beta r + \beta m)\bar{k} = \left(\bar{p}\underbrace{(\beta q + r)}_{\bar{q}} + \underbrace{er + em}_{\bar{r}} + \underbrace{(p/t)m}_{\bar{\alpha}m} \right) \bar{k}.$$

Therefore, it is sufficient to append the following "transformation key" to the bootstrapping keys:

$$\mathsf{tk} := (\bar{p}\mathbf{q} + \mathbf{r})\bar{k} + \mathbf{g} \cdot \beta \in \bar{R}^{\bar{\ell}}.$$

Besides that, we would need define $\mathfrak{K}_\delta$ with $\alpha := 1$ and all noise terms generated by GAHE.EncScalar and GAHE.EncVec, and also the noise terms of ek_i would be multiplied by t, so that the extracted messages have the correct format at the end of the refreshing procedure.

Chapter 8

Conclusion

In this thesis, we have studied the AGCD problem and its common variants, firstly, revisiting the known attacks and proposing a new orthogonal-lattice attack on the multi-prime AGCD problem, then, discussing efficiency issues of cryptographic schemes based in this problem and proposing randomized variants, as the VAGCD and the RAGCD, to improve the memory requirements and the running times of AGCD-based cryptographic primitives. We believe that, in general, existing cryptosystems can easily be changed by replacing the AGCD problem by the corresponding randomized version, with the benefit of having smaller parameters and more efficient procedures. Moreover, constructions that would be unpractical over the AGCD problem are possible by using the VAGCD or the RAGCD problem.

In particular, using the multi-prime VAGCD problem, we described the first implementation of an N-party one-round key-exchange (NOKE) protocol that is secure against all the existing attacks and argued that it would be totally unpractical if the parameters were chosen using the multi-prime AGCD problem. Investigating how to improve the time needed to generate the parameters of our key exchange is an interesting problem. Moreover, providing a proof of security for our NOKE protocol is left as an open problem. Although being relevant from a theoretical point of view, the practical importance of proofs of security involving multilinear maps is debatable, because they generally use very restricted security models, as the generic colored matrix model [GGH13a], which supposes that an attacker can only perform some specific matrix operations on the encoded values,

or the generic multilinear map model [BGK+14], which uses an idealized oracle to model the multilinear map, in particular, it supposes that the zero-test procedure outputs a single bit determining whether the given value encodes a zero or not, but, in practice, zero-tested values can also leak some information about the secret parameters of the multilinear map, indeed, Cheon et al. attack is only possible because there are known linear relations between zero-tested encodings of zero and the secret parameters of the CLT13 multilinear map.

To show another application of the VAGCD problem, we also proposed a leveled homomorphic scheme that operates with vectors and matrices natively. Compared to other homomorphic schemes based on the AGCD problem, the size of the ciphertexts and the cost of the homomorphic operations are good, specially when we compare the evaluation of functions with high multiplicative degree, since previous AGCD-based schemes require very large parameters in this case. We showed two applications of our scheme: homomorphic evaluation of nondeterministic finite-state automata and a simple homomorphic classifier, but many other applications are possible, including more complex classifier, or other machine-learning related applications, as the principal component analysis, which is already naturally described in terms of vectors and matrices operations. One example of a more general application is the evaluation of private matrix branching programs (MBP) on a known input. Namely, one could transform a circuit into an MBP, encrypt the matrices, and send the ciphertexts to a third party, who could then evaluate the program in an input of their choice by using our homomorphic vector-matrix product. We also notice that it is still possible to improve a little the efficiency of our scheme by using the Chinese Remainder Theorem to encrypt about γ/η matrices (or vectors) in a same ciphertext and to perform homomorphic operations in parallel. However, since γ/η is typically small in our parameter choice, the efficiency gain is not significant.

Finally, we showed that bootstrapping procedures for FHE schemes over the integers are not inherently slower than for (R)LWE-based schemes. Our proposed bootstrapping for AGCD-based FHE schemes are still slower than the best bootstrapping methods based on the RLWE problem, but at least now we can hope that the current difference of

efficiency is due to practical reasons, as implementation aspects, rather than some theoretical impossibility result. It would be interesting to find an efficient way of performing our multi-value bootstrapping on a base scheme that encrypts a polynomial m instead of an integer. If m were a binary polynomial of low degree, we could evaluate m at 2 (using our functional key-switching) to obtain an integer encryption, then, perform one single multi-value bootstrapping, but with one extraction key ek_i for each bit of the integer $m(2)$, which would correspond to each coefficient of m. However, the general case remains as an open problem. Another natural question that arises is how to perform similar fast bootstrapping procedures to AGCD-based schemes that use the Chinese Remainder Theorem to pack several messages into one ciphertext, since that could significantly improve the amortized cost of bootstrapping.

More broadly, we expect that the randomized problems proposed in this thesis will be valuable for the scientific community and will still be studied and cryptanalyzed in other works, and also used to construct more cryptographic schemes.

References

[APS15] Martin R. Albrecht, Rachel Player, and Sam Scott. On the concrete hardness of learning with errors. *J. Math. Cryptol.*, 9(3):169–203, 2015.
→ Cited on pages 17, 37, and 45.

[ASP14] Jacob Alperin-Sheriff and Chris Peikert. Faster bootstrapping with polynomial error. In *Advances in Cryptology – CRYPTO 2014*, pages 297–314, Berlin, Heidelberg, 2014. Springer Berlin Heidelberg.
→ Cited on pages 152, 156, 157, and 158.

[Bar89] David A Barrington. Bounded-width polynomial-size branching programs recognize exactly those languages in nc1. *Journal of Computer and System Sciences*, 38(1):150–164, 1989.
→ Cited on page 64.

[BBL17] Daniel Benarroch, Zvika Brakerski, and Tancrède Lepoint. FHE over the Integers: Decomposed and Batched in the Post-Quantum Regime. In *Public-Key Cryptography – PKC 2017*, pages 271–301, Berlin, Heidelberg, 2017. Springer Berlin Heidelberg.
→ Cited on pages 22, 29, 34, 92, 93, 94, 96, 100, 109, 114, 127, 151, 152, and 164.

[BCLO09] Alexandra Boldyreva, Nathan Chenette, Younho Lee, and Adam O'Neill. Order-preserving symmetric encryption. In Antoine Joux, editor, *Advances in Cryptology - EUROCRYPT 2009*, pages 224–241, Berlin, Heidelberg,

2009. Springer Berlin Heidelberg.
→ Cited on page 2.

[BDGL16] Anja Becker, Léo Ducas, Nicolas Gama, and Thijs Laarhoven. New directions in nearest neighbor searching with applications to lattice sieving. In *Proceedings of the Twenty-Seventh Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2016, Arlington, VA, USA, January 10-12, 2016*, 2016.
→ Cited on page 79.

[Ber04] Daniel J. Bernstein. How to find smooth parts of integers, 2004. Available at https://cr.yp.to/papers.html#smoothparts.
→ Cited on page 47.

[BGI+01] Boaz Barak, Oded Goldreich, Rusell Impagliazzo, Steven Rudich, Amit Sahai, Salil Vadhan, and Ke Yang. On the (im)possibility of obfuscating programs. In Joe Kilian, editor, *Advances in Cryptology — CRYPTO 2001*, pages 1–18, Berlin, Heidelberg, 2001. Springer Berlin Heidelberg.
→ Cited on page 63.

[BGK+14] Boaz Barak, Sanjam Garg, Yael Tauman Kalai, Omer Paneth, and Amit Sahai. Protecting obfuscation against algebraic attacks. In *Advances in Cryptology – EUROCRYPT 2014 - Proceedings*, 2014.
→ Cited on pages 58, 74, 75, and 180.

[BGN05] Dan Boneh, Eu-Jin Goh, and Kobbi Nissim. Evaluating 2-dnf formulas on ciphertexts. In Joe Kilian, editor, *Theory of Cryptography*, pages 325–341, Berlin, Heidelberg, 2005. Springer Berlin Heidelberg.
→ Cited on page 3.

[BGV12] Zvika Brakerski, Craig Gentry, and Vinod Vaikuntanathan. (Leveled) Fully Homomorphic Encryption Without Bootstrapping. In *Proceedings of the 3rd Innovations in Theoretical Computer Science Conference*, ITCS '12, pages

309–325, New York, NY, USA, 2012. ACM.
→ Cited on pages 26, 29, 127, and 151.

[BPTG15] Raphael Bost, Raluca Ada Popa, Stephen Tu, and Shafi Goldwasser. Machine learning classification over encrypted data. In *NDSS*, volume 4324, page 4325, 2015.
→ Cited on page 121.

[Bra12] Zvika Brakerski. Fully Homomorphic Encryption without Modulus Switching from Classical GapSVP. In *Advances in Cryptology – CRYPTO 2012*, pages 868–886, Berlin, Heidelberg, 2012. Springer Berlin Heidelberg.
→ Cited on page 169.

[BS03] Dan Boneh and Alice Silverberg. Applications of multilinear forms to cryptography. In *Contemporary Mathematics*, volume 324, pages 71–90. American Mathematical Society, 2003.
→ Cited on pages 59 and 67.

[BSW11] Dan Boneh, Amit Sahai, and Brent Waters. Functional encryption: Definitions and challenges. In Yuval Ishai, editor, *Theory of Cryptography*, pages 253–273, Berlin, Heidelberg, 2011. Springer Berlin Heidelberg.
→ Cited on page 2.

[BV11] Z. Brakerski and V. Vaikuntanathan. Efficient Fully Homomorphic Encryption from (Standard) LWE. In *2011 IEEE 52nd Annual Symposium on Foundations of Computer Science*, pages 97–106, Oct 2011.
→ Cited on pages 29 and 151.

[CCK+13] Jung Hee Cheon, Jean-Sébastien Coron, Jinsu Kim, Moon Sung Lee, Tancrède Lepoint, Mehdi Tibouchi, and Aaram Yun. Batch fully homomorphic encryption over the integers. In *Advances in Cryptology – EUROCRYPT 2013*, pages 315–335, Berlin, Heidelberg, 2013. Springer.
→ Cited on page 151.

[CGGI16a] Ilaria Chillotti, Nicolas Gama, Mariya Georgieva, and Malika Izabachène. Faster Fully Homomorphic Encryption: Bootstrapping in Less Than 0.1 Seconds. In *Advances in Cryptology – ASIACRYPT 2016*, pages 3–33, Berlin, Heidelberg, 2016. Springer Berlin Heidelberg.
→ Cited on pages 7, 92, 137, 152, 153, 155, 158, 162, 163, 175, and 176.

[CGGI16b] Ilaria Chillotti, Nicolas Gama, Mariya Georgieva, and Malika Izabachène. A homomorphic lwe based e-voting scheme. In Tsuyoshi Takagi, editor, *Post-Quantum Cryptography*, pages 245–265, Cham, 2016. Springer International Publishing.
→ Cited on page 26.

[CGGI19] Ilaria Chillotti, Nicolas Gama, Mariya Georgieva, and Malika Izabachène. TFHE: Fast Fully Homomorphic Encryption Over the Torus. *Journal of Cryptology*, Apr 2019.
→ Cited on pages 7 and 29.

[CGH+15] Jean-Sébastien Coron, Craig Gentry, Shai Halevi, Tancrède Lepoint, Hemanta K. Maji, Eric Miles, Mariana Raykova, Amit Sahai, and Mehdi Tibouchi. Zeroizing without low-level zeroes: New MMAP attacks and their limitations. In *CRYPTO 2015, Part I*, volume 9215 of *LNCS*. Springer, 2015.
→ Cited on pages 58 and 81.

[CHL+15] Jung Hee Cheon, Kyoohyung Han, Changmin Lee, Hansol Ryu, and Damien Stehlé. Cryptanalysis of the multilinear map over the integers. In *EUROCRYPT 2015, Part I*, volume 9056 of *LNCS*. Springer, 2015.
→ Cited on pages 39, 40, 41, 57, and 80.

[CIM19] Sergiu Carpov, Malika Izabachène, and Victor Mollimard. New Techniques for Multi-value Input Homomorphic Evaluation and Applications. In *Topics in Cryptology – CT-RSA 2019*, pages 106–126, Cham, 2019. Springer Inter-

national Publishing.
→ Cited on pages 153, 155, 176, and 177.

[CLLT17] Jean-Sébastien Coron, Moon Sung Lee, Tancrède Lepoint, and Mehdi Tibouchi. Zeroizing attacks on indistinguishability obfuscation over CLT13. In *Public-Key Cryptography - PKC 2017 - Proceedings, Part I*, 2017.
→ Cited on pages 58, 83, and 84.

[CLT13] Jean-Sébastien Coron, Tancrède Lepoint, and Mehdi Tibouchi. Practical multilinear maps over the integers. In *CRYPTO*, volume 8042 of *LNCS*. Springer, 2013.
→ Cited on pages 5, 31, 39, 51, 52, 57, 60, 62, 87, and 89.

[CMNT11] Jean-Sébastien Coron, Avradip Mandal, David Naccache, and Mehdi Tibouchi. Fully homomorphic encryption over the integers with shorter public keys. In Phillip Rogaway, editor, *Advances in Cryptology – CRYPTO 2011*, pages 487–504, Berlin, Heidelberg, 2011. Springer Berlin Heidelberg.
→ Cited on pages 34 and 35.

[CN11] Yuanmi Chen and Phong Q. Nguyen. Bkz 2.0: Better lattice security estimates. In Dong Hoon Lee and Xiaoyun Wang, editors, *Advances in Cryptology – ASIACRYPT 2011*, pages 1–20, Berlin, Heidelberg, 2011. Springer Berlin Heidelberg.
→ Cited on page 79.

[CN12] Yuanmi Chen and Phong Nguyen. Faster Algorithms for Approximate Common Divisors: Breaking Fully Homomorphic Encryption Challenges over the Integers. In *Advances in Cryptology – EUROCRYPT 2012*, volume 7237 of *Lecture Notes in Computer Science*. Springer, 2012.
→ Cited on pages 35, 37, and 46.

[CNT12] Jean-Sébastien Coron, David Naccache, and Mehdi Tibouchi. Public key compression and modulus switching for fully homomorphic encryption over

the integers. In *Advances in Cryptology – EUROCRYPT 2012*, pages 446–464, Berlin, Heidelberg, 2012. Springer Berlin Heidelberg.
→ Cited on pages 46, 47, 129, 151, 153, and 155.

[CP19a] Jean-Sebastien Coron and Hilder V. L. Pereira. On Kilian's Randomization of Multilinear Map Encodings. In *ASIACRYPT 2019*, 2019. https://eprint.iacr.org/2018/1129.
→ Cited on pages 7, 9, and 93.

[CP19b] Jean-Sebastien Coron and Hilder V. L. Pereira. Source code of key exchange based on CLT13 multilinear maps, 2019. Publicly available at https://github.com/coron/cltexchangeimpl.
→ Cited on pages 9, 77, 81, 82, 83, 86, and 88.

[CS15] Jung Hee Cheon and Damien Stehlé. Fully Homomophic Encryption over the Integers Revisited. In *EUROCRYPT 2015 - 34th Annual International Conference on the Theory and Applications of Cryptographic Techniques*, Sofia, Bulgaria, April 2015.
→ Cited on pages 17, 34, 35, 37, 45, 164, and 169.

[DGHV10] Marten van Dijk, Craig Gentry, Shai Halevi, and Vinod Vaikuntanathan. Fully Homomorphic Encryption over the Integers. In *Advances in Cryptology – EUROCRYPT 2010*, volume 6110 of *Lecture Notes in Computer Science*. Springer, 2010.
→ Cited on pages 4, 26, 29, 31, 33, 35, 127, 151, 164, 170, and 177.

[DK07] Hans Delfs and Helmut Knebl. *Introduction to Cryptography: Principles and Applications*. Information Security and Cryptography. Springer-Verlag Berlin Heidelberg, second edition, 2007.
→ Cited on page 1.

[DM15] Léo Ducas and Daniele Micciancio. FHEW: Bootstrapping Homomorphic Encryption in Less Than a Second. In *Advances in Cryptology – EURO-*

CRYPT 2015, pages 617–640, Berlin, Heidelberg, 2015. Springer.
→ Cited on pages 7, 9, 29, 152, 153, 155, 157, 162, 163, 175, and 176.

[Gen09] Craig Gentry. *A fully homomorphic encryption scheme*. PhD thesis, Stanford University, 2009. crypto.stanford.edu/craig.
→ Cited on pages 3, 26, 27, 28, 91, and 151.

[GGH13a] Sanjam Garg, Craig Gentry, and Shai Halevi. Candidate multilinear maps from ideal lattices. In *EUROCRYPT*, volume 7881 of *LNCS*. Springer, 2013.
→ Cited on pages 5, 57, 59, 60, and 179.

[GGH+13b] Sanjam Garg, Craig Gentry, Shai Halevi, Mariana Raykova, Amit Sahai, and Brent Waters. Candidate indistinguishability obfuscation and functional encryption for all circuits. In *2013 IEEE 54th Annual Symposium on Foundations of Computer Science*, pages 40–49, 2013.
→ Cited on pages 26, 58, 63, 64, 66, and 89.

[GGH15] Craig Gentry, Sergey Gorbunov, and Shai Halevi. Graph-induced multilinear maps from lattices. In *TCC*, volume 9015 of *LNCS*, 2015.
→ Cited on pages 57 and 60.

[GGH+19] Nicholas Genise, Craig Gentry, Shai Halevi, Baiyu Li, and Daniele Micciancio. Homomorphic encryption for finite automata. In *ASIACRYPT 2019*, 2019. https://eprint.iacr.org/2019/176.
→ Cited on pages 91, 92, 93, 94, 96, 98, 107, 117, 118, 125, and 126.

[GGI+15] Craig Gentry, Jens Groth, Yuval Ishai, Chris Peikert, Amit Sahai, and Adam Smith. Using fully homomorphic hybrid encryption to minimize non-interative zero-knowledge proofs. *Journal of Cryptology*, 28(4):820–843, October 2015.
→ Cited on page 26.

[GGM16] Steven D. Galbraith, Shishay W. Gebregiyorgis, and Sean Murphy. Algorithms for the approximate common divisor problem. *LMS Journal of Com-*

putation and Mathematics, 19(A):58–72, 2016.
→ Cited on pages 36 and 39.

[GHS12] Craig Gentry, Shai Halevi, and Nigel P. Smart. Fully homomorphic encryption with polylog overhead. In *Proceedings of the 31st Annual International Conference on Theory and Applications of Cryptographic Techniques*, EUROCRYPT'12, pages 465–482, Berlin, Heidelberg, 2012. Springer-Verlag.
→ Cited on pages 29 and 151.

[GSW13] Craig Gentry, Amit Sahai, and Brent Waters. Homomorphic Encryption from Learning with Errors: Conceptually-Simpler, Asymptotically-Faster, Attribute-Based. In *Advances in Cryptology – CRYPTO 2013*, pages 75–92, Berlin, Heidelberg, 2013. Springer Berlin Heidelberg.
→ Cited on pages 5, 26, 29, 92, 93, 94, and 151.

[Hal17] Shai Halevi. *Homomorphic Encryption*, pages 219–276. Springer International Publishing, Cham, 2017.
→ Cited on page 28.

[HAO15] Ryo Hiromasa, Masayuki Abe, and Tatsuaki Okamoto. Packing Messages and Optimizing Bootstrapping in GSW-FHE. In *Public-Key Cryptography – PKC 2015*, pages 699–715, Berlin, Heidelberg, 2015. Springer Berlin Heidelberg.
→ Cited on page 96.

[HG01] Nick Howgrave-Graham. Approximate integer common divisors. In *Cryptography and Lattices*, pages 51–66, Berlin, Heidelberg, 2001. Springer.
→ Cited on page 33.

[HPS11] Guillaume Hanrot, Xavier Pujol, and Damien Stehlé. Analyzing blockwise lattice algorithms using dynamical systems. In *CRYPTO 2011*, 2011.
→ Cited on pages 17, 37, 41, and 45.

[KL14] Jonathan Katz and Yehuda Lindell. *Introduction to Modern Cryptography*. Chapman & Hall/CRC Cryptography and Network Security Series. Chapman and Hall/CRC, second edition, 2014.
→ Cited on page 1.

[Lip03] Helger Lipmaa. Verifiable homomorphic oblivious transfer and private equality test. In Chi-Sung Laih, editor, *Advances in Cryptology - ASIACRYPT 2003*, pages 416–433, Berlin, Heidelberg, 2003. Springer Berlin Heidelberg.
→ Cited on page 26.

[LS14] Hyung Tae Lee and Jae Hong Seo. Security analysis of multilinear maps over the integers. In *Advances in Cryptology - CRYPTO 2014 - Proceedings, Part I*, 2014.
→ Cited on pages 35, 38, 46, and 136.

[MF71] A. R. Meyer and M. J. Fischer. Economy of description by automata, grammars, and formal systems. In *12th Annual Symposium on Switching and Automata Theory (swat 1971)*, pages 188–191, Oct 1971.
→ Cited on page 117.

[MG02] Daniele Micciancio and Shafi Goldwasser. *Complexity of Lattice Problems*. Springer US, first edition, 2002.
→ Cited on page 16.

[MP12] Daniele Micciancio and Chris Peikert. Trapdoors for lattices: Simpler, tighter, faster, smaller. In David Pointcheval and Thomas Johansson, editors, *Advances in Cryptology – EUROCRYPT 2012*, pages 700–718, Berlin, Heidelberg, 2012. Springer Berlin Heidelberg.
→ Cited on page 97.

[NS97] Phong Nguyen and Jacques Stern. Merkle-hellman revisited: A cryptanalysis of the qu-vanstone cryptosystem based on group factorizations. In Burton S. Kaliski, editor, *Advances in Cryptology — CRYPTO '97*, pages 198–212,

Berlin, Heidelberg, 1997. Springer Berlin Heidelberg.
→ Cited on page 15.

[NS01] Phong Q. Nguyen and Jacques Stern. The two faces of lattices in cryptology. In *Cryptography and Lattices, International Conference, CaLC 2001, Providence, RI, USA, March 29-30, 2001, Revised Papers*, 2001.
→ Cited on page 35.

[Pai99] Pascal Paillier. Public-key cryptosystems based on composite degree residuosity classes. In Jacques Stern, editor, *Advances in Cryptology — EUROCRYPT '99*, pages 223–238, Berlin, Heidelberg, 1999. Springer Berlin Heidelberg.
→ Cited on pages 3 and 25.

[Per20a] Hilder Vitor Lima Pereira. Bootstrapping fully homomorphic encryption over the integers in less than one second. IACR e-print, 2020. https://eprint.iacr.org/2020/491.
→ Cited on page 8.

[Per20b] Hilder Vitor Lima Pereira. Efficient AGCD-based homomorphic encryption for matrix and vector arithmetic. International Conference on Applied Cryptography and Network Security (ACNS), 2020. https://eprint.iacr.org/2020/491.
→ Cited on pages 8 and 9.

[Per20c] Hilder Vitor Lima Pereira. Source code of HEVaM: Homomorphic Encryption for Vector And Matrix arithmetic. GitHub, 2020. GitHub repository: https://github.com/hilder-vitor/HEVaM.
→ Cited on pages 9, 94, and 114.

[Per20d] Hilder Vitor Lima Pereira. Source code of several attacks on the AGCD problem, VAGCD problem, and their variants. GitHub, 2020. GitHub repository:

https://github.com/hilder-vitor/attacks-AGCD-VAGCD.
→ Cited on page 45.

[Per20e] Hilder Vitor Lima Pereira. Source code of the RAGCD-based homomorphic scheme and our bootstrapping procedures. GitHub, 2020. GitHub repository: https://github.com/hilder-vitor/boot-FHE-Z.
→ Cited on pages 9, 147, and 174.

[PKK+18] Heejin Park, Pyung Kim, Heeyoul Kim, Ki-Woong Park, and Younho Lee. Efficient machine learning over encrypted data with non-interactive communication. *Computer Standards and Interfaces*, 58:87 – 108, 2018.
→ Cited on page 122.

[Reg05] Oded Regev. On lattices, learning with errors, random linear codes, and cryptography. In *Proceedings of the Thirty-Seventh Annual ACM Symposium on Theory of Computing*, STOC '05, page 84–93, New York, NY, USA, 2005. Association for Computing Machinery.
→ Cited on page 4.