THÈSE DE DOCTORAT

de l'Université de recherche Paris Sciences et Lettres
PSL Research University

Préparée à l'École normale supérieure

Fully Homomorphic Encryption for Machine Learning

École doctorale n°386
Sciences Mathématiques de Paris Centre

Spécialité Informatique

Soutenue par
Michele MINELLI
le 26 octobre 2018

Dirigée par
Michel FERREIRA ABDALLA
Hoeteck WEE

Fully Homomorphic Encryption for Machine Learning

Michele MINELLI

Supervisors: Michel FERREIRA ABDALLA and Hoeteck WEE

Abstract

Fully homomorphic encryption enables computation on encrypted data without leaking any information about the underlying data. In short, a party can encrypt some input data, while another party, that does not have access to the decryption key, can blindly perform some computation on this encrypted input. The final result is also encrypted, and it can be recovered only by the party that possesses the secret key.

In this thesis, we present new techniques/designs for FHE that are motivated by applications to machine learning, with a particular attention to the problem of homomorphic inference, i.e., the evaluation of already trained cognitive models on encrypted data.

First, we propose a novel FHE scheme that is tailored to evaluating neural networks on encrypted inputs. Our scheme achieves complexity that is essentially independent of the number of layers in the network, whereas the efficiency of previously proposed schemes strongly depends on the topology of the network.

Second, we present a new technique for achieving circuit privacy for FHE. This allows us to hide the computation that is performed on the encrypted data, as is necessary to protect proprietary machine learning algorithms. Our mechanism incurs very small computational overhead while keeping the same security parameters.

Together, these results strengthen the foundations of efficient FHE for machine learning, and pave the way towards practical privacy-preserving deep learning.

Finally, we present and implement a protocol based on homomorphic encryption for the problem of private information retrieval, i.e., the scenario where a party wants to query a database held by another party without revealing the query itself.

Résumé

Le chiffrement totalement homomorphe permet d'effectuer des calculs sur des données chiffrées sans fuite d'information sur celles-ci. Pour résumer, un utilisateur peut chiffrer des données, tandis qu'un serveur, qui n'a pas accès à la clé de déchiffrement, peut appliquer à l'aveugle un algorithme sur ces entrées. Le résultat final est lui aussi chiffré, et il ne peut être lu que par l'utilisateur qui possède la clé secrète.

Dans cette thèse, nous présentons des nouvelles techniques et constructions pour le chiffrement totalement homomorphe qui sont motivées par des applications en apprentissage automatique, en portant une attention particulière au problème de l'inférence homomorphe, c'est-à-dire l'évaluation de modèles cognitifs déjà entrainé sur des données chiffrées.

Premièrement, nous proposons un nouveau schéma de chiffrement totalement homomorphe adapté à l'évaluation de réseaux de neurones artificiels sur des données chiffrées. Notre schéma atteint une complexité qui est essentiellement indépendante du nombre de couches dans le réseau, alors que l'efficacité des schéma proposés précédemment dépend fortement de la topologie du réseau.

Ensuite, nous présentons une nouvelle technique pour préserver la confidentialité du circuit pour le chiffrement totalement homomorphe. Ceci permet de cacher l'algorithme qui a été exécuté sur les données chiffrées, comme nécessaire pour protéger les modèles propriétaires d'apprentissage automatique. Notre mécanisme rajoute un coût supplémentaire très faible pour un niveau de sécurité égal. Ensemble, ces résultats renforcent les fondations du chiffrement totalement homomorphe efficace pour l'apprentissage automatique, et représentent un pas en avant vers l'apprentissage profond pratique préservant la confidentialité.

Enfin, nous présentons et implémentons un protocole basé sur le chiffrement totalement homomorphe pour le problème de recherche d'information confidentielle, c'est-à-dire un scénario où un utilisateur envoie une requête à une base de donnée tenue par un serveur sans révéler cette requête.

Contents

Chapter 1

Introduction

In this chapter, we introduce the topic of this thesis in the area of cryptography. First of all, we motivate the interest for cryptography, we give some historical facts about this science, and we introduce the main points of this thesis. Finally, we present the organization of this manuscript.

Contents

Cryptography is usually perceived as a tool of war, involved in military operations or intelligence activities. From a long time ago, when it was used by Roman generals to securely send orders to the legions on the field, to the Second World War, with the Nazis using the famous Enigma machine to keep the Allied from understanding the content of their communications, to our days, when it is used to protect military secrets from being intercepted by spies or enemy states, or to communicate with James-Bond-like characters deployed on the field. Although this romantic aura surrounding cryptography has many elements of truth, this is far from being the entire picture. Cryptography is also used by common people, to carry out daily tasks that require some attention to concepts like privacy, integrity, and reliability. For example, almost everybody sends text messages or pictures from their phones. Despite the fact that cryptography is often used transparently in these cases (i.e., the users do not realize messages are encrypted/decrypted), nowadays it is normally the case that content travels through the internet in an encrypted version, so that a potential eavesdropper would gain no information from intercepting the message. Although some people would argue that this is paranoid, since their messages are not sensitive at all and they have nothing to hide[1], this is clearly not the case when it comes to sensitive applications like home banking. Even in this case, cryptography allows people to carry out daily life operations without the risk of information being stolen or falling in the hands of malicious criminals. Finally, cryptography is a tool that, in many cases, literally saves lives: it is used by whistle-blowers or activists to conceal their communications from repressive governments or organizations, thus protecting their physical safety.

More in general, the traditional goal of cryptography is to allow two parties to communicate over an insecure channel, in a way that preserves the confidentiality of the communication, and prevents anyone else from eavesdropping on the messages that are being exchanged. Historically, we usually refer to these parties as Alice and Bob, while the third party, referred to as the *adversary*, is usually called Eve.

What Alice and Bob would use is an *encryption scheme*, which intuitively is a protocol designed to allow for this protected communication, and consists of two procedures called *encryption* and *decryption*. Assuming Alice wants to send a message to Bob, she first applies the encryption procedure to the message, then sends the result to Bob who will apply the decryption procedure and recover the original message. This means that what passes through the insecure channel (and is thus potentially known to the adversary) is only the encryption of the message, called *ciphertext*. We are then in a situation where both Bob and the adversary know the ciphertext, so it must be the case that Bob knows something that the adversary does not, otherwise the adversary could simply proceed in the same way as Bob and recover the message. This extra piece of information is called the *secret key*. We can then assume that encryption and decryption are public procedures, and that the secret key is fed to them along with the message (for the encryption procedure) or the ciphertext (for the decryption), in order to obtain the expected result. We also assume that this secret key is shared between Alice and Bob. This corresponds to the case of *private-key* encryption, also referred to as *symmetric* encryption.

At this point, the reasonable question to ask is about the security of this encryption scheme: what does the adversary learn from seeing the ciphertext going through the com-

[1]This way of thinking is widely regarded as dangerous in the crypto community, as it leads to an oversimplification of the very complex problem of "right to privacy". Also, it is safe to assume that everyone has something they are embarrassed, ashamed, or frightened of. And that everyone has secrets.

munication channel? Ideally, we would like to answer “nothing”, meaning that the ciphertext contains absolutely no information on the message. This notion is called *information theoretic* or *perfect* security, and can be achieved by a trivial scheme, known as *one-time pad*.

We now give an intuitive toy example of this scheme. Let us consider the case where Alice wants to send the message “iloveyou” to Bob, and let us assume that they had previously agree upon the secret key “4,7,2,9,5,8,7,1”. They can then proceed as follows: for each character of the original message, they “shift” it by a number of spots which is given by the corresponding entry in the secret key (wrapping around when necessary, i.e., X $\to$ Y $\to$ Z $\to$ A $\to \ldots$). Alice then does the following:

$$\text{Encryption:} \qquad \begin{array}{cccccccc} \mathrm{i} & \mathrm{l} & \mathrm{o} & \mathrm{v} & \mathrm{e} & \mathrm{y} & \mathrm{o} & \mathrm{u} \\ + & + & + & + & + & + & + & + \\ 4 & 7 & 2 & 9 & 5 & 8 & 7 & 1 \\ \hline \mathrm{m} & \mathrm{s} & \mathrm{q} & \mathrm{e} & \mathrm{j} & \mathrm{g} & \mathrm{v} & \mathrm{v} \end{array}$$

The message that is sent is then “msqejgvv”. Upon receiving it, Bob performs the inverse operation:

$$\text{Decryption:} \qquad \begin{array}{cccccccc} \mathrm{m} & \mathrm{s} & \mathrm{q} & \mathrm{e} & \mathrm{j} & \mathrm{g} & \mathrm{v} & \mathrm{v} \\ - & - & - & - & - & - & - & - \\ 4 & 7 & 2 & 9 & 5 & 8 & 7 & 1 \\ \hline \mathrm{i} & \mathrm{l} & \mathrm{o} & \mathrm{v} & \mathrm{e} & \mathrm{y} & \mathrm{o} & \mathrm{u} \end{array}$$

This way of encrypting and decrypting things is perfectly secure, meaning that a party that does not have the secret key learns no information about the message (apart from its length). In fact, let us say that the adversary sees the ciphertext “msqejgvv”. This could also correspond to the message “ihateyou” (under the key “4,11,16,11,5,8,7,1”), or “ihavenot” (under the key “4,11,16,9,5,19,7,2”), or “abcdefgh” (under the key “12,17,14,1,5,1,15,14”). In a nutshell, this ciphertext could represent any 8-character message, and is thus completely hiding what Alice wanted to communicate to Bob.

One could then think that this is everything we need from cryptography: we have a way to achieve perfect security, and this is sufficient. However, this is far from being the case. In fact, it is easy to see that in the previous example, the key had to be at least as long as the message it was intended to hide. Also these keys cannot be used more than once[2], otherwise some dangerous attacks arise, that can compromise the security of the scheme. This means that Alice and Bob must be able to share a huge amount of secret bits, which quickly becomes impractical, or are limited to communicating short messages. Also, they have to agree on the secret key *before* communicating, either by meeting in person or in another way: this only adds to the impracticality of this system.

With the breakthrough discoveries of *key exchange* in 1976 by Diffie and Hellman [DH76], *public-key encryption* (also known as *asymmetric* encryption), and *digital signatures* in 1978 by Rivest, Shamir and Adleman [RSA78], the scope of cryptology broadened considerably. In public-key cryptography, each party has a pair of keys: a public one and a private (or secret) one. The public one can be published, e.g., on the Internet, and allows anyone to encrypt a message, that can only be decrypted with the corresponding private key. In order to explain this concept, a famous analogy is often used: the public key corresponds to an open lock, whereas the private key corresponds to the lock’s key. Publishing the public key

[2]Hence, the name “one-time pad”.

is equivalent to making the open lock available; then anyone can write a message, put it in a box, and close the box with the provided lock. The sealed box is then sent to the recipient, who can open it with the appropriate key.

1.1 Computation outsourcing

A moment's thought reveals that there is a fundamental limit in the concept of encryption as it was presented. In fact, once something is encrypted, it can only be stored in order to be kept safe and retrieved at a later time, or it can be sent to someone who has the possibility (i.e., a suitable secret) to decrypt it and recover it. This means that the encrypted version of the data is completely useless to any party that does not hold the secret piece of information to decrypt it. On the bright side, this means that no unauthorized party can access the encrypted information; on the not-so-bright side, it also means that no party can perform operations on the encrypted data. The reason why we would like such a thing to be possible lies in the concept of *computation outsourcing*, i.e., when a party receives some input data x, performs some computation on it, and returns the result.

Nowadays, everybody owns or produces enormous quantities of data, whether it be pictures, documents, bills, records, etc., and a big part of the users' activities are concentrated on small devices, with limited storage and limited computational capacities. This is one of the reasons why *Cloud computing* has gained a lot of momentum over the last years and is becoming a key paradigm in the way we interact with our data. In a nutshell, the model encompasses a user with limited capabilities and a powerful remote server (the Cloud), that has massive storage devices and powerful computing resources as its disposal. The idea is that the user uploads its data to the Cloud and then asks the Cloud to perform some task and return the result.

The issue with this model is clearly data privacy. In fact, the remote server is usually *untrusted*, meaning that it is run by a party that we do not know or control in any way, and whose honesty we cannot trust. The problem becomes even more evident if the date we are talking about consists of highly private and sensitive information like medical records, bank statements, personal communications, etc. This situation introduces a conflict: we can either take advantage of the Cloud computing paradigm, but we have to allow the Cloud to know our personal information, or we have to give up the potential of Cloud computing and perform all the computations locally, on trusted machines. For years, researchers have been trying to devise a way to reconcile the desire for data privacy and that for computation outsourcing, in order to obtain *privacy-preserving computation outsourcing*.

1.1.1 Homomorphic encryption

This part is a very general overview on the topic. These concepts are expanded, together with a more technical presentation, in Chapter 3.

A *homomorphic* encryption scheme is an encryption scheme that admits some kind of computations on encrypted data. In particular, given an input x encrypted as $\mathsf{Enc}\,(x)$, it should be possible to *publicly* compute $\mathsf{Enc}\,(f(x))$ for a function f taken from some class of functions. The key point here is "*publicly*", meaning that carrying out this computation has to be possible without having access to any secret information.

For a first trivial example, consider the following scenario: let us say that an encryption schemes consists of taking a message and adding a secret number S to it. Then an encryption of m_1 will be $c_1 = m_1 + S$, while an encryption of m_2 will be $c_2 = m_2 + S$. Then, any party that receives these two encryptions, can sum them together and obtain $c' = m_1 + m_2 + 2S$, without knowing either m_1, m_2, or S. A party that knows S and receives the result c' can simply subtract $2S$ (or take the result modulo S, if some other conditions hold) and recover $m_1 + m_2$. The final result is that the operation of adding two numbers has been delegated to another party, without this party knowing the operands or the result.

The "scheme" used in this toy example is obviously insecure, but it turns out that many widely-used encryption schemes already have some homomorphic properties. For example, let us consider the famous RSA cryptosystem [RSA78], in which an encryption of a message m is $m^e \mod N$, where e is a public exponent and N is a public modulus. It is easy to see that, given two ciphertexts $c_1 = m_1^e \mod N$ and $c_2 = m_2^e \mod N$ that encrypt messages m_1 and m_2, we can multiply them together and obtain $c' = (m_1 m_2)^e \mod N$, which is an encryption of $m_1 m_2$. This means that we can homomorphically multiply the messages, meaning that we can take two ciphertexts, perform only public operations on them, and obtain an encryption of the product of whatever was encrypted in the ciphertexts, without knowing any message or any secret key. For this reason we say that the RSA cryptosystem is *multiplicatively homomorphic*. Another example is the well known El Gamal encryption scheme [ElG84], which is also multiplicatively homomorphic. Notice that, for these schemes, we do not know how to perform an homomorphic addition, which amounts to computing a ciphertext that decrypts to the sum of the two original plaintexts.

Instead let us consider the famous Paillier cryptosystem [Pai99], based on the decisional composite residuosity assumption. In this scheme, an encryption of a message m is of the form $c = g^m \cdot r^n \mod n^2$, where g and n are public and r is random. In this case, a party that is given two ciphertexts c_1, c_2 encrypting messages m_1, m_2, can compute $c' = c_1 \cdot c_2 = g^{m_1+m_2} \cdot (r_1 r_2)^n \mod n^2$, which is a Paillier encryption of $m_1 + m_2$. This means that we can homomorphically sum two ciphertexts and produce an encryption of the two original messages, without knowing them or any secret piece of information. We then say that the Paillier cryptosystem is *additively* homomorphic. Notice, however, that in this case we do not know how to homomorphically multiply two ciphertexts.

We call *partially* homomorphic encryption schemes those schemes that support either addition or multiplication, but not both. Schemes that are both additively and multiplicatively homomorphic are harder to come by. An example of such scheme is the DGHV encryption scheme [DGHV10], which will be described in details later in the manuscript (cf. Chapter 6). In most of these encryption schemes, a "noise" term is added during the encryption for security purposes. This noise (sometimes called "error") grows when performing homomorphic additions and multiplications, and must remain below a certain threshold in order for the ciphertext to be correctly decryptable. In the majority of the homomorphic schemes known so far, the way noise grows with multiplication is more severe than in the case of addition. For this reason, the most important metric when quantifying the hardness of homomorphically evaluating a circuit is its *multiplicative depth*.

Another important definition is that of *somewhat* homomorphic encryption (SHE) schemes. We use this term to refer to encryption schemes that can evaluate a certain number of operations on encrypted inputs, but for which proceeding further would result in losing decryption correctness, meaning that the result will not decrypt to what is expected. Every encryption scheme is instantiated with some parameters, e.g., the size of the primes which are used, the size of the secret key, etc. We say that a homomorphic encryption scheme is *leveled* if, for any multiplicative depth L fixed a priori, it is possible to find a set of parameters such that the encryption scheme instantiated with those parameters is able to homomorphically evaluate any circuit of depth L. It is easy to see that this is a stronger notion than that of somewhat homomorphic encryption scheme.

However, the main obstacle that researchers faced for more than 30 years (since the suggestion of [RAD78]) was the following: in any case, the number of operations that can be evaluated is bounded. At some point, the noise level will become too large and it will be impossible to proceed without losing correctness. The goal of constructing a *fully* homomorphic encryption (FHE) scheme, i.e., a scheme that can homomorphically evaluate an unbounded number of operations, remained a dream for a long time, and some famous researchers claimed it was never going to be reached. The turning point came in 2009 with the breakthrough by Craig Gentry, then a Ph.D. student at Stanford under the supervision of Dan Boneh. In his dissertation [Gen09a], Gentry put forward the first plausible construction for an FHE scheme based on the hardness of some lattice problems, and that of the approximate GCD problem (cf. Chapter 2 for the details). This breakthrough had the effect of reigniting FHE as a topic of research, and since then many important results followed. Although the techniques improved greatly, and the schemes became simpler and more efficient, the original blueprint presented in Gentry's thesis continues to underlie all known FHE constructions.

The key idea that was proposed in Gentry's work is that of *bootstrapping*. By this term, we denote the process of *refreshing* a ciphertext in order to produce a new ciphertext that encrypts the same message, but with a lower level of noise so that more homomorphic operations can be evaluated on it. This operation is at the very core of any FHE schemes known to date, and consists of homomorphically evaluating the decryption circuit of the scheme. Roughly speaking, it is like decrypting the ciphertext with the secret key, and then re-encrypting the message, with the difference that the secret key is not known and it is replaced by *an encryption* of the secret key, called the *bootstrapping key*. This requires an additional hardness assumption, called *circular security assumption*, meaning that we must assume it is safe to publish an encryption of the secret key under itself. Although this assumption is still not well studied and understood, and it is sometimes regarded with suspicion, no attacks that exploit this extra piece of information have been proposed.

1.2 FHE in the user-server scenario

Fully homomorphic encryption has huge consequences in the world of delegated computations, as it essentially enables a user to safely hand all of its (encrypted) data to an untrusted remote party, and let it process the information, with the guarantee that the remote party will learn neither the input nor the output of the computation. Practical consequences are fairly easy to see; we present here some simple use cases:

- Emails could be stored encrypted, so that the email provider does not know the content of the messages. Also emails could be searchable, without the provider knowing what a user is looking for.
- Pictures could be uploaded to websites offering image processing capabilities, without the site learning the content of the original picture or that of the final picture.
- Medical data could be safely shared in order to extract statistics or make predictions on one's health condition, without revealing any sensitive information. For example,

in the case of estimating the cost of a life insurance, this could be done by running an algorithm on encrypted data (the applicant's personal information), and returning an encrypted answer (the predicted cost of the insurance policy).

- One could even go so far as to imagining a completely homomorphic search engine, that can process encrypted search queries and return an encrypted list of matches.

The list of potential examples is unsurprisingly long, and the consequences of an introduction of FHE on a large scale would be very serious, with a strong enhancement of user's privacy.

However, it might be worth pointing out something which might not be obvious at a first glance at the subject. While FHE certainly helps protecting user's privacy against a curious server, the fact that the server is no longer able to collect users' data would necessarily produce a change in the business plan of numerous internet companies. These enterprises normally provide services for free, in exchange for the possibility to harvest people's information in order to create tailored advertisement campaigns, extract statistics, ...[3] And while someone might argue that this is wrong on a moral ground and that anything that brings this to a stop is welcome, it is also true that companies need revenues for their survival. Preventing companies from generating profits from users' data would likely force them to adopt different strategies, such as that of making people pay for services that are normally expected to be free (rather, paid for through personal data), like email addresses, storage space, ... This means that, just like any major technology, FHE has potentially serious and life-changing consequences, that have to be evaluated carefully, on the basis of the specific application, the privacy requirements, the costs, etc.

What currently keeps FHE from becoming a widespread tool is essentially efficiency. As we said before, all the solutions we know of for achieving the ability to perform unbounded computations are based on the operation of bootstrapping. Although this procedure has been improved and refined over the years, it remains costly and can considerably limit the efficiency of the scheme. A more viable alternative is that of relying on custom-made SHE instantiations, tailored to the specific application that is targeted. This solution is certainly less flexible, as it has to be adapted on a case-by-case basis, but offers the advantage that no bootstrapping is required. For real-world applications, SHE is usually the chosen approach, and more and more examples of practically efficient SHE are surfacing regularly.

1.3 User and server: different problems for different players

Since the user-server scenario is a world with two players, it is expectable that each of them has a different perspective, different security requirements, and different goals. We analyze them in the following.

1.3.1 The user's point of view

The user is mainly interested in having the privacy of his input data guaranteed by a strong encryption scheme, but is also concerned by the workload he faces in order to encrypt the input and decrypt the output. Another important point regards the communication complexity, or bandwidth usage: the user would like to minimize the amount of data he

[3]As the saying goes, "If you do not pay for the product, then you are the product".

has to send to or download from the Cloud, which implies making ciphertexts as small as possible.

An emerging and fast-growing application that lies within the boundaries of Cloud computing is that of machine-learning-as-a-service (MLaaS), where a user submits data to a remote server, that applies a previously-trained predictive model (e.g., a neural network) to the data. The challenge, in this scenario, is usually represented by the complexity of the computations involved in the homomorphic evaluation of the predictive model, especially in the case of *deep learning*, where the depth of the circuit can be particularly large. For this kind of applications, the main goal is then to improve the efficiency in order to produce accurate results in a fast and reliable way.

1.3.2 The server's point of view

Although the previous part was mainly devoted to the user's needs and how to protect his input data, there can be security requirements from the server's side as well. In fact, a server might provide some data processing capabilities based on proprietary algorithms, that represent a key company asset and constitute sensitive intellectual property. A typical requirement is then that of *circuit privacy* [SYY99; IP07], meaning that the result returned by the server to the user at the end of the computation should leak no information on the algorithm (or circuit) that was applied to the encrypted inputs. In fact, it turns out that the noise term that is added for security purposes and that we mentioned before, not only *grows* with homomorphic operations, but changes in a way that somehow *reflects* (or *depends on*) the operations that are performed. Therefore, the noise contained in the final ciphertext that the user receives can leak information about how the result was produced, therefore jeopardizing the privacy of the algorithm. Instead, we would like the final result to contain no information at all on the computation, and we model this requirement as follows: let x be an input, encrypted as $\mathsf{Enc}(x)$, and let f be a function. Homomorphically evaluating f on $\mathsf{Enc}(x)$ means performing a certain number of homomorphic operations in order to produce a ciphertext that decrypts to $f(x)$. The intuitive goal of circuit privacy is then for the outcome to "look like" a *somewhat fresh* encryption of $f(x)$, which in turn guarantees that the result contains no information on f. The rough idea is that we want to keep the correct result, i.e., $f(x)$, while removing all the information on f.

1.4 Our results

In this manuscript we propose contributions that improve the previous state of the art on both ends of the user-server use case.

1.4.1 A new framework for homomorphic evaluation of neural networks

With respect to the user's point of view, we propose a new framework for efficiently evaluating discretized neural networks on encrypted data. In this framework, the user encrypts his input data under a lattice-based encryption scheme, and sends the encrypted data to the other party. This party is able to homomorphically evaluate a neural network of arbitrary depth on the input, and finally returns the encrypted result. The substantial difference between our work [BMMP18] and previous works is that the instantiation of the encryption schemes, i.e., the parameters that define the scheme, does not have to depend on the network that has to be evaluated. In previous works, the chosen approach was to tailor a somewhat homomorphic encryption scheme to a specific network, with parameters that were good enough for that application. The problem is that this solution is (1) not flexible, and (2) not suited for evaluating deep neural network, that would force to take extremely large and cumbersome parameters for correctness to hold, thus making the scheme completely inefficient and impractical.

On the other hand, our framework heavily relies on a recent and efficient implementation of the bootstrapping operation, through which we can refresh ciphertexts after the evaluation of each layer of the neural network. In turn, this allows us to "keep going with the computations", meaning that there is no a priori bound on the number of layers that can be evaluated. The disadvantage of this approach is that, in order to make our neural networks "FHE-friendly", we have to impose some limitations, namely the inputs and some internal values (weights and biases) have to be discretized. However, we note that this simplified model of neural networks has already been studied in the literature, and is known to achieve near-state-of-the-art performance.

1.4.2 A new technique for circuit privacy

With respect to the server's point of view, in [BPMW16] we propose a conceptually different approach to the problem of circuit privacy that, unlike previous works, is not based on annihilating the noise term by adding another large noise term to it (this technique is called *noise flooding*), or by using bootstrapping in order to "wash away" any trace of the previous computations. Instead, we directly analyze the distribution of the noise term and we show a way of removing the information on the circuit from the ciphertext in a simple and efficient way. In particular, we target the GSW FHE encryption scheme for branching program evaluation, and we introduce a modification that achieves circuit privacy essentially without worsening the overall efficiency of the scheme.

This work also gives better insights into the algebraic structure and the noise distribution in the GSW scheme, and provide new tools for analyzing noise randomization that could be of independent interest.

The fundamental downside of our approach is that it is fundamentally limited to the GSW encryption scheme, while it is not clear how to apply our techniques to other FHE schemes. In fact, although GSW is quite ubiquitous in the FHE world and it represents the most recent and asymptotically best instantiation of FHE, other schemes can be preferable and more efficient for specific applications.

Another limiting problem is that our technique only works in the *honest-but-curious* adversary model, i.e., we have to assume the adversary follows the protocol and play by the rules, but tries to extract as much information as it can from what it sees. This is opposed to the *malicious* adversary model, where the adversary can do whatever it wants and deviate in any way from the prescribed execution, including submitting malformed inputs. In the latter case, our technique breaks, as it relies on well-formed input ciphertexts.

1.4.3 A protocol for private information retrieval

Another contribution of this work is given by [BBB+17] and regards privacy-preserving interactions with a database held by another party. In particular, we devise a protocol for *private information retrieval*, meaning that a party A can query a database held by a party B, without B learning what A is looking for. We also show that, under appropriate conditions and with some formalization caveats, our protocol also achieves the property that A learns nothing more from the database than the records that match the query, thus achieving a stronger notion known as *oblivious transfer*. We also propose a C++ implementation of the protocol that shows the practicality of our approach.

1.5 Other contributions

Apart from input data privacy, Cloud computing introduces another crucial challenge: can we trust the computation done by the Cloud? Can we be sure that the result is indeed the correct one and not some random or forged value? Outsourcing computations is clearly useless without a certain level of confidence in the correctness of the result, which is why we would like the Cloud to be able to "convince us" that the result is indeed valid, but without having to re-run the entire computation step by step.

In zero-knowledge (ZK) proofs (introduced in [GMR89]), a powerful prover can prove to a weaker verifier that a particular statement is true, without revealing *why* this is the case. More formally, given an NP language L with corresponding witness relation $\mathcal{R}$, the prover will know a witness w that demonstrates the truth of a certain statement $x \in L$. With a zero-knowledge proof, the prover will be able to convince the verifier that the statement is indeed true, without revealing any information on the witness w. Non-interactive zero-knowledge proofs [BFM88] and succinct ZK arguments [Kil92; Mic94] were introduced shortly thereafter, but remained of mainly theoretical interest until more recently, when several breakthroughs have shown that these proofs can be used in practical applications (see e.g., Pinocchio [PHGR13]).

A particularly interesting concept is that of SNARK, i.e., succinct non-interactive argument of knowledge. By this term, we denote a proof system which is *non-interactive* (it does not require multiple rounds of communication), *complete* (all correctly generated proofs are accepted), *succinct* (the size of the proof is linear in the security parameter), and *knowledge-sound* (for any prover able to produce a valid proof, there is an algorithm capable of extracting a witness for the statement). Recently, in two companion papers [BISW17; BISW18], Boneh *et al.* provided the first designated-verifier SNARKs construction based on lattice assumptions.

In [GMNO18] we build zero knowledge SNARKs (succinct non-interactive arguments of knowledge) from lattice assumptions for square span programs (SSPs), which are a way to characterize NP introduced in [DFGK14]. Compared with previous works, our result achieves zero-knowledge, relies on weaker assumptions, and is simpler and more efficient. Also, for a reasonable set of parameters, we achieve post-quantum security. In contrast, the size of our proofs is considerably larger than in previous works. Moreover, our construction is designated-verifier, meaning that the verification procedure requires access to some secret piece of information, as opposed to publicly verifiable, where the verification procedure can be run based solely on public information.

1.6 Organization of the manuscript

This manuscript is organized as follows: Chapter 2 introduces the notation used in the manuscript, and gives some definitions and some general notions; Chapter 3 presents fully homomorphic encryption in details, and constitutes a sort of survey on the area; Chapter 4 addresses the problem of homomorphic evaluation of deep neural networks, and presents an efficient framework that allows one to evaluate an already-trained predictive model on encrypted inputs; Chapter 5 considers the other side of the coin for outsourced computations, and examines the problem of circuit privacy, resulting in a new and more efficient way of avoiding undesired leakages from the result of homomorphic computations; Chapter 6 presents the design and the concrete implementation of a protocol based on FHE for private information retrieval; finally, Chapter 7 draws some conclusions and outlines several questions that remain open and that can be the topic for extending the research presented in this manuscript.

Personal Publications

[BPMW16] Florian Bourse, Rafaël del Pino, Michele Minelli, and Hoeteck Wee. "FHE Circuit Privacy Almost for Free". In: *CRYPTO 2016, Part II*. Ed. by Matthew Robshaw and Jonathan Katz. Vol. 9815. LNCS. Springer, Heidelberg, Aug. 2016, pp. 62–89. DOI: 10.1007/978-3-662-53008-5_3 (cit. on p. 9).

[BBB+17] Anthony Barnett, Charlotte Bonte, Carl Bootland, Joppe W. Bos, Wouter Castryck, Anamaria Costache, Louis Goubin, Ilia Iliashenko, Tancrède Lepoint, Michele Minelli, Pascal Paillier, Nigel P. Smart, Frederik Vercauteren, Srinivas Vivek, and Adrian Waller. *Processing Encrypted Data Using Homomorphic Encryption*. Workshop on Data Mining with Secure Computation, SODA project. 2017 (cit. on p. 9).

[BMMP18] Florian Bourse, Michele Minelli, Matthias Minihold, and Pascal Paillier. *Fast Homomorphic Evaluation of Deep Discretized Neural Networks*. CRYPTO 2018. https://eprint.iacr.org/2017/1114. 2018 (cit. on p. 8).

[GMNO18] Rosario Gennaro, Michele Minelli, Anca Nitulescu, and Michele Orrù. *Lattice-Based zk-SNARKs from Square Span Programs*. ACM CCS 2018. https://eprint.iacr.org/2018/275. 2018 (cit. on p. 10).

Chapter 2

Preliminaries

In this chapter, we introduce the notation used in this manuscript, and give some definitions and notions.

We begin with standard notation, then we give basic definitions about cryptographic primitives and introduce the concept of provable security. After this, we give an introduction to lattices, and we present some background notions that will be useful in the rest of the manuscript, including the LWE problem and a brief overview on Gaussian distributions and some of their properties. Finally, we conclude with hardness assumptions that will be used throughout this work.

These preliminaries are inspired and adapted from several PhD theses' preliminaries, such as the ones by Florian Bourse [Bou17], Pierrick Méaux [Méa17], Alain Passelègue [Pas16], Geoffroy Couteau [Cou17], and Fabrice Ben Hamouda–Guichoux [Ben16]. The part on lattices is heavily inspired by Chris Peikert's survey on the area [Pei15a].

Contents

2.1 Notation and preliminaries

The notation used in this work is rather standard in cryptography and computer science: we explicitly define it here for completeness and to make this manuscript self-contained.

2.1.1 Mathematical notation

2.1.1.1 Sets, rings, integers

We denote the set of integers by $\mathbb{Z}$, the set of non-negative integers by $\mathbb{N}$, the set of reals by $\mathbb{R}$ and the set of integers modulo some q by $\mathbb{Z}_q$. We use $\mathbb{T}$ to indicate $\mathbb{R}/\mathbb{Z}$, i.e., the torus of real numbers modulo 1. We denote by $\{0,1\}^n$ a bitstring of length n; sometimes we use $\mathbb{B}$ for $\{0,1\}$. For any ring $\mathcal{R}$, we use $\mathcal{R}[X]$ for polynomials in the variable X with coefficients in $\mathcal{R}$. We use $\mathbb{R}_N[X]$ to denote $\mathbb{R}[X]/\left(X^N+1\right)$, and $\mathbb{Z}_N[X]$ to denote $\mathbb{Z}[X]/\left(X^N+1\right)$, and we write their quotient as $\mathbb{T}_N[X] = \mathbb{R}_N[X]/\mathbb{Z}_N[X]$, i.e., the ring of polynomials in X quotiented by $\left(X^N+1\right)$, with real coefficients modulo 1.

For two integers a, b such that $a < b$, we write $\{a, \ldots, b\}$ to denote the set of integers between a and b (a and b included). If $a = 1$, then we use $[b]$ for $\{1, \ldots, b\}$. For a finite set $\mathcal{S}$, we denote its cardinality by $|\mathcal{S}|$.

For $x \in \mathbb{Z}$, $x \mod q$ denotes the remainder of the Euclidean division of x by q. For $y \in \mathbb{R}$, $|y|$ denotes its absolute value and $\log y$ its logarithm in basis 2, $\lceil y \rceil$ is the smallest integer z such that $y \leq z$, $\lfloor y \rfloor$ is the biggest integer z such that $z \leq y$, and $\lfloor y \rceil$ is the closest integer to y, with ties broken upward.

2.1.1.2 Vectors and matrices

Vectors are denoted by lower-case bold letters, like $\mathbf{v}$, and are always columns. We indicate a vector $\mathbf{v}$'s i-th entry by v_i (not in bold). We use $\mathbf{v}^\intercal$ for the transpose of a vector $\mathbf{v}$, which is then a row. For a vector $\mathbf{v}$, we define the L_p norm as

$$\|\mathbf{v}\|_p = \left(\sum_{i=1}^{n} |v_i|^p\right)^{1/p},$$

and for $p = \infty$, we define $\|\mathbf{v}\|_\infty = \max_i |v_i|$. When we omit the subscript and simply write $\|\mathbf{v}\|$, we refer to the L_2 norm of the vector $\mathbf{v}$. Given two vectors of the same length $\mathbf{a}, \mathbf{b} \in \mathbb{R}^n$, we use either $\langle \mathbf{a}, \mathbf{b} \rangle$ or $\mathbf{a} \cdot \mathbf{b}$ to denote their inner (or dot) product $\sum_{i \in [n]} a_i\, b_i$.

Matrices are denoted by upper-case bold letters, like $\mathbf{A}$, and we use $\mathbf{A}^\intercal$ for the transpose of the matrix $\mathbf{A}$. When a square matrix is invertible, we denote by $\mathbf{A}^{-1}$ its inverse. Sometimes we consider a $n \times m$ matrix as the ordered sequence of its columns: in this case we write $\mathbf{A} = (\mathbf{a}_1, \ldots, \mathbf{a}_m)$. We will also use $\mathbf{0}$ for either the zero-vector or the zero-matrix (it will be clear from the context), and $\mathbf{I}_n$ for the $n \times n$ identity matrix.

For a d_1-dimensional vector $\mathbf{a}$ and a d_2-dimensional vector $\mathbf{b}$, we write $(\mathbf{a}\|\mathbf{b})$ to denote the $(d_1 + d_2)$-dimensional vector obtained by concatenating $\mathbf{a}$ and $\mathbf{b}$. We will use a similar notation for concatenating matrices with matrices or matrices with vectors.

2.1.1.3 Distributions and probabilities

Given a set $\mathcal{S}$, we write $x \leftarrow_{\$} \mathcal{S}$ to indicate that x is sampled uniformly at random from $\mathcal{S}$ (independently of everything else). Similarly, if $\mathcal{D}$ is a probability distribution, then we write $x \leftarrow \mathcal{D}$ to denote that x is sampled according to $\mathcal{D}$. We write $\Pr[X = x]$ to indicate the probability of a random variable X taking the value x. Given two distributions X, Y over a finite or countable domain D, their statistical distance is defined as $\Delta(X, Y) = \frac{1}{2}\sum_{v \in D} |X(v) - Y(v)|$. When it is clear from the context, we sometimes write “random” for “uniformly at random”.

2.1.1.4 Asymptotic behaviors

When f and g are two functions $\mathbb{N} \to \mathbb{R}$, we write $f = \mathcal{O}(g)$ to indicate that there exist a constant c and an integer $n_0 \in \mathbb{N}$ such that for any integer $n \geq n_0$, $|f(n)| \leq c \cdot |g(n)|$. We say that a function $\varepsilon : \mathbb{N} \to [0,1]$ is *negligible* (or $1 - \varepsilon$ is *overwhelming*) if, for any constant $c \in \mathbb{N}$, there exists $\eta_0 \in \mathbb{N}$ such that for any $\eta \geq \eta_0$, $\varepsilon \leq \frac{1}{\eta^c}$.

2.1.2 Algorithms

For simplicity, unless otherwise stated, we consider all the algorithms as probabilistic Turing machines. This means that we implicitly assume that they can use an additional tape containing random bits (also referred to as random coins). In the rest of this thesis, we use the term “PPT algorithm” for “Probabilistic Polynomial-Time algorithm”, and we say an algorithm is *efficient* if it is a PPT algorithm. For an algorithm $\mathcal{A}$, we write $y \leftarrow \mathcal{A}(x)$ to denote the fact that we run $\mathcal{A}$ on input x with fresh random coins and we obtain an output y. In case $\mathcal{A}$ is a deterministic algorithm, we will instead write $y = \mathcal{A}(x)$.

2.1.3 Provable security

Almost any problem can be solved if enough computational power or enough time is available, e.g., by enumerating and trying every possible solution until finding the right one (this approach is known as *brute-force attack*). With this in mind, the goal of provable security is then to assess the amount of effort required to solve a certain problem (i.e., break a certain system). A cryptosystem is deemed “secure” if breaking it would require an unreasonable effort (with regards to computations, elapsed time, data storage, ...) from a reasonable attacker. The choice of what is reasonable depends on the security model (e.g., what kind of adversary we are considering), and on the security guarantees that we require[1]. These considerations are numerically represented by a *security parameter*, that we denote with κ: a cryptosystem provides κ bits of security if it requires 2^κ elementary operations to be broken. Although usually omitted in order to ease notation, all the integers, vectors, and matrices that we define will implicitly be a function of κ. However, sometimes we explicitly write that an algorithm (e.g., Setup) takes 1^κ as an argument. The reason for doing this is that we want the algorithm to run in a time that is polynomial in κ, so its input must have size κ. The adversaries we consider in this work are PPT algorithms and we denote an adversary’s advantage when attacking a scheme, a protocol, or a generic construction, by Adv.

[1]For example, spending a year to decrypt a friend’s message could be pointless, but spending a year to decipher a valuable industrial or military secret might be perfectly acceptable.

2.2 Cryptographic primitives

In this part we give a basic introduction to secret-key and public-key encryption schemes, outline some generic algorithms and provide some definitions related to the security of encryption schemes.

Definition 2.2.1 (Secret-key encryption scheme)**.** *A secret-key encryption scheme consists of the following algorithms:*

- $\mathsf{KeyGen}\,(1^\kappa) \mapsto (\mathsf{k}, pp)$*: on input the security parameter* κ*, outputs the key* k *and some public parameters* pp*;*
- $\mathsf{Encrypt}\,(\mathsf{k}, m) \mapsto c$*: on input the key* k *and a message* m*, outputs a ciphertext* c*;*
- $\mathsf{Decrypt}\,(\mathsf{k}, c) \mapsto m$*: on input the key* k *and a ciphertext* c*, outputs a message* m*.*

While in a secret-key encryption scheme *the same key* is used both for encrypting and decrypting, in a public-key encryption scheme there are two separate keys, a public and a private (or secret) one, which are used for encrypting and decrypting, respectively.

Definition 2.2.2 (Public-key encryption scheme)**.** *A public-key encryption scheme consists of the following algorithms:*

- $\mathsf{KeyGen}\,(1^\kappa) \mapsto (\mathsf{pk}, \mathsf{sk}, pp)$*: on input the security parameter* κ*, outputs the public key* pk*, the secret key* sk*, and some public parameters* pp*;*
- $\mathsf{Encrypt}\,(\mathsf{pk}, m) \mapsto c$*: on input the public key* pk *and a message* m*, outputs a ciphertext* c*;*
- $\mathsf{Decrypt}\,(\mathsf{sk}, c) \mapsto m$*: on input the secret key* sk *and a ciphertext* c*, outputs a message* m*.*

We now give a standard definition of security for an encryption scheme, namely that of *indistinguishability under chosen-plaintext attacks* (IND-CPA).

Definition 2.2.3 (Indistinguishability under chosen-plaintext attacks)**.** *For a public-key encryption scheme* $\mathcal{E}$*, we define* IND-CPA *security via the game depicted in Figure 2.1. Let*

$$\mathsf{Adv}_{\mathcal{A}}^{\text{IND-CPA}}(\kappa) = \left|\Pr\left[\text{IND-CPA}_{\mathcal{E},\kappa}\right] - \frac{1}{2}\right|$$

be the advantage of an adversary $\mathcal{A}$ *when playing the game* $\text{IND-CPA}_{\mathcal{E},\kappa}$*. We say that* $\mathcal{E}$ *is* IND-CPA *secure if, for any PPT adversary* $\mathcal{A}$*, it holds that* $\mathsf{Adv}_{\mathcal{A}}^{\text{IND-CPA}}(\kappa) = \mathsf{negl}\,(\kappa)$*.*

```
Game IND-CPA$_{\mathcal{E},\kappa}$
$(\mathsf{pk}, \mathsf{sk}, pp) \leftarrow \mathsf{KeyGen}(1^\kappa)$
$(m_0, m_1) \leftarrow \mathcal{A}(1^\kappa, \mathsf{pk}, pp)$
$b \leftarrow_\$ \{0,1\}$
$c \leftarrow \mathcal{E}.\mathsf{Encrypt}(\mathsf{pk}, m_b)$
$b' \leftarrow \mathcal{A}(1^\kappa, \mathsf{pk}, pp, c)$
return $b' = b$
```

Figure 2.1: Game for IND-CPA security.

2.3 Lattices

In this part we present some basic definitions about lattices and we recall several fundamental computational problems.

2.3.1 Basic definitions

Definition 2.3.1 (Lattice)**.** *An n-dimensional lattice Λ is a discrete additive subgroup of $\mathbb{R}^n$. For an integer $k < n$ and a rank k matrix $\mathbf{B} \in \mathbb{R}^{n \times k}$, $\Lambda(\mathbf{B}) = \left\{\mathbf{Bx} \in \mathbb{R}^n \mid \mathbf{x} \in \mathbb{Z}^k\right\}$ is the lattice generated by the columns of $\mathbf{B}$. The columns of $\mathbf{B}$ form a* basis *of the lattice.*

In this work, we are only interested in full-rank lattices, i.e., those for which $k = n$

Remark 2.3.2. *A lattice basis $\mathbf{B}$ is never unique: for any unimodular matrix $\mathbf{U} \in \mathbb{Z}^{n \times n}$ (i.e., such that $\det(\mathbf{U}) = \pm 1$), the matrix $\mathbf{B} \cdot \mathbf{U}$ is also a basis of $\Lambda(\mathbf{B})$. In Figure 2.2 we show a two-dimensional lattice with two different bases.*

Definition 2.3.3 (Minimum distance)**.** *The* minimum distance *of a lattice Λ is the length of a shortest nonzero lattice vector:*

$$\lambda_1(\Lambda) := \min_{\mathbf{0} \neq \mathbf{v} \in \Lambda} \|\mathbf{v}\|$$

Definition 2.3.4 (Successive minima)**.** *Given a lattice Λ, its i-th successive minimum $\lambda_i(\Lambda)$ is the smallest positive $\ell \in \mathbb{R}$ such that Λ has i linearly independent vectors of norm at most ℓ.*

Definition 2.3.5 (Fundamental parallelepiped)**.** *For any lattice basis $\mathbf{B}$, we define*

$$\mathcal{P}(\mathbf{B}) = \{\mathbf{Bx} : \mathbf{x} \in \mathbb{R}^n, 0 \leq x_i < 1 \ \forall i\}$$

Definition 2.3.6 (Determinant of a lattice)**.** *Let $\Lambda(\mathbf{B})$ be a lattice with rank n. We define the determinant of Λ, denoted as $\det(\Lambda)$, as the n-dimensional volume of its fundamental parallelepiped $\mathcal{P}(\mathbf{B})$. In symbols, we can write this as $\det(\Lambda) := \sqrt{\det(\mathbf{B}^\mathsf{T}\mathbf{B})}$.*

Definition 2.3.7 (Dual and orthogonal lattice)**.** *The* dual *of a lattice $\Lambda \subset \mathbb{R}^n$ is defined as*

$$\Lambda^* := \{\mathbf{v} : \langle \mathbf{v}, \Lambda \rangle \subseteq \mathbb{Z}\},$$

i.e., the set of points whose inner products with the vectors in Λ are all integers. It is easy to verify that the dual of a lattice is still a lattice. Given a rank k matrix $\mathbf{B} \in \mathbb{R}^{n \times k}$, we define the q-ary orthogonal *of a lattice Λ as*

$$\Lambda_q^\perp(\mathbf{B}) := \{\mathbf{v} \in \mathbb{Z}^n : \mathbf{B}^\mathsf{T}\mathbf{v} = \mathbf{0} \mod q\}.$$

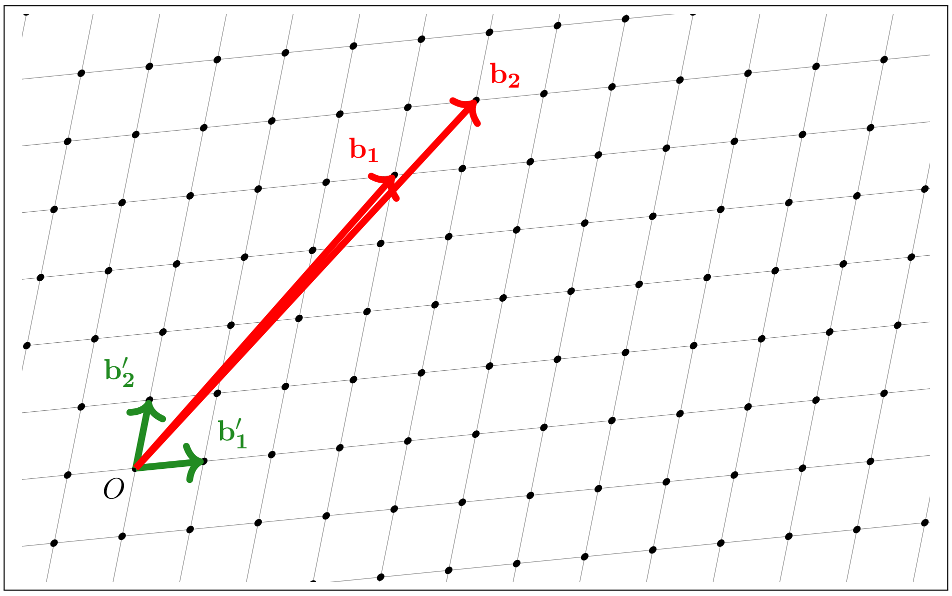


Figure 2.2: A two-dimensional lattice with two different bases.

2.3.2 Computational problems

In this part we recall several classical computational problems on lattices.

Definition 2.3.8 (Shortest Vector Problem (SVP)). *Given an arbitrary basis* $\mathbf{B}$ *of some lattice* $\Lambda = \Lambda(\mathbf{B})$, *find a shortest*[2] *nonzero lattice vector, i.e., a vector* $\mathbf{v} \in \Lambda$ *such that* $\|\mathbf{v}\| = \lambda_1(\Lambda)$.

In the following, we present several approximation problems, parameterized by an approximation factor $\gamma \geq 1$, usually taken as a function of the lattice dimension n, i.e., $\gamma = \gamma(n)$.

Definition 2.3.9 (Approximate Shortest Vector Problem (SVP_γ)). *Given an arbitrary basis* $\mathbf{B}$ *of a lattice* $\Lambda = \Lambda(\mathbf{B})$, *find a nonzero vector* $\mathbf{v} \in \Lambda$ *such that* $\|\mathbf{v}\| \leq \gamma \cdot \lambda_1(\Lambda)$.

Definition 2.3.10 (Decisional Approximate SVP (GapSVP_γ)). *Given an arbitrary basis* $\mathbf{B}$ *of a lattice* $\Lambda = \Lambda(\mathbf{B})$, *where* $\lambda_1(\Lambda) \leq 1$ *or* $\lambda_1(\Lambda) > \gamma$, *determine which is the case.*

Definition 2.3.11 (Approximate Shortest Independent Vector Problem (SIVP_γ)). *Given an arbitrary basis* $\mathbf{B}$ *of an* n*-dimensional lattice* $\Lambda = \Lambda(\mathbf{B})$, *output a set* $\mathcal{S} = \{\mathbf{v}_1, \ldots, \mathbf{v}_n\} \subset \Lambda$ *of* n *linearly independent lattice vectors such that* $\|\mathbf{v}_i\| \leq \gamma \cdot \lambda_i(\Lambda)$ *for all* $i \in [n]$.

Definition 2.3.12 (Approximate Closest Vector Problem (CVP_γ)). *Given an arbitrary basis* $\mathbf{B}$ *of an* n*-dimensional lattice* $\Lambda = \Lambda(\mathbf{B})$ *and a target vector* $\mathbf{t} \in \mathbb{R}^n$, *find a vector* $\mathbf{v} \in \Lambda$ *such that* $0 < \|\mathbf{v} - \mathbf{t}\| \leq \gamma \cdot \mathsf{D}(\mathbf{t}, \Lambda) = \gamma \cdot \inf_{\mathbf{x} \in \Lambda} \|\mathbf{x} - \mathbf{t}\|$.

Definition 2.3.13 (Approximate Bounded Distance Decoding Problem (BDD_γ)). *Given an arbitrary basis* $\mathbf{B}$ *of an* n*-dimensional lattice* $\Lambda = \Lambda(\mathbf{B})$ *and a target vector* $\mathbf{t} \in \mathbb{R}^n$ *such that* $\mathsf{D}(\mathbf{t}, \Lambda) \leq \gamma^{-1} \cdot \lambda_1(\Lambda)$, *find a vector* $\mathbf{v} \in \Lambda$ *such that* $\|\mathbf{v} - \mathbf{t}\| = \mathsf{D}(\mathbf{t}, \Lambda)$.

[2]Note that we do not ask for "the shortest", as a shortest vector is never unique. For a vector $\mathbf{v}$ that satisfies the condition, the lattice vector $-\mathbf{v}$ does as well.

2.3.3 Worst-case hardness

For a problem to be useful in cryptography, we require it to be hard *on average*, i.e., a *random instance* of this problem should be hard to solve. Instead, a more common definition of hardness is related to the *worst case*, i.e., a problem is hard if *some instances* of this problem are hard to solve. In [Ajt96], Ajtai gave a connection between these two situations and proved that there are cryptographically useful problems that are hard on average, as long as certain lattice problems are hard in the worst case. This essentially makes implementing cryptosystems considerably easier, as one can just pick a random instance of the problem and be reasonably sure that this instance is hard. On the other hand, this does not happen with many number theory problems, where implementing requires checking that a specific instance of the problem is hard to break.

2.3.4 Gaussians

In this part, we introduce some notions about discrete Gaussian distributions that we will use throughout this work.

Definition 2.3.14 (Gaussian function and discrete distribution)**.** *For any $\alpha > 0$ the spherical Gaussian function with parameter α (omitted if 1) is defined as*

$$\rho_\alpha(\mathbf{x}) \coloneqq \exp\left(\frac{-\pi\,\|\mathbf{x}\|}{\alpha^2}\right)$$

for any $\mathbf{x} \in \mathbb{R}^n$. Given a lattice $\Lambda \subseteq \mathbb{R}^n$, a parameter $r \in \mathbb{R}$, and a vector $\mathbf{c} \in \mathbb{R}^n$, the spherical Gaussian distribution with parameter r and support $\Lambda + \mathbf{c}$ is defined as

$$\mathcal{D}_{\Lambda+\mathbf{c},r}(\mathbf{x}) \coloneqq \frac{\rho_r(\mathbf{x})}{\rho_r(\Lambda + \mathbf{c})}, \quad \forall \mathbf{x} \in \Lambda + \mathbf{c}$$

where $\rho_r(\Lambda + \mathbf{c})$ denotes $\sum_{\mathbf{v}\in\Lambda+\mathbf{c}} \rho_r(\mathbf{v})$.

Remark 2.3.15. *Note that $\rho_r(\mathbf{x}) = \rho(r^{-1}\mathbf{x})$.*

Definition 2.3.16 (Sub-Gaussian distribution)**.** *A distribution $\mathcal{D}$ is sub-Gaussian with parameter α if there exists $M > 0$ such that for all $\mathbf{x} \in \mathbb{R}^n$,*

$$\mathcal{D}(\mathbf{x}) \leq M \cdot \rho_\alpha(\mathbf{x}).$$

We now report a very well known result about additivity of Gaussian distributions.

Lemma 2.3.17 (Pythagorean additivity of Gaussians)**.** *Let $\mathcal{D}_1$ and $\mathcal{D}_2$ be Gaussian distributions with parameters σ_1 and σ_2, respectively. Then $\mathcal{D}^+$, obtained by sampling $\mathcal{D}_1$ and $\mathcal{D}_2$ and summing the results, is a Gaussian with parameter $\sqrt{\sigma_1^2 + \sigma_2^2}$.*

Another important quantity for a lattice Λ is its *smoothing parameter.* Roughly speaking, this can be seen as the minimum amount of Gaussian "blur" required to "smooth out" all the discrete structure of Λ. We now give a more formal definition.

Definition 2.3.18 (Smoothing parameter [MR04])**.** *For a lattice $\Lambda \subseteq \mathbb{R}^n$ and a positive real $\varepsilon > 0$, the* smoothing parameter *$\eta_\varepsilon(\Lambda)$ is the smallest real $r > 0$ such that $\rho_{1/r}(\Lambda^* \setminus \{\mathbf{0}\}) \leq \varepsilon$, where Λ^* is defined as per Definition 2.3.7.*

2.3.5 Short integer solution (SIS)

The short integer solution problem was first introduced in the work by Ajtai [Ajt96], and has served as the foundation for one-way functions, collision-resistant hash functions, and other primitives.

Definition 2.3.19 (The SIS problem)**.** *The short integer solution problem is parametrized by four integers: the lattice dimension n, the number of samples m, the modulus q, and the constraint β.*

Given m uniformly random vectors $\mathbf{a}_i \in \mathbb{Z}_q^n$, forming the columns of a matrix $\mathbf{A} \in \mathbb{Z}_q^{n\times m}$, find a nonzero integer vector $\mathbf{z} \in \mathbb{Z}^m$ of norm $\|\mathbf{z}\| \leq \beta$ such that

$$\mathbf{A}\mathbf{z} = \sum_i \mathbf{a}_i z_i = \mathbf{0} \in \mathbb{Z}_q^n.$$

Note that it must be $\beta < q$, otherwise the vector $(q, 0, \dots, 0) \in \mathbb{Z}_q^n$ would always be a valid solution. On the other hand, β must be large enough that a solution is guaranteed to exist. For example, let $m = n \log q$. Then, $\beta \geq \sqrt{m}$ is enough to have this guarantee. In fact, there are more than q^n vectors in $\{0,1\}^m$, so there must be two distinct vectors $\mathbf{x}, \mathbf{x}' \in \{0,1\}^m$ such that $\mathbf{A}\mathbf{x} = \mathbf{A}\mathbf{x}'$. In turn, this means that $\mathbf{A}(\mathbf{x} - \mathbf{x}') = \mathbf{0} \in \mathbb{Z}_q^n$. The conclusion is that the vector $\mathbf{y} = \mathbf{x} - \mathbf{x}' \in \{-1, 0, 1\}^m$ is a solution of norm at most $\sqrt{m} = \beta$. By the same argument, we can also conclude that the function

$$f_{\mathbf{A}} : \{0,1\}^m \to \mathbb{Z}_q^n$$
$$f_{\mathbf{A}}(\mathbf{x}) := \mathbf{A}\mathbf{x} = \sum_i \mathbf{a}_i x_i \in \mathbb{Z}_q^n$$

is collision-resistant, assuming the hardness of the corresponding SIS problem. The reason is immediately clear: finding a collision means finding two distinct vectors $\mathbf{x}, \mathbf{x}' \in \{0,1\}^m$ such that $\mathbf{A}\mathbf{x} = \mathbf{A}\mathbf{x}'$, but this immediately yields the solution $\mathbf{x} - \mathbf{x}'$ to the SIS problem for the matrix $\mathbf{A}$.

2.3.6 Learning with errors (LWE)

The learning with errors (LWE) problem was introduced by Regev in [Reg05], and has become one of the most known problems in lattice-based cryptography. It has been used to construct several cryptosystems, and it is believed to be hard even for quantum computers. This problem comes in two flavors, search and decision: we present them both in the following.

LWE is parameterized by two positive integers, n and q, and an error distribution χ over $\mathbb{Z}$, usually a discrete Gaussian of width αq, $0 < \alpha < 1$.

Definition 2.3.20 (LWE distribution)**.** *Given a secret vector $\mathbf{s} \in \mathbb{Z}_q^n$, the LWE distribution $\mathsf{lwe}_{\mathbf{s},\chi}$ over $\mathbb{Z}_q^n \times \mathbb{Z}_q$ is sampled by picking $\mathbf{a} \leftarrow_{\$} \mathbb{Z}_q^n$, an error $e \leftarrow \chi$, and returning $(\mathbf{a}, b = \langle \mathbf{s}, \mathbf{a} \rangle + e)$.*

We now present the two versions of the LWE problem.

Definition 2.3.21 (Search-LWE $\mathsf{slwe}_{n,q,\chi,m}$)**.** *Given m independent samples $(\mathbf{a}_i, b_i) \leftarrow \mathsf{lwe}_{s,\chi}$, for a fixed $\mathbf{s} \leftarrow_{\$} \mathbb{Z}_q^n$, find $\mathbf{s}$.*

Definition 2.3.22 (Decisional-LWE $\mathsf{dlwe}_{n,q,\chi,m}$)**.** *Given m independent samples $(\mathbf{a}_i, b_i) \in \mathbb{Z}_q^n \times \mathbb{Z}_q$, where every sample is either distributed according to $\mathsf{lwe}_{\mathbf{s},\chi}$ for some fixed $\mathbf{s} \in \mathbb{Z}_q^n$ or uniformly random in $\mathbb{Z}_q^n \times \mathbb{Z}_q$, distinguish which is the case.*

2.3.6.1 Ring LWE

A version of the LWE problem over rings was introduced in [SSTX09; LPR10]. We give here the definition without fractional ideals.

Definition 2.3.23 (Ring-LWE distribution)**.** *Let $\mathcal{R}$ be a polynomial ring such that $\mathcal{R} = \mathbb{Z}[X]/f(X)$, with f some cyclotomic polynomial of degree n. Also, let $q \geq 2$ be an integer modulus, and let $\mathcal{R}_q = \mathcal{R}/q\mathcal{R}$ be the quotient ring. Finally, let χ be an error distribution over $\mathcal{R}$. For a fixed secret $s \in \mathcal{R}_q$, the ring-LWE distribution $\mathsf{ring\text{-}lwe}_{s,\chi}$ is sampled by taking $a \leftarrow\!\!\$\, \mathcal{R}_q$, $e \leftarrow \chi$, and outputting $(a, b = s \cdot a + e)$. All the computations are, as usual, done modulo q.*

Analogously to what was done for LWE, we now present the two versions of the ring-LWE problem.

Definition 2.3.24 (Search ring-LWE)**.** *Given m independent samples $(a, b) \leftarrow \mathsf{ring\text{-}lwe}_{s,\chi}$, for a fixed $s \leftarrow\!\!\$\, \mathcal{R}$, find s.*

Definition 2.3.25 (Decisional ring-LWE)**.** *Given m independent samples $(a, b) \in \mathcal{R} \times \mathcal{R}$, where every sample is either distributed according to $rlwe_{s,\chi}$ for some fixed $s \in \mathcal{R}$ or uniformly random in $\mathcal{R} \times \mathcal{R}$, distinguish which is the case.*

The advantage of using ring-LWE instead of plain LWE is compactness and efficiency. In fact, in the case of ring-LWE, each sample gives a n-dimensional pseudorandom ring element $b \in \mathcal{R}$, instead of just a pseudorandom scalar $b \in \mathbb{Z}_q$. We can thus say that a single ring-LWE sample with $a \in \mathcal{R}$ takes the place of n LWE samples with vectors $\mathbf{a}_i \in \mathbb{Z}_q^n$. Moreover, thanks to techniques like FFT, the multiplication between ring elements can be performed in quasi-linear time. The essential drawback of ring-LWE is its conjectured hardness, which is not as well-established as for LWE. We will not give details about this, but refer the interested reader to specialized literature (e.g., [APS15; BF17]). However, results on ring-LWE hardness are not conclusive and the question stands wide open, with some prominent figures in the community like Dan Bernstein notoriously expressing concerns (see e.g., [Ber14]), and other famous researchers like Chris Peikert, Damien Stehlé, Vadim Lyubashevsky, and Léo Ducas usually more skeptical and less convinced that disastrous attacks really exist[3]. In the past, this topic has even generated heated discussions (cf. e.g., [BPL+15; BPDS16]), showing that the questions related to ideal lattice cryptography and its security are very important ones and need more attentive investigation.

2.3.6.2 Link between LWE and lattice-based problems

We now clarify the link between LWE and the lattice problems presented in Section 2.3.2.

Let $\mathbf{A} \leftarrow\!\!\$\, \mathbb{Z}_q^{n \times m}$, and let Λ be the following lattice

$$\Lambda = \Lambda(\mathbf{A}) := \left\{ \mathbf{A}^\intercal \mathbf{v} : \mathbf{v} \in \mathbb{Z}_q^n \right\} + q\mathbb{Z}^m$$

This is an m-dimensional q-ary lattice, since $\mathbf{A}^\intercal$ has m rows and the lattice is defined modulo the integer q.

[3]For example, in [Pei15b], Chris Peikert wrote "Healthy skepticism is always warranted (especially in cryptography), but based on these new developments, I don't believe our level of skepticism should rise much from where it was before."

Let $\mathbf{s} \leftarrow\!\!\$\, \mathbb{Z}_q^n$, let χ be an LWE error distribution, $\mathbf{e} \leftarrow \chi^m$, and $\mathbf{b} = \mathbf{A}^\intercal \mathbf{s} + \mathbf{e} \in \mathbb{Z}_q^m$. Now it is easy to see that, for sufficiently small error terms, the vector $\mathbf{b}$ is rather close to a specific vector (or point) of the lattice Λ, namely $\mathbf{A}^\intercal \mathbf{s}$, whereas a random vector $\mathbf{u} \leftarrow\!\!\$\, \mathbb{Z}_q^m$ will be far from Λ with high probability. We can then conclude that solving search-LWE amounts to solving an average instance of the BDD problem on the lattice Λ. In fact, once the vector $\mathbf{A}^\intercal \mathbf{s} \in \Lambda$ has been found, $\mathbf{s}$ can be trivially recovered, e.g., by Gaussian elimination.

2.4 Complexity assumptions

In this subsection, we list the complexity assumptions we rely on in this manuscript.

Assumption 2.4.1 (Hardness of lattice problems)**.** *It is hard to solve the lattice problems presented in Section 2.3.2.*

This assumption simply says that the lattice problems presented in Section 2.3.2, despite having been studied intensively over the years, are intractable, except for very large approximation factors γ. We currently know of some polynomial-time algorithms, like the famous LLL algorithm by Lenstra, Lenstra, and Lovász [LLL82], that obtain only slightly subexponential approximation factors $\gamma(n) = 2^{\Theta(n \log\log n/\log n)}$ for all the problems stated in Section 2.3.2. When requiring polynomial (meaning $\mathsf{poly}(n)$) or better factors γ, the algorithms we currently know either require superexponential $2^{\Theta(n \log n)}$ time or exponential $2^{\Theta(n)}$ time and space. Between these two extremes, there are intermediate solutions that achieve $\gamma = 2^k$ approximations factors in $2^{\tilde{\Theta}(n/k)}$ time [Sch87].

Interestingly, this is also the state of the art for *quantum* algorithms, although some constant factors might be smaller. In a nutshell, quantum attacks on lattice problems do not achieve anything beyond generic quantum speedups, which justifies the following assumption.

Assumption 2.4.2 (*Quantum* hardness of lattice problems)**.** *It is hard to solve the problems presented in Section 2.3.2, even with a quantum computer.*

Most of the constructions presented in this manuscript will be based on the decisional LWE (dlwe) problem. In [Reg05], Regev proved the following theorem on the hardness of dlwe:

Theorem 2.4.3 (Hardness of dlwe)**.** *For any $m = \mathsf{poly}(n)$, any modulus $q \leq 2^{\mathsf{poly}(n)}$, and any (discretized) Gaussian error distribution χ of parameter $\alpha q \geq 2\sqrt{n}$ (where $0 < \alpha < 1$), solving the $\mathsf{dlwe}_{n,q,\chi,m}$ problem is at least as hard as quantumly solving GapSVP_γ and SIVP_γ on arbitrary n-dimensional lattices, for some $\gamma = \widetilde{\mathcal{O}}(n/a)$.*

This justifies the following assumption:

Assumption 2.4.4 (dlwe)**.** *For a correct choice of parameters, it is hard to break the* dlwe *problem defined in Definition 2.3.22.*

Finally, we will use an assumption on the so-called *approximate GCD* problem: we now present the problem and the related assumption. The approximate-GCD problem (AGCD) was introduced in 2001 by Howgrave-Graham [How01] and is parametrized by integers $\gamma, \eta, \rho \in \mathbb{N}$.

Definition 2.4.5 (Approximate GCD distribution)**.** *The AGCD distribution* $\mathsf{agcd}_\rho(p, q_0)$ *for a given* η*-bit integer* p *and a* $q_0 \leftarrow_{\$} [0, 2^\gamma/p)$ *is the set of integers* $x_i = p \cdot q_i + r_i$*, where* $q_i \leftarrow_{\$} [0, q_0)$ *and* $r_i \leftarrow_{\$} [0, 2^\rho)$*.*

Definition 2.4.6 (Approximate GCD problem)**.** *For a* η*-bit integer* p *and a uniformly chosen* $q_0 \leftarrow_{\$} [0, 2^\gamma/p)$*, given polynomially many samples from* $\mathsf{agcd}_\rho(p, q_0)$*, find* p*.*

Assumption 2.4.7 (Hardness of the approximate GCD problem)**.** *For appropriately set parameters* $\gamma, \eta, \rho \in \mathbb{N}$*, it is hard to break the AGCD problem of Definition 2.4.6.*

For a review on several kind of attacks to this problem, we refer the reader to, e.g., [How01; CH11; CMNT11; CNT12; CN12].

Chapter 3

Fully homomorphic encryption

In this chapter we present fully homomorphic encryption in details, recalling its history from the first plausible construction, through successive improvements, to today's "almost practical" instantiations. We also go through several homomorphic cryptosystems that have been proposed, analyzing their pros and cons, in hope to give a sort of "bird's eye view" on this field. Some of the contents about this part are inspired by the PhD thesis of Tancrède Lepoint [Lep14]. We will conclude this chapter with a part on implementation, reviewing some libraries that have been developed over the years, and assessing their performances, their potential, and their limitations.

The idea is for this chapter to be of independent interest as a sort of survey on FHE.

Contents

3.1 Introduction

Homomorphic encryption is a kind of encryption that allows one to perform operations on ciphertexts based solely on publicly available information, and in particular without having access to any secret key. The reason why it is called "homomorphic" is that, roughly speaking, there exists a correspondence between the space of the messages and the space of the ciphertexts, in such a way that operations performed on ciphertexts are somehow reflected in operations on the messages they encrypt. For example, an *additively homomorphic* encryption scheme allows anyone to take a ciphertext c_1 encrypting a message m_1, a ciphertext c_2 encrypting a message m_2, and produce a ciphertext c_+ that decrypts to $m_1 + m_2$. Analogously, a *multiplicatively homomorphic* encryption scheme allows one to produce a ciphertext $c_\times$ that decrypts to $m_1 \cdot m_2$. And this is possible *without having access* to either m_1 or m_2 or any secret information. We say that an encryption scheme is *partially* homomorphic if it supports only some operations, but not others: for example, a scheme could support homomorphic addition but not multiplication.

In most of the homomorphic encryption schemes known to date, an error term is injected during the encryption procedure for security purposes. The reason is that these encryption schemes rely on the hardness of solving "noisy" problems, i.e., problems where the relations are not exact, but are perturbed by a moderate quantity of error. Combining multiple ciphertexts through homomorphic operations has the side effect of combining the noises as well, thus increasing the magnitude of the error in the resulting encryption. When the error grows beyond a certain threshold, correctness is lost, meaning that the decryption procedure will not return the expected result. We say that an encryption scheme is *somewhat* homomorphic if it can evaluate a certain number of homomorphic operations, before the error grows too much to maintain the correctness of the evaluation.

When practically instantiating an encryption scheme, one of the most delicate steps is that of choosing parameters. This usually requires finding a sensible trade-off between security and efficiency, and remains one of the potentially weak points for theoretically secure schemes. This is also the case for homomorphic encryption schemes, but in this case the problem of finding parameters is even more important. In fact, parameters like the size of the modulus or the standard deviation of noise terms define the threshold below which the noise must remain in order to guarantee a correct decryption. By doing so, these parameters control how many homomorphic operations can be carried out. We say that an encryption scheme is *leveled* homomorphic if, for any multiplicative depth L fixed a priori, we can find a set of parameters such that the encryption scheme instantiated with these parameters can evaluate any circuit with multiplicative depth equal to L. We stress that, in this setting, the bound on the number of operations has to be known at setup time, i.e., when setting the parameters. This means that the scheme will tend to be tailored for a specific application, rather than being a generic tool for evaluating any function.

On the other hand, a *fully* homomorphic encryption (FHE) scheme is a homomorphic scheme that allows for the evaluation of arbitrarily complex computations over encrypted data. The problem of designing such scheme was suggested by Rivest, Adleman and Dertouzos in 1978 [RAD78] but, despite moderate progress [GM82; Pai99; BGN05; IP07], it remained the "Holy Grail of cryptography" (cit. [Mic10]) until the breakthrough result of Gentry in 2009 [Gen09b]. In the case of FHE, there is no need to set an a priori bound on the number of homomorphic operations, thus making the scheme more flexible. In contrast, FHE schemes tend to be considerably less efficient than leveled ones, which, for specific applications, can be noticeably faster and, as a consequence, more appealing. In a nutshell, the matter boils down to a trade-off between flexibility on one side and efficiency/optimization on the other side.

Finally, we note that it is not rare to see in the literature leveled homomorphic encryption being called *somewhat* homomorphic encryption. Although the meaning is usually clear from the context, a somewhat homomorphic encryption scheme denotes a scheme with limited homomorphic properties, for example one that can only evaluate additions or multiplications.

Over the years, homomorphic encryption has been a very active field for research, and this has led to a long list of works and improvements, (e.g., [DGHV10; SS10; SV10; BV11a; BV11b; BGV12; GHS12; GSW13; BV14; AP14]). This comprised both theoretical works that put forth new ideas and constructions, and implementation works that optimized existing constructions with the goal of achieving the best possible efficiency.

The FHE constructions that we know of can roughly be divided into three groups, usually referred to as *generations*. "first generation" FHE usually denotes the one stemming directly from Gentry's seminal work [Gen09b] and based on ideal lattices and the approximate GCD problem; "second generation" usually indicates constructions proposed in a sequence of works by Brakerski and Vaikuntanathan [BV11a; BV11b] and based on the LWE problem; "third generation" usually denotes the GSW FHE scheme by Gentry, Sahai, and Waters [GSW13] and subsequent works (e.g., [AP14; HAO15]).

In the following, we review all three generations and try to explain the main ideas behind them and the principal changes and improvements when moving from one generation to another. Before doing so, we give some definitions and a high-level introduction to a key technique called *bootstrapping*, that remains at the very core of all currently known FHE schemes.

3.2 Homomorphic encryption scheme

A homomorphic (secret-key) encryption scheme $\mathcal{E}$ with message space $\mathcal{M}$ is composed of the following algorithms:

$\mathcal{E}.\mathsf{KGen}(1^\kappa) \to (\mathsf{sk}, \mathsf{evk})$ **:** given the security parameter, output a secret key sk and an evaluation key evk.

$\mathcal{E}.\mathsf{Enc}(\mathsf{sk}, \mu) \to c$ **:** given the secret key sk and a message $\mu \in \mathcal{M}$, output a ciphertext c.

$\mathcal{E}.\mathsf{Dec}(\mathsf{sk}, c) \to \mu$ **:** given the secret key sk and a ciphertext c, output a message $\mu \in \mathcal{M}$.

$\mathcal{E}.\mathsf{Eval}(\mathsf{evk}, f, c_1, \ldots, c_\ell)$ **:** given the evaluation key evk, a function $f : \mathcal{M}^\ell \to \mathcal{M}$ and ciphertexts $c_1, \ldots, c_\ell$, apply the function f on ciphertexts c_i and output a ciphertext c_f.

We now give a fundamental definition for any FHE scheme, i.e., evaluation correctness.

Definition 3.2.1 (Evaluation correctness)**.** *We say that the* $\mathcal{E}.\mathsf{Eval}$ *algorithm correctly evaluates all functions in* $\mathcal{F}$ *if, for any function* $f \in \mathcal{F} : \mathcal{M}^\ell \to \mathcal{M}$ *and respective inputs* $x_1, \ldots, x_\ell$*, it holds that*

$$\Pr\left[\mathcal{E}.\mathsf{Dec}\left(\mathsf{sk}, \mathcal{E}.\mathsf{Eval}\left(\mathsf{evk}, f, c_1, \ldots, c_\ell\right)\right) \neq f\left(x_1, \ldots, x_\ell\right)\right] = \mathsf{negl}\left(\kappa\right),$$

where $\mathsf{sk} \leftarrow \mathcal{E}.\mathsf{KGen}(1^\kappa)$*, and* $c_i \leftarrow \mathcal{E}.\mathsf{Enc}(\mathsf{sk}, x_i), \forall i \in [\ell]$.

3.3 Bootstrapping and key-switching

Every FHE construction that we know of is based on "noisy" encryption, i.e., each ciphertext contains a certain amount of noise (or error) that is vital for the security of the scheme. Whenever performing homomorphic operations on ciphertexts, the noise term contained in the output ciphertext is larger than those in the input ciphertexts, as a result of some sort of interaction that happens during the homomorphic computation. We can then say that homomorphic operations make the error grow[1]. We encounter problems when the noise grows beyond a certain threshold and makes the ciphertext "too noisy" to be correctly decrypted. This means that decrypting the ciphertext will produce a message which is different from the expected one; one can think of the noise becoming so large that it changes the value of the message.

A trivial solution is that, for a specific circuit that has to be homomorphically evaluated, one can choose the parameters of the encryption scheme so that this threshold is high enough to carry out all the computations that are needed. However, this bound has to be known in advance, before setting the parameters and instantiating the encryption scheme. Since homomorphic operations increase the noise in the ciphertexts, it is impossible to carry out an unbounded number of homomorphic operations without reaching the threshold and, therefore, loosing correctness. This remained a key obstacle on the path to FHE, until Gentry's original work [Gen09b] proposed a beautiful idea called *bootstrapping*.

Given a ciphertext c that encrypts a message m and that contains a large (but still within the threshold) noise e, if we had the correct secret key we could simply decrypt c, recover m, pick a smaller error e' and re-encrypt m with this noise. In other words, if we had the secret key we could *refresh* a ciphertext by decreasing the noise it contains: this would open the way to unbounded computations. In fact, it would be sufficient to perform homomorphic operations and refresh the ciphertexts before loosing correctness. However, as previously stated, the goal with FHE is to *publicly* perform operations on encrypted data, i.e., without having access to private information, namely the secret key.

The key intuition for this bootstrapping procedure is to perform this decrypt-and-encrypt procedure *homomorphically*! In formulas, what we want to achieve is this

$$c' = \mathsf{Enc}\left(\mathsf{Dec}\left(\mathsf{sk}, c\right)\right), \tag{3.1}$$

i.e., produce a *fresh* ciphertext c' that encrypts the same message as c, but with smaller noise. When moving to homomorphic operations, let us consider the following:

$$\boxed{c'} = \mathsf{Enc}\left(\mathsf{Dec}\left(\boxed{\mathsf{sk}}, \boxed{c}\right)\right), \tag{3.2}$$

where boxed terms are encrypted. Equation (3.2) can be seen as being the encrypted "equivalent" of Equation (3.1), where all the inputs and the outputs are encrypted. Two things are immediately noticeable: (1) this procedure requires an encryption of the secret key sk, and (2) the final result is not a fresh encryption c', but an encryption of an encryption, i.e., we added a layer of encryption. The second issue is due to the fact that the procedure outlined in Equation (3.2) was purposely inaccurate, and only meant to give an intuition. What happens in reality is that the ciphertext c is hardwired in the decryption circuit, so

[1]The way, i.e., the amount by which, the noise grows depends on the specific encryption scheme and also on which operation is homomorphically performed.

that the encryption of the secret key is the only input. The final output is then a ciphertext, without any additional layer of encryption, as was the case for Equation (3.1). In formulas,

$$c' = \mathsf{Dec}_c\left(\boxed{\mathsf{sk}}\right) \tag{3.3}$$

In conclusion, what we need is for the encryption scheme to be able to homomorphically evaluate its decryption function.

As far as the encryption of the secret key is concerned, this term is usually referred to as *bootstrapping key* (bk), and it appears to be necessary for the bootstrapping to work. The question is then the following:

Under which key should the secret key be encrypted?

There are two possibilities:

1. The secret key sk is encrypted under itself, i.e., $\mathsf{bk} = \mathsf{Enc}\left(\mathsf{sk}, \mathsf{sk}\right)$;
2. The secret key sk is encrypted under another key sk', i.e., $\mathsf{bk} = \mathsf{Enc}\left(\mathsf{sk}', \mathsf{sk}\right)$.

The first alternative leads to the situation where the refreshed ciphertext c' is encrypted under the same key as the original ciphertext c. This is advantageous because the secret key remains the same and there is no collection of keys to handle (see, e.g., [BGV12, Section 5.5]). However, following this alternative forces to make the so-called *circular security assumption.*

Assumption 3.3.1 (Circular security (informal))**.** *It is safe to publish an encryption of the secret key* sk *under itself.*

For a more formal statement of this assumption, consider the following: let $\mathcal{E}$ be a homomorphic encryption scheme, and let $\mathsf{Adv}_{\mathcal{A}}^{\text{IND-CPA}}(x)$ be the advantage that the adversary $\mathcal{A}$ running on input x has when trying to break the semantic security of $\mathcal{E}$ (see Figure 2.1 for the depiction of the IND-CPA game). Then the circular security assumption can be formulated as follows:

Assumption 3.3.2 (Circular security)**.** *For any PPT adversary $\mathcal{A}$,*

$$\left|\mathsf{Adv}_{\mathcal{A}}^{\text{IND-CPA}}\left(1^\kappa, \mathsf{bk}\right) - \mathsf{Adv}_{\mathcal{A}}^{\text{IND-CPA}}\left(1^\kappa\right)\right| = \mathsf{negl}\left(\kappa\right),$$

where bk *is a bootstrapping key for the scheme* $\mathcal{E}$*.*

It is currently unknown whether we can prove that revealing such an encryption is secure but, although this assumption is sometimes regarded with suspicion, no concrete attacks have ever been shown.

Instead, the second alternative (i.e., encrypting the secret key under a different key) leads to the situation where the refreshed ciphertext c' is encrypted under a different secret key and, for this reason, this procedure is also referred to as *key switching.* It has the advantage of not requiring any circular security assumption, but it does require handling several keys and, above all, agreeing on these keys beforehand. Moreover, a fundamental limitation of this approach is about the number of operations that can be performed, that depends on the number of keys that were agreed upon in advance. In fact, if there are N keys available, it is clear that it will be possible to do at most N bootstrapping operations: this means that the scheme will not be *fully* homomorphic but will achieve only a *leveled* homomorphism. Also, note that switching to any previously-used key, e.g., $sk_1 \to \mathsf{sk}_2 \to \cdots \to \mathsf{sk}_N \to \mathsf{sk}_1$, anyway requires assuming circular security. The conclusion is that, as of today, the instantiation of a fully-fledged FHE scheme does require the circular security assumption. Any other approach will only achieve limited homomorphic properties, that will allow for the evaluation of a bounded circuit.

Finally, a note on efficiency: bootstrapping is a complex procedure, that involves the homomorphic computation of a somewhat intricate function. For this reason, it is not very efficient and remains one of the major bottlenecks in FHE implementations. Over the years, several works aimed at optimizing bootstrapping, both from the point of view of the running time and from that of when to perform this operation during the evaluation of a circuit (e.g., see [AP13; AP14; DM15; CGGI16b; CGGI17; ZYL+17; BLMZ17]).

We now present a basic version of a well-known encryption scheme based on the LWE assumption, usually referred to as the Regev encryption scheme [Reg05]. The ideas that this construction is based on will be re-used several times in this manuscript.

Construction 3.3.3 (Regev's symmetric encryption scheme)**.** *This encryption scheme operates on bits, i.e., the message space is $\mathcal{M} = \{0,1\}$. It is composed of the following algorithms:*

$\mathsf{KGen}\left(1^\kappa\right) \to (\mathsf{sk}, pp)$ **:** *given the security parameter in unary, choose an integer $n = n(\kappa)$, a modulus $q = q(\kappa)$, an error distribution $\chi = \chi(\kappa)$, and output a secret key $\mathsf{sk} \leftarrow_{\$} \mathbb{Z}_q^n$ and public parameters $pp = (n, q, \chi)$. In the following, pp is an implicit argument to all the algorithms.*

$\mathsf{Enc}\left(\mathsf{sk}, m \in \mathcal{M}\right) \to c$ **:** *given the secret key sk and a message m, sample $\mathbf{a} \leftarrow_{\$} \mathbb{Z}_q^n$, $e \leftarrow \chi$, and return $c = \left(\mathbf{a}, b = \langle \mathsf{sk}, \mathbf{a}\rangle + e + m\,\frac{q}{2}\right) \in \mathbb{Z}_q^{n+1}$.*

$\mathsf{Dec}\left(\mathsf{sk}, c\right) \to m'$ **:** *given the secret key sk and a ciphertext $c = (\mathbf{a}, b) \in \mathbb{Z}_q^{n+1}$, compute $b - \langle \mathsf{sk}, \mathbf{a}\rangle$ and return $m' = 0$ if this quantity is closer to 0 than to $\frac{q}{2}$. Otherwise return 1.*

Correctness and security of Construction 3.3.3 are straightforward: correctness holds as long as $|e| < \frac{q}{4}$, whereas security comes directly from the LWE assumption. In particular, the term $\langle \mathsf{sk}, \mathbf{a}\rangle + e$ plays the role of a random mask for the message m.

Remark 3.3.4 (Extending the message space)**.** *Construction 3.3.3 can be trivially extended to $\mathcal{M} = \mathbb{Z}_p$, for $q > p \in \mathbb{N}$. It is sufficient to multiply m by $\frac{q}{p}$ instead of $\frac{q}{2}$ during encryption, and round appropriately during decryption. Naturally, this implies a different condition on the error for correctness to hold: in this case, it is necessary that $|e| < \frac{q}{2p}$.*

3.4 Three generations of FHE

After giving these introductory notions on FHE and a basic construction, we finally turn to analyzing the different generations of fully homomorphic encryptions.

3.4.1 First generation FHE

The first generation of FHE is that which stems directly from Gentry's seminal work (mainly his STOC paper [Gen09b] and his Ph.D. thesis [Gen09a]), and it was the first concrete proposal for an encryption schemes that allows for unbounded computation on encrypted data. We now give a brief and high-level presentation of this construction, and we refer the reader to the cited material and to [GH10] for more details. This scheme is a public-key encryption scheme *à la* [GGH97] over ideal lattices in a polynomial ring $\mathcal{R}$. In the ring $\mathcal{R}$, two ideals I and J are chosen: the public key is a "bad basis" of the ideal lattice J, whereas the private key is a "good basis" of this ideal. Then, a single bit plaintext is embedded in $\mathcal{R}/I$ and added to an error term sampled from I. The sum is then reduced using the public key (i.e., the public basis of J), thus obtaining the ciphertext. This way, the message part and the error part are separated, which will be extremely important for homomorphic operations. If the error term is small enough, then the secret basis of J allows one to separate the fractional part of the ciphertext, thus recovering the plaintext and the error. On the other hand, if the error is too large, then this separation cannot be performed and the message is lost. The homomorphic operations are straightforward: homomorphic addition amounts to adding the two ciphertexts, whereas homomorphic multiplication amounts to multiplying the two ciphertexts. In the case of homomorphic addition, it is easy to see that, in the two ciphertexts, the message part and the error part can be treated separately. In the case of homomorphic product, the mixed terms are still in the ideal I, which means that they end up being part of the error of the output ciphertext. Naturally, all these operations yield correct results only if the final error is still small enough to allow for decryption. In particular, with this scheme, the noise is roughly doubled in case of an addition and squared in case of a multiplication.

This work also introduced the concept of *squashing the decryption circuit.* Roughly speaking, the decryption function of the scheme was not simple enough to allow for a sufficient reduction of the noise in the bootstrapped ciphertexts. For this reason, this decryption function is replaced by a simpler one, under an additional security assumption, namely that the sparse subset sum problem (SSSS) is hard. The goal of subsequent works then became instantiating FHE from encryption schemes with simpler decryption circuits, in order to avoid this additional step and the need for additional assumptions.

3.4.2 Second generation FHE

Second generation FHE is mainly due to the works of Brakerski and Vaikuntanathan (BV) [BV11a; BV11b; BGV12], who constructed FHE based on standard assumptions supported by worst-case hardness, namely LWE (cf. Assumption 2.4.4). The original BV supports a message space $\mathcal{M} = \{0, 1\}$, but can be easily extended to $\mathbb{Z}_p$ for a sufficiently small integer p. It allows one to perform homomorphic additions and multiplications modulo two: it is easy to see that this is sufficient for building FHE, as a composition of these two operations is enough to evaluate any boolean circuit. We now present in details a public-key version of the BV scheme.

Construction 3.4.1 (BV encryption scheme)**.** *This is a public-key encryption scheme that operates on bits, i.e., the message space is $\mathcal{M} = \{0, 1\}$. Note that all the computations are done modulo q. It is composed of the following algorithms:*

$\mathsf{KGen}\left(1^\kappa\right) \to (\mathsf{sk}, \mathsf{pk}, pp)$ **:** *given the security parameter in unary, choose integers $n = n\left(\kappa\right)$, $m = m\left(\kappa\right)$, a modulus $q = q\left(\kappa\right)$, and an error distribution $\chi = \chi\left(\kappa\right)$ over the integers. Sample a vector $\mathbf{s} \leftarrow\!\!\$\, \mathbb{Z}_q^n$, and a matrix $\mathbf{A} \leftarrow\!\!\$\, \mathbb{Z}_q^{m \times n}$. Output a secret key $\mathsf{sk} = \mathbf{s}$, and* *a public key* $\mathsf{pk} = (-\mathbf{A}, \mathbf{b} = \mathbf{A}\mathbf{s} + 2\mathbf{e})$*, where* $\mathbf{e} \leftarrow \chi^m$*. Also, output public parameters* $pp = (n, m, q)$*, which are implicitly given as inputs to all the algorithms.*

$\mathsf{Enc}(\mathsf{pk}, \mu \in \mathcal{M}) \rightarrow c$ **:** *given a public key* $\mathsf{pk} = (-\mathbf{A}, \mathbf{b})$ *and a message* μ*, sample* $\mathbf{r} \leftarrow_{\$} \{0,1\}^m$ *and return* $c = (\mathbf{c}_1, c_2) = (-\mathbf{A}^\intercal \mathbf{r}, \mathbf{b}^\intercal \mathbf{r} + \mu)$*.*

$\mathsf{Dec}(\mathsf{sk}, c) \rightarrow \mu'$ **:** *given a secret key* sk *and a ciphertext* $c = (\mathbf{c}_1, c_2) \in \mathbb{Z}_q^{n+1}$*, compute*

$$c_2 + \langle \mathsf{sk}, \mathbf{c}_1 \rangle = \mathbf{s}^\intercal \mathbf{A}^\intercal \mathbf{r} + 2\mathbf{e}^\intercal \mathbf{r} + \mu - \mathbf{s}^\intercal \mathbf{A}^\intercal \mathbf{r} = 2\mathbf{e}^\intercal \mathbf{r} + \mu.$$

Return the parity of this result, i.e., return $(2\mathbf{e}^\intercal \mathbf{r} + \mu) \mod 2$*.*

It is easy to see that the scheme is correct as long as $2\mathbf{e}^\intercal \mathbf{r} < \frac{q}{4}$. We now detail how to perform homomorphic operations, namely additions and multiplications, with this encryption scheme.

Homomorphic addition. The homomorphic addition is trivial: let $\mathcal{E}$ be an instantiation of Construction 3.4.1, and let $(\mathsf{sk}, \mathsf{pk}) \leftarrow \mathcal{E}.\mathsf{KGen}(1^\kappa)$. Let $c_1 = (\mathbf{c}_{11}, c_{12}) \leftarrow \mathsf{Enc}(\mathsf{pk}, \mu_1)$ and $c_2 = (\mathbf{c}_{21}, c_{22}) \leftarrow \mathsf{Enc}(\mathsf{pk}, \mu_2)$, for messages $\mu_1, \mu_2 \in \{0,1\}$. Then $c^+ = (\mathbf{c}_{11} + \mathbf{c}_{21}, c_{12} + c_{22})$ is an encryption of $\mu_1 + \mu_2 \mod 2$, as long as the error remains small enough.

In fact, the decryption of $\mathbf{c}^+$ can be written as

$$c_{12} + c_{22} + \langle \mathsf{sk}, \mathbf{c}_{11} \rangle + \langle \mathsf{sk}, \mathbf{c}_{21} \rangle = \cdots = 2\mathbf{e}^\intercal (\mathbf{r}_1 + \mathbf{r}_2) + (\mu_1 + \mu_2),$$

where $\mathbf{r}_i$ is the random binary vector used for encrypting μ_i. Again, the decryption will be successful if $2\mathbf{e}^\intercal (\mathbf{r}_1 + \mathbf{r}_2) < \frac{q}{4}$.

We can immediately notice that performing an homomorphic addition increases the noise level in the output ciphertext: for this reason, the number of additions that can be evaluated is bounded a priori by the choice of the parameters for the scheme.

Homomorphic multiplication. The homomorphic multiplication is considerably trickier. First of all, we can notice that the decryption procedure can be written as a single inner product: for a secret vector $\mathbf{s}$ and a ciphertext $c = (\mathbf{c}_1, c_2)$, let $\hat{\mathbf{s}} := (\mathbf{s} \| 1)$ and $\hat{\mathbf{c}} := (\mathbf{c}_1 \| c_2)$. Then decrypting amounts to computing $\langle \hat{\mathbf{s}}, \hat{\mathbf{c}} \rangle$ and reducing modulo 2.

Next we recall the *tensor* (or Kronecker) product between two vectors $\mathbf{v}, \mathbf{w} \in \mathbb{Z}_q^n$:

$$\mathbf{v} \otimes \mathbf{w} = (v_i \cdot w_j)_{i,j} \in \mathbb{Z}_q^{n^2}.$$

We also recall the *mixed-product* property of the tensor product: for appropriately sized vectors $\mathbf{a}, \mathbf{b}, \mathbf{c}, \mathbf{d}$,

$$\langle \mathbf{a} \otimes \mathbf{b}, \mathbf{c} \otimes \mathbf{d} \rangle = \langle \mathbf{a}, \mathbf{c} \rangle \cdot \langle \mathbf{b}, \mathbf{d} \rangle.$$

Then, it is immediate to see that the tensor product of two ciphertexts is an encryption of the product of the messages, under a different key (namely, the tensor product of the secret key with itself). In formulas:

$$\langle \hat{\mathbf{s}} \otimes \hat{\mathbf{s}}, \hat{\mathbf{c}}_1 \otimes \hat{\mathbf{c}}_2 \rangle = \langle \hat{\mathbf{s}}, \hat{\mathbf{c}}_1 \rangle \cdot \langle \hat{\mathbf{s}}, \hat{\mathbf{c}}_2 \rangle = (2(\bullet) + \mu_1) \cdot (2(\bullet) + \mu_2) = 2(\bullet) + (\mu_1 \cdot \mu_2),$$

where we use the symbol $\bullet$ to hide terms that we do not care about. In the end, the message part, i.e., the part which is not multiplied by 2, is $(\mu_1 \cdot \mu_2)$ as desired.

Once again, performing an homomorphic multiplication increases the noise level in the output ciphertext, so the number of multiplications that can be evaluated is bounded a priori by the choice of the parameters for the scheme.

However, the biggest problem with homomorphic multiplication is that, because of the tensor product, the size of the ciphertexts grows exponentially in the number of operations. This clearly impacts efficiency, as well as composability, meaning that the output ciphertexts are not in the same space as the input ones. In order to solve this issue, Brakerski and Vaikuntanathan [BV11a] propose a *dimension reduction* technique, which can be seen as an example of key switching; in the literature, this procedure is also referred to as *relinearization*. In a nutshell, by publishing an encryption of several terms in the original secret key $\hat{\mathbf{s}}$ under a different secret key $\hat{\mathbf{t}}$, they are able to re-linearize the ciphertext, going back from a quadratic size (roughly $n^2/2$) to the original $n+1$.

Modulus switching. As we presented in the preceding part, second generation FHE ciphertexts are vectors in $\mathbb{Z}_q$, where q is some modulus. These ciphertexts contain a certain amount of noise for security purposes, and remain decryptable as long as the magnitude of this noise is below $q/4$. We also saw that homomorphic operations increase the noise level contained in ciphertexts; in particular, homomorphic addition roughly doubles the noise, while homomorphic multiplication roughly squares it. Let us say that the initial noise magnitude was B: after evaluating L multiplications, it will grow to B^{2^L}. In turn, this means that a very large modulus $q \approx B^{2^L}$ is required for correct decryption, thus affecting both efficiency (larger parameters mean the scheme is less efficient) and security (which is determined by the modulus-to-noise ratio). A substantial improvement came from [BGV12], where they proposed a modulus switching technique for better management of the noise growth. It essentially consists in *scaling down* the ciphertext after every multiplication by a certain scaling factor K. This means that the modulus goes from q to q/K, while the noise goes from B to B/K. Indeed, the ratio between modulus and noise remains the same, but this technique is helpful for choosing better parameters. In fact, let us consider a scaling factor $K = B$. After a multiplication, the noise becomes roughly B^2, but gets scaled back down to B, together with the modulus, which becomes q/B. After L multiplications, the modulus will become q/B^L, while the error will always be B. In conclusion, the condition to have decryption correctness is that the noise must be smaller than the modulus divided by 4. In this case, this means that it is sufficient to take $q \approx B^{L+1}$, which is considerably better than having to take $q \approx B^{2^L}$ as before.

However, even if using the techniques described above helps with improving the capabilities of the scheme, homomorphic operations still increase the noise level. For this reason, the number of operations that can be evaluated remains bounded: constructing an FHE scheme (i.e., having the possibility of evaluating an unbounded number of operations) still requires using bootstrapping (cf. Section 3.3).

3.4.3 Third generation FHE

By "third generation FHE" we refer to a line of work initiated by Gentry, Sahai, and Waters (GSW) [GSW13], and prosecuted by several other works (e.g., [AP14; HAO15; KGV14]). These schemes are based on the so-called *approximate eigenvector problem*, and their main advantage is that they do not require any key switching for homomorphic multiplication. We now show a construction inspired by [GSW13], following the presentation of [AP14].

Gadget matrix and (randomized) binary decomposition. The first ingredient for this construction is the so-called *gadget matrix*, already introduced in [MP12]. Let q be a modulus, such that $\ell = \lceil \log q \rceil$; then the matrix is defined as $\mathbf{G} = \mathbf{g}^\intercal \otimes \mathbf{I}_n \in \mathbb{Z}_q^{n \times n\ell}$, where $\mathbf{g} = \left(1, 2, 4, \ldots, 2^{\ell-1}\right)$ is the vector of the powers of 2. The matrix $\mathbf{G}$ is thus a block-diagonal matrix:

$$\mathbf{G} = \begin{bmatrix} 1 & 2 & \cdots & 2^{\ell-1} & 0 & 0 & \cdots & 0 & \cdots & 0 & 0 & 0 & 0 \\ 0 & 0 & \cdots & 0 & 1 & 2 & \cdots & 2^{\ell-1} & \cdots & 0 & \cdots & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & \cdots & 0 & 0 & 0 & \cdots & 0 & \cdots & 1 & 2 & \cdots & 2^{\ell-1} \end{bmatrix}.$$

The cryptosystem. We now describe the GSW encryption scheme according to the presentation in [AP14]. In particular, we will focus on a secret-key version of the scheme, with message space $\mathcal{M} = \mathbb{Z}_q$. The scheme is composed of the following algorithms:

$\mathsf{KGen}(1^\kappa) \to (\mathsf{sk}, pp)$ **:** choose an integer $n = n(\kappa)$, a modulus $q = q(\kappa)$, and let $\ell = \lceil \log q \rceil$, and $m = n\ell$. Sample a secret vector $\mathbf{s} \leftarrow_\$ \mathbb{Z}_q^{n-1}$ and output the secret key $\mathsf{sk} = \hat{\mathbf{s}} := (\mathbf{s} \| 1)$, and public parameters $pp = (n, q)$. As usual, pp will be an implicit argument to all the algorithms.

$\mathsf{Enc}(\mathsf{sk}, \mu \in \mathcal{M}) \to \mathbf{C}$ **:** given a secret key sk and a message μ, sample a random matrix $\mathbf{A} \leftarrow_\$ \mathbb{Z}_q^{(n-1)\times m}$, an error vector $\mathbf{e} \leftarrow \chi^m$ for some error distribution χ, and output

$$\mathbf{C} = \begin{pmatrix} -\mathbf{A} \\ \mathbf{s}^\intercal \mathbf{A} + \mathbf{e}^\intercal \end{pmatrix} + \mu \mathbf{G} \in \mathbb{Z}_q^{n \times m}.$$

$\mathsf{Dec}(\mathsf{sk}, \mathbf{C}) \to \mu'$ **:** given a secret key sk and a ciphertext $\mathbf{C}$, compute

$$\mathsf{sk}^\intercal \mathbf{C} = \hat{\mathbf{s}}^\intercal \left(\begin{pmatrix} -\mathbf{A} \\ \mathbf{s}^\intercal \mathbf{A} + \mathbf{e}^\intercal \end{pmatrix} + \mu \mathbf{G} \right) = \mathbf{e}^\intercal + \mu \, \hat{\mathbf{s}}^\intercal \mathbf{G},$$

and return μ', the closest integer to $\hat{\mathbf{s}}^\intercal \mathbf{G}$.

Condition for correct decryption. In order for the decryption procedure to return the correct value, it is necessary that the error is small enough, so that it does not perturb the final result. In particular, it is easy to imagine the space $\mathbb{Z}_q$ divided into several segments, with each one corresponding to a possible plaintext value. The absolute value of the error will then have to be smaller than half the width of each segment. In conclusion, it must hold that

$$\|\mathbf{e}\|_\infty < \frac{q}{2\,|\mathcal{M}|}.$$

Notice that for $\mathcal{M} = \{0, 1\}$, we recover the well-known condition $\|\mathbf{e}\|_\infty < \frac{q}{4}$.

Extension to public-key setting. It is easy to turn the secret-key encryption scheme presented above into a public-key one. The main idea is that the LWE matrix that plays the role of the mask will be the public key; for encrypting, one will first re-randomize this public key by multiplying it by a random matrix with small entries, and then proceed as before. The decryption will work exactly like in the secret-key case. We now give the details of the public-key version of the GSW cryptosystem.

$\mathsf{KGen}(1^\kappa) \to (\mathsf{sk}, \mathsf{pk}, pp)$ **:** choose an integer $n = n(\kappa)$, a modulus $q = q(\kappa)$, and let $\ell = \lceil \log q \rceil$, and $m = n\ell$. Sample a secret vector $\mathbf{s} \leftarrow_\$ \mathbb{Z}_q^{n-1}$, a matrix $\mathbf{A} \leftarrow_\$ \mathbb{Z}_q^{(n-1)\times m}$, an error vector $\mathbf{e} \leftarrow \chi^m$ for some error distribution χ, and output the secret key $\mathsf{sk} = \hat{\mathbf{s}} := (\mathbf{s} \| 1)$, the public key $\mathsf{pk} = \begin{pmatrix} -\mathbf{A} \\ \mathbf{s}^\intercal \mathbf{A} + \mathbf{e}^\intercal \end{pmatrix}$, and public parameters $pp = (n, q)$. As usual, pp will be an implicit argument to all the algorithms.

$\mathsf{Enc}(\mathsf{pk}, \mu \in \mathcal{M}) \to \mathbf{C}$ **:** given a public key pk and a message μ, sample a random matrix $\mathbf{R} \leftarrow_\$ \{0,1\}^{m\times m}$, and output

$$\mathbf{C} = \begin{pmatrix} -\mathbf{A} \\ \mathbf{s}^\intercal \mathbf{A} + \mathbf{e}^\intercal \end{pmatrix} \mathbf{R} + \mu \mathbf{G} \in \mathbb{Z}_q^{n\times m}.$$

$\mathsf{Dec}(\mathsf{sk}, \mathbf{C}) \to \mu'$ **:** given a secret key sk and a ciphertext $\mathbf{C}$, compute

$$\mathsf{sk}^\intercal \mathbf{C} = \hat{\mathbf{s}}^\intercal \left(\begin{pmatrix} -\mathbf{A} \\ \mathbf{s}^\intercal \mathbf{A} + \mathbf{e}^\intercal \end{pmatrix} \mathbf{R} + \mu \mathbf{G} \right) = \mathbf{e}^\intercal \mathbf{R} + \mu\, \hat{\mathbf{s}}^\intercal \mathbf{G},$$

and return μ', the closest integer to $\hat{\mathbf{s}}^\intercal \mathbf{G}$.

We now turn to homomorphic operations. In particular, we will describe how to perform addition and multiplication with encrypted inputs, which allow for the evaluation of any function.

Homomorphic addition. The homomorphic addition is trivial: given a ciphertext $\mathbf{C}_1$ that encrypts a message μ_1, and a ciphertext $\mathbf{C}_2$ that encrypts a message μ_2, the ciphertext $\mathbf{C}^+ := \mathbf{C}_1 + \mathbf{C}_2$ is an encryption of $\mu_1 + \mu_2$ (provided that the errors in the input ciphertexts are not too big). In fact,

$$\hat{\mathbf{s}}^\intercal \mathbf{C}^+ = \hat{\mathbf{s}}^\intercal (\mathbf{C}_1 + \mathbf{C}_2) = (\mathbf{e}_1^\intercal + \mathbf{e}_2^\intercal) + (\mu_1 + \mu_2)\, \hat{\mathbf{s}}^\intercal \mathbf{G},$$

and one can recover $\mu_1 + \mu_2$ by rounding as before.

Noise growth formula for homomorphic addition. Although it is immediate, we report here for completeness the noise growth formula for the homomorphic addition of two GSW ciphertexts: given a common secret key sk, and two GSW ciphertexts $\mathbf{C}_1, \mathbf{C}_2 \in \mathbb{Z}_q^{n\times m}$, such that $\mathbf{C}_i$ is an encryption under sk of a message μ_i with error vector $\mathbf{e}_i$, the error vector contained in $\mathbf{C}^+ := \mathbf{C}_1 + \mathbf{C}_2$ is

$$\mathbf{e}_{\text{out}} = \mathbf{e}_1 + \mathbf{e}_2.$$

This means that, for the GSW cryptosystem, the error in homomorphic additions is additive.

The $\mathbf{G}^{-1}(\cdot)$ algorithm. Before describing the homomorphic multiplication, we introduce a key part of this construction, which is the $\mathbf{G}^{-1}(\cdot)$ algorithm. Let $\ell := \lceil \log q \rceil$, then $\mathbf{G}^{-1}(\cdot) : \mathbb{Z}_q^n \to \{0,1\}^{n\ell}$ is a deterministic algorithm defined as follows:

$$\mathbf{G}^{-1}(\mathbf{x}) = \mathbf{y} \iff \mathbf{G}\mathbf{y} = \mathbf{x}, \quad \forall \mathbf{x} \in \mathbb{Z}_q^n.$$

In a nutshell, $\mathbf{G}^{-1}(\mathbf{x})$ is simply the binary decomposition of $\mathbf{x}$. In the literature, there exists also a randomized version of this algorithm, which will be extremely important in Chapter 5 and will be introduced there. A key feature of the $\mathbf{G}^{-1}(\cdot)$ algorithm is that its output is a vector with *small* entries, whether they be binary or slightly larger (as will be the case in Chapter 5).

Note that, given a matrix $\mathbf{M} \in \mathbb{Z}_q^{r\times c}$, when we write $\mathbf{N} = \mathbf{G}^{-1}(\mathbf{M}) \in \{0,1\}^{r\ell\times c}$, we mean that the $\mathbf{G}^{-1}(\cdot)$ algorithm is applied independently to all the columns of $\mathbf{M}$, and the resulting vectors are the columns of $\mathbf{N}$.

Homomorphic multiplication. The homomorphic multiplication is, once again, the tricky part. Given a ciphertext $\mathbf{C}_1 = \begin{pmatrix} -\mathbf{A}_1 \\ \mathbf{s}^\intercal\mathbf{A}_1 + \mathbf{e}_1^\intercal \end{pmatrix} + \mu_1\mathbf{G}$ that encrypts a message μ_1, and a ciphertext $\mathbf{C}_2 = \begin{pmatrix} -\mathbf{A}_2 \\ \mathbf{s}^\intercal\mathbf{A}_2 + \mathbf{e}_2^\intercal \end{pmatrix} + \mu_2\mathbf{G}$ that encrypts a message μ_2, then

$$\mathbf{C}^\times = \mathbf{C}_1 \cdot \mathbf{G}^{-1}(\mathbf{C}_2) \tag{3.4}$$

is an encryption of $\mu_1 \cdot \mu_2$. Verifying this fact is straightforward:

$$\mathbf{C}_1 \cdot \mathbf{G}^{-1}(\mathbf{C}_2) = \begin{pmatrix} -\mathbf{A}_1\mathbf{G}^{-1}(\mathbf{C}_2) \\ \mathbf{s}^\intercal\mathbf{A}_1\mathbf{G}^{-1}(\mathbf{C}_2) + \mathbf{e}_1^\intercal\mathbf{G}^{-1}(\mathbf{C}_2) \end{pmatrix} + \mu_1 \begin{pmatrix} -\mathbf{A}_2 \\ \mathbf{s}^\intercal\mathbf{A}_2 + \mathbf{e}_2^\intercal \end{pmatrix} + \mu_1\mu_2\mathbf{G}$$

This can be rewritten as

$$\mathbf{C}_1 \cdot \mathbf{G}^{-1}(\mathbf{C}_2) = \begin{pmatrix} -\mathbf{A}^* \\ \mathbf{s}^\intercal\mathbf{A}^* + \mathbf{e}^{*\intercal} \end{pmatrix} + \mu_1\mu_2\mathbf{G},$$

where $\mathbf{A}^* = \mathbf{A}_1\mathbf{G}^{-1}(\mathbf{C}_2) + \mu_1\mathbf{A}_2$, and $\mathbf{e}^{*\intercal} = \mathbf{e}_1^\intercal\mathbf{G}^{-1}(\mathbf{C}_2) + \mu_1\mathbf{e}_2^\intercal$. It is trivial to see that this decrypts to $\mu_1\mu_2$, as long as the new error is small enough.

Noise growth formula for homomorphic multiplication. We report here the analytical noise growth formula for a GSW multiplication: given a common secret key sk, and two GSW ciphertexts $\mathbf{C}_1, \mathbf{C}_2 \in \mathbb{Z}_q^{n\times m}$, such that $\mathbf{C}_i$ is an encryption under sk of a message μ_i with error vector $\mathbf{e}_i$, the error vector contained in $\mathbf{C}_1 \cdot \mathbf{G}^{-1}(\mathbf{C}_2)$ is

$$\mathbf{e}_{\text{out}} = \mathbf{e}_1\mathbf{G}^{-1}(\mathbf{C}_2) + \mu_1\mathbf{e}_2. \tag{3.5}$$

Remark 3.4.2. *An important feature of this encryption scheme is the fact that its noise growth for homomorphic multiplication is asymmetric, i.e., it depends on the order of the operands. In fact, the second noise vector is multiplied as it is by the first message, while the first noise vector is multiplied by the second ciphertext, but after applying the* $\mathbf{G}^{-1}(\cdot)$ *operation, which ensures the output is small. With this in mind, one can notice that also* $\mathbf{C}_2 \cdot \mathbf{G}^{-1}(\mathbf{C}_1)$ *leads to a result which is similar to Equation (3.4), but with a different output noise. The choice of the order of the operands then depends on which of the two ciphertexts has the highest noise, i.e., which one is "fresher". Taking the noise growth formula of Equation (3.5) into account, it is usually convenient to apply the* $\mathbf{G}^{-1}(\cdot)$ *operation to the one that contains more noise.*

Remark 3.4.3. *Looking once again at the noise growth formula in Equation (3.5), it is easy to see why this encryption scheme is not very well suited for large message spaces. In fact, the term* $\mu_1\mathbf{e}_2$ *can be very large if* μ_1 *is in a large domain. Therefore, it is not uncommon to see the GSW encryption scheme used only for messages in* $\{0,1\}$ *or* $\{-1,0,1\}$.

3.4.4 Message packing

One of the most evident downsides of the GSW encryption scheme is its expansion factor when encrypting, meaning the ratio between the size of a ciphertext and the size of the plaintext. For example, with reasonable parameters, this problem can be as bad as requiring almost 1 Gbit to encrypt a single bit[2]! A work by Hiromasa *et al.* [HAO15] exploits the redundancy in the ciphertexts, and manages to amortize the size of an encryption by packing several messages in the same ciphertext. In particular, r plaintexts can be encrypted in a matrix of size $(r+n) \times (r+n) \log q$, which takes $(r+n)^2 \log^2 q$ bits of storage. This is a clear improvement on naively encrypting r plaintexts separately, which would take $r \cdot n^2 \log^2 q$ bits of storage.

3.5 Advanced constructions

Although FHE has gained momentum only recently (the first construction having been proposed in 2009), a lot of research effort has been put on this topic, because of its novelty, the number of open questions, the vast number of results that can be achieved through FHE, and the important consequences that practical instantiations would have on people's lives. Apart from "basic" FHE schemes, where the operations are encryption, decryption, and circuit evaluation, some more advanced constructions have been proposed. Analyzing each of them in detail would require far too much space, and it is beyond the scope of this chapter. However, we quickly and informally introduce some interesting topics that FHE has been applied to, and we refer the interested reader to the referenced material for more details.

Threshold FHE A threshold FHE scheme with N parties is essentially an FHE scheme where key generation and decryption are no longer algorithms but N-party protocols. The basic idea is that, at the beginning, the parties engage in a multi-party computation (MPC) protocol that produces a common public key pk, a common evaluation key evk, and gives each party P_i a share sk_i of the secret key sk. Then, each party can encrypt its input x_i under the common pk, and all the parties can independently and homomorphically evaluate a function f on these ciphertext[3]. Finally the parties engage in another MPC protocol where each party participates with his share of the secret key sk_i and in the end receives the result $\mu = f(x_1, \dots, x_N)$. For more details on the topic, we refer the reader to e.g., [AJL+12; JRS17].

Multi-key FHE This primitive was introduced in [LTV12], with improved constructions later given e.g., in [CM15; MW16]. The basic idea is essentially to enable homomorphic computations over data encrypted under different keys. More in details, each party individually encrypts its input under its key, then broadcasts the ciphertext. Next, all the parties can homomorphically compute a multi-key encryption of the output, that, however, cannot be decrypted under any of the secret keys. Finally, the parties broadcast a partial decryption of the output obtained with their secret keys; these partial decryptions can be combined together in order to recover the output itself.

[2]Let $n = 450$ and $q = 2^{64}$. Then a ciphertext is a matrix with $nm = n \cdot n \log q$ elements in $\mathbb{Z}_q$, which take $n \cdot n \log q \cdot \log q$ bits of space. With the given numbers, this amounts to 0.772 Gbit.

[3]This computation can even be delegated to another party, e.g., the Cloud.

Fully homomorphic signatures The basic idea of this primitive, proposed by Gorbunov *et al.* in [GVW15] is as follows. A signer signs some initial data x under its public key, and produces a signature σ_x. Then, an untrusted party that only knows the public key, x, and σ_x (but not the secret signing key), can apply an arbitrary function f to x, and compute a corresponding signature $\sigma_{f(x)}$ for the value $f(x)$. Finally, a verifier that is given the public key, f, $f(x)$, and $\sigma_{f(x)}$ (but not x), can verify that $f(x)$ is indeed the output of f applied to the input x that was signed at the beginning.

3.6 Libraries and practical implementations

Beside theoretical research, the field of homomorphic encryption has been strongly active with respect to implementation efforts. In fact, given the strongly applied nature of this sort of constructions, it is appropriate that the development of new mathematical techniques and theoretical breakthrough proceeds hand in hand with top quality implementation. On one hand, this contributes to assessing where we actually stand and how far the goal of practically usable homomorphic encryption is. On the other hand, this also helps pinpointing the limitations and the main difficulties still remaining, therefore suggesting to researchers and theoreticians where to concentrate their efforts.

Although several implementations have been released over the years by numerous authors, via means like GitHub, we will consider only a few of them, in order to give a broad idea on the current state of the art on this matter. For more information about efforts to standardize and implement FHE, we refer the reader to [org].

In particular, in this section, we will focus on HElib [HS14a; HS14b; HS15], SEAL [CHH+16], and TFHE [CGGI16b; CGGI17; CGGI16a].

HElib. This library, authored mainly by Shai Halevi and Victor Shoup, implements the BGV scheme [BGV12] (which can be put in the category of "2nd generation FHE") and it is one of the most widely used libraries in applications. It allows for packing of ciphertexts and SIMD computations, amortizing the cost for certain tasks. It is able to perform additions and multiplications in an efficient way, but the bootstrapping operation is significantly slow. In practice, this library is often used as a somewhat homomorphic encryption scheme. One of the drawbacks of this library regards the parameter selection, which remains a complicated and error-prone operation. In fact, in [HS15, Appendix A], the authors note that "*The BGV implementation in HElib relies on a myriad of parameters, some of which are heuristically chosen, and so it takes some experimentation to set them all so as to get a working implementation with good performance*".

SEAL. This library is written in C++11 and implements the FV encryption scheme [FV12]. It should be noted that this scheme is already implemented in [BEHZ16; Lep16], both of which use the ideal lattice library NFLlib [MBG+16]. Unlike in HElib, with the SEAL library it is considerably easier to set good parameters for the performance and the security of the implemented scheme.

TFHE. This library implements a generalized version of the GSW encryption scheme [GSW13; AP14], and thus belongs to the category of "3rd generation FHE". The library features a very efficient operation bootstrapping, with timings that are in the order of fractions of a second on average machines. As a downside, this bootstrapping operation has to be applied after computing every gate of the circuit. When used for realizing an FHE scheme, this library is more efficient than HElib. However, for simple tasks requiring small computational depth, HElib used as a somewhat homomorphic encryption scheme will generally perform better. Moreover, TFHE is currently not capable of amortizing large SIMD computations as well as HElib does. However, it should be noted that the code of TFHE is still largely under development, and more features (e.g., multithreading) are likely to be added soon. TFHE will be at the core of our construction in Chapter 4. Therefore, it will be analyzed more in details there.

3.7 FHE constructions from non-lattice assumptions

For completeness, we also mention that there are other constructions of fully homomorphic encryption schemes which are not based on lattice assumptions. This line of work is usually referred to as FHE *over the integers*. Notably, this is the case for the DGHV scheme [DGHV10; CMNT11; CCK+13; NK15], which relies on the assumed hardness of the approximate GCD problem (cf. Definition 2.4.6). The DGHV scheme is central in the constructions presented in Chapter 6, so we refer the reader to this chapter for its presentation and the details on this instantiation of FHE.

Chapter 4

Homomorphic evaluation of deep neural networks

In this chapter we address the challenge of homomorphically and efficiently evaluating a (deep) neural network. First, we give an introduction to the problem and some potential applications; then, we provide some necessary background notions on neural networks and we review the current state of the art for privacy-preserving evaluation of these models. We then move on to presenting a fast framework for FHE and bootstrapping by Chillotti *et al.* [CGGI16b; CGGI17], which will be central in our constructions. Finally, we present our contributions, explain in details our framework FHE-DiNN, and give some encouraging experimental results obtained on a typical dataset.

Contents

4.1 Introduction to the problem

Machine Learning As a Service (MLAS) refers to a cluster of applications where a remote entity provides a service that can be accessed through the Internet and that takes advantage of some machine learning algorithm to provide a result to the users' queries. This type of applications is becoming more and more popular because of its versatility and the impressive performance (e.g., better-than-human accuracies) that can be achieved.

A typical use-case involves a user (client) and a remote service (server), with the client uploading some data to the server, which runs some predictive model on it and returns some answer to the client. For a more concrete example, let us consider the following situation: Alice is a user who would like to assess her medical condition through an online diagnosis service[1]. In order to do so, she connects to a remote server, that we will call RemoteDoc, uploads her medical data and fills in some form where she answers several questions about her lifestyle and her habits. Then, RemoteDoc processes this information and, based on an archive of past diagnosis that was used for training some predictive model, returns an evaluation on Alice's situation. Optionally, RemoteDoc charges Alice with a fee for using the service. It is easy to see that this model of interaction might have tremendous consequences on Alice's privacy: the data she submits is likely to contain extremely sensitive information, that she does not want to disclose. Note that here we are not concerned about an external intruder, since we can safely assume that the channel through which Alice and RemoteDoc communicate is secure (e.g., via HTTPS); in this case, the problem is represented by disclosing private information to RemoteDoc itself. In this situation, there seems to be an unsolvable conflict: on one hand, Alice wants to keep her medical information private, while on the other hand she wants to obtain a reliable diagnosis, that necessarily has to be based on true data and honest answers.

Of course, a trivial solution would be that of evaluating RemoteDoc's predictive model *locally* on Alice's machine. We assume this is either infeasible (because of high computational or storage requirements) or forbidden (because the model is considered as sensitive intellectual property, or because the remote server wants to charge the user with a fee per every request) and we focus on the situation where the evaluation has to take place remotely, i.e., on RemoteDoc's side.

Homomorphic encryption offers an elegant way to solve this apparent paradox, by allowing Alice to submit *encrypted data* which is then *blindly processed* by RemoteDoc; the encrypted result (i.e., the encrypted diagnosis) is then sent back to Alice, who proceeds to decrypting and recovering the result. This way, the evaluation takes place on the server side and still RemoteDoc learns nothing on Alice's medical data, since all it sees is encryptions produced with a secret key that never leaves Alice's controlled domain. Another bonus of using homomorphic encryption is that it achieves non-interactivity: Alice submits an input and later receives an output, without the need for any additional communications between her and the remote party.

The "predictive models" we are referring to are often neural networks, which are *trained* on pairs consisting of some input and the expected output, and then *evaluated* on some fresh input (i.e., an input that has possibly never been seen before) to obtain a predicted output.

[1]We are not suggesting this is a good practice or an advisable/reliable way to obtain a medical evaluation. This is just an example to show some privacy-related implications of using web services.

We are then interested in designing an *efficient* framework for evaluating arbitrarily complex neural networks over encrypted data.

4.2 Refresher on neural networks

Neural networks are the second best way of doing just about anything.

– John Denker

In this section we give some necessary background notions on neural networks.

4.2.1 Basic definitions

Giving a proper, formal definition of a neural network is not an easy task and the literature contains several attempts at doing so. For example, we can consider the following:

Definition 4.2.1 (Artificial neural network)**.** *An artificial neural network (ANN, or simply NN) is a computing system that attempts to identify underlying relationships in a set of data, by mimicking the way a biological brain works. It is composed of a population of (artificial) neurons arranged in layers that process information (inputs, which can be seen as* stimuli*) coming from the external world.*

We now proceed to giving a necessary mathematical background on NNs, and how they are trained and evaluated.

Computation performed by a neuron. Each neuron of which a NN is composed accepts n_I real-valued inputs $\mathbf{x} = (x_1, \ldots, x_{n_I}) \in \mathbb{R}^{n_I}$ and performs the following two computations:

1. It computes a value $y = \sum_{i=1}^{n_I} w_i x_i + \beta \in \mathbb{R}$, which is a weighted sum of the inputs with real values called *weights* (w_i is the weight associated to the input x_i), and $\beta \in \mathbb{R}$, which is referred to as the *bias* of the neuron;

2. It applies a non-linear function f, called the *activation function*, and returns $f(y)$.

A neuron's output can conveniently be written as $f(\langle \mathbf{w}, \mathbf{x} \rangle) = \sum_{i=0}^{n_I} w_i x_i$ if one extends the inputs and the neuron's weight vectors by setting $\mathbf{w} = (\beta, w_1, \ldots, w_{n_I})$ and $\mathbf{x} = (1, x_1, \ldots, x_{n_I})$. In the following, we will use $\mathbf{W}_\ell = (\mathbf{w}_{\ell,1}, \ldots, \mathbf{w}_{\ell,n_2})$ to denote the $n_1 \times n_2$ matrix composed of all the weights associated to the connections that go from layer $\ell - 1$ (containing n_1 neurons) to layer ℓ (containing n_2 neurons). The vectors $\mathbf{w}_{\ell,i}$'s will contain the weights associated to the i-th neuron of the ℓ-th layer. When computing the output given by the network when presented with an input pattern x, we will usually write $\mathcal{N}(x)$.

Neural networks could in principle be *recurrent systems* (see, e.g., [Hop88; Jor89; Elm90]), as opposed to the purely feed-forward ones, where each neuron is evaluated only once. In this work, we consider only feed-forward networks, as they are more widely used, easier to understand, and simpler to evaluate homomorphically.

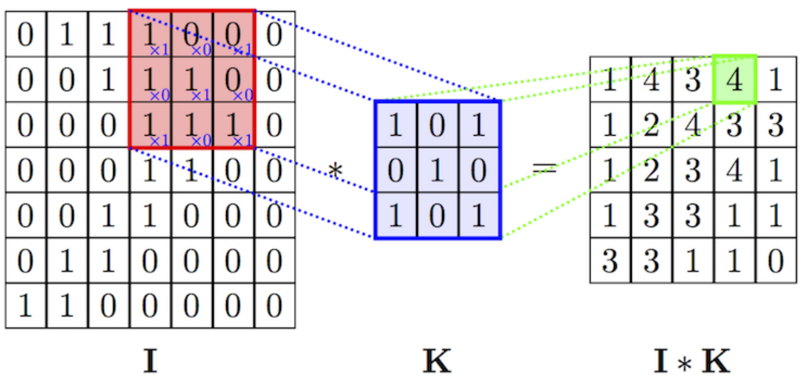


Figure 4.1: Example of convolution between an input **I** and a kernel **K**.

4.2.2 Neural networks' layers

The neurons of a NN are organized in successive layers, which are categorized according to their activation function. Neurons of one layer are connected to the neurons of the next layer by paths that are associated to weights. An input layer composed of the network's inputs, as well as an output layer made of the network's output values, are also added to the network. Internal layers are called *hidden*, since they are not directly accessible from the external world. Neural networks are usually composed of layers of various types. The "fusion" of diverse building blocks helps unlocking the full potential of neural networks, while keeping their size reasonable. We now provide a short description of the most widely used layer types:

Fully connected layer: in this kind of layer, every neuron takes all incoming signals as inputs.

Convolutional layer: this layer's learnable parameters consist of (a set of) small filters, called *kernels*, which are nothing more than a small matrix of values. The layer's output is then computed by "sliding" these matrices along the layer's input and computing the dot product between the input and the kernel. In Figure 4.1 we show an example of this computation. Note that this kind of layer *can reduce the size of the data*, depending on how the borders are handled (e.g., one could cut them, or pad them with zeros, or adopt other strategies).

Max pooling layer: this layer has no learnable parameters. Given a window of size $d_1 \times d_2$, one simply slides the window along the layer's input and takes as output the maximum among the $d_1 \cdot d_2$ input values. Note that this kind of layers *reduces the size* of the data, since it maps $d_1 \cdot d_2$ values to just one.

Mean pooling layer: this layer is similar to the max pooling one, except that it takes the average of the values instead of their max. Just like the previous kind of layer, it reduces the size of the data.

4.2.3 Activation functions

A key ingredient for the construction of an effective neural network is the choice of a neuron's activation function. This essentially determines how the neuron reacts to the inputs and what kind of information it forwards to the following layers. Roughly speaking, it serves as a mean to highlight some characteristics of the data over some others: for example, we might be interested in just separating positive values from negative ones (sign function), or we might want to give importance to whether the value's magnitude exceeds a certain threshold or not (function with a saturation point), and so forth. Several activation functions have been studied and used in the literature: we now list some of them and try to briefly outline some of their features.

Step: for any input $x \in \mathbb{R}$, this function is 0 if $x < 0$ and $+1$ if $x \geq 0$. It only discriminates between positive and negative values, completely disregarding their magnitudes. It presents an inherent problem, which is the fact that its derivative is zero almost everywhere. The reason why this represents a problem will be clear in Section 4.2.5, when we will outline a classical method for training a neural network.

Sign: for any input $x \in \mathbb{R}$, this function is -1 if $x < 0$ and $+1$ if $x \geq 0$. It is very similar to the step function, with the exception that negative values here do give a contribution, which is the opposite of that given by positive values. Like the previous one, its derivative is zero almost everywhere.

Sigmoid (or logistic): for any input $x \in \mathbb{R}$, this function is defined as

$$f(x) = \frac{1}{1+e^{-x}} = \frac{e^x}{e^x+1}.$$

It is continuous, bounded in the range $(0, 1)$, differentiable, and has a non-negative derivative at each point. This function can be seen as a continuous approximation of the step function.

Hyperbolic tangent: for any input $x \in \mathbb{R}$, this function is defined as

$$f(x) = \tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}} = \frac{e^{2x}-1}{e^{2x}+1}.$$

It is continuous, bounded in the range $(-1, 1)$, differentiable, and has a non-negative derivative at each point. This function can be seen as a continuous approximation of the sign function.

Arctangent: for any input $x \in \mathbb{R}$, this function is defined as $f(x) = \arctan(x)$. It is continuous, bounded in the range $(-\frac{\pi}{2}, \frac{\pi}{2})$, differentiable, and has a non-negative derivative at each point. This function can be seen as a scaled continuous approximation of the step function.

Square: for any input $x \in \mathbb{R}$, this function is defined as $f(x) = x^2$. Although less common than other functions, it has been employed, e.g., in [DGL+16]. It is continuous, non-negative, differentiable, and even (i.e., $f(-x) = f(x) \forall x \in \mathbb{R}$). This function ignores the sign of the input and only considers its magnitude.

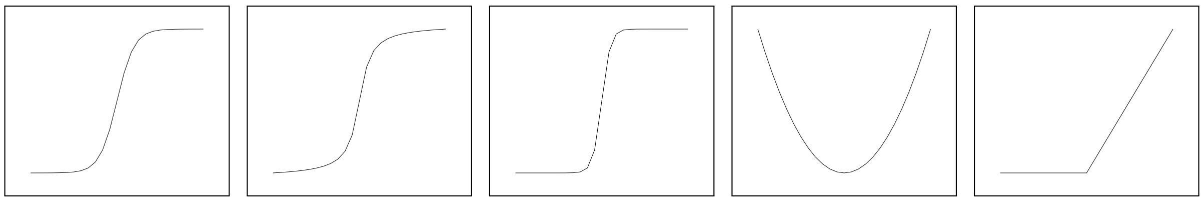

(a) sigmoid (x) (b) arctan (x) (c) tanh (x) (d) x^2 (e) ReLU (x)

Figure 4.2: Some possible activation functions for a neuron.

Rectifier: for any input $x \in \mathbb{R}$, this function is defined as $f(x) = \mathsf{ReLU}(x) = \max(0, x)$. It has recently become one of the most widely used activation functions in the machine learning community (e.g., [NH10; MHN13; ZRM+13]). It has the characteristic of propagating positive values untouched, while completely disregarding negative ones. An important thing to remember when using this function (and especially when implementing neural networks in software) is that it is *unbounded*, thus applying it multiple times without any form of rescaling or data normalization can quickly increase the values and produce overflows.

Softmax: this is a special kind of activation function since, unlike the previous ones, it is not defined on a single value but on a vector of values. It can be seen as a generalization of the logistic function, and it maps a vector $\mathbf{x} \in \mathbb{R}^d$ to a vector $\mathsf{softmax}(\mathbf{x}) \in (0,1)^d$, such that its values add up to 1. Formally, given a vector $\mathbf{x} = (x_1, \dots, x_d) \in \mathbb{R}^d$, this function is defined as

$$\mathsf{softmax}(\mathbf{x})_i = \frac{e^{x_i}}{\sum_{j=1}^{d} e^{x_j}}.$$

$\mathsf{softmax}$ is often used in *multiclass classification* problems, i.e., when the goal is to classify input patterns into one of N_C classes (with $N_C \geq 3$), and has the effect of converting "scores" into *probabilities*, while having no effect on the ordering of the values. Also, given the presence of the exponential function, it has the effect of "spreading" the outputs, by assigning high probabilities to the heaviest values and separating them from the others. Given these features, it is mostly used for the output layer of a neural network, when the goal is to output the probability that the pattern belongs to class C_i, for $i \in [N_C]$.

Some commonly used activation functions are plotted in Figure 4.2.

4.2.4 Perceptrons, multilayer perceptrons, and deep NNs

Historically, one of the first results towards building neural networks was the *perceptron*, introduced by Frank Rosenblatt [Ros57]. A perceptron is essentially a neural network composed solely of an input layer and an output layer, i.e., without any hidden layer. Rosenblatt proved that a perceptron converges to the correct solution in a finite number of steps, provided that a correct solution exists. However, the perceptron model is greatly limited by the fact that it can learn only *linearly separable* functions so, for example, it can learn the

AND and the OR function, but not the XOR. Also, this limitation makes it useless for classification problems with three or more classes.

Decisive progress in the field of machine learning was made with the introduction of the *multilayer perceptron*, i.e., a neural network with at least one hidden layer. A famous result, known as the *universal approximation theorem* (see, e.g., [Cyb89; Hor91]), states the following:

Theorem 4.2.2 (Universal approximation theorem (informal))**.** *A neural network with a single hidden layer that contains a finite amount of neurons can approximate any continuous function.*

It is then clear that neural networks have a huge potential, but this theorem does not specify precisely *how many* neurons there must be in the single hidden layer of the network. It turns out that, for complex function, this quantity can grow exponentially and quickly make the problem of training and evaluating the model infeasible.

Deep neural networks. A way to solve this issue is represented by *deep neural networks* (deep NNs), i.e., neural networks that comprise several hidden layers stacked one on top of the other (for a visual example, see Figure 4.3). These layers of non-linearities allow the network to extract increasingly complex features of the input patterns and can lead to a better ability to generalize (i.e., to learn the underlying structures and relationships in the dataset), especially in the case of more complex tasks.

The term "deep learning" was introduced to the machine learning community in 1986, and to artificial neural networks in 2000, but it was only starting from 2006 that, mainly thanks to the works of Hinton and Salakhutdinov (e.g., [HS06]), this area became a prominent field of research. More specifically, Salakhutdinov (advised by Hinton) has the merit of shifting the computationally intense process of training a deep network from a CPU to a GPU, thus taking advantage of the large number of cores available on these boards and, in turn, fostering the development of specifically designed hardware for machine learning. From that moment, deep learning has been applied with impressive results to a vast number of problems: from speech recognition (e.g., [HDY+12; Rav17; ZPB+17]), to image classification (see e.g., [LBBH98; Kag]), from adding colors to black & white images (e.g., [Avn16]), to even creating new astonishing works of art (e.g., [MOT15a; MOT15b]). The potential of deep learning is considered enormous, and new applications surface regularly, with active research still happening in the field.

4.2.5 Training and evaluating neural networks

In this section we give an overview of the algorithms for training and evaluating a neural network, i.e., we explain how to construct a model from a training set and later evaluate its performance on a test set.

The way a neural networks extracts (or approximates) the underlying relationships in a set of data is by *learning*. This means that the network is presented with several examples, i.e., several input patterns and the corresponding expected outputs, and learns a mapping from the input space to the output space. It is worth noting that this goes well beyond the simple storage and successive retrieval of information. In other words, the networks does not (or, at least, should not) learn the example "by heart", but it should be able to *generalize*, i.e., extract general relationships that pertain to the data and not to the specific

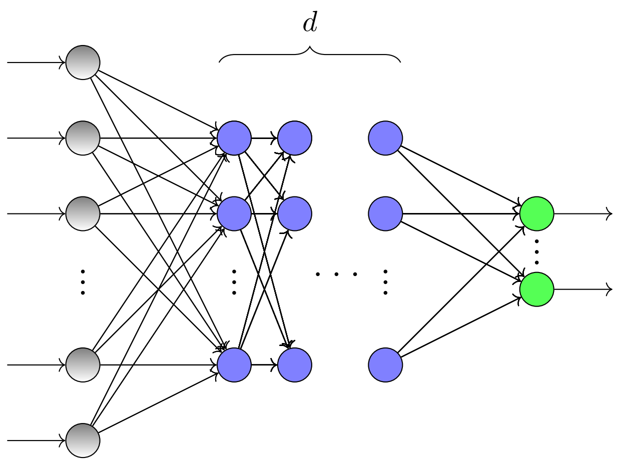


Figure 4.3: A generic feed-forward neural network of arbitrary depth d: one input layer, followed by d hidden layers of variable size, and one output layer.

examples. In fact, once trained, the network should be able to give correct answers even when presented with input patterns that were not used during the training phase. It is simple to see an analogy with the way a human brains evolves during the course of a person's life: through examples and by experiencing different situations, it learns how to behave and how to respond to new stimuli.

For a neural network, we can distinguish between *supervised* and *unsupervised* learning: in the first case, the network is given a set of pairs composed of an input and an expected output, whereas in the second case, the network is only given input patterns and has to organize itself in order to infer a function that describes hidden structures in the data. In this work, we will focus exclusively on supervised learning.

Error (or cost) function. Given a neural network, we can define a function $L(\widehat{y}, y)$ that measures some notion of error between the expected output $\widehat{y}$ and the actual network's output y. This function is sometimes referred to as the *cost* function or the *loss* function. Choosing an appropriate loss function is extremely important, as this defines the quantity that the training phase will try to minimize. We now present some typical loss functions that can be used when training a neural network; in the following, N indicates the number of training samples, i.e., the number of pairs (input, expectedOutput) in the training set.

L_1**:** this is one of the simplest loss functions, and it simply measures how far the predicted value is from the correct one:

$$L_1(\widehat{y}, y) = \sum_{i=1}^{N} |\widehat{y}_i - y_i|$$

L_2**:** similar to the previous one, but this time the differences are squared:

$$L_2(\widehat{y}, y) = \sum_{i=1}^{N} (\widehat{y}_i - y_i)^2$$

Mean Squared Error (MSE): similar to the previous one, but this time the value is averaged on the training set:

$$\mathsf{MSE}\left(\widehat{y}, y\right) = \frac{1}{N}\sum_{i=1}^{N}\left(\widehat{y}_i - y_i\right)^2$$

0/1-**loss:** this function gives a simple hit-or-miss measurement of the loss:

$$L\left(\widehat{y}_i, y_i\right) = \begin{cases} 0, & y_i = \widehat{y}_i \\ 1, & y_i \neq \widehat{y}_i \end{cases} \quad \forall i \in [N]$$

Cross-entropy (or log loss): this is one of the most well known and widely used cost functions in the machine learning field. It measures the performance of a classification model whose output is a probability value between 0 and 1.

$$L\left(\widehat{y}, y\right) = -\sum_{i=1}^{N}\widehat{y}_i \log\left(y_i\right)$$

Once an appropriate loss function has been chosen (perhaps through trial-and-error), it is possible to proceed with the real training. Here is an high-level overview of the procedure:

1. Present an input pattern x to the network;
2. Calculate the network output $y = \mathcal{N}\left(x\right)$ by feeding x forward and performing all the computations until the output layer;
3. Compute the loss $L\left(\widehat{y}, y\right)$;
4. Modify the weights of the network in order to reduce the error;
5. Repeat for all the samples in the training set.

Clearly, Item 4 is the key one, and the obvious question is *how* to modify the network parameters in order to reduce the loss. Roughly speaking, the answer is that the weights have to be moved in the direction opposed to that of the gradient of the loss function, calculated with respect to the weights. In fact, the final goal is to find a set of weights that minimize the loss. In Algorithm 1 we show a basic and unoptimized version of the *gradient descent* algorithm. It is easy to see that this procedure is rather inefficient: in fact, it needs to process *all* the N training samples *before* updating the weights; in other words, the weights are updated only M times, where M is the number of iterations of the algorithm, usually called *epochs*.

An alternative approach is that of updating the weights *after each training sample*. This technique goes by the name of stochastic gradient descent (SGD) and it is a method for approximating the "true" gradient by the gradient at a single training sample. This way, the weights are updated in a "less precise", heuristic way, but more frequently. This method is also used for the so-called *online learning*, i.e., when new training samples keep coming in, and they are used to continuously "update" the network.

A trade-off between these extremes is the *batched* gradient descent: in this case, the training set is divided into a number of batches (the size of which becomes a parameter for the training), and the weights are updated after processing each batch. This method offers a good compromise between the accuracy of the gradient computation and the speed of convergence of the procedure, making it widely used for training neural networks.

Several optimizations are possible: learning rate decay, weight regularization, data normalization, SGD with momentum, ... These enhancements are extremely useful for training effective networks and can often improve the results dramatically. For an exhaustive review on this topic, we refer the interested reader to one of the many resources available in the machine learning literature.

Algorithm 1 Gradient descent

```
    Input: a neural network $\mathcal{N}$ with $d$ hidden layers, a loss function $L$, training samples
    $(x_i, \widehat{y}_i)$ for $i \in [N]$, a prescribed number of iterations $M$, a learning rate $\varepsilon \in \mathbb{R}$
    Output: a trained neural network $\mathcal{N}^*$
 1: for all epoch $\in [M]$ do
 2:     for all $i \in [N]$ do
 3:         compute $y_i = \mathcal{N}(x_i)$ by feeding $x$ forward through the network
 4:         compute the loss $L(\widehat{y}_i, y_i)$
 5:         update $\Delta\mathbf{W}_{d+1} = \Delta\mathbf{W}_{d+1} + \frac{\partial L(\widehat{y}_i, y_i)}{\partial \mathbf{W}_{d+1}}$, i.e., the gradient of the loss w.r.t. the
    weights that connect the last hidden layer to the output layer
 6:         for $\ell = d, d-1, \ldots, 1$ do
 7:             update $\Delta\mathbf{W}_\ell = \Delta\mathbf{W}_\ell + \frac{\partial L(\widehat{y}_i, y_i)}{\partial \mathbf{W}_\ell}$ via the chain rule for derivatives
 8:         end for
 9:     end for
10:     for $j = 1, \ldots, d+1$ do
11:         update the weights by setting $\mathbf{W}_j = \mathbf{W}_j - \varepsilon \cdot \frac{\Delta\mathbf{W}_j}{N}$
12:     end for
13: end for
```

4.2.6 MNIST: a typical dataset for NNs

A classical problem for neural networks is that of image classification, i.e., classifying images based on what they depict. In particular, a well studied problem in the machine learning literature is that of digit recognition. The MNIST (Modified National Institute of Standards and Technology) dataset contains images representing digits that have been handwritten by more than 500 different writers, and is commonly used as a benchmark for machine learning systems [LBBH98]. It contains 60000 training images (i.e., used for building, or training, the model) and 10000 testing images (i.e., used for evaluating the model's performance). In Figure 4.4 we show, as an example, some of the images that compose the MNIST dataset. The format of the images is 28×28 and the value of each pixel represents an 8-bit level of gray. Moreover, each image is labeled with the digit is depicts; this serves both during the training phase for "telling the network" which is the expected output, and during the testing phase for verifying if the classification was correct or not. A typical neural network for the MNIST dataset has $28 \cdot 28 = 784$ input nodes (one per pixel), an arbitrary number of hidden layers, each of which is composed of an arbitrary number of neurons, and finally

Figure 4.4: Some images from the MNIST dataset

an output layer composed of 10 neurons (one per possible digit or class). The output values can be interpreted as scores[2] given by the network, with the predicted class being the one associated to the highest score.

Over the years, the MNIST dataset has been a typical benchmark for classifiers, and many approaches have been applied: linear classifiers, principal component analysis, support vector machines, neural networks, convolutional neural networks, etc. For a more complete review of these approaches, we refer the reader to, e.g., [LBBH98]. Neural networks are known to perform well on this dataset. For example, [LBBH98] proposes different architectures for neural networks and obtains more than 97% of correct classifications. More recent works even surpassed 99% of accuracy [CMS12]. For a nice overview on the results obtained on this dataset and on the techniques that were used, we refer the reader to [LCB98].

4.3 State of the art for privacy-preserving predictions

We now turn to the problem of homomorphically making predictions on input data. In a nutshell, given a model $\mathcal{N}$, the goal is to take an encryption of an input pattern $\mathsf{Enc}(x)$ and return an encrypted answer c such that $\mathsf{Dec}(c) = \mathcal{N}(x)$, i.e., the evaluation of the model $\mathcal{N}$ on the input x.

This problem has already been addressed in the past, mainly via multiparty computation (MPC), Yao's garbled circuits [Yao86], and homomorphic encryption (see, e.g., [BPTG15], SecureML[MZ17], PICS [MRSV17], MiniONN [LJLA17], DeepSecure [RRK17], Gazelle [JVC18]). In the case of MPC, a possible architecture is the following: a party A holds the input data x, a party B holds the trained model $\mathcal{N}$, and they engage in a protocol that, through several round of interactions, allows A to obtain $\mathcal{N}(x)$. The main drawback of this approach is essentially that it requires interactivity between the parties, meaning that there are several points in time where the client A is involved in the protocol, as opposed to a simpler case where A sends the input and later receives the output. Even though practical performances of MPC-based solutions have largely surpassed those of FHE-based solutions, they

[2]Or probabilities, if the $\mathsf{softmax}$ function is applied.

incur issues like network latencies and high bandwidth usage. Because of these downsides, FHE-based solutions appear more scalable for real-life applications, even though MPC-based solutions might be faster for specific instantiations.

As far as FHE-based solutions are concerned, Cryptonets [DGL+16] was the first initiative to address the challenge of achieving blind, non-interactive classification. The main idea consists in applying a leveled somewhat homomorphic encryption scheme such as BGV [BGV12] to the network inputs and propagating the signals across the network homomorphically, thereby consuming levels of homomorphic evaluation whenever non-linearities are met. We remind the reader that, in neural networks, non-linearities come from activation functions which are usually picked from a small set of non-linear functions of reference (logistic sigmoid, hyperbolic tangent, ...) chosen for their mathematical convenience. To optimally accommodate the underlying SHE scheme, Cryptonets replace their standard activation by the (depth 1) square function, which only consumes one level of homomorphic multiplication but does not resemble the typical sigmoidal shape. A number of subsequent works have followed the same approach and improved it, typically by adopting higher degree polynomials as activation functions for more training stability [ZYC16], or by renormalizing weighted sums prior to applying the approximate function, so that its degree can be kept as low as possible [CWM+17]. Practical experiments have shown that training can accommodate approximated activations and generate NNs with very good accuracy.

However, this approach suffers from an inherent limitation: the homomorphic computation, local to a single neuron, depends on the total number of levels required to implement the network, which is itself roughly proportional to the number of its activated layers. Therefore, the overall performance of the homomorphic classification heavily depends on the total multiplicative depth of the circuit and rapidly becomes prohibitive as the number of layers increases. This approach does not scale well and is not adapted to deep neural networks, that can contain tens, hundreds or sometimes thousands of layers [HZRS15; ZK16].

4.4 TFHE: a framework for efficient bootstrapping

In 2016, Chillotti *et al.* proposed a framework [CGGI16b] (later refined in [CGGI17]) that improved dramatically the performances of FHE and, particularly, those of the bootstrapping operation. These works propose an alternative – although equivalent to the traditional one[3] – representation of the LWE problem (cf. Section 2.3.6) over the torus (i.e., the reals modulo 1) and feature an improved bootstrapping procedure that is considerably more efficient than the previous state of the art. In the following, we review some of the contributions introduced by these works, that will be of key importance for our constructions. For all the details, we refer the reader to [CGGI16b; CGGI17].

4.4.1 LWE over the torus and related constructions

In this section we present the formulation of the LWE problem over the torus, and we introduce related constructions that will be used in the rest of this chapter.

LWE over the torus. First, we provide a definition of the LWE problem [Reg05] over the torus $\mathbb{T} = \mathbb{R}/\mathbb{Z}$.

[3]If the framework is implemented with precision of φ bits, then switching from one representation to the other is just a matter of multiplying or dividing by 2^{φ} and rounding.

Definition 4.4.1 (LWE over the torus)**.** *Let n be a positive integer and χ be a probability distribution over $\mathbb{R}$ for the noise. For any vector $\mathbf{s} \in \{0,1\}^n$, the LWE distribution $\mathsf{lwe}_{\mathbf{s},\chi}$ over $\mathbb{T}^n \times \mathbb{T}$ is sampled by picking $\mathbf{a} \leftarrow_{\$} \mathbb{T}^n$, an error $e \leftarrow \chi$, and returning $(\mathbf{a}, b = \langle \mathbf{s}, \mathbf{a} \rangle + e)$. Then the LWE assumption states that, for some fixed $\mathbf{s} \in \{0,1\}^n$, it is hard to distinguish between $(\mathbf{a}, b) \leftarrow \mathsf{lwe}_{\mathbf{s},\chi}$ and $(\mathbf{u}, v) \leftarrow_{\$} \mathbb{T}^{n+1}$.*

From now on, when mentioning the LWE problem, we will refer to this formulation over the torus.

TLWE. TLWE is a generalization of LWE and Ring-LWE [LPR10].

Definition 4.4.2 (TLWE)**.** *Let $k \geq 1$ be an integer, N be a power of 2 and χ be an error distribution over $\mathbb{R}_N[X]$. A TLWE secret key $\bar{\mathbf{s}} \in \mathbb{B}_N[X]^k$ is a vector of k polynomials over $\mathbb{Z}_N[X]$ with binary coefficients. The TLWE distribution $\mathsf{tlwe}_{\bar{\mathbf{s}},\chi}$ is sampled by picking $\mathbf{a} \leftarrow_{\$} \mathbb{T}_N[X]^k$, an error $e \leftarrow \chi$, and returning $(\mathbf{a}, b = \bar{\mathbf{s}} \cdot \mathbf{a} + e)$. Given a message encoded as a polynomial $\mu \in \mathbb{T}_N[X]$, a fresh TLWE encryption of μ under the key $\bar{\mathbf{s}}$ is obtained by taking $(\mathbf{a}, b) + (\mathbf{0}, \mu)$, where $(\mathbf{a}, b) \leftarrow \mathsf{tlwe}_{\bar{\mathbf{s}},\chi}$, and $\mathbf{0}$ is a vector of k zero-polynomials.*

Remark 4.4.3. *Notice that if N is large and $k = 1$, then TLWE becomes Ring-LWE with binary secret, whereas if $N = 1$ and k is large, then TLWE becomes the standard LWE problem with binary secret.*

With this definition, one can define the search and the decision versions of the TLWE problem, analogously to what was done for LWE in Definitions 2.3.21 and 2.3.22.

TGSW. TGSW is a generalized version of the GSW FHE scheme [GSW13; AP14] (cf. Section 3.4.3). It can roughly be seen as the matrix version of the TLWE scheme, just like GSW can be seen the matrix-equivalent of LWE. A key difference is in the gadget decomposition, which in TGSW is approximated, whereas in the classical GSW cryptosystem this operation is exact.

Definition 4.4.4 (Approximate gadget decomposition [CGGI16b, Definition 3.7])**.** *Let $\mathbf{H} \in \mathbb{T}_N[X]^{(k+1)\ell \times (k+1)}$ be as in Equation (4.1). We say that $Dec_{\mathbf{H},\beta,\varepsilon}(\mathbf{v})$ is a decomposition algorithm on the gadget $\mathbf{H}$ with quality β and precision ε if and only if for any TLWE sample $\mathbf{v} \in \mathbb{T}_N[X]^{k+1}$, it publicly and efficiently outputs a small vector $\mathbf{u} \in \mathcal{R}^{(k+1)\ell}$ such that $\|\mathbf{u}\|_\infty \leq \beta$ and $\|\mathbf{u} \cdot \mathbf{H} - \mathbf{v}\|_\infty \leq \varepsilon$. Furthermore, the expectation of $\mathbf{u} \cdot \mathbf{H} - \mathbf{v}$ must be* 0 *when $\mathbf{v}$ is uniformly distributed in $\mathbb{T}_N[X]^{k+1}$.*

$$\mathbf{H} = \begin{pmatrix} 1/B_g & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 1/B_g^\ell & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & 1/B_g \\ \vdots & \ddots & \vdots \\ 0 & \cdots & 1/B_g^\ell \end{pmatrix} \in \mathbb{T}_N[X]^{(k+1)\ell \times (k+1)}. \tag{4.1}$$

Definition 4.4.5 (TGSW [CGGI16b, Definition 3.9])**.** *Let ℓ and $k \geq 1$ be two integers, $\alpha \geq 0$ be a noise parameter, $\mathbf{H}$ the gadget defined in Equation (4.1), and let $\mathbf{s} \in \mathbb{B}_N[X]^k$ be a TLWE* *key. We say that* $\mathbf{C} \in \mathbb{T}[X]^{(k+1)\ell \times (k+1)}$ *is a fresh TGSW sample of a polynomial* $\mu \in \mathcal{R}/\mathbf{H}^{\perp}$ *with noise parameter* α *if and only if* $\mathbf{C} = \mathbf{Z} + \mu\mathbf{H}$*, where each row of* $\mathbf{Z} \in \mathbb{T}_N[X]^{(k+1)\ell \times (k+1)}$ *is a TLWE encryption of the zero polynomial with Gaussian noise parameter* α*.*

From a TLWE encryption $\bar{c}$ of a polynomial $\mu \in \mathbb{T}_N[X]$ under a TLWE key $\bar{\mathbf{s}}$ we can extract a LWE encryption $c' = \mathsf{Extract}(\bar{c})$ of the constant term of μ under an extracted key $\mathbf{s}' = \mathsf{ExtractKey}(\bar{\mathbf{s}})$. For the details of the algorithms Extract and ExtractKey, we refer the reader to [CGGI16b, Definition 4.1].

4.4.2 External product and bootstrapping procedure

The core idea for the efficiency of the new bootstrapping procedure is the so-called external product $\boxdot$, that performs the following mapping:

$$\boxdot : \text{TGSW} \times \text{TLWE} \to \text{TLWE}.$$

Roughly speaking, the external product of a TGSW encryption of a polynomial $\mu_1 \in \mathbb{T}_N[X]$ and a TLWE encryption of a polynomial $\mu_2 \in \mathbb{T}_N[X]$ is a TLWE encryption of $(\mu_1 \cdot \mu_2) \in \mathbb{T}_N[X]$.

Now the bootstrapping procedure of an n-LWE sample (here, n denotes the dimension) consists of the following three functions:

BlindRotate: $\text{TGSW}^n \times \text{TLWE} \times n\text{-LWE} \to \text{TLWE}$
: On input TGSW encryptions of $(s_i)_{i\in[n]}$, a (possibly noiseless) TLWE encryption of testVector and an n-LWE sample $(\mathbf{a}, b)$, computes a TLWE encryption of $X^{\phi} \cdot$ testVector, where $\phi = b - \langle \mathbf{s}, \mathbf{a} \rangle$;

Extract: $\text{TLWE} \to N\text{-LWE}$
: On input a TLWE encryption of polynomial $\mu \in \mathbb{T}_N[X]$, computes an N-LWE encryption of the constant term $\mu(0)$;

KeySwitch: $n\text{-LWE}^N \times N\text{-LWE} \to n\text{-LWE}$
: On input n-LWE encryptions of $(s'_i)_{i\in[N]}$, and an N-LWE sample $(\mathbf{a}, b)$ computes an n-LWE encryption of $b - \langle \mathbf{s}', \mathbf{a} \rangle$.

Then we can define a function $\mathsf{Bootstrap}(\cdot, \cdot, \cdot)$ that takes as input a bootstrapping key bk, a keyswitching key ksk, and a ciphertext and outputs a new ciphertext. Roughly speaking,

$$\mathsf{Bootstrap} = \mathsf{KeySwitch} \circ \mathsf{Extract} \circ \mathsf{BlindRotate}.$$

We note that BlindRotate works on LWE samples with values in $[2N]$ instead of $\mathbb{T}$, thus the first step is to map $\mathbb{T}$ to $[2N]$ by multiplying and rounding.

4.5 Our contributions

In this section we present our contributions for privacy-preserving evaluation of neural networks.

As already stated, the biggest limitation of using a somewhat homomorphic encryption scheme and then evaluating non-linear activation functions is that the overall efficiency of the scheme depends on the total multiplicative depth that is needed to evaluate the *entire* neural network. This way, choosing the parameters for the scheme quickly becomes prohibitive as the depth of the network increases.

Instead, we propose a *scale-invariant* approach to the problem, i.e., each neuron's output is refreshed through bootstrapping, resulting in that arbitrarily deep networks can be homomorphically evaluated. Of course, the entire homomorphic evaluation of the network will still take time proportional to the number of its neurons or, if parallelism is involved, to the number of its layers. However, evaluating one neuron is now essentially independent of the dimensions of the network: it just relies on system-wide parameters.

We start by proposing a new model of neural network, which is designed to be FHE-friendly, and we explain the procedure to homomorphically evaluate it on encrypted inputs; finally, we present some improvements on the TFHE framework and show experimental results obtained on the MNIST dataset.

4.5.1 Definition of a discretized neural network

First of all, we recall that state-of-the-art fully homomorphic encryption schemes cannot support operations over real messages. Traditional neural networks have real-valued weights, and this incompatibility motivates investigating alternative architectures.

Definition 4.5.1. *A Discretized Neural Network (DiNN) is a feed-forward artificial neural network whose inputs are integer values in $\{-I, \dots, I\}$ and whose weights are integer values in $\{-W, \dots, W\}$, for some $I, W \in \mathbb{N}$. For every neuron of the network, the activation function maps the inner product between the incoming inputs vector and the corresponding weights to integer values in $\{-I, \dots, I\}$.*

In particular, for a first attempt we chose $\{-1, 1\}$ as the input space and $\mathsf{sign}(\cdot)$ as the activation function for the hidden layers:

$$\mathsf{sign}(x) = \begin{cases} -1, & x < 0, \\ +1, & x \geq 0. \end{cases} \tag{4.2}$$

These choices are inspired by the fact that the model was designed with the idea of performing homomorphic evaluations over encrypted input. As a consequence, we wanted the message space to be as small as possible, which, in turn, would allow us to increase the efficiency of the overall evaluation.

We also note that using an activation function whose output is in the same range as the network's input allows us to maintain the same semantics across different layers. In our case, what enters a neuron is always a weighted sum of values in $\{-1, 1\}$. In order to make the evaluation of the network compatible with FHE schemes, discretizing the input space is not sufficient: we also need to have discrete values for the weights of the network[4].

[4]As all the computations are done over the torus (i.e., modulo 1), scaling a ciphertext by any integer factor preserves the relations that make the decryption correct. However, this does not hold for non-integer factors.

4.5.2 Simple conversion from a traditional NN to a DiNN

In this subsection we show a very simple method to convert an already-trained canonical neural network (i.e., with real weights) into a DiNN. This method is not guaranteed to be the best way to obtain such a conversion; it indeed introduces a visible loss in the classification accuracy and would probably be best used as a first step in the conversion procedure. However, we remind the reader that this work is aimed at the homomorphic evaluation of a network, thus we decided not to put too much effort in the construction of a sophisticated cleartext model. This procedure allows us to obtain a network which respects our constraints and that can be evaluated over encrypted inputs, so it is sufficient for our purposes.

It turns out that the only thing that we need to do is discretizing the weights and biases of the network. To this purpose, we define the function

$$\mathsf{processWeight}\left(w, \tau\right) = \tau \cdot \left\lfloor \frac{w}{\tau} \right\rceil \tag{4.3}$$

where $\tau \in \mathbb{N}$ is a parameter that controls the precision of the discretization. In the following, we implicitly take all the weights as discretized after being processed through the formula in Equation (4.3). After fixing a value τ, the network obtained by applying $\mathsf{processWeight}\left(\cdot, \tau\right)$ to all the weights and biases is a DiNN. The parameter τ has to be chosen carefully, since it defines the message space that our encryption scheme must support. Thus, we want the bound on $\langle \mathbf{w}, \mathbf{x} \rangle$ to be small for all neurons, where $\mathbf{w}$ and $\mathbf{x}$ are the discretized weights and the inputs associated to the neuron, respectively. In Figure 4.5, we show the evaluation of a single neuron: we first compute $\langle \mathbf{w}, \mathbf{x} \rangle$, which we refer to as a *multisum*, and then apply the sign function to the result.

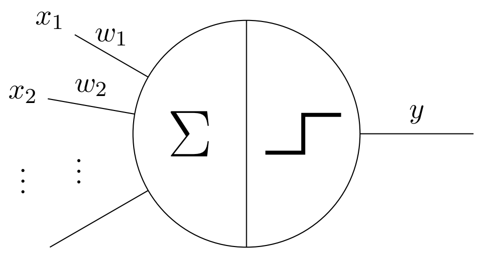


Figure 4.5: Evaluation of a single neuron. The output value is $y = \mathsf{sign}\left(\langle \mathbf{w}, \mathbf{x} \rangle\right)$, where w_i are the discretized weights associated to the incoming wires and x_i are the corresponding input values.

4.5.3 Homomorphic evaluation of a DiNN

We now give a high level description of our procedure to homomorphically evaluate a DiNN, called FHE-DiNN. We basically need two ingredients: we need to be able to compute the multisum between the encrypted inputs and the weights and we need to homomorphically extract the sign of the result. In order to maintain the scalability of our scheme across the layers of a given DiNN, we perform a bootstrapping operation for every neuron in hidden layers. This ensures that the ciphertext encrypting the sign of the result after applying one layer of the DiNN can be used for further computations without an initially fixed limit on the number of layers that the network can contain. Hence we can choose parameters that are independent of the number of layers and evaluate arbitrarily deep neural networks.

4.5.3.1 Evaluating the multisum

In our framework, the weights of the network are available in clear, so we can evaluate the multisum just by using homomorphic additions. The only things that need our attention are the message space of our encryption scheme, which has to be large enough to accommodate for all possible values of the multisums, and the noise level that might grow too much and lead to incorrect results.

Extending the message space. In order for our FHE scheme to be able to correctly evaluate the multisum, we need all the possible values of the multisum to be inside our message space. To this end, we extend our LWE encryption scheme as follows. This idea was already used in previous works such as [PW08; KTX08; ABDP15; ALS16].

Construction 4.5.2 (Extended LWE-based private-key encryption scheme)**.** *Let B be a positive integer and let $m \in [-B, B]$ be a message. Then we split the torus into $2B+1$ slices, one for each possible message, and we encrypt and decrypt as follows:*

$\mathsf{Setup}(\kappa)$*: for a security parameter κ, fix $n = n(\kappa), \sigma = \sigma(\kappa)$; return $\mathbf{s} \leftarrow_{\$} \mathbb{T}^n$*

$\mathsf{Enc}(\mathbf{s}, m)$*: return $(\mathbf{a}, b)$, with $\mathbf{a} \leftarrow_{\$} \mathbb{T}^n$ and $b = \langle \mathbf{s}, \mathbf{a} \rangle + e + \frac{m}{2B+1}$, where $e \leftarrow \chi_\sigma$*

$\mathsf{Dec}(\mathbf{s}, (\mathbf{a}, b))$*: return $\lfloor (b - \langle \mathbf{s}, \mathbf{a} \rangle) \cdot (2B+1) \rceil$*

An input message is mapped to the center of its corresponding torus slice by scaling it by $1/(2B+1)$ during encryption, and decoded by scaling it by $2B+1$ during decryption.

Correctness of homomorphically evaluating the multisum. Note that ciphertexts can be homomorphically added and scaled by a known integer constant: for any two messages $m_1, m_2 \in [-B, B]$, any secret key $\mathbf{s}$, any $c_1 = (\mathbf{a}_1, b_1) \leftarrow \mathsf{Enc}(\mathbf{s}, m_1)$, $c_2 = (\mathbf{a}_2, b_2) \leftarrow \mathsf{Enc}(\mathbf{s}, m_2)$, and constant $w \in \mathbb{Z}$, we have that

$$\mathsf{Dec}(\mathbf{s}, c_1 + w \cdot c_2) = \mathsf{Dec}(\mathbf{s}, (\mathbf{a}_1 + w \cdot \mathbf{a}_2, b_1 + w \cdot b_2)) = m_1 + w \cdot m_2$$

as long as (1) $m_1 + w \cdot m_2 \in [-B, B]$, and (2) the noise did not grow too much.

The first condition is easily met by choosing $B \geq \|\mathbf{w}\|_1$ for all weight vectors $\mathbf{w}$ in the network (e.g., we can take the max).

Fixing the noise. Increasing the message space has an impact on the choice of parameters. Evaluating the multisum with a given weight vector $\mathbf{w}$ means that, if the standard deviation of the initial noise is σ, then the standard deviation of the output noise can be as high as $\|\mathbf{w}\|_2 \cdot \sigma$ (see Lemma 2.3.17), which in turn means that our initial standard deviation must be smaller than the one in [CGGI16b] by a factor $\max_{\mathbf{w}} \|\mathbf{w}\|_2$. Moreover, for correctness to hold, we need the noise to remain smaller than half a slice of the torus. As we are splitting the torus into $2B+1$ slices rather than 2, we need to further decrease the noise by a factor B. Special attention must be paid to security: taking a smaller noise might in fact compromise the security of the scheme. In order to mitigate this problem, we can increase the dimension of the LWE problem n, but this in turn induces more noise overhead in the bootstrapping procedure due to rounding errors.

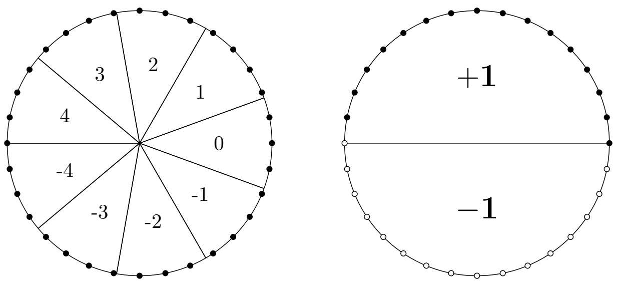


Figure 4.6: On the left, we show the first step of the bootstrapping, which consists in mapping the torus (the continuous circle) to the wheel (the $2N$ ticks on it) by rounding to the closest tick. Each slice corresponds to one of the possible results of the multisum operation. On the right we show the final result of the bootstrapping: each tick of the top part of the wheel is mapped to its sign which is $+1$ and each tick of the bottom part to -1. This can roughly be seen as embedding the wheel back to the torus.

4.5.3.2 Homomorphic computation of the sign function

We take advantage of the flexibility of the bootstrapping technique introduced by Chillotti *et al.* [CGGI16b] in order to perform the sign extraction and the bootstrapping at the same time. Concretely, in the call to BlindRotate, we change the value of testVector to

$$\frac{-1}{2B+1}\sum_{i=0}^{N-1} X^i.$$

Then, if the value of the phase $b - \langle \mathbf{s}, \mathbf{a} \rangle$ is between 1 and N (positive), the output will be an encryption of 1, otherwise if it is between $N+1$ and $2N$ (negative), the output will be an encryption of -1.

In order to give more intuition, we present an illustration of the bootstrapping technique in Figure 4.6. The first step of the bootstrapping basically consists in mapping the torus $\mathbb{T}$ to an object that we will refer to as the wheel. This wheel is split into $2N$ "ticks" that are associated to the possible values that are encrypted in the bootstrapped ciphertext. The bootstrapping procedure then consists in choosing a value for each tick, rotating the wheel by $b - \langle \mathbf{s}, \mathbf{a} \rangle$ ticks counter-clockwise, and picking the value of the rightmost tick. We note that the values on the wheel are encoded in the testVector variable, which contains values for the ticks on the top part of the wheel. The bottom values are then fixed by the anticyclic property of $\mathbb{T}_N[X]$ (the value at tick $N+i$ is minus the value at tick i).

From now on, we say that a bootstrapping is *correct* if, given a valid encryption of a message μ, its output is a valid encryption of $\mathsf{sign}(\mu)$ with overwhelming probability.

4.5.3.3 Scale-invariance

If the parameters are set correctly then, by using the two operations described above, we can homomorphically evaluate neural networks of any depth. In particular, the choice of parameters is independent of the depth of the neural network. This result cannot be achieved with previous techniques relying on somewhat homomorphic evaluations of the network. In fact, they have to choose parameters that accommodate for the whole computation, whereas our method only requires the parameters to accommodate for the evaluation of a single neuron. The rest of the computation follows by induction. More precisely, our choice of parameters only depends on bounds on the norms ($\|\cdot\|_1$ and $\|\cdot\|_2$) of the input weights of a neuron. In the following, we denote these bounds by M_1 and M_2, respectively.

We say that the homomorphic evaluation of the neural network is *correct* if the decryptions of its output scores are equal to the scores given by its evaluation in the clear with overwhelming probability. Then, the scale-invariance is formally defined by the following theorem:

Theorem 4.5.3 (Scale-invariance of our homomorphic evaluation)**.** *For any* DiNN *of any depth, any correctly generated bootstrapping key* bk *and keyswitching key* ksk*, and any ciphertext c, let σ be a Gaussian parameter such that the noise of* $\mathsf{Bootstrap}(\mathsf{bk}, \mathsf{ksk}, c)$ *is sub-Gaussian with parameter σ. Then, if the bootstrapping is correct on input ciphertexts with sub-Gaussian noise of parameter $\frac{\sigma}{M_2}$ and message space larger than $2M_1 + 1$, the result of the homomorphic evaluation of the* DiNN *is correct.*

Proof. The proof is a simple induction on the structure of the neural network. First, the correctness of the evaluation of the first layer is implied by the choice of parameters for the encryption[5].

If the evaluation is correct for all neurons of the ℓ-th layer, then the correctness for all neurons of the $(\ell + 1)$-th layer follows from the two observations made in the previous subsections:

- The result of the homomorphic evaluation of the multisum is a valid encryption of the multisum;
- The result of the bootstrapping is a valid encryption of the sign of the multisum.

The first fact is implied by the choice of the message space, since the multisum value is contained in $[-M_1, M_1]$. The second one comes directly from the correctness of the bootstrapping, because the homomorphic computation of the multisum on ciphertexts with sub-Gaussian noise of parameter σ yields a ciphertext with sub-Gaussian noise of parameter at most σM_2 (cf. Theorem 2.3.17).

Then, the correctness of the encryption scheme ensures that the final ciphertexts are valid encryptions of the scores. □

4.5.4 Refinements of TFHE

In this section, we present several improvements that helped us achieving better efficiency for the actual FHE-DiNN implementation. These various techniques can without any doubt be applied in other FHE-based applications.

4.5.4.1 Reducing bandwidth usage

One of the drawbacks of our evaluation process is that encrypting individual values for each input neuron yields a very large ciphertext, which is inconvenient from a user perspective, as a high bandwidth requirement is the direct consequence. In order to mitigate this issue, we

[5]If it is not, we can bootstrap all input ciphertexts in order to ensure this holds.

"pack" multiple values into one ciphertext. We use the standard technique of encrypting a polynomial (using the TLWE scheme instead of LWE) whose coefficients correspond to the different values we want to encrypt:

$$ct = \text{TLWE.Encrypt}\left(\sum_i x_i X^i\right),$$

where the x_i's represent the values of the input neurons to be encrypted[6]. This packing technique is what made Ring-LWE an attractive variant to the standard LWE problem, as was already presented in [LPR10], and is widely used in FHE applications to amortize the cost of operations [HS14a; HS15].

Then, we observe that for each neuron in the first hidden layer, we can compute the multisum with coefficients w_i by scaling the input TLWE ciphertext by a factor

$$\sum_i w_i X^{-i}.$$

Indeed, it is easy to verify that the constant term of $\left(\sum_i x_i X^i\right) \cdot \left(\sum_i w_i X^{-i}\right)$ is $\sum_i w_i x_i$, and we can obtain an LWE encryption of this value by invoking $\mathsf{Extract}$.

Remark 4.5.4. *We note that this computation is actually equivalent to doing the multisum directly on LWE ciphertexts, so the resulting noise growth of this approach is* exactly *the same as before. We end up saving bandwidth usage (by a factor up to N, the degree of the polynomials) basically for free. Furthermore, as the weights of the neural network never change, we can precompute and store the FFT representation of the polynomials $\sum w_i X^{-i}$, thus saving time during the online classification.*

In a nutshell, we reduce the size of the ciphertexts for N elements from N LWE ciphertexts to 1 TLWE ciphertext. In terms of numbers of elements in $\mathbb{T}$, the cost dropped from $N(n+1)$ to $N(k+1)$.

We remark that the resulting ciphertext is an LWE ciphertext in dimension N, and not the original n, thus requiring key-switching to become a legitimate ciphertext. However, this is not a problem thanks to the trick presented in the following subsection.

4.5.4.2 Moving $\mathsf{KeySwitch}$ around

The main goal of key-switching here is to reduce the LWE dimension. The benefits in memory usage and efficiency of this reduction are extremely important, since the size of the bootstrapping key, the final noise level, and the number of external products (the most costly operation) all depend linearly on this parameter. However, we noticed that reducing this dimension in the beginning of the bootstrapping procedure instead of the end gave much better results, hence the new bootstrapping function:

$$\mathsf{Bootstrap} = \mathsf{Extract} \circ \mathsf{BlindRotate} \circ \mathsf{KeySwitch}.$$

The intuition is that, with this technique, the noise produced by $\mathsf{KeySwitch}$ will not be multiplied by $\|\mathbf{w}\|_2$ when performing the computation of the multisum, but will only be

[6]If the number of input neurons is bigger than the maximal degree of the polynomials N, we can pack the ciphertext by groups of N, compute partial multisums with our technique, and aggregate them afterwards

added at the end. Basically, we moved the noise of the output ciphertext produced by KeySwitch to an overhead noise.

Doing this, we reverse the usage of the two underlying LWE schemes: everything is now done on high dimensional N-LWE, whereas the low dimensional n-LWE scheme is only used during the bootstrapping operation. Since the noise in the key-switching key is not used for any computation anymore, we can allow it to be bigger, thus reducing the dimension we need for the same security to hold and, in turn, gaining in time per bootstrapping.

The only downside is that working with higher dimensional N-LWE samples means slightly more memory usage for the server, bigger output ciphertext[7], and slightly slower addition of ciphertexts. However, as this operation is instantaneous when compared to other operations such as bootstrapping, this is not an issue.

4.5.4.3 Dynamically changing the message space

In Section 4.5.3, we showed how to evaluate the whole neural network by induction, using a message space of $2B+1$ slices, where B is a bound on the values of the multisums across the whole evaluation. However, in order to be able to reduce the probability of errors along the way, we are able to use different message spaces for each layer of the DiNN, and adapt the number of slots to the values given by the local computations, depending on the values of the weights $\mathbf{w}$. In order to do so, we change the value of testVector to

$$\frac{-1}{2B_\ell + 1} \sum_{i=0}^{N-1} X^i,$$

where B_ℓ is now indexed by the current layer ℓ, and is a bound on the values of the multisums for the next layer $\ell+1$. The point of this manoeuvre is that if the number of slots is smaller, the slices are bigger, and the noise would have to be bigger in order to change the plaintext message. This trick might seem superfluous, because it decreases a probability that is already negligible. However sometimes, in practical scenarios, the correctness of the scheme is relaxed, and this trick allows us to obtain results closer to the expected values without costing any extra computation or storage.

4.5.4.4 Alternative BlindRotate implementations

Following the technique of [ZYL+17], we try to gain efficiency in the bootstrapping by reducing the number of external products that we have to compute. In order to do so, they slightly unfold the loop computing $X^{\langle \mathbf{s},\mathbf{a}\rangle}$ in the BlindRotate algorithm. They group the terms of the sum two by two, using the following formula for each of the new terms:

$$X^{as+a's'} = ss'X^{a+a'} + s(1-s')X^a + (1-s)s'X^{a'} + (1-s)(1-s').$$

In order to compute this new function, they change the bootstrapping key to contain encryptions of the values $ss', s(1-s'), (1-s)s'$, and $(1-s)(1-s')$, thus expanding the size of the bootstrapping key by a factor 2. Using this idea, they cut the number of iterations of the loop by half, thus computing only half the amount of external products, which is the most costly operation of the bootstrapping. However, by doing so, they introduce the

[7]This can be circumvented by applying one last round of KeySwitch at the end of the protocol, if needed.

Algorithm 2 Alternative BlindRotate algorithm.

```
**Input:** an $n$-LWE ciphertext $(\mathbf{a}, b)$ with coefficients in $\mathbb{Z}_{2N}$, a (possibly noiseless) TLWE encryption $\mathbf{C}$ of testVector, the bootstrapping key bk such that for all $i$ in $[n/2]$, $\mathsf{bk}_{3i}, \mathsf{bk}_{3i+1}$, and $\mathsf{bk}_{3i+2}$ are respectively TGSW encryptions of $s_{2i}s_{2i+1}, s_{2i}(1-s_{2i+1})$, and $s_{2i+1}(1-s_{2i})$
**Output:** a TLWE encryption of $X^{b-\langle \mathbf{s},\mathbf{a}\rangle} \cdot \mathsf{testVector}$
1: $ACC \leftarrow X^b \cdot \mathbf{C}$
2: for $i = 1 \ldots n/2$ do
3:     $ACC \leftarrow ((X^{a_{2i}+a_{2i+1}} - 1)\mathsf{bk}_{3i} + (X^{a_{2i}} - 1)\mathsf{bk}_{3i+1} + (X^{a_{2i+1}} - 1)\mathsf{bk}_{3i+2}) \boxdot ACC$
4: end for
5: return ACC
```

computation of 4 scalings of TGSW ciphertexts (which are matrices) by constant polynomials, and 3 TGSW additions, when TFHE's BlindRotate only needed 1 scaling of a TLWE ciphertext, and 1 TLWE addition. Another benefit is that the homomorphic computation of $\langle \mathbf{s}, \mathbf{a}\rangle$ induces rounding errors on only $n/2$ terms instead of n. The noise of the output ciphertext is also different. On the bright side, the technique of [ZYL+17] reduces the noise induced by the precision errors during the gadget decomposition by a factor 2. On the other hand, it increases the noise coming from the bootstrapping key by a factor 2.

In this work, we suggest to use another formula in order to compute each term of the slightly unfolded sum. Observing that $ss' + s(1-s') + (1-s)s' + (1-s)(1-s') = 1$, we can save 1 element in the bootstrapping key:

$$X^{as+a's'} = ss'(X^{a+a'} - 1) + s(1-s')(X^a - 1) + (1-s)s'(X^{a'} - 1) + 1.$$

The resulting BlindRotate algorithm is described in Algorithm 2. Having a 1 in the decomposition is a valuable advantage, because it means that we can move it out of the external product and instead add the previous value of the accumulator to the result. Thus, efficiency-wise, we halved the number of external products at the cost of only 3 scalings of TGSW ciphertexts by constant polynomials, 2 TGSW additions, and 1 TLWE addition. We note that while multiplying naively by a monomial might be faster than multiplying by a degree 2 polynomial, the implementation pre-computes and stores the FFT representation of the bootstrapping keys in order to speed up polynomial multiplication. Thus, multiplying by a polynomial of any degree has the same cost. The size of the bootstrapping key is now 3/2 times larger than the size of the one in TFHE, which is a compromise between the two previous methods. As in [ZYL+17], the noise induced by precision errors and roundings is halved compared to TFHE. On the other hand, now we increase the noise coming from the bootstrapping key by a factor 3 instead. However, we note that it is possible to reduce this noise without impacting efficiency by reducing the noise in the bootstrapping key, trading off security (depending on what the bottleneck for security of the scheme is, this could come for free), whereas in order to reduce the noise induced by the precision errors, efficiency will be impacted. We recapitulate these numbers in Table 4.1.

We note that this idea could be generalized to unfoldings consisting of more than two terms, yielding more possible trade-offs, but we did not explore further because of the dissuasive exponential growth in the number of operands in the general formula.

		TFHE	**ZYLZD17**	FHE-DiNN
Efficiency	External products	n	$n/2$	$n/2$
	Scaled TGSW add.	0	4	3
	Scaled TLWE add.	1	0	1
	Noise overhead	δ	$\delta/2$	$\delta/2$
Out noise (average)	roundings	$n(1+kN)\varepsilon^2$	$\frac{n}{2}(1+kN)\varepsilon^2$	$\frac{n}{2}(1+kN)\varepsilon^2$
	from BK	$n(k+1)\ell N\beta^2\sigma_{bk}^2$	$2n(k+1)\ell N\beta^2\sigma_{bk}^2$	$3n(k+1)\ell N\beta^2\sigma_{bk}^2$
Out noise (worst)	roundings	$n(1+kN)\varepsilon$	$\frac{n}{2}(1+kN)\varepsilon$	$\frac{n}{2}(1+kN)\varepsilon$
	from BK	$n(k+1)\ell N\beta\mathcal{A}_{bk}$	$2n(k+1)\ell N\beta\mathcal{A}_{bk}$	$3n(k+1)\ell N\beta\mathcal{A}_{bk}$
Storage	TGSW in the BK	n	$2n$	$3n/2$

Table 4.1: Comparison of the three alternative BlindRotate algorithms. n denotes the LWE dimension after keyswitching; δ refers to the noise introduced by rounding the LWE samples into $[2N]$ before we can BlindRotate; N is the degree of the polynomials in the TLWE scheme; k is the dimension of the TLWE ciphertexts; ε is the precision $(1/2\beta)^\ell/2$ of the gadget matrix (tensor product between the identity Id_{k+1} and the powers of $1/2\beta$ arranged as ℓ-dimensional vector $(1/2\beta, \ldots, (1/2\beta)^\ell)$); σ_{bk} is the standard deviation of the noise of the TGSW encryptions in the bootstrapping key, and $\mathcal{A}_{bk}$ is a bound on this noise. These values were derived using the theorems for noise analysis in [CGGI17]

4.5.5 Experimental results

We implemented the proposed approach to test its accuracy and efficiency. This section is divided into two main parts: the first one describes the training of the neural network over data in the clear and the second one details the results obtained when evaluating the network over encrypted inputs.

4.5.5.1 Pre-processing the MNIST database

In order to respect the constraint of having inputs in $\{-1, 1\}$, we binarized all the images with a threshold value equal to 128: any pixel whose value is smaller than the threshold is mapped to -1; the others are mapped to $+1$. This actually reduces the amount of information available, as each 8-bit grayscale value is clamped to a single bit, and one could wonder if this could impact the accuracy of the classification. Although this is possible, a quick visual inspection of the result shows that the digits depicted in the images are still clearly recognizable.

4.5.5.2 Building a DiNN from data in the clear

In order to train the neural network, we first chose its topology, i.e., the number of hidden layers and neurons per hidden layer. We experimented with several values, always keeping in mind that a smaller number of neurons per layer is preferable: having more neurons means that the value of the multisum will be potentially higher, thus requiring a larger message space in the homomorphic evaluation, which in turn forces to choose bigger parameters for the scheme. After some tries, we decided to show the feasibility of our approach through the homomorphic evaluation of two neural networks. Both have 784 neurons in the input layer (one per pixel), a single hidden layer, and an output layer composed of 10 neurons (one per class). The difference between the two models is the size of the hidden layer: the first network has 30 neurons, while the second has 100.

In order to build a DiNN, we use the simple approach described in Section 4.5.2: we (1) train a traditional neural network (i.e., with real weights and biases), and then we (2) discretize all the values by applying the function in Equation (4.3). For step (1) we take advantage of the library keras [Cho+15] with Tensorflow [MAP+15], which offers a simple and highly customizable framework for defining, training and evaluating even complex models of neural networks. Through a fairly simple Python script and in little time, we are able to define and train our models as desired. Given its similarity with (a scaled and shifted version of) the sign function, as an activation function we used the version of hard_sigmoid defined in Tensorflow and whose formula is in Equation (4.4).

$$\mathsf{hard\_sigmoid}\left(x\right) = \begin{cases} 0, & x < -2.5 \\ 0.2\,x + 0.5, & -2.5 \leq x \leq 2.5 \\ 1, & x > 2.5 \end{cases} \tag{4.4}$$

The reason behind this choice is that we know we will substitute this activation function with the true $\mathsf{sign}\left(x\right)$. Thus, using a function which is already similar to it helps reducing the errors introduced by this switch.

Once we obtain the trained model, we proceed to choose a value $\tau \in \mathbb{N}$ and discretize the weights and the biases of the network, as per Equation (4.3), thus finally obtaining a DiNN that we can later evaluate over encrypted inputs. The choice of τ is an important part of the process: on one hand, picking a very small value will give little resolution to the network[8], potentially degrading the accuracy largely; on the other hand, picking a very large value will minimize the loss in accuracy but increase the message space that we will need to support for homomorphic evaluation, thus forcing us to choose larger parameters and making the overall evaluation less efficient. Also, note that it is possible to choose different values of the parameter τ for different layers of the network. Although there might be better choices, we did not invest too much efforts in optimizing the cleartext model and simply chose the value $\tau = 10$ for both layers of each model. Finally, we switched all the activation functions from $\mathsf{hard\_sigmoid}\left(\cdot\right)$ to $\mathsf{sign}\left(\cdot\right)$. In order to assess the results of the training and how the accuracy varies because of these changes, in Table 4.2 we report the accuracies obtained on the MNIST test set. Note that these values are referred to the evaluation *over cleartext inputs*.

4.5.5.3 Classifying encrypted inputs

Implementing the homomorphic evaluation of the neural network over encrypted input was more than a mere coding exercise, but allowed us to discover several interesting properties of our DiNNs.

[8]This means that the number of values that the weights will be able to take will be fairly limited.

	Original NN	**DiNN + hard_sigmoid**	**DiNN + sign**
30 neurons	94.76 %	93.76 % (-1 %)	93.55 % (-1.21 %)
100 neurons	96.75 %	96.62 % (-0.13 %)	96.43 % (-0.32 %)

Table 4.2: Accuracy obtained when evaluating the models in the clear on the MNIST test set. The first value refers to the evaluation of the model as output by the training; the second refers to the model where all the values for weights and biases have been discretized; the third refers to the same model, but with $\mathsf{sign}(\cdot)$ as the activation function for all the neurons in the hidden layer.

Ciphertext	**Dimension**	α	**Estimated security**
input	1024	2^{-30}	$>$ 150 bits
keyswitching key	450	2^{-17}	$>$ 80 bits
bootstrapping key	1024	2^{-36}	$>$ 100 bits

Table 4.3: The security parameters we use for the different kinds of ciphertexts. The estimated security has been extracted from the plot in [CGGI16b] and later verified with the estimator from Albrecht *et al.* [APS15].

The starting point was the TFHE library by Chillotti *et al.*, which is freely available on GitHub [CGGI16a] and which was used to efficiently perform the bootstrapping operation. The library takes advantage of FFT processors for fast polynomial multiplication and, although not parallelized, achieves excellent timing results. We extended the code to apply this fast bootstrapping procedure to our use case.

Parameters. We now present our setting of the parameters, following the notation of [CGGI16b], to which we refer the reader for extra details. In Table 4.3 we highlight the main security parameters regarding our ciphertexts, together with an estimate of the security level that this setting achieves. Other additional parameters, related to the various operations we need to perform, are the following:

- Degree of the polynomials in the ring: $N = 1024$;
- Dimension of the TLWE problem: $k = 1$;
- Basis for the decomposition of TGSW ciphertexts: $B_g = 1024$;
- Length of the decomposition of TGSW ciphertexts: $\ell = 3$;
- Basis for the decomposition during key switching: 8;
- Length of the decomposition during key switching: $t = 5$;

With this choice of parameters, we achieve a minimum security level of 80 bits and a single bootstrapping operation takes roughly 15 ms on a single core of an Intel Core i7-4720HQ CPU @ 2.60GHz. Also, we note that by exploiting the packing technique presented

	FHE-DiNN 30			FHE-DiNN 100		
	$\max_{\mathbf{w}} \|\mathbf{w}\|_1$	theor.	exp.	$\max_{\mathbf{w}} \|\mathbf{w}\|_1$	theor.	exp.
1st layer	2338	4676	2500	1372	2744	1800
2nd layer	399	798	800	488	976	1000

Table 4.4: Message space: theoretically required values and how we set them in our experiments with FHE-DiNN.

in Section 4.5.4.1, we save a factor 172 in the size of the input ciphertext: instead of having $784 \cdot (450 + 1)$ torus elements (corresponding to a 450-LWE ciphertext for each of the 784 pixels in an image), we now have only $2 \cdot 1024$ torus elements (corresponding to the two polynomials that form a TLWE sample).

Finally, we calculated the maximum value of the norms of the weight vectors associated to each neuron, both for the first and the second layer. These values, which can be computed at setup time (since the weights are available in the clear), define the theoretical bounds on the message space that our scheme should be able to support. In practice, we evaluated the actual values of the multisums on the training set, and took a message space slightly larger[9] than what we computed. We note that with this method, it is possible that some input could make the multisum go out of bounds, but this was not observed when evaluating the network on the test set. Moreover, this allows us to take a considerably smaller message space in some cases, and thus reduce the probability of errors. In Table 4.4 we report the theoretical message space we would need to support and the message space we actually used for our implementation.

In order to pinpoint our noise parameters, we also calculated the maximum L_2-norms of the weight vectors in each layer: for the network with 30 hidden neurons, we have $\max_{\mathbf{w}} \|\mathbf{w}\|_2 \approx 119$ for the first layer and ≈ 85 for the second layer; for the network with 100 hidden neurons, we have $\max_{\mathbf{w}} \|\mathbf{w}\|_2 \approx 69$ for the first layer and ≈ 60 for the second layer.

Evaluation. Our homomorphic evaluation follows the outline presented in Figure 4.7 in order to classify an encrypted image,

1. Encrypt the image as a TLWE ciphertext;
2. Multiply the TLWE ciphertext by the polynomial which encodes the weights associated to the hidden layer. This operation takes advantage of FFT for speeding up the calculations;
3. From each of the so-computed ciphertexts, extract a 1024-LWE ciphertext, which encrypts the constant term of the result;
4. Perform a key switching in order to move from a 1024-LWE ciphertext to a 450-LWE one;
5. Bootstrap to decrease the noise level. By setting the testVector, this operation also applies the sign function and changes the message space of our encryption scheme for free.

[9] As we do not achieve perfect correctness with our parameters, the message can be shifted. This fact has to be taken into account when choosing the number of slots.

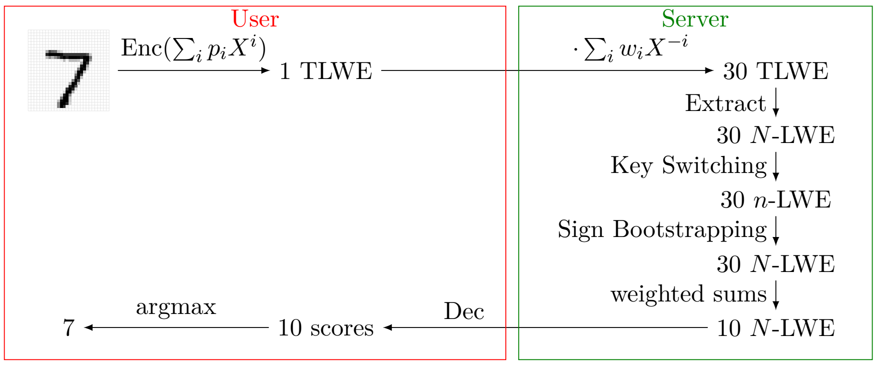


Figure 4.7: Refined homomorphic evaluation of a 784:30:10 neural network with activation function sign. The whole image (784 pixels) is packed into 1 TLWE ciphertext to minimize bandwidth usage. After evaluation, the user recovers 10 ciphertexts corresponding to the scores assigned by the network to each digit.

	Accur.	**Disag.**	**Wrong BS**	**Disag. (wrong BS)**	**Time**
30 or	93.71%	273 (105–121)	3383/300000	196/273	0.515 s
30 un	93.46%	270 (119–110)	2912/300000	164/270	0.491 s
100 or	96.26%	127 (61–44)	9088/1000000	105/127	1.679 s
100 un	96.35%	150 (66–58)	7452/1000000	99/150	1.64 s

Table 4.5: Results of homomorphic evaluation of two DiNNs on the full test set. The second column gives the number of disagreements (images classified differently) between the evaluation in the clear and the homomorphic one; the numbers in parentheses give the disagreements in favor of the cleartext evaluation and those in favor of the homomorphic evaluation, respectively. The third column gives the number of wrong bootstrapping, i.e., when the sign is flipped. The fourth value gives the number of disagreements in which at least one bootstrapping was wrong. Finally, the last column gives the time required to classify a single image.

6. Perform the multisum of the resulting ciphertext and the weights leading to the output layer, through the technique showed in Section 4.5.3.1[10]

7. Return the 10 ciphertexts corresponding to the 10 scores assigned by the neural network. These ciphertext can be decrypted and the argmax can be computed to obtain the classification given by the network.

In Table 4.5 we present the complete results of our experiments, both when using the original BlindRotate algorithm from [CGGI16b] (denoted by or) and when using the modified algorithm presented in Section 4.5.4.4 (denoted by un, unfolded).

The homomorphic evaluation of the network on the entire test set was compared to its classification in the clear and we observed the following facts:

[10]Note that we do not apply any activation function to the output neurons: we are only interested in being able to retrieve the scores and sorting them to recover the classification given by the network.

Remark 4.5.5. *The accuracy achieved when classifying encrypted images is close to that obtained when classifying images in the clear.*

In the case of the network with 30 hidden neurons, we obtain a classification accuracy of 93.55% in the clear (cf. Table 4.2) and of 93.71% homomorphically. In the case of the network with 100 hidden neurons, we have 96.43% accuracy in the clear and 96.35% on encrypted inputs. These gaps are explained by the following observations.

Remark 4.5.6. *During the evaluation, some signs are flipped during the bootstrapping but this does not significantly harm the accuracy of the network.*

We use aggressive internal parameters (e.g., N and, in general, all the parameters that control the precision) for the homomorphic evaluation, knowing that this could sometimes lead the bootstrapping procedure to return an incorrect result when extracting the sign of a message. In fact, we conjectured that the neural network would be resilient to perturbations and experimental results proved that this is indeed the case: when running our experiment over the full test set, we noticed that the number of wrong bootstrappings is 3383 (respectively, 9088) but this did not change the outcome of the classification in more than 196 (respectively, 105) cases (cf. Table 4.5).

Remark 4.5.7. *The classification of an encrypted image might disagree with the classification of the same image in the clear but this does not significantly worsen the overall accuracy.*

This is a property that we expected during the implementation phase and our intuition to explain this fact is the following: the network is assigning 10 scores to each image, one per digit, and when two scores are close (i.e., the network is hesitating between two classes), it can happen that the classification in the clear is correct and the one over the encrypted image is wrong. But the opposite can also be true, thus leading to classifying correctly an encrypted sample that was misclassified in the clear. We experimentally verified that disagreements between the evaluations do not automatically imply that the homomorphic classification is worse than the one in the clear: out of 273 (respectively, 127) disagreements, the classification in the clear was correct 105 (respectively, 61) times, against 121 (respectively, 44) times in favor of the homomorphic one[11] (cf. Table 4.5).

Remark 4.5.8. *Using the modified version of the* BlindRotate *algorithm presented in Section 4.5.4.4 decreases the number of wrong bootstrappings.*

Before stating some open problems, we conclude with the following note: using a bigger neural network generally leads to a better classification accuracy, at the cost of performing more calculations and, above all, more bootstrapping operations. However, the evaluation time will always grow linearly with the number of neurons. Although it is true that evaluating a bigger network is computationally more expensive, we stress that the bootstrapping operations are independent of each other and can thus be performed in parallel. Ideally, parallelizing the execution across a number of cores equal to the number of neurons in a layer (30 or 100 in our work) would result in that the evaluation of the layer would take roughly the time of a bootstrapping (i.e., around 15 ms).

[11]In the remaining cases, the classifications were different but they were both wrong.

Future directions and open problems. This work opens a number of possibilities and, thus, raises several interesting open problems. The first one is about the construction of our DiNNs. In this work, we did not pay too much attention to this step and, as a consequence, we considerably worsened the accuracy when moving from a canonical neural network to a DiNN. In order to improve the classification given by these discretized networks, it would be interesting to *train* a DiNN, rather than simply discretizing an already-trained model. Using discrete values and the sign function for the activation makes some calculations (e.g., some derivatives) impossible. Techniques to overcome these limitations have already been proposed in the literature (e.g., [CB16]) and they can be applied to our DiNNs as well. Also, another potentially interesting approach would be mixing these two ways of constructing a DiNN, for example by first discretizing a given model and then training the resulting network to refine it. Another natural question is whether we can batch several bootstrappings together, in order to improve the overall efficiency of the evaluation. Moreover, the speed of the evaluation would benefit from taking advantage of multi-core processing units, like GPUs.

Most interestingly, our FHE-DiNN framework is flexible and can be adapted to more generic cognitive architectures: we leave this as an interesting open problem. In particular, excellent results have been obtained by using Convolutional Neural Networks (see e.g., [LBBH98]), and we believe that trying to apply FHE-DiNN to these models would be an interesting line of research. Achieving this goal would require extending the current capabilities of FHE. For example, we would need to be able to homomorphically evaluate the max function, which is required to construct the widely-used max pooling layers. To the best of our knowledge, a technique for an efficient homomorphic evaluation of the max function is currently not known. Finally, the methodology presented in this work is by no means limited to image recognition, but can be applied to other machine learning problems as well.

4.5.6 Comparison with Cryptonets [DGL+16]

We now give a complete comparison between our framework FHE-DiNN and Cryptonets, with respect to communication complexity (i.e., size of ciphertexts), classification accuracy, and timings.

In Cryptonets, propagated signals are reals properly encoded into compatible plaintexts and a single encrypted input (i.e., an image pixel) takes $2 \cdot 382 \cdot 8192$ bits ($= 766$ kB). Therefore, an entire image takes $28 \cdot 28 \cdot 766\,\text{kB} \approx 586\,\text{MB}$. However, with the same storage requirements, Cryptonets can batch 8192 images together, so that the amortized size of an encrypted image is reduced to 73.3 kB. In the case of FHE-DiNN, we are able to exploit the batching technique *on a single image*, resulting in that each encrypted image takes $\approx 8.2\,\text{kB}$. In the case of Cryptonets, the complete homomorphic evaluation of the network takes 570 seconds, whereas in our case it takes 0.49 s (or 1.6 s in the case of a slightly larger network). However, it should be noted that (a) the networks that we use for our experiments are considerably smaller than that used in Cryptonets, so we also compare the time-per-neuron and, in this case, our solution is faster by roughly a factor 36; moreover (b) once again Cryptonets support image batching, so 8192 images can be classified in 570 seconds, resulting in only 0.07 s per image. Cryptonets' ability to batch images together can be useful in some applications where the same user wants to classify a large number of samples together. In the simplest case where the user only wants a single image to be classified, this feature does not help.

	Neurons	**Size of ct.**	**Accuracy**	**Time enc**	**Time eval**	**Time dec**
Cryptonets	945	586 MB	98.95%	122 s	570 s	5 s
Cryptonets*	945	73.3 kB	98.95%	0.015 s	0.07 s	0.0006 s
FHE-DiNN 30	30	$\approx$ 8.2 kB	93.71%	0.000168 s	0.49 s	0.0000106 s
FHE-DiNN 100	100	$\approx$ 8.2 kB	96.35%	0.000168 s	1.65 s	0.0000106 s

Table 4.6: Comparison with Cryptonets and its amortized version (denoted by Cryptonets*). FHE-DiNN 30 and FHE-DiNN 100 refer to neural networks with one hidden layer composed of 30 and 100 neurons, respectively.

Regarding classification accuracy, the NN used by Cryptonets achieves 98.95 % of correctly classified samples, when evaluated on the MNIST dataset. In our case, a loss of accuracy occurs due to the preliminary simplification of the MNIST images, and especially because of the discretization of the network. We stress however that our prime goal was not accuracy but to achieve a qualitatively better homomorphic evaluation at the neuron level.

Finally, we also achieve scale-invariance, meaning that we can keep on computing over the encrypted outputs of our network, whereas Cryptonets are bounded by the initial choice of parameters. In Table 4.6 we present a detailed comparison with Cryptonets.

Chapter 5

Circuit privacy for homomorphic computations

In the context of secure computation outsourcing, protecting the input data is only one side of the coin. On the other side, it might be that the party that performs the computation wants to keep the circuit it evaluates (i.e., the algorithm) private. In fact, it turns out that, without appropriate countermeasures, the output of an homomorphic computation can leak information on the computation itself, thus potentially compromising the privacy of an algorithm which could be a sensitive intellectual property. In this chapter of the manuscript, we address the problem of "circuit privacy", and we show how to efficiently eliminate this leakage, thus providing a way for a party to perform a homomorphic computation without revealing more than its result and some generic side information.

Contents

5.1 Introduction

As presented in Chapter 3, a fully homomorphic encryption (FHE) scheme is an encryption scheme which supports computation on encrypted data: given a ciphertext that encrypts some data μ, one can compute a ciphertext that encrypts $f(\mu)$ for any efficiently computable function f, without ever needing to decrypt the data or know the decryption key. FHE has numerous theoretical and practical applications, the canonical one being to the problem of outsourcing computation to a remote server without compromising one's privacy. In 2009, Gentry put forth the first candidate construction of FHE based on ideal lattices [Gen09b]. Since then, substantial progress has been made [DGHV10; SS10; SV10; BV11a; BV11b; BGV12; GHS12; GSW13; BV14; AP14], offering various improvements in conceptual and technical simplicity, efficiency, security guarantees, assumptions, etc; in particular, Gentry, Sahai and Waters presented a very simple FHE (hereafter called the GSW cryptosystem) based on the standard learning with errors (LWE) assumption.

Circuit privacy. An additional requirement in many FHE applications is that the evaluated ciphertext should also hide the function f, apart from what is inevitably leaked through the outcome of the computation $f(\mu)$; we refer to this requirement as *circuit privacy* [SYY99; IP07]. In the context of outsourcing computation, a server may wish to hide its proprietary algorithm from the client, while in the context of homomorphic evaluation of neural networks, a company might want to protect the "internal details" of the cognitive model, such as its topology, the weights, ... In fact, training an effective neural network is usually a complex and time-consuming procedure, where considerable effort is invested to achieve small improvements that make one's product better than those of the competitors. In these situations, protecting the privacy of these trade secrets becomes of vital importance for the profitability of the product. Circuit privacy is also a requirement when we use FHE for low-communication secure two-party computation. In all existing FHE schemes, there is a "noise" term in the ciphertext, which is necessary for security. The noise grows and changes as a result of performing homomorphic operations and, in particular, could leak information about the function f. The main challenge for achieving FHE circuit privacy lies precisely in avoiding the leakage from the noise term in the evaluated ciphertext.

Prior works. Prior works achieve circuit privacy by essentially canceling out the noise term in the evaluated ciphertext. There are two main approaches for achieving this. The first is "noise flooding" introduced in Gentry's thesis, where we add a much larger noise at the end of the computation; in particular, the noise that is added needs to be super-polynomially larger than the noise that accumulates amidst homomorphic operations, which in turn requires that we start with a super-polynomial modulus-to-noise ratio[1]. This is a fairly mild assumption for the early constructions of FHE schemes, which required a quasi-polynomial modulus-to-noise ratio just to support homomorphic operations for circuits in NC^1 (i.e., circuits of logarithmic depth). The second is to decrypt and re-encrypt the evaluated ciphertext, also known as bootstrapping in the FHE literature. This can be achieved securely without having to know the secret key in the clear in one of two ways: (i) with the use of garbled circuits [OPP14; GHV10], and (ii) via homomorphic evaluation of the decryption circuit given an encryption of the secret key under itself [DS16], which requires the additional assumption of circular security (cf. Assumption 3.3.1).

[1]Recall that LWE hardness depends on the modulus-to-noise ratio: the smaller the ratio, the harder the problem.

Both of the prior approaches have some theoretical and practical draw-backs, if we consider FHE for NC^1 circuits (the rest of the discussion also applies to leveled FHE for general circuits). First, recall that we now have FHE for NC^1 circuits under the LWE assumption with a polynomial modulus-to-noise ratio [BV14; AP14], and we would ideally like to achieve circuit privacy under the same assumption. Relying on noise flooding for circuit privacy would require quantitatively stronger assumptions with a super-polynomial modulus-to-noise ratio, which in turn impacts practical efficiency due to the use of larger parameters. Similarly, the use of bootstrapping for circuit privacy can also be computationally expensive (indeed, the bootstrapping operation is the computational bottleneck in existing FHE schemes, cf. [DM15; HS15]). Moreover, realizing bootstrapping via an encryption of the secret key requires an additional circular security assumption (cf. Assumption 3.3.1), which could in turn also entail the use of larger parameters in order to account for potential weaknesses introduced by circular security. Realizing bootstrapping via garbled circuits avoids the additional assumption, but is theoretically and practically unsatisfying as it requires encoding the algebraic structure in existing FHEs as boolean computation, and sacrifices the multi-hop property in that we can no longer perform further homomorphic computation on the evaluated ciphertexts.

5.1.1 Our results

Our main result is a circuit-private FHE for NC^1 circuits – and a circuit-private leveled FHE for general circuits – under the LWE assumption with a polynomial modulus-to-noise ratio, and whose efficiency essentially matches that of existing variants of the GSW cryptosystem in [BV14; AP14]; in other words, we avoid noise flooding or bootstrapping and obtain circuit privacy almost for free.

We obtain our main result via a conceptually different approach from prior works: instead of canceling out the noise term in the evaluated ciphertext, we directly analyze the *distribution* of the noise term (prior works on FHE merely gave a bound on the noise term). Concretely, we show that adding a small noise in each step of homomorphic evaluation in the GSW cryptosystem already hides the computation itself which yields circuit privacy. Along the way, we gain better insights into the algebraic structure and the noise distribution in GSW scheme and provide new tools for analyzing noise randomization which we believe could be of independent interest.

As an immediate corollary, we obtain a two-party protocol for secure function evaluation where Alice holds x, Bob holds a branching program f, and we want Alice to learn $f(x)$ while protecting the privacy of x and f to the largest extent possible, that is, Bob learns nothing about x and Alice learns nothing about f (apart from a bound on the size of f). Our protocol achieves semi-honest security under the standard LWE assumption with polynomial hardness, and where the total communication complexity and Alice's computation are poly-logarithmic in the size of f.

The core of our analysis is a variant of the Gaussian leftover hash lemma [AGHS13; AR13]: given a "small" vector $\mathbf{e}$ and any vector $\mathbf{v}$, we have

$$\mathbf{e}^{\mathsf{T}} \cdot \mathbf{G}^{-1}_{\text{rand}}(\mathbf{v}) + y \approx_s e'$$

where

- $\mathbf{G}^{-1}_{\text{rand}}(\mathbf{v})$ outputs a random short vector $\mathbf{x}$ satisfying $\mathbf{G}\mathbf{x} = \mathbf{v} \mod q$ according to a discrete Gaussian with parameter $r = \widetilde{\mathcal{O}}(1)$;

plaintext	$v_{\text{out}} = v_x = xv_1 + (1-x)v_0$
[GSW13; BV14]	$\mathbf{V}_{\text{out}} = \mathbf{C} \cdot \mathbf{G}^{-1}_{\text{det}}(\mathbf{V}_1) + (\mathbf{G} - \mathbf{C}) \cdot \mathbf{G}^{-1}_{\text{det}}(\mathbf{V}_0)$
[AP14]	$\mathbf{V}_{\text{out}} = \mathbf{C} \cdot \mathbf{G}^{-1}_{\text{rand}}(\mathbf{V}_1) + (\mathbf{G} - \mathbf{C}) \cdot \mathbf{G}^{-1}_{\text{rand}}(\mathbf{V}_0)$
[this work]	$\mathbf{V}_{\text{out}} = \mathbf{C} \cdot \mathbf{G}^{-1}_{\text{rand}}(\mathbf{V}_1) + (\mathbf{G} - \mathbf{C}) \cdot \mathbf{G}^{-1}_{\text{rand}}(\mathbf{V}_0) + \begin{pmatrix} \mathbf{0} \\ \mathbf{y}^{\mathsf{T}} \end{pmatrix}$

Table 5.1: The first row of the table shows the plaintext computation that happens at each step of the evaluation of a branching program (cf. Section 5.4.3.1). The next three rows describe how this computation is carried out homomorphically on ciphertexts $\mathbf{V}_0, \mathbf{V}_1, \mathbf{C}$ corresponding to encryptions of the input bits v_0, v_1, x. In the [GSW13; BV14] FHE schemes, homomorphic evaluation is deterministic, whereas in [AP14] and this work, homomorphic evaluation is randomized. In particular, our construction introduces an additional small Gaussian shift on top of [AP14].

- both y and e' are drawn from discrete Gaussians with parameter $\mathcal{O}(r \cdot \|\mathbf{e}\|)$ (the norm of e' will be slightly larger than that of y).

We stress that the distribution of e' is independent of $\mathbf{v}$ and that the norm of y, e' are polynomially related to that of $\|\mathbf{e}\|$. Indeed, a similar statement is true via noise flooding, where we pick y, e' to have norm super-polynomially larger than that of $\|\mathbf{e}\|$. Using this leftover hash lemma to hide the argument of $\mathbf{G}^{-1}_{\text{rand}}(\cdot)$ is new to this work and will be crucial in proving circuit privacy. In Table 5.1 we show a comparison with previous works on how to perform a step of computation for branching program evaluation.

5.1.2 Technical overview

We proceed with a technical overview of our construction. We build up to our main construction in three steps.

Generating fresh LWE samples. How do we generate a fresh LWE sample from a large but bounded number of samples? That is, we need to randomize $(\mathbf{A}, \mathbf{s}^{\mathsf{T}}\mathbf{A} + \mathbf{e}^{\mathsf{T}})$. The first idea, going back to [Reg05; GPV08; ACPS09] is to choose $\mathbf{x}$ according to a discrete Gaussian with parameter $r = \widetilde{\mathcal{O}}(1)$ and a small "smoothing" noise y from a discrete Gaussian with parameter $\mathcal{O}(r \cdot \|\mathbf{e}\|)$ and output

$$\mathbf{A}\mathbf{x}, (\mathbf{s}^{\mathsf{T}}\mathbf{A} + \mathbf{e}^{\mathsf{T}})\mathbf{x} + y$$

The vector $\mathbf{A}\mathbf{x}$ is statistically close to uniform (by leftover hash lemma), and the error $\mathbf{e}^{\mathsf{T}}\mathbf{x} + y$ in the resulting sample is statistically close to a discrete Gaussian with parameter $\mathcal{O}(r \cdot \|\mathbf{e}\|)$. We stress that the norm of y is polynomially related to that of $\mathbf{e}$, which is better than naive noise flooding. One draw-back compared to noise flooding is that the error in the new sample leaks $\|\mathbf{e}\|$. In the case of generating fresh LWE samples, we just need to repeat the process to generate many more samples than what we started out with.

Randomizing GSW ciphertexts. Next, we note that the above idea can also be used to randomize GSW ciphertexts. Recall that a GSW encryption of a message μ is of the form

$$\mathbf{C} = \begin{pmatrix} \mathbf{A} \\ \mathbf{s}^\intercal \mathbf{A} + \mathbf{e}^\intercal \end{pmatrix} + \mu \mathbf{G} \in \mathbb{Z}_q^{n \times (n \log q)}$$

where $\mathbf{s} \in \mathbb{Z}_q^n$ is the secret key and $\mathbf{G}$ is the "powers of 2" gadget matrix. We can randomize $\mathbf{C}$ to be a fresh encryption of μ by computing

$$\mathbf{C} \cdot \mathbf{G}_{\text{rand}}^{-1}(\mathbf{G}) + \begin{pmatrix} \mathbf{0} \\ \mathbf{y}^\intercal \end{pmatrix}$$

where $\mathbf{G}_{\text{rand}}^{-1}(\mathbf{G})$ is chosen according to a discrete Gaussian of parameter r satisfying $\mathbf{G} \cdot \mathbf{G}_{\text{rand}}^{-1}(\mathbf{G}) = \mathbf{G}$ and $\mathbf{y}$ is again a small smoothing noise vector. Here, we need an extension of the previous lemma showing that each coordinate in $\mathbf{e}^\intercal \cdot \mathbf{G}_{\text{rand}}^{-1}(\mathbf{G}) + \mathbf{y}^\intercal$ is statistically close to a discrete Gaussian; this in turn follows from an extension of the previous lemma where the vector $\mathbf{x}$ is drawn from discrete Gaussian over the coset of a lattice (cf. Lemma 5.3.3). And again, the norm of $\mathbf{y}$ is polynomially related to that in $\mathbf{e}$, which is better than naive noise flooding.

Scaling GSW ciphertexts. More interesting, given a constant $a \in \{0, 1\}$, we can scale a GSW encryption of μ to obtain a fresh encryption of $a \cdot \mu$ while revealing no information about a beyond what is leaked in $a \cdot \mu$. In particular, if $\mu = 0$, then the resulting ciphertext should completely hide a. To achieve this, we simply proceed as before, except we use $\mathbf{G}_{\text{rand}}^{-1}(a \cdot \mathbf{G})$ so that $\mathbf{G} \cdot \mathbf{G}_{\text{rand}}^{-1}(a \cdot \mathbf{G}) = a \cdot \mathbf{G}$. Here, we crucially rely on the fact that the error $\mathbf{e}^\intercal \cdot \mathbf{G}_{\text{rand}}^{-1}(a \cdot \mathbf{G}) + \mathbf{y}^\intercal$ in the resulting ciphertext is independent of a.

Circuit-private homomorphic evaluation. The preceding construction extends to the setting where we are given a GSW encryption $\mathbf{C}'$ of a instead of a itself, so that we output

$$\mathbf{C} \cdot \mathbf{G}_{\text{rand}}^{-1}(\mathbf{C}') + \begin{pmatrix} \mathbf{0} \\ \mathbf{y}^\intercal \end{pmatrix}.$$

We can handle homomorphic encryption as in GSW; this then readily extends to a circuit-private homomorphic evaluation for branching programs, following [BV14; AP14].

Branching programs are a relatively powerful representation model. In particular, any logarithmic space or NC^1 computation can be carried out by a family of polynomial-size branching programs. Branching programs can also directly capture several representation models often used in practice such as decision trees, OBDDs, and deterministic finite automaton.

The key insight from Brakerski and Vaikuntanathan [BV14] is that when homomorphically evaluating a branching program, we will only need to perform homomorphic additions along with homomorphic multiplications of ciphertexts $\mathbf{V}_j, \mathbf{C}_i$ where $\mathbf{V}_j$ is the encryption of an intermediate computation and $\mathbf{C}_i$ is an encryption of the input variable x_i. To obtain decryption correctness with polynomial noise growth, they computed the product as

$$\mathbf{C}_i \cdot \mathbf{G}_{\text{det}}^{-1}(\mathbf{V}_j),$$

where $\mathbf{G}_{\text{det}}^{-1}(\cdot)$ is the deterministic binary decomposition, cleverly exploiting the asymmetric noise growth in GSW ciphertexts and the fact that the noise in $\mathbf{C}_i$ is smaller than that in

$\mathbf{V}_j$. To obtain circuit privacy, we will compute the product as

$$\mathbf{C}_i \cdot \mathbf{G}^{-1}_{\text{rand}}(\mathbf{V}_j) + \begin{pmatrix} \mathbf{0} \\ \mathbf{y}_j^\intercal \end{pmatrix}.$$

Note that we made two modifications:

- First, we switched to a randomized $\mathbf{G}^{-1}_{\text{rand}}(\cdot)$. The use of a randomized $\mathbf{G}^{-1}_{\text{rand}}(\cdot)$ for homomorphic evaluation was first introduced in [AP14], but for the very different purpose of a mild improvement in the noise growth (i.e., *efficiency*); here, we crucially exploit randomization for *privacy*.
- Next, we introduced an additional Gaussian shift $\mathbf{y}_j^\intercal$.

Interestingly, it turns out that computing the product as $\mathbf{C}_i \cdot \mathbf{G}^{-1}_{\text{rand}}(\mathbf{V}_j)$ instead of $\mathbf{V}_j \cdot \mathbf{G}^{-1}_{\text{rand}}(\mathbf{C}_i)$ is useful not only for polynomial noise growth, but also useful for circuit privacy. Roughly speaking, the former hides which $\mathbf{V}_j$ is used, which corresponds to hiding the intermediate states that lead to the final output state, which in turn hides the branching program.

We highlight a subtlety in the analysis: $\mathbf{V}_j$ could in principle encode information about $\mathbf{C}_i$, if the variable x_i has been read prior to reaching the intermediate state encoded in $\mathbf{V}_j$, whereas to apply our randomization lemma, we crucially rely on independence between $\mathbf{C}_i$ and $\mathbf{V}_j$. The analysis proceeds by a careful induction argument showing that $\mathbf{V}_j$ looks like a fresh GSW ciphertext independent of input ciphertexts $\mathbf{C}_1, \ldots, \mathbf{C}_\ell$ apart from some dependencies on the norm of the noise terms in the input ciphertexts (see Lemma 5.4.7 for a precise statement). These dependencies mean that homomorphic evaluation leaks the number of times each variable appears in the branching program, but that can be easily fixed by padding the branching program.

5.2 Additional preliminaries

In this section we give some additional notions that are needed to properly understand the following. These preliminaries are local to this part.

5.2.1 Randomized $\mathrm{G}^{-1}(\cdot)$ algorithm

First of all, we introduce a randomized version of the $\mathbf{G}^{-1}(\cdot)$ algorithm already mentioned in Section 3.4.3. This algorithm and its properties will be fundamental for our construction.

Definition 5.2.1 (The $\mathbf{G}^{-1}_{\text{rand}}(\cdot)$ algorithm, adapted from [MP12],[AP14, Claim 3.1])**.** *There is a randomized, efficiently computable function $\mathbf{G}^{-1}_{\text{rand}}(\cdot) : \mathbb{Z}_q^n \to \mathbb{Z}^m$, where $m = n\lceil \log q \rceil$ such that $\mathbf{x} \leftarrow \mathbf{G}^{-1}_{\text{rand}}(\mathbf{v})$ is drawn from a distribution close to a Gaussian with parameter $r = \widetilde{\mathcal{O}}(1)$ conditioned on $\mathbf{Gx} = \mathbf{v} \mod q$, i.e., $\mathbf{G}^{-1}_{\text{rand}}(\mathbf{v})$ outputs a sample from the distribution $\mathcal{D}_{\Lambda_q^\perp + \mathbf{G}^{-1}_{\text{det}}(\mathbf{v}), r}$ where $\mathbf{G}^{-1}_{\text{det}}(\cdot)$ denotes (deterministic) bit decomposition. We will also write $\mathbf{X} \leftarrow \mathbf{G}^{-1}_{\text{rand}}(\mathbf{M})$ to denote that the columns of the matrix $\mathbf{X} \in \mathbb{Z}^{m \times p}$ are obtained by applying the algorithm separately to each column of a matrix $\mathbf{M} \in \mathbb{Z}_q^{n \times p}$.*

In particular, using the exact sampler in [BLP+13, Section 5] (which is a variant of the algorithm presented in [GPV08]), $\mathbf{G}^{-1}_{\text{rand}}(\mathbf{v})$ outputs a sample from the discrete Gaussian

$$\mathcal{D}_{\Lambda_q^\perp + \mathbf{G}^{-1}_{\text{det}}(\mathbf{v}), r}.$$

5.2.2 Probability results

We will also need the following probability results.

Lemma 5.2.2 (Simplified version of [Pei10, Theorem 3.1])**.** *Let $\varepsilon > 0$, $r_1, r_2 > 0$ be two Gaussian parameters, and $\Lambda \subseteq \mathbb{Z}^m$ be a lattice. If $\frac{r_1 r_2}{\sqrt{r_1^2+r_2^2}} \geq \eta_\varepsilon(\Lambda)$, then*

$$\Delta\left(\mathbf{y}_1 + \mathbf{y}_2, \mathbf{y}'\right) \leq 8\varepsilon,$$

where $\mathbf{y}_1 \leftarrow \mathcal{D}_{\Lambda, r_1}$, $\mathbf{y}_2 \leftarrow \mathcal{D}_{\Lambda, r_2}$, and $\mathbf{y}' \leftarrow \mathcal{D}_{\Lambda, \sqrt{r_1^2+r_2^2}}$.

Lemma 5.2.3 ([AP14, Lemma 2.1])**.** *There exists a universal constant $C > 0$, such that*

$$\Pr\left[\|\mathbf{x}\| > Cr\sqrt{m}\right] \leq 2^{-\Omega(m)},$$

where $\mathbf{x} \leftarrow \mathcal{D}_{\mathbb{Z}^m, r}$.

5.2.3 Results on lattices and Gaussian distributions

Finally, we recall some additional results.

Lemma 5.2.4 ([MR07, Lemma 3.3])**.** *Let Λ be any rank-m lattice and ε be any positive real. Then*

$$\eta_\varepsilon(\Lambda) \leq \lambda_m(\Lambda) \cdot \sqrt{\frac{\ln\left(2m\left(1 + 1/\varepsilon\right)\right)}{\pi}},$$

where $\lambda_m(\Lambda)$ is the m-th successive minimum of the lattice Λ (cf. Definition 2.3.4).

Lemma 5.2.5 ([GPV08, Corollary 2.8])**.** *Let $\Lambda \subseteq \mathbb{Z}^m$ be a lattice, $0 < \varepsilon < 1$, $r > 0$. For any vector $\mathbf{c} \in \mathbb{R}^m$, if $r \geq \eta_\varepsilon(\Lambda)$, then we have*

$$\rho_r(\Lambda + \mathbf{c}) \in \left[\frac{1-\varepsilon}{1+\varepsilon}, 1\right] \cdot \rho_r(\Lambda).$$

Lemma 5.2.6 ([Reg05, Claim 3.8])**.** *Let $\Lambda \subseteq \mathbb{Z}^m$ be any lattice, $\mathbf{c} \in \mathbb{R}^m$, $\varepsilon > 0$ and $r \geq \eta_\varepsilon(\Lambda)$. Then*

$$\rho_r(\Lambda + \mathbf{c}) \in \frac{r^m}{\det(\Lambda)}(1 \pm \varepsilon).$$

5.2.4 Entropy and leftover hash lemma

Here, we recall the definition of min-entropy and a version of the leftover hash lemma.

Definition 5.2.7 (Min-entropy)**.** *The min-entropy of a random variable X is defined as*

$$H_\infty(X) \coloneqq -\log\left(\max_x \Pr[X = x]\right).$$

Next, we state here a simplified version of the leftover hash lemma (originally introduced by Impagliazzo *et al.* [ILL89]), which is sufficient for our use.

Lemma 5.2.8 (Leftover hash lemma [DRS04])**.** *Let $\mathbf{e}$ be any random variable over $\mathbb{Z}_q^m$ and $f : \mathbb{Z}_q^m \to \mathbb{Z}_q^k$. Then*

$$\Delta\left(\left(\mathbf{X}\mathbf{e}, \mathbf{X}, f(\mathbf{e})\right), \left(\mathbf{r}, \mathbf{X}, f(\mathbf{e})\right)\right) \leq \sqrt{q^{n+k} \cdot 2^{-H_\infty(\mathbf{e})}}.$$

5.2.5 Permutation branching programs

Here we recall the definition of a permutation branching program, that we also call simply *branching program*, and sometimes abbreviate as BP.

Definition 5.2.9 (Permutation branching program). *A permutation branching program* Π *of length* L *and width* W *with input space* $\{0,1\}^\ell$ *is a sequence of* L *tuples of the form* $(\mathrm{var}(t), \pi_{t,0}, \pi_{t,1})$*, where*

- $\mathrm{var} : [L] \to [\ell]$ *is a function that associates the* t*-th tuple with an input bit* $x_{\mathrm{var}(t)}$
- $\pi_{t,0}, \pi_{t,1} : [W] \to [W]$ *are permutations that dictate the* t*-th step of the computation.*

On input $(x_1, \ldots, x_\ell)$*,* Π *outputs* 1 *iff*

$$\pi_{L,x_{var(L)}}(\cdots(\pi_{1,x_{var(1)}}(1))\cdots) = 1.$$

Following [BV14; IP07], we will evaluate Π recursively as follows. We associate each $t \in [L]$ with the characteristic vector $\mathbf{v}_t \in \{0,1\}^W$ of the current "state", starting with $\mathbf{v}_0 = (1, 0, \ldots, 0)$. We can then compute the w-th entry of $\mathbf{v}_t$ (denoted by $\mathbf{v}_t[w]$): for all $t \in [L], w \in [W]$,

$$\mathbf{v}_t[w] = \mathbf{v}_{t-1}\left[\pi^{-1}_{t,x_{\mathrm{var}(t)}}(w)\right] = x_{\mathrm{var}(t)} \cdot \mathbf{v}_{t-1}\left[\pi^{-1}_{t,1}(w)\right] + \left(1 - x_{\mathrm{var}(t)}\right) \cdot \mathbf{v}_{t-1}\left[\pi^{-1}_{t,0}(w)\right]. \quad (5.1)$$

5.3 Core randomization lemma

We are now ready to state and prove our main randomization lemma for LWE samples. Note that in the remainder of this chapter, we set q to be a power of 2, and $m = n\log q$. We discuss the use of a q which is not a power of 2 in Section 5.4.5.

Lemma 5.3.1 (Core randomization lemma). *Let* $\varepsilon, \varepsilon' > 0$*,* $r > \eta_\varepsilon\left(\Lambda_q^\perp(\mathbf{G}^\intercal)\right)$ *be a Gaussian parameter. For any* $\mathbf{e} \in \mathbb{Z}_q^m$*,* $\mathbf{v} \in \mathbb{Z}_q^n$*, if*

$$r \geq \max\left(4\left((1-\varepsilon)(2\varepsilon')^2\right)^{-\frac{1}{m}}, \sqrt{5}(1+\|\mathbf{e}\|)\sqrt{\frac{\ln(2m(1+1/\varepsilon))}{\pi}}\right),$$

then

$$\Delta\left((\mathbf{A}, \mathbf{A}\mathbf{x}, \mathbf{e}^\intercal\mathbf{x} + y), (\mathbf{A}, \mathbf{u}, e')\right) < \varepsilon' + 2\varepsilon$$

where $\mathbf{x} \leftarrow \mathbf{G}^{-1}_{\mathrm{rand}}(\mathbf{v})$*,* $\mathbf{A} \leftarrow_\$ \mathbb{Z}_q^{(n-1)\times m}$*,* $\mathbf{u} \leftarrow_\$ \mathbb{Z}_q^{n-1}$*,* $y \leftarrow \mathcal{D}_{\mathbb{Z},r}$ *and* $e' \leftarrow \mathcal{D}_{\mathbb{Z},r\sqrt{1+\|\mathbf{e}\|^2}}$*.*
Asymptotically, $r = \widetilde{\Theta}(\|\mathbf{e}\|\sqrt{\kappa})$ *is enough to obtain negligible statistical distance.*

Remark 5.3.2 (on the necessity of randomization). *We note here that the use of randomization in* $\mathbf{G}^{-1}_{\mathrm{rand}}(\cdot)$ *and the shift are both necessary.*

First, the shift is necessary for both distributions to have the same support. For example, $\mathbf{e}^\intercal\mathbf{G}^{-1}_{\mathrm{rand}}((1,0,\ldots,0))$ *and* $\mathbf{e}^\intercal\mathbf{G}^{-1}_{\mathrm{rand}}(\mathbf{0})$ *might lie in two different cosets of the lattice* $\mathbf{e}^\intercal\Lambda_q^\perp(\mathbf{G}^\intercal)$*, depending on the value of* $\mathbf{e}$*: if the first coordinate of* $\mathbf{e}$ *is odd and all the others are even, then* $\mathbf{e}^\intercal\mathbf{G}^{-1}_{\mathrm{rand}}((1,0,\ldots,0))$ *will be odd, while* $\mathbf{e}^\intercal\mathbf{G}^{-1}_{\mathrm{rand}}(\mathbf{0})$ *will be even, for an even* q*. The shift by a Gaussian over* $\mathbb{Z}$ *ensures that the support of the two distributions is* $\mathbb{Z}$*.*

Proving that $\mathbf{e}^\intercal \Lambda_q^\perp(\mathbf{G}^\intercal) = \mathbb{Z}$ *with overwhelming probability over the choice of* $\mathbf{e}$ *is still an open question that would remove the necessity of the shift, thus proving circuit privacy for standard GSW only using randomized* $\mathbf{G}^{-1}_{\text{rand}}(\cdot)$*.*

Finally, the randomization of $\mathbf{G}^{-1}_{\text{rand}}(\cdot)$ *is necessary for both distributions to have the same center. Using the same example,* $\mathbf{e}^\intercal \mathbf{G}^{-1}_{\text{det}}((1, 0, \ldots, 0)) + y$ *and* $\mathbf{e}^\intercal \mathbf{G}^{-1}_{\text{det}}(\mathbf{0}) + y$ *would be two Gaussians, centered respectively on* e_1 *(the first coordinate of* $\mathbf{e}$*) and on* 0*. Instead, using the randomized algorithm* $\mathbf{G}^{-1}_{\text{rand}}(\cdot)$*, the center of both distributions will be* 0*.*

5.3.1 Proof of randomization lemma

We first prove that given $\mathbf{e}$, the new error term $\mathbf{e}^\intercal \mathbf{x} + y$ is indeed a Gaussian with parameter $r\sqrt{1 + \|\mathbf{e}\|^2}$. This proof is inspired by [AR13], which in turn is an improvement of [AGHS13], but it is different in two aspects: on one hand, in [AR13] the proof is done for the specific case where $\mathbf{x}$ is drawn from a Gaussian over a coset of $\mathbb{Z}^m$; on the other hand, they consider the more general case of an ellipsoidal Gaussian distribution.

Lemma 5.3.3 (adapted from [AR13, Lemma 3.3])**.** *Let* $\varepsilon, r > 0$*. For any* $\mathbf{e} \in \mathbb{Z}^m$*,* $\mathbf{c} \in \mathbb{R}^m$*, if* $r \geq \sqrt{5}(1 + \|\mathbf{e}\|) \cdot \sqrt{\frac{\ln(2m(1+1/\varepsilon))}{\pi}}$*, then*

$$\Delta\left(\mathbf{e}^\intercal \mathbf{x} + y, e'\right) < 2\varepsilon$$

where $\mathbf{x} \leftarrow \mathcal{D}_{\Lambda_q^\perp(\mathbf{G}^\intercal)+\mathbf{c},r}$*,* $y \leftarrow \mathcal{D}_{\mathbb{Z},r}$*, and* $e' \leftarrow \mathcal{D}_{\mathbb{Z},r\sqrt{1+\|\mathbf{e}\|^2}}$*.*

Asymptotically, $r = \widetilde{\Theta}(\|\mathbf{e}\| \sqrt{\kappa})$ is enough to obtain negligible statistical distance. We stress that the distribution of e' does not depend on the coset $\mathbf{c}$.

Proof. Let $\widehat{\mathbf{e}} = (\mathbf{e}, 1) \in \mathbb{Z}^{m+1}, \widehat{\mathbf{c}} = (\mathbf{c}, 0) \in \mathbb{Z}^{m+1}$ and $\widehat{\Lambda} = \Lambda_q^\perp(\mathbf{G}^\intercal) \times \mathbb{Z}$. We want to show that

$$\Delta\left(\widehat{\mathbf{e}}^\intercal \mathcal{D}_{\widehat{\Lambda}+\widehat{\mathbf{c}},r}, \mathcal{D}_{\mathbb{Z},\|\widehat{\mathbf{e}}\|r}\right) \leq 2\varepsilon$$

The support of $\widehat{\mathbf{e}}^\intercal \mathcal{D}_{\widehat{\Lambda}+\widehat{\mathbf{c}},r}$ is $\widehat{\mathbf{e}}^\intercal \widehat{\Lambda} + \widehat{\mathbf{e}}^\intercal \widehat{\mathbf{c}} = \mathbf{e}^\intercal \Lambda_q^\perp(\mathbf{G}^\intercal) + \mathbb{Z} + \mathbf{e}^\intercal \mathbf{c} = \mathbb{Z}$. Fix some $z \in \mathbb{Z}$. The probability mass assigned to z by $\widehat{\mathbf{e}}^\intercal \mathcal{D}_{\widehat{\Lambda}+\widehat{\mathbf{c}},r}$ is proportional to $\rho_r(\mathcal{L}_z)$, where

$$\mathcal{L}_z = \left\{\mathbf{v} \in \widehat{\Lambda} + \widehat{\mathbf{c}} : \widehat{\mathbf{e}}^\intercal \mathbf{v} = z\right\}$$

We define the lattice $\mathcal{L} = \left\{\mathbf{v} \in \widehat{\Lambda} : \widehat{\mathbf{e}}^\intercal \mathbf{v} = 0\right\}$; note that $\mathcal{L}_z = \mathcal{L} + \mathbf{w}_z$ for any $\mathbf{w}_z \in \mathcal{L}_z$. Let $\mathbf{u}_z = \frac{z}{\|\widehat{\mathbf{e}}\|^2 r}\widehat{\mathbf{e}}$, then $\mathbf{u}_z$ is clearly proportional to $\widehat{\mathbf{e}}$. Observe that $\mathbf{u}_z$ is orthogonal to $r^{-1}\mathcal{L}_z - \mathbf{u}_z$, indeed for any $\mathbf{t} \in r^{-1}\mathcal{L}_z$ we have $\widehat{\mathbf{e}}^\intercal(\mathbf{t} - \mathbf{u}_z) = 0$. From this we have $\rho(\mathbf{t}) = \rho(\mathbf{u}_z) \cdot \rho(\mathbf{t} - \mathbf{u}_z)$, and by summing for $\mathbf{t} \in r^{-1}\mathcal{L}_z$:

$$\rho(r^{-1}\mathcal{L}_z) = \rho(\mathbf{u}_z) \cdot \rho\left(r^{-1}\mathcal{L}_z - \mathbf{u}_z\right)$$

Observe that we have $r^{-1}\mathcal{L}_z - \mathbf{u}_z = r^{-1}(\mathcal{L} - \mathbf{c}')$ for some $\mathbf{c}'$ in the vector span of the lattice $\mathcal{L}$ (because $\mathcal{L}_z - r\mathbf{u}_z = \mathcal{L} + \mathbf{w}_z - r\mathbf{u}_z$ and $\widehat{\mathbf{e}}^\intercal(\mathbf{w}_z - r\mathbf{u}_z) = 0$). Thus using Lemmas 5.2.5

and 5.3.4 with $r \geq \sqrt{5}\,(1+\|\mathbf{e}\|)\cdot\sqrt{\frac{\ln(2m(1+1/\varepsilon))}{\pi}} \geq \eta_\varepsilon(\mathcal{L})$, we obtain

$$\begin{aligned}\rho(r^{-1}\mathcal{L}_z) &= \rho(\mathbf{u}_z)\cdot\rho_r(\mathcal{L}-\mathbf{c}') \\ &\in \left[\frac{1-\varepsilon}{1+\varepsilon},\ 1\right]\cdot\rho_r(\mathcal{L})\cdot\rho(\mathbf{u}_z) \\ &= \left[\frac{1-\varepsilon}{1+\varepsilon},\ 1\right]\cdot\rho_r(\mathcal{L})\cdot\rho\left(\frac{z}{\|\widehat{\mathbf{e}}\|^2 r}\widehat{\mathbf{e}}\right) \\ &= \left[\frac{1-\varepsilon}{1+\varepsilon},\ 1\right]\cdot\rho_r(\mathcal{L})\cdot\rho_{\|\widehat{\mathbf{e}}\|r}(z)\end{aligned}$$

This implies that the statistical distance between $\widehat{\mathbf{e}}^\intercal\mathcal{D}_{\widehat{\Lambda}+\widehat{\mathbf{c}},r}$ and $\mathcal{D}_{\mathbb{Z},\|\widehat{\mathbf{e}}\|r}$ is at most $1-\frac{1-\varepsilon}{1+\varepsilon}\leq 2\varepsilon$. □

In order to conclude the previous proof, we now give a bound on the smoothing parameter of the lattice $\mathcal{L}$.

Lemma 5.3.4. *Let $\varepsilon > 0$. For any $\mathbf{e}\in\mathbb{Z}^m$, let $\mathcal{L}$ be as defined in Lemma 5.3.3. Then we have:*

$$\eta_\varepsilon(\mathcal{L}) \leq \sqrt{5}(1+\|\mathbf{e}\|)\cdot\sqrt{\frac{\ln\left(2m\left(1+1/\varepsilon\right)\right)}{\pi}}$$

Proof. We use Lemma 5.2.4 to bound the smoothing parameter of $\mathcal{L}$. Since $\widehat{\Lambda} = \Lambda_q^\perp\left(\mathbf{G}^\intercal\right)\times\mathbb{Z}$ is of dimension $m+1$ and $\mathcal{L}$ is the sublattice of $\widehat{\Lambda}$ made of the vectors that are orthogonal to $\mathbf{e}$, we have that $\mathcal{L}$ is of dimension m. We thus exhibit m independent short vectors of $\mathcal{L}$ to obtain an upper bound on $\lambda_m\left(\mathcal{L}\right)$. We first define the matrix

$$\overline{\mathbf{B}} = \begin{pmatrix} 2 & & & \\ -1 & \ddots & & \\ & \ddots & \ddots & \\ & & -1 & 2\end{pmatrix} \in \mathbb{Z}^{(\log q)\times(\log q)}$$

and remark that it is a basis for the lattice $\Lambda_q^\perp\left(\mathbf{g}^\intercal\right)$. The lattice $\widehat{\Lambda}$ is then generated by the columns of the matrix:

$$\mathbf{B} = (\mathbf{b}_1 \mid \ldots \mid \mathbf{b}_{m+1}) = \left(\begin{array}{c|c} \mathbf{I}_n\otimes\overline{\mathbf{B}} & \mathbf{0} \\ \hline \mathbf{0}^\intercal & 1\end{array}\right) \in \mathbb{Z}^{(m+1)\times(m+1)}$$

For $k\leq m$ let $\mathbf{u}_k = \mathbf{b}_k - \mathbf{b}_{m+1}\cdot\widehat{\mathbf{e}}^\intercal\mathbf{b}_k$, since $\widehat{\mathbf{e}}^\intercal\mathbf{b}_{m+1} = 1$ we directly have $\widehat{\mathbf{e}}^\intercal\mathbf{u}_k = 0$ and thus $\mathbf{u}_k\in\mathcal{L}$. The vectors $\mathbf{u}_1,\ldots,\mathbf{u}_m$ are linearly independent since $\operatorname{span}\left(\mathbf{u}_1,\ldots,\mathbf{u}_m,\mathbf{b}_{m+1}\right) = \operatorname{span}\left(\mathbf{b}_1,\ldots,\mathbf{b}_m,\mathbf{b}_{m+1}\right) = \mathbb{R}^{m+1}$ (which comes from the fact that $\mathbf{B}$ is a basis of an $(m+1)$-dimensional lattice). We now bound the norm of $\mathbf{u}_k$:

$$\begin{aligned}\|\mathbf{u}_k\| &\leq \|\mathbf{b}_k\| + \|\mathbf{b}_{m+1}\|\,\|\mathbf{e}\|\,\|\mathbf{b}_k\| \\ &= \sqrt{5}(1+\|\mathbf{e}\|)\end{aligned}$$

Note that $|\widehat{\mathbf{e}}^\intercal\mathbf{b}_k| \leq \|\mathbf{e}\|\,\|\mathbf{b}_k\|$ since the last coefficient of $\mathbf{b}_k$ is 0. Finally we obtain $\lambda_m(\mathcal{L}) \leq \max_{k\leq m}\|\mathbf{u}_k\| \leq \sqrt{5}(1+\|\mathbf{e}\|)$ and the result. □

The final proof of Lemma 5.3.1 will necessitate a call to the leftover hash lemma, so before continuing we analyze the min-entropy of $\mathbf{x} \leftarrow \mathcal{D}_{\Lambda_q^\perp(\mathbf{G}^\intercal)+\mathbf{c},r}$.

Lemma 5.3.5. *Let $\varepsilon > 0$, $r \geq \eta_\varepsilon\left(\Lambda_q^\perp\left(\mathbf{G}^\intercal\right)\right)$. For any $\mathbf{c} \in \mathbb{R}^m$, we have*

$$H_\infty\left(\mathcal{D}_{\Lambda_q^\perp(\mathbf{G}^\intercal)+\mathbf{c},r}\right) \geq \log\left(1-\varepsilon\right) + m\log\left(r\right) - m$$

Proof. For any $\mathbf{v} \in \Lambda_q^\perp\left(\mathbf{G}^\intercal\right) + \mathbf{c}$

$$\begin{aligned}\mathcal{D}_{\Lambda_q^\perp(\mathbf{G}^\intercal)+\mathbf{c},r}(\mathbf{v}) &\leq \mathcal{D}_{\Lambda_q^\perp(\mathbf{G}^\intercal)+\mathbf{c},r}(\mathbf{v}_0), \text{ for } \mathbf{v}_0 \text{ the point of } \Lambda_q^\perp\left(\mathbf{G}^\intercal\right)+\mathbf{c} \text{ closest to } \mathbf{0}\\ &= \frac{\rho_r(\mathbf{v}_0)}{\rho_r(\Lambda_q^\perp\left(\mathbf{G}^\intercal\right)+\mathbf{c})}\\ &\leq \frac{1}{\rho_r(\Lambda_q^\perp\left(\mathbf{G}^\intercal\right)+\mathbf{c})}, \text{ since } \rho_r(\mathbf{v}_0) < 1\\ &\leq (1-\varepsilon)\frac{r^m}{\det\left(\Lambda_q^\perp\left(\mathbf{G}^\intercal\right)\right)}, \text{ by Lemma 5.2.6 since } r \geq \eta_\varepsilon\left(\Lambda_q^\perp\left(\mathbf{G}^\intercal\right)\right)\end{aligned}$$

The lattice $\Lambda_q^\perp\left(\mathbf{G}^\intercal\right)$ is generated by the basis $\mathbf{I}_n \otimes \overline{\mathbf{B}}$, with $\overline{\mathbf{B}}$ defined as above, which has determinant $\left(2^{\log q}\right)^n = 2^m$. The result follows:

$$H_\infty\left(\mathcal{D}_{\Lambda_q^\perp(\mathbf{G}^\intercal)+\mathbf{c},r}\right) \geq \log\left(1-\varepsilon\right) + m\log\left(r\right) - m$$

□

We are now ready to prove Lemma 5.3.1.

Proof. The proof is done in two steps. First, by Lemma 5.3.5, we know that $\mathbf{x}$ has min entropy at least $\log(1-\varepsilon) + m\log(r) - m \geq (n+1)\log(q) - 2\log(\varepsilon') - 2$. Moreover, $\mathbf{e}^\intercal\mathbf{x} + y$ is in $\mathbb{Z}_q$. Applying the leftover hash lemma (Lemma 5.2.8), we obtain

$$\Delta\left(\left(\mathbf{A}, \mathbf{A}\mathbf{x}, \mathbf{e}^\intercal\mathbf{x} + y\right), \left(\mathbf{A}, \mathbf{u}, \mathbf{e}^\intercal\mathbf{x} + y\right)\right) < \varepsilon'$$

where $\mathbf{u} \leftarrow_\$ \mathbb{Z}_q^{n-1}$. Now, using Lemma 5.3.3, we know that

$$\Delta\left(\mathbf{e}^\intercal\mathbf{x} + y, e'\right) < 2\varepsilon$$

Summing the two statistical distances concludes the proof. □

5.3.2 Rerandomizing LWE samples

We finally describe a simple application of Lemma 5.3.1. Generating fresh LWE samples for a fixed secret $\mathbf{s}$ from a bounded number of samples is very useful, for example to build a public key encryption scheme from a symmetric one. It has already been shown in the succession of papers [Reg05; GPV08; ACPS09] that multiplying a matrix of m LWE samples $(\mathbf{A}, \mathbf{s}^\intercal\mathbf{A} + \mathbf{e}^\intercal)$ by a discrete Gaussian $\mathbf{x} \leftarrow \mathcal{D}_{\mathbb{Z}^m,r}$ and adding another Gaussian term $y \leftarrow \mathcal{D}_{\mathbb{Z},r}$ to the error part yields a fresh LWE sample $(\mathbf{a}', \mathbf{s}^\intercal\mathbf{a}' + e')$ with a somewhat larger Gaussian noise e'. Here we have shown that picking $\mathbf{x}$ according to a discrete Gaussian distribution over a coset $\mathbf{c}$ of $\Lambda_q^{\perp}(\mathbf{G}^{\mathsf{T}})$ is enough for this rerandomization process. Moreover, we show that the distribution of the final error is independent of the coset $\mathbf{c}$, which will come in handy for hiding homomorphic evaluations. We note that this could be extended to any other lattice with a small public basis (cf. Section 5.4.5), but we mainly focus on $\Lambda_q^{\perp}(\mathbf{G}^{\mathsf{T}})$ because this is sufficient for our use.

5.4 Our scheme: circuit-private homomorphic evaluation for GSW

In this section, we prove that a slight modification of the GSW encryption scheme is enough to guarantee circuit privacy, i.e., that an evaluation of any branching program does not reveal anything more than the result of the computation and the length of the branching program, as long as the secret key holder is honest.

5.4.1 Rerandomizing and scaling GSW ciphertexts

First, we show how we can rerandomize and scale a GSW ciphertext. In the following we write $\mathsf{Enc}_\gamma(\mathsf{sk}, \mu)$ to denote that, when calling the encryption algorithm, the error is sampled from a discrete Gaussian distribution with parameter γ. We let α be the parameter of fresh ciphertexts, and implicitly use $\mathsf{Enc}(\mathsf{sk}, \mu)$ to denote $\mathsf{Enc}_\alpha(\mathsf{sk}, \mu)$. Recall from Section 3.4.3 that the form of a GSW ciphertext is

$$\mathbf{C} = \begin{pmatrix} -\mathbf{A} \\ \mathbf{s}^{\mathsf{T}}\mathbf{A} + \mathbf{e}^{\mathsf{T}} \end{pmatrix} + \mu\mathbf{G}.$$

Using the rerandomization of LWE samples presented in Section 5.3.2, it is possible to generate a fresh encryption of 0 by computing $\mathbf{C} \cdot \mathbf{G}_{\text{rand}}^{-1}(\mathbf{V})$, where $\mathbf{C}$ is an encryption of 0 and $\mathbf{V}$ is any matrix in $\mathbb{Z}_q^{n \times m}$.

Lemma 5.4.1. *Let $r > 0$. For any $\mathbf{V} \in \mathbb{Z}_q^{n\times m}$, if $r = \Omega(\alpha\sqrt{\kappa\, m \log m})$, with α being the Gaussian parameter of fresh encryptions, then*

$$\left(\mathbf{C} \cdot \mathbf{G}_{\text{rand}}^{-1}(\mathbf{V}) + \begin{pmatrix} \mathbf{0} \\ \mathbf{y}^{\mathsf{T}} \end{pmatrix}, \mathbf{C}\right) \approx_s (\mathbf{C}', \mathbf{C})$$

where $\mathbf{C} = \begin{pmatrix} -\mathbf{A} \\ \mathbf{s}^{\mathsf{T}}\mathbf{A} + \mathbf{e}^{\mathsf{T}} \end{pmatrix} \leftarrow \mathsf{Enc}(\mathsf{sk}, 0)$, $\mathbf{C}' \leftarrow \mathsf{Enc}_\gamma(\mathsf{sk}, 0)$, *with* $\gamma = r\sqrt{1 + \|\mathbf{e}\|^2}$.

Proof. Fix $\mathbf{v} \in \mathbb{Z}_q^m$ and $\mathbf{e}$ such that $\|\mathbf{e}\| \leq C\alpha\sqrt{m}$, where C is as in Lemma 5.2.3. Then by applying Lemma 5.3.1 with $r = \Omega(\alpha\sqrt{\kappa\, m \log m})$ and $\varepsilon' = \varepsilon = 2^{-\kappa}$ we have

$$\Delta\left((\mathbf{A}, \mathbf{A}\mathbf{x}, \mathbf{e}^{\mathsf{T}}\mathbf{x} + y), (\mathbf{A}, \mathbf{u}, e')\right) < 3 \cdot 2^{-\kappa}$$

where $\mathbf{A} \leftarrow_{\$} \mathbb{Z}_q^{(n-1)\times m}$, $\mathbf{x} \leftarrow \mathbf{G}_{\text{rand}}^{-1}(\mathbf{v})$ and $y \leftarrow \mathcal{D}_{\mathbb{Z},r}$. From this we obtain that for $\mathbf{e} \leftarrow$

$\mathcal{D}_{\mathbb{Z}^m,\alpha}$:

$$\begin{aligned}
&\Delta\left((\mathbf{A},\mathbf{e},\mathbf{A}\mathbf{x},\mathbf{e}^\intercal\mathbf{x}+y),(\mathbf{A},\mathbf{e},\mathbf{u},e')\right)\\
&=\sum_{\mathbf{w}\in\mathbb{Z}^m}\Delta\left((\mathbf{A},\mathbf{A}\mathbf{x},\mathbf{w}^\intercal\mathbf{x}+y),(\mathbf{A},\mathbf{u},w')\right)\cdot\Pr[\mathbf{e}=\mathbf{w}]\\
&\le\sum_{\|\mathbf{w}\|<C\alpha\sqrt{m}}3\cdot2^{-\kappa}\Pr[\mathbf{e}=\mathbf{w}]+\sum_{\|\mathbf{w}\|\ge C\alpha\sqrt{m}}\Pr[\mathbf{e}=\mathbf{w}]\\
&\le 3\cdot2^{-\kappa}+\Pr\left[\|\mathbf{e}\|\ge C\alpha\sqrt{m}\right]\\
&\le 3\cdot2^{-\kappa}+2^{-\Omega(\kappa)}
\end{aligned}$$

In the left operand of the third equation we bound the statistical distance by $3\cdot2^{-\kappa}$ and in the right operand we bound it by 1. To obtain the last inequality we use Lemma 5.2.3 and have $\Pr[\|\mathbf{e}\|>C\alpha\sqrt{m}]\le2^{-\Omega(m)}\le2^{-\Omega(\kappa)}$ since $m\ge\kappa$. By rewriting this distance we have for any $\mathbf{v}\in\mathbb{Z}_q^m$

$$\left(\mathbf{C}\cdot\mathbf{G}_{\text{rand}}^{-1}(\mathbf{v})+\begin{pmatrix}\mathbf{0}\\y\end{pmatrix},\mathbf{C}\right)\approx_s\left(\begin{pmatrix}-\mathbf{u}\\\mathbf{s}^\intercal\mathbf{u}+e'\end{pmatrix},\mathbf{C}\right)$$

By writing $\mathbf{V}=(\mathbf{v}_1\mid\ldots\mid\mathbf{v}_m)$ and $\mathbf{y}=(y_1,\ldots,y_m)$, we have

$$\mathbf{C}\cdot\mathbf{G}_{\text{rand}}^{-1}(\mathbf{V})+\begin{pmatrix}\mathbf{0}\\\mathbf{y}^\intercal\end{pmatrix}=\left(\mathbf{C}\cdot\mathbf{G}_{\text{rand}}^{-1}(\mathbf{v}_1)+\begin{pmatrix}\mathbf{0}\\y_1\end{pmatrix}\mid\ldots\mid\mathbf{C}\cdot\mathbf{G}_{\text{rand}}^{-1}(\mathbf{v}_m)+\begin{pmatrix}\mathbf{0}\\y_m\end{pmatrix}\right)$$

We define the distributions $(D_i)_{0\le i\le m}$ in which the first i columns of $\mathbf{C}\cdot\mathbf{G}_{\text{rand}}^{-1}(\mathbf{V})+\begin{pmatrix}\mathbf{0}\\\mathbf{y}^\intercal\end{pmatrix}$ are replaced with “fresh” $\begin{pmatrix}-\mathbf{u}\\\mathbf{s}^\intercal\mathbf{u}+e'\end{pmatrix}$ and we obtain through a hybrid argument that

$$\Delta\left(\left(\mathbf{C}\cdot\mathbf{G}_{\text{rand}}^{-1}(\mathbf{V})+\begin{pmatrix}\mathbf{0}\\\mathbf{y}^\intercal\end{pmatrix},\mathbf{C}\right),\left(\begin{pmatrix}-\mathbf{A}'\\\mathbf{s}^\intercal\mathbf{A}'+\mathbf{e}'^\intercal\end{pmatrix},\mathbf{C}\right)\right)\le m(3\cdot2^{-\kappa}+2^{-\Omega(\kappa)})$$

□

As a direct corollary we remark that the scaling of a GSW encryption $\mathbf{C}$ of μ by a bit a, defined as $\mathbf{C}\cdot\mathbf{G}_{\text{rand}}^{-1}(a\cdot\mathbf{G})+\begin{pmatrix}\mathbf{0}\\\mathbf{y}^\intercal\end{pmatrix}$, where $\mathbf{y}\leftarrow\mathcal{D}_{\mathbb{Z}^m,r}$, does not depend on a, but only on $a\mu$.

5.4.2 Circuit privacy: definition and main theorem

We now state our definition of circuit privacy, similar to [IP07, Definition 7], which is stronger than the one given in [Gen09a, Definition 2.1.6] in the sense that it is simulation based, but weaker in the sense that we leak information about the length of the branching program.

Definition 5.4.2 (Simulation-based circuit privacy). *We say that a homomorphic encryption scheme $\mathcal{E}$ is circuit private if there exists a PPT algorithm* Sim *such that for any branching program Π of length $L=\mathsf{poly}(\kappa)$ on ℓ variables, any $x_1,\ldots,x_\ell\in\{0,1\}$, the following* *holds:*

$$\begin{aligned}&\left(\mathcal{E}.\mathsf{Eval}\left(\mathsf{evk},\Pi,\left(\mathbf{C}_1,\ldots,\mathbf{C}_\ell\right)\right),\mathbf{C}_1,\ldots,\mathbf{C}_\ell,1^\kappa,\mathsf{sk}\right)\\ \approx_s\ &\left(\mathsf{Sim}\left(1^\kappa,\Pi\left(x_1,\ldots,x_\ell\right),1^L,\left(\mathbf{C}_1,\ldots,\mathbf{C}_\ell\right)\right),\mathbf{C}_1,\ldots,\mathbf{C}_\ell,1^\kappa,\mathsf{sk}\right)\end{aligned}$$

where $\mathsf{sk} \leftarrow \mathcal{E}.\mathsf{Setup}\left(1^\kappa\right)$*,* $\mathbf{C}_i \leftarrow \mathcal{E}.\mathsf{Enc}\left(\mathsf{sk}, x_i\right)$ *for* $i \in [\ell]$*.*

We can now state our main theorem:

Theorem 5.4.3 (Main theorem)**.** *There exists a fully homomorphic encryption scheme for branching programs that is circuit private and whose security is based on the LWE assumption with polynomial noise-to-modulus ratio.*

Remark 5.4.4. *The aforementioned scheme is also multi-hop (see definition in [GHV10]) for branching programs, as long as the noise does not grow beyond $q/4$. This means that the output of an evaluation can be used as input for further computation, while the property of circuit privacy is maintained for every hop. More in detail, the evaluation can be carried out by multiple parties and any subset of these parties is not able to gain information about the branching program applied by an evaluator which is not in the subset, beside its length, input and output, even given access to the secret key.*

5.4.3 Modified Eval algorithm for the GSW encryption scheme

5.4.3.1 Homomorphic evaluation for branching programs

Here we present our $\mathsf{Eval}\left(\Pi,\left(\mathbf{C}_1,\ldots,\mathbf{C}_\ell\right)\right)$ algorithm (note that it does not require any evaluation key), which homomorphically evaluates a branching program Π over ciphertexts $\mathbf{C}_1,\ldots,\mathbf{C}_\ell$. The first state vector is encrypted without noise: the initial encrypted state vector is $\mathbf{V}_0 = (\mathbf{G},\mathbf{0},\ldots,\mathbf{0})$, i.e. $\mathbf{V}_0[1] = \mathbf{G}$ and $\mathbf{V}_0[w] = \mathbf{0}$, for $2 \le w \le W$. Note that $\mathbf{G}$ and $\mathbf{0}$ are noiseless encryptions of 1 and 0, respectively. The encrypted state vector is then computed at each step by homomorphically applying Equation (5.1) and adding a noise term: for $t \in [L]$ and $w \in [W]$

$$\mathbf{V}_t\left[w\right] \leftarrow \mathbf{C}_{\mathrm{var}(t)}\cdot\mathbf{G}^{-1}_{\mathrm{rand}}\left(\mathbf{V}_{t-1}\left[\pi^{-1}_{t,1}\left(w\right)\right]\right)+\left(\mathbf{G}-\mathbf{C}_{\mathrm{var}(t)}\right)\cdot\mathbf{G}^{-1}_{\mathrm{rand}}\left(\mathbf{V}_{t-1}\left[\pi^{-1}_{t,0}\left(w\right)\right]\right)+\boxed{\begin{pmatrix}\mathbf{0}\\ \mathbf{y}^{\mathsf{T}}_{t,w}\end{pmatrix}} \tag{5.2}$$

where $\mathbf{y}_{t,w} \leftarrow \mathcal{D}_{\mathbb{Z}^m, r\sqrt{2}}$. The output of the evaluation algorithm is $\mathbf{V}_L[0] \in \mathbb{Z}_q^{n\times m}$.

Remark 5.4.5 (Comparison with [BV14; AP14]. Cf. also Table 5.1)**.** *The differences between our homomorphic evaluation procedure and the previous ones are as follows:*

- *We added an additional Gaussian noise to the computation, as captured in the boxed term;*
- *[BV14] uses the deterministic $\mathbf{G}^{-1}_{\mathrm{det}}(\cdot)$ whereas [AP14] introduced the randomized $\mathbf{G}^{-1}_{\mathrm{rand}}(\cdot)$ for efficiency. Here, we crucially exploit the randomized $\mathbf{G}^{-1}_{\mathrm{rand}}(\cdot)$ for privacy.*

Simulator. Towards proving circuit privacy, we need to specify a simulator Sim. We first describe a simulator that is given access to the number of times each variable is used and prove that its output distribution is statistically close to the result of Eval (Lemma 5.4.8). We can then pad the branching program so that each variable is used the same number of times. Given the security parameter κ, the length L of the branching program Π, the number of times τ_i that Π uses the i-th variable, the final value x_f of the evaluation of Π on input $(x_1, \ldots, x_\ell)$, the ciphertexts $\mathbf{C}_i$ encrypting x_i for $i \in [\ell]$, Sim mimics the way error grows in the states of Eval by doing τ_i dummy steps of computation with the i-th variable. This gives a new encryption $\widehat{\mathbf{A}}_f$ of 0 with the same noise distribution as the ciphertext output by the Eval procedure. Sim then adds the message part x_f to this ciphertext and outputs $\mathbf{C}_f = \widehat{\mathbf{A}}_f + x_f\mathbf{G}$.

In other words,

$$\mathsf{Sim}\left(1^\kappa, x_f, \left(1^{\tau_1}, \ldots, 1^{\tau_\ell}\right), (\mathbf{C}_1, \ldots, \mathbf{C}_\ell)\right) \leftarrow \sum_{i=1}^{\ell} \sum_{t=1}^{\tau_i} \left(\mathbf{C}_i \cdot \left(\mathbf{G}^{-1}_{\text{rand}}(\mathbf{0}) - \mathbf{G}^{-1}_{\text{rand}}(\mathbf{0}) \right) + \begin{pmatrix} \mathbf{0} \\ \mathbf{y}_t^{\mathsf{T}} \end{pmatrix} \right) + x_f\mathbf{G}$$

where $\mathbf{y}_t \leftarrow \mathcal{D}_{\mathbb{Z}^m, r\sqrt{2}}$ for $t \in [L]$.

We note that the sum of $2\tau_i$ samples $\mathbf{G}^{-1}_{\text{rand}}(\mathbf{0})$ can be sampled at once using the $\mathbf{G}^{-1}_{\text{rand}}(\cdot)$ algorithm with a larger parameter $r\sqrt{2\tau_i}$, and the sum of τ_i samples from $\mathcal{D}_{\mathbb{Z}^m, r\sqrt{2}}$ is close to a sample from $\mathcal{D}_{\mathbb{Z}^m, r\sqrt{2\tau_i}}$.

5.4.3.2 Proof of circuit privacy

We proceed to establish circuit privacy in two steps. We first analyze how the ciphertext distribution changes in a single transition, and then proceed by induction to reason about homomorphic evaluation of the entire branching program.

Step 1. We begin with the following lemma, which is useful for analyzing the output of Equation (5.2). Roughly speaking, this lemma says that if at step t, the state vector consists of fresh GSW encryptions with some noise parameter ζ, then at step $t+1$, the state vector is statistically close to fresh GSW encryptions with a somewhat larger noise which depends on the error in the input ciphertext and on ζ.

Lemma 5.4.6. *For any $x, v_0, v_1 \in \{0,1\}$ and $\mathsf{sk} = \left(-\mathbf{s}, 1\right) \leftarrow \mathsf{KGen}\left(1^\kappa\right)$, the following holds:*

$$\left(\mathbf{C} \cdot \mathbf{G}^{-1}_{\text{rand}}(\mathbf{V}_1) + (\mathbf{G} - \mathbf{C}) \cdot \mathbf{G}^{-1}_{\text{rand}}(\mathbf{V}_0) + \begin{pmatrix} \mathbf{0} \\ \mathbf{y}^{\mathsf{T}} \end{pmatrix}, \mathbf{C} \right) \approx_s \left(\mathbf{V}'_x, \mathbf{C}\right)$$

where $\mathbf{V}_b \leftarrow \mathsf{Enc}_\gamma\left(\mathsf{sk}, v_b\right)$ for $b \in \{0,1\}$, $\mathbf{C} = \begin{pmatrix} \mathbf{A} \\ \mathbf{s}^{\mathsf{T}}\mathbf{A} + \mathbf{e}^{\mathsf{T}} \end{pmatrix} + x\mathbf{G} \leftarrow \mathsf{Enc}\left(\mathsf{sk}, x\right)$, $\mathbf{y} \leftarrow \mathcal{D}_{\mathbb{Z}^m, r\sqrt{2}}$ and $\mathbf{V}'_x \leftarrow \mathsf{Enc}_\zeta\left(\mathsf{sk}, v_x\right)$, with $\zeta = \sqrt{\gamma^2 + 2r^2\left(1 + \|\mathbf{e}\|^2\right)}$.

Proof. We begin with a simple identity which is useful in the remainder of the proof:

$$\mathbf{C} \cdot \mathbf{G}^{-1}_{\text{rand}}(\mathbf{V}_1) + (\mathbf{G} - \mathbf{C}) \cdot \mathbf{G}^{-1}_{\text{rand}}(\mathbf{V}_0) = \widehat{\mathbf{A}} \cdot \left(\mathbf{G}^{-1}_{\text{rand}}(\mathbf{V}_1) - \mathbf{G}^{-1}_{\text{rand}}(\mathbf{V}_0) \right) + \mathbf{V}_x$$

where $\widehat{\mathbf{A}} = \begin{pmatrix} \mathbf{A} \\ \mathbf{s}^{\mathsf{T}}\mathbf{A} + \mathbf{e}^{\mathsf{T}} \end{pmatrix}$ and $\mathbf{V}_0, \mathbf{V}_1, \mathbf{C}$ are as defined in the statement of the Lemma.

Showing this identity is correct just requires performing the calculations:

$$\begin{aligned}
&\mathbf{C}\cdot\mathbf{G}^{-1}_{\text{rand}}\left(\mathbf{V}_1\right)+\left(\mathbf{G}-\mathbf{C}\right)\cdot\mathbf{G}^{-1}_{\text{rand}}\left(\mathbf{V}_0\right)\\
=&\left(\widehat{\mathbf{A}}+x\mathbf{G}\right)\cdot\mathbf{G}^{-1}_{\text{rand}}\left(\mathbf{V}_1\right)+\left(\left(1-x\right)\mathbf{G}-\widehat{\mathbf{A}}\right)\cdot\mathbf{G}^{-1}_{\text{rand}}\left(\mathbf{V}_0\right)\\
=&\,\widehat{\mathbf{A}}\cdot\left(\mathbf{G}^{-1}_{\text{rand}}\left(\mathbf{V}_1\right)-\mathbf{G}^{-1}_{\text{rand}}\left(\mathbf{V}_0\right)\right)+x\mathbf{V}_1+\left(1-x\right)\mathbf{V}_0\\
=&\,\widehat{\mathbf{A}}\cdot\left(\mathbf{G}^{-1}_{\text{rand}}\left(\mathbf{V}_1\right)-\mathbf{G}^{-1}_{\text{rand}}\left(\mathbf{V}_0\right)\right)+\mathbf{V}_x
\end{aligned}$$

Then we observe that by applying Lemma 5.2.2 we have

$$\begin{pmatrix}\mathbf{0}\\ \mathbf{y}^{\mathsf{T}}\end{pmatrix}\approx_s\begin{pmatrix}\mathbf{0}\\ \mathbf{y}_1^{\mathsf{T}}\end{pmatrix}-\begin{pmatrix}\mathbf{0}\\ \mathbf{y}_0^{\mathsf{T}}\end{pmatrix}$$

where $\mathbf{y}_b\leftarrow\mathcal{D}_{\mathbb{Z}^m,r}, b\in\{0,1\}$. Lemma 5.4.1 also gives

$$\left(\widehat{\mathbf{A}}\cdot\mathbf{G}^{-1}_{\text{rand}}\left(\mathbf{V}_b\right)+\begin{pmatrix}\mathbf{0}\\ \mathbf{y}_b^{\mathsf{T}}\end{pmatrix},\mathbf{C}\right)\approx_s\left(\mathbf{C}_b,\mathbf{C}\right)$$

where $\mathbf{C}_b\leftarrow\mathsf{Enc}_{\zeta'}(\mathsf{sk},0)$, for $b\in\{0,1\}$, with $\zeta'=r\sqrt{1+\|\mathbf{e}\|^2}$. We now have

$$\left(\mathbf{C}\cdot\mathbf{G}^{-1}_{\text{rand}}\left(\mathbf{V}_1\right)+\left(\mathbf{G}-\mathbf{C}\right)\cdot\mathbf{G}^{-1}_{\text{rand}}\left(\mathbf{V}_0\right)+\begin{pmatrix}\mathbf{0}\\ \mathbf{y}^{\mathsf{T}}\end{pmatrix},\mathbf{C}\right)\approx_s\left(\mathbf{C}_1-\mathbf{C}_0+\mathbf{V}_x,\mathbf{C}\right)$$

By additivity of variance on independent variables, we obtain that $\mathbf{C}_1-\mathbf{C}_0+\mathbf{V}_x=\mathbf{V}'_x$ looks like a fresh encryption of $0-0+v_x=v_x$ with parameter $\sqrt{\gamma^2+2r^2(1+\|\mathbf{e}\|^2)}$. □

Step 2. We now prove that, at each step of the evaluation, each entry of the encrypted state $\mathbf{V}_t$ looks like a fresh GSW encryption of the corresponding entry of the state $\mathbf{v}_t$, even given the GSW encryptions of the input bits, except for a small correlation in the noise.

Lemma 5.4.7 (Distribution of the result of Eval)**.** *For any branching program Π of length L on ℓ variables, we define $\tau_{t,i}$ to be the number of times the i-th variable has been used after t steps of the evaluation, i.e., $\tau_{t,i}=\left|\text{var}^{-1}\left(i\right)\cap\left[t\right]\right|$, for i in $[\ell]$ and $t\in[L]$.*
For any $x_1,\ldots,x_\ell\in\{0,1\}$, any $\mathsf{sk}=\left(-\mathbf{s},1\right)\leftarrow\mathsf{KGen}\left(1^\kappa\right)$, at each step $t\in[L]$, for all indexes $w\in[W]$, the following holds:

$$\left(\mathbf{V}_t\left[w\right],\left(\mathbf{C}_i\right)_{i\in[\ell]}\right)\approx_s\left(\mathbf{C}'_{t,w},\left(\mathbf{C}_i\right)_{i\in[\ell]}\right)$$

where $\mathbf{C}_i=\begin{pmatrix}\mathbf{A}_i\\ \mathbf{s}^{\mathsf{T}}\mathbf{A}_i+\mathbf{e}_i^{\mathsf{T}}\end{pmatrix}+x_i\mathbf{G}\leftarrow\mathsf{Enc}\left(\mathsf{sk},x_i\right)$ for $i\in[\ell]$, $\mathbf{C}'_{t,w}\leftarrow\mathsf{Enc}_{r_t}\left(\mathsf{sk},\mathbf{v}_t\left[w\right]\right)$ for $(t,w)\in[L]\times[W]$ and $r_t=r\sqrt{2\sum_{i=1}^{\ell}\tau_{t,i}\left(1+\|\mathbf{e}_i\|^2\right)}$.

Proof. We prove this lemma by induction on $t\in[L]$. At step $t>1$, for index $w\in[W]$ we use a series of hybrid distributions $\mathcal{H}_{t,w,k}$ for $0\leq k\leq 2$ to prove that $\left(\mathbf{V}_t\left[w\right],\left(\mathbf{C}_i\right)_{i\in[\ell]}\right)\approx_s\left(\mathbf{C}'_{t,w},\left(\mathbf{C}_i\right)_{i\in[\ell]}\right)$. In particular $\mathcal{H}_{t,w,0}=\left(\mathbf{V}_t\left[w\right],\left(\mathbf{C}_i\right)_{i\in[\ell]}\right)$, and $\mathcal{H}_{t,w,2}=\left(\mathbf{C}'_{t,w},\left(\mathbf{C}_i\right)_{i\in[\ell]}\right)$.

Hybrid $\mathcal{H}_{t,w,0}$. Let $w_b = \pi_{t,b}^{-1}(w)$ for $b \in \{0,1\}$. We write w_β to denote $w_{x_{\mathrm{var}(t)}}$, i.e. w_0 or w_1, depending on the value of the variable which is used at time t.

$$\mathcal{H}_{t,w,0} = \left(\mathbf{V}_t[w], (\mathbf{C}_i)_{i\in[\ell]}\right)$$
$$= \left(\mathbf{C}_{\mathrm{var}(t)} \cdot \mathbf{G}_{\mathrm{rand}}^{-1}(\mathbf{V}_{t-1}[w_1]) + \left(\mathbf{G} - \mathbf{C}_{\mathrm{var}(t)}\right) \cdot \mathbf{G}_{\mathrm{rand}}^{-1}(\mathbf{V}_{t-1}[w_0]) + \begin{pmatrix}\mathbf{0}\\ \mathbf{y}_{t,w}^{\mathsf{T}}\end{pmatrix}, (\mathbf{C}_i)_{i\in[\ell]}\right)$$

where $\mathbf{C}_i \leftarrow \mathsf{Enc}(\mathsf{sk}, x_i)$ and $\mathbf{y}_{t,w} \leftarrow \mathcal{D}_{\mathbb{Z}^m, r\sqrt{2}}$.

Hybrid $\mathcal{H}_{t,w,1}$. We set

$$\mathcal{H}_{t,w,1} = \left(\mathbf{C}_{\mathrm{var}(t)} \cdot \mathbf{G}_{\mathrm{rand}}^{-1}\left(\mathbf{C}'_{t-1,w_1}\right) + \left(\mathbf{G} - \mathbf{C}_{\mathrm{var}(t)}\right) \cdot \mathbf{G}_{\mathrm{rand}}^{-1}\left(\mathbf{C}'_{t-1,w_0}\right) + \begin{pmatrix}\mathbf{0}\\ \mathbf{y}_{t,w}^{\mathsf{T}}\end{pmatrix}, (\mathbf{C}_i)_{i\in[\ell]}\right)$$

where $\mathbf{C}_i \leftarrow \mathsf{Enc}(\mathsf{sk}, x_i)$, $\mathbf{y}_{t,w} \leftarrow \mathcal{D}_{\mathbb{Z}^m, r\sqrt{2}}$ and $\mathbf{C}'_{t-1,w_b} \leftarrow \mathsf{Enc}_{r_{t-1}}(\mathsf{sk}, \mathbf{v}_{t-1}[w_b])$ for $b \in \{0,1\}$. By induction hypothesis we have $\mathcal{H}_{t-1,w_b,0} \approx_s \mathcal{H}_{t-1,w_b,2}$ for $b \in \{0,1\}$, i.e.,

$$\left(\mathbf{V}_{t-1}[w_b], (\mathbf{C}_i)_{i\in[\ell]}\right) \approx_s \left(\mathbf{C}'_{t-1,w_b}, (\mathbf{C}_i)_{i\in[\ell]}\right)$$

where $\mathbf{C}_i \leftarrow \mathsf{Enc}(\mathsf{sk}, x_i)$ and $\mathbf{C}'_{t-1,w_b} \leftarrow \mathsf{Enc}_{r_{t-1}}(\mathsf{sk}, \mathbf{v}_{t-1}[w_b])$ for $b \in \{0,1\}$.
We use the fact that applying a function to two distributions does not increase their statistical distance to obtain $\mathcal{H}_{t,w,0} \approx_s \mathcal{H}_{t,w,1}$.

Hybrid $\mathcal{H}_{t,w,2}$. Let

$$\mathcal{H}_{t,w,2} = \left(\mathbf{C}', (\mathbf{C}_i)_{i\in[\ell]}\right)$$

with $\mathbf{C}_i \leftarrow \mathsf{Enc}(\mathsf{sk}, x_i)$, $\mathbf{C}' \leftarrow \mathsf{Enc}_\zeta(\mathsf{sk}, \mathbf{v}_{t-1}[w_\beta])$ and $\zeta = \sqrt{r_{t-1}^2 + 2r^2\left(1 + \left\|\mathbf{e}_{\mathrm{var}(t)}\right\|^2\right)}$.

By Lemma 5.4.6 we have:

$$\left(\mathbf{C}_{\mathrm{var}(t)} \cdot \mathbf{G}_{\mathrm{rand}}^{-1}\left(\mathbf{C}'_{t-1,w_1}\right) + \left(\mathbf{G} - \mathbf{C}_{\mathrm{var}(t)}\right) \cdot \mathbf{G}_{\mathrm{rand}}^{-1}\left(\mathbf{C}'_{t-1,w_0}\right) + \begin{pmatrix}\mathbf{0}\\ \mathbf{y}_{t,w}^{\mathsf{T}}\end{pmatrix}, (\mathbf{C}_i)_{i\in[\ell]}\right)$$
$$\approx_s \left(\mathbf{C}', (\mathbf{C}_i)_{i\in[\ell]}\right)$$

where $\mathbf{C}_i \leftarrow \mathsf{Enc}(\mathsf{sk}, x_i)$, $\mathbf{y}_{t,w} \leftarrow \mathcal{D}_{\mathbb{Z}^m, r\sqrt{2}}$, $\mathbf{C}'_{t-1,w_b} \leftarrow \mathsf{Enc}_{r_{t-1}}(\mathsf{sk}, \mathbf{v}_{t-1}[w_b])$ for $b \in \{0,1\}$ and $\mathbf{C}' \leftarrow \mathsf{Enc}_\zeta(\mathsf{sk}, \mathbf{v}_{t-1}[w_\beta])$. Note that $\mathbf{v}_{t-1}[w_\beta] = \mathbf{v}_t[w]$ and $r_t = \sqrt{r_{t-1}^2 + 2r^2\left(1 + \left\|\mathbf{e}_{\mathrm{var}(t)}\right\|^2\right)} = \zeta$ from which we have that $\mathbf{C}'$ and $\mathbf{C}'_{t,w}$ are identically distributed, and directly $\mathcal{H}_{t,w,1} \approx_s \mathcal{H}_{t,w,2}$.

We note that this recursive formula does not apply to step $t = 0$, we thus use $t = 1, w \in [W]$ as the base case. We only describe the steps that differ from the case $t > 1$.

Hybrid $\mathcal{H}_{1,w,1}$. We have $\mathbf{G}_{\mathrm{rand}}^{-1}(\mathbf{V}_0[w_b]) = \mathbf{G}_{\mathrm{rand}}^{-1}(\mathbf{v}_0[w_b] \cdot \mathbf{G})$ for $b \in \{0,1\}$. Notice that we now have exactly $\mathcal{H}_{1,w,1} = \mathcal{H}_{1,w,0}$.

Hybrids $\mathcal{H}_{1,w,2}$. The proof for $\mathcal{H}_{1,w,1} \approx_s \mathcal{H}_{1,w,2}$ is identical to the one of Lemma 5.4.6 except for the fact that the ciphertext $\mathbf{V}_x$ from the proof is now of the form $\mathbf{v}_0[w_\beta]\mathbf{G}$. The resulting ciphertext $\mathbf{C}'_{1,w}$ is now only the sum of two encryptions of 0 and $\mathbf{v}_0[w_\beta]$ and has a Gaussian parameter $r\sqrt{2\left(1 + \left\|\mathbf{e}_{\mathrm{var}(1)}\right\|^2\right)} = r_1$. This implies $\mathcal{H}_{1,w,1} \approx_s \mathcal{H}_{1,w,2}$. □

We now proceed to proving circuit privacy. We will first prove the following lemma, which states that the Eval algorithm presented in Section 5.4.3.1 only leaks the final result of the evaluation and the number of times each variable is used.

Lemma 5.4.8. *Let $\mathcal{E}$ be the GSW scheme defined in Section 3.4.3 with evaluation defined as in this section, and Sim be the corresponding simulator. Then for any branching program Π of length $L = \mathsf{poly}(\kappa)$ on ℓ variables, such that the i-th variable is used τ_i times, and any $x_1, \dots, x_\ell \in \{0,1\}$, the following holds:*

$$\begin{aligned}&\left(\mathcal{E}.\mathsf{Eval}\left(\Pi, (\mathbf{C}_1, \dots, \mathbf{C}_\ell)\right), \mathbf{C}_1, \dots, \mathbf{C}_\ell, 1^\kappa, \mathsf{sk}\right)\\ \approx_s &\left(\mathsf{Sim}\Big(1^\kappa, \Pi\left(x_1, \dots, x_\ell\right), (1^{\tau_1}, \dots, 1^{\tau_\ell}), (\mathbf{C}_1, \dots, \mathbf{C}_\ell)\Big), \mathbf{C}_1, \dots, \mathbf{C}_\ell, 1^\kappa, \mathsf{sk}\right)\end{aligned}$$

where $\mathsf{sk} \leftarrow \mathcal{E}.\mathsf{KGen}\left(1^\kappa\right)$*,* $\mathbf{C}_i \leftarrow \mathcal{E}.\mathsf{Enc}\left(\mathsf{sk}, x_i\right)$ *for* i *in* $[\ell]$*.*

Proof. As shown in Lemma 5.4.7, the final result of the homomorphic evaluation of the branching program Π is of the form

$$\mathbf{V}_L\left[0\right] \approx_s \begin{pmatrix} \mathbf{A} \\ \mathbf{s}^\intercal \mathbf{A} + \mathbf{f}^\intercal \end{pmatrix} + x_f \mathbf{G}$$

where $\mathbf{A} \leftarrow_\$ \mathbb{Z}_q^{(n-1)\times m}$, $\mathbf{f} \leftarrow \mathcal{D}_{\mathbb{Z}^m, r_L}$ and $r_L = r\sqrt{2\sum_{i=1}^{\ell}\left(1 + \|\mathbf{e}_i\|^2\right)\tau_i}$.
Now we prove that the output of Sim is statistically close to the same distribution. This proof follows from the fact that scaling GSW ciphertexts yields a result which is independent of the argument of $\mathbf{G}_{\text{rand}}^{-1}\left(\cdot\right)$. Let $\mathbf{A}_{i,t}, \mathbf{A}'_{i,t} \leftarrow_\$ \mathbb{Z}_q^{(n-1)\times m}$, $\mathbf{f}_{i,f}, \mathbf{f}'_{i,t} \leftarrow \mathcal{D}_{\mathbb{Z}^m, r\sqrt{1+\|\mathbf{e}_i\|}}$, then the joint distribution of the output of Sim and ciphertexts $\left(\mathbf{C}_i\right)_{i\in[\ell]}$ is

$$\begin{aligned}\left(\mathcal{S}, \left(\mathbf{C}_i\right)_{i\in[\ell]}\right) &= \left(\sum_{i=1}^{\ell} \mathbf{C}_i \sum_{t=1}^{\tau_i}\left(\mathbf{G}_{\text{rand}}^{-1}\left(\mathbf{0}\right) - \mathbf{G}_{\text{rand}}^{-1}\left(\mathbf{0}\right)\right) + \begin{pmatrix}\mathbf{0}\\ \mathbf{y}_t^\intercal\end{pmatrix} + x_f\mathbf{G}, \left(\mathbf{C}_i\right)_{i\in[\ell]}\right)\\ &\approx_s \left(\sum_{i=1}^{\ell}\sum_{t=1}^{\tau_i}\begin{pmatrix}\mathbf{A}_{i,t}\\ \mathbf{s}^\intercal\mathbf{A}_{i,t} + \mathbf{f}_{i,t}\end{pmatrix} + \begin{pmatrix}\mathbf{A}'_{i,t}\\ \mathbf{s}^\intercal\mathbf{A}'_{i,t} + \mathbf{f}'_{i,t}\end{pmatrix}, \left(\mathbf{C}_i\right)_{i\in[\ell]}\right)\end{aligned}$$

by Lemma 5.3.1

$$\approx_s \left(\begin{pmatrix}\mathbf{A}\\ \mathbf{s}^\intercal\mathbf{A} + \mathbf{f}^\intercal\end{pmatrix}, \left(\mathbf{C}_i\right)_{i\in[\ell]}\right)$$

by Lemma 5.2.2 and summing uniform variables.

The result is the same as the joint distribution of the output of Eval and ciphertexts $\left(\mathbf{C}_i\right)_{i\in[\ell]}$, thus concluding the proof. □

We are now ready to prove Theorem 5.4.3.

Main theorem. Theorem 5.4.3 follows from Lemma 5.4.8 by tweaking the Eval algorithm of $\mathcal{E}$: it is sufficient that this algorithm pads the branching program Π so that each variable is used L times. This padding is done by using the identity permutation for all steps after the L-th. After this proof, we discuss more efficient ways to pad branching program evaluations. It is easy to see that this step is enough to reach the desired circuit privacy property: the only information leaked besides the final result is $\tau_i = L$. □

Padding branching program evaluations. In order to pad a branching program Π that uses the i-th variable τ_i times to one that uses the i-th variable L times, we add $L - \tau_i$ steps, using the identity permutation at each one of these. Given $\mathbf{V}_L[0]$ the final result of the computation, this padding corresponds to steps $t \in [L+1, 2L-\tau_i]$ defined as follows:

$$\mathbf{V}_t[0] \leftarrow \mathbf{V}_{t-1}[0] + \mathbf{C}_i\left(\mathbf{G}^{-1}_{\text{rand}}(\mathbf{V}_{t-1}[0]) - \mathbf{G}^{-1}_{\text{rand}}(\mathbf{V}_{t-1}[0])\right) + \begin{pmatrix}\mathbf{0}\\ \mathbf{y}^{\mathsf{T}}_{t,0}\end{pmatrix}$$

Using the same proof as Lemma 5.4.8 the final output will be

$$\begin{aligned}\mathbf{V}_{2L-\tau_i}[0] &\leftarrow \mathbf{V}_L[0] + \mathbf{C}_i\sum_{t=L}^{2L-\tau_i-1}\left(\mathbf{G}^{-1}_{\text{rand}}(\mathbf{V}_t[0]) - \mathbf{G}^{-1}_{\text{rand}}(\mathbf{V}_t[0])\right) + \begin{pmatrix}\mathbf{0}\\ \mathbf{y}^{\mathsf{T}}_{t,0}\end{pmatrix}\\ &\approx_s \mathbf{V}_L[0] + \mathbf{C}_i\sum_{t=L}^{2L-\tau_i-1}\left(\mathbf{G}^{-1}_{\text{rand}}(\mathbf{0}) - \mathbf{G}^{-1}_{\text{rand}}(\mathbf{0})\right) + \begin{pmatrix}\mathbf{0}\\ \mathbf{y}^{\mathsf{T}}_{t,0}\end{pmatrix}\end{aligned}$$

Observe that by using Lemma 5.2.2 we have that

$$\begin{aligned}&\sum_{t=L}^{2L-\tau_i-1}\left(\mathbf{G}^{-1}_{\text{rand}}(\mathbf{0}) - \mathbf{G}^{-1}_{\text{rand}}(\mathbf{0})\right) \approx_s \mathcal{D}_{\Lambda_q^{\perp}(\mathbf{G}^{\mathsf{T}}),r_f}\\ &\sum_{t=L}^{2L-\tau_i-1}\begin{pmatrix}\mathbf{0}\\ \mathbf{y}^{\mathsf{T}}_{t,0}\end{pmatrix} \approx_s \mathcal{D}_{\mathbb{Z}^m,r_f}\end{aligned}$$

Where $r_f = r\sqrt{2(L-\tau_i)}$. We can thus do all the steps at once by outputting $\mathbf{V}_L[0] + \mathbf{C}_i \cdot \mathbf{X} + \begin{pmatrix}\mathbf{0}\\ \mathbf{y}^{\mathsf{T}}_f\end{pmatrix}$, where $\mathbf{X} \leftarrow \mathcal{D}^m_{\Lambda_q^{\perp}(\mathbf{G}^{\mathsf{T}}),r_f}$ and $\mathbf{y}_f \leftarrow \mathcal{D}_{\mathbb{Z}^m,r_f}$. We note that $\mathbf{X}$ can be sampled using the $\mathbf{G}^{-1}_{\text{rand}}(\cdot)$ algorithm with parameter r_f instead of r.

5.4.4 Setting the parameters

In this section we show that, for appropriate values of the parameters, the output of the homomorphic evaluation $\mathbf{V}_L[0]$ decrypts to $\Pi(x_1, \dots, x_\ell)$ with overwhelming probability and guarantees circuit privacy.

We first recall the bounds on the parameters needed for both correctness and privacy. Let $n = \Theta(\kappa)$, $q = \mathsf{poly}(n)$, $m = n\log q$, α be the Gaussian parameter of fresh encryptions, r be the parameter of $\mathbf{G}^{-1}_{\text{rand}}(\cdot)$. Let $B = \Theta(\alpha\sqrt{m})$ be a bound on the norm of the error in fresh encryptions (using a tail cutting argument we can show that $B = C\alpha\sqrt{m}$ is sufficient to have a bound with overwhelming probability), $L_{\max} = \mathsf{poly}(n)$ be a bound on the size of the branching programs we consider and $\ell_{\max} = \mathsf{poly}(n)$ an upper bound on their number of variables. Let $\varepsilon = \mathcal{O}(2^{-\kappa})$ and $\varepsilon' = \mathcal{O}(2^{-\kappa})$.

We have the following constraints:

- $\alpha = \Omega(\sqrt{m})$ for the hardness of $\mathsf{lwe}_{n-1,q,\mathcal{D}_{\mathbb{Z},\alpha}}$
- $r \geq \sqrt{\frac{5\ln(2m(1+1/\varepsilon))}{\pi}}$ for the correctness of $\mathbf{G}^{-1}_{\text{rand}}(\cdot)$ sampling
- $r \geq 4\left((1-\varepsilon)(2\varepsilon')^2\right)^{-\frac{1}{m}}$ for the leftover hash lemma

- $r \geq \sqrt{5}\,(1+B)\sqrt{\frac{\ln(2m(1+1/\varepsilon))}{\pi}}$ for Lemma 5.3.4
- $q = \Omega\left(\sqrt{m} r\alpha\,(m\,L_{\max}\,\ell_{\max})^{1/2}\right)$ for the correctness of decryption

We can thus set the parameters as follows:

- $n = \Theta(\kappa)$,
- $L_{\max} = \mathsf{poly}\,(n)$,
- $\ell_{\max} = \mathsf{poly}\,(n)$,
- $\alpha = \Theta(\sqrt{n})$,
- $r = \widetilde{\Theta}\,(n)$,
- $q = \widetilde{\Theta}\left(n^{5/2} \cdot L_{\max} \cdot \ell_{\max}\right)$, a power of 2.

Note that the ciphertext size grows with $\log L_{\max}$. Correctness follows directly.

Lemma 5.4.9 (Correctness)**.** *For any branching program Π of length L on ℓ variables, any $x_1, \ldots, x_\ell \in \{0,1\}$, the result of the homomorphic evaluation $\mathbf{C}_f = \mathsf{Eval}\,(\Pi, (\mathbf{C}_1, \ldots, \mathbf{C}_\ell))$ decrypts to $\Pi\,(x_1, \ldots, x_\ell)$ with overwhelming probability, where $\mathbf{C}_i \leftarrow \mathsf{Enc}\,(\mathsf{sk}, x_i)$ for $i \in [\ell]$ and $\mathsf{sk} \leftarrow \mathsf{KGen}\,(1^\kappa)$.*

Proof. Lemma 5.4.7 shows that the noise distribution of the output $\mathbf{C}_f$ of Eval has parameter $r_f = r\sqrt{2\sum_{i=1}^{\ell}\tau_i\left(1+\|\mathbf{e}_i\|^2\right)}$, that is $r\sqrt{2L\sum_{i=1}^{\ell}\left(1+\|\mathbf{e}_i\|^2\right)}$ because of the padding we applied to Π. We have $r_f \leq r\sqrt{2L\ell\,(1+C^2\alpha^2 m)}$ with C the universal constant defined in Lemma 5.2.3. Using the bounds $L_{\max}$ and $\ell_{\max}$ we have $r_L = \widetilde{\mathcal{O}}\left(r\alpha\,(m\,L_{\max}\,\ell_{\max})^{1/2}\right)$. Finally, by a tail cutting argument, $q = \widetilde{\Theta}\,(r_L\sqrt{n}) = \widetilde{\Theta}\left(n^{5/2}L_{\max}\ell_{\max}\right)$ is enough for decryption to be correct with overwhelming probability. ☐

5.4.5 Extension to arbitrary moduli and trapdoor matrices

In this paragraph we show how to instantiate our proofs in a more generic setting.

Our GSW ciphertext rerandomization can be straightforwardly adapted to any matrix $\mathbf{H}$ and modulus q, as long as the lattice $\Lambda_q^\perp\,(\mathbf{H}^\mathsf{T})$ has a small public basis, i.e., a small public trapdoor. Observe that the conditions needed to apply GSW ciphertext rerandomization are given in Lemma 5.3.4, which bounds the smoothing parameter of the lattice

$$\mathcal{L} = \left\{\mathbf{v} \in \Lambda_q^\perp\,(\mathbf{H}^\mathsf{T}) \times \mathbb{Z} : \hat{\mathbf{e}}^\mathsf{T}\mathbf{v} = 0\right\}$$

and in Lemma 5.3.5 which gives the min-entropy of a Gaussian over $\Lambda_q^\perp\,(\mathbf{H}^\mathsf{T})$.

Let $\beta \geq \|\mathbf{t}_i\|$, where $\mathbf{T} = \{\mathbf{t}_1, \ldots, \mathbf{t}_m\}$ is the public trapdoor of $\mathbf{H}$ (i.e., $\mathbf{T}$ is a small basis of $\Lambda_q^\perp\,(\mathbf{H}^\mathsf{T})$), we show that the previous two lemmas can be proven for $\mathbf{H}$ and the parameter r only grows by a factor β.

First, observe that Lemma 5.3.4 aims to find m small independent vectors in $\mathcal{L}$. By noticing that

$$\mathcal{L} = \left\{(\mathbf{v}, -\mathbf{v}^\mathsf{T}\mathbf{e}) : \mathbf{v} \in \Lambda_q^\perp\,(\mathbf{H}^\mathsf{T})\right\}$$

we can exhibit m small vectors $\mathbf{u}_i = \left(\mathbf{t}_i, -\mathbf{t}_i^\intercal \mathbf{e}\right), i \in [m]$ which are of norm

$$\|\mathbf{u}_i\| \leq \|\mathbf{t}_i\| \left(1 + \|\mathbf{e}\|\right) \leq \beta \left(1 + \|\mathbf{e}\|\right)$$

This bound is the one we obtain in Lemma 5.3.4 for $\Lambda_q^\perp\left(\mathbf{G}^\intercal\right)$ where $\|\mathbf{T}\| = \sqrt{5}$.

Second, we show that the bound on the min-entropy of Lemma 5.3.5 can be expressed as a function of β, by simply using the fact that $\det\left(\mathbf{T}\right) \leq \|\mathbf{T}\|^m = \beta^m$. From this we have the following bound on the min-entropy:

$$H_\infty\left(\mathcal{D}_{\Lambda_q^\perp(\mathbf{H}^\intercal)+\mathbf{c},r}\right) \geq \log\left(1-\varepsilon\right) + m\log\left(r\right) - m\log\left(\beta\right)$$

This bound is slightly worse than the one we obtain in Lemma 5.3.5 for $\mathbf{G}$ (where we had 2 instead of β). However this is not a problem as it is a weaker bound than the one obtained in Lemma 5.3.4.

By using these two lemmas we can rerandomize GSW ciphertexts and ensure circuit privacy for arbitrary modulus q, and any matrix $\mathbf{H}$ with public trapdoor by setting the Gaussian parameter of $\mathbf{H}^{-1}\left(\cdot\right)$ to $r = \widetilde{\Theta}\left(\beta n\right)$.

5.5 Discussions

We conclude with some considerations and a critical analysis of the results presented in this part of the manuscript. Finally we state some open problems and outline possible directions for future research.

A draw-back of our approach is that it is specific to the GSW cryptosystem and variants there-of, whereas previous approaches based on noise flooding and bootstrapping are fairly generic. Nonetheless, we stress that the GSW cryptosystem turns out to be ubiquitous in many applications outside of FHE, including attribute-based encryption and fully homomorphic signatures [BGG+14; GVW15]. Another issue is that we need to pad the branching program so that each variable appears the same number of times, thus decreasing the efficiency of the evaluation for no real reason. However, one can argue that the impact on efficiency is fairly limited, especially in comparison with generic homomorphic operations. We are optimistic that the additional insights we gained into the noise distributions of GSW ciphertexts will find applications outside of FHE.

Open problems. We conclude with several open problems pertaining to FHE circuit privacy. The first is to achieve circuit privacy against malicious adversaries [OPP14]: namely, the result of a homomorphic evaluation should leak no information about the circuit f, even if the input ciphertexts are maliciously generated. Our analysis breaks down in this setting as it crucially uses fresh uniform randomness in the input ciphertexts for left-over hash lemma, and the fact that the noise in the input ciphertexts are small (but does not need to be discrete Gaussian). Another goal is to achieve circuit-private CCA1-secure FHE [LMSV12]; here, the technique that [DS16] uses to achieve circuit privacy cannot obtain such a result since giving out an encryption of the secret key violates CCA1-security. A third major open problem is to extend the techniques in this work to other FHE schemes, such as those in [BV11a; DM15; HS15].

Chapter 6

Private information retrieval through homomorphic encryption

In this chapter we present an implementation of homomorphic encryption, applied to the problem of private information retrieval (PIR). In particular, we will design and implement a protocol that allows a client to query a remote database held by a server, with the guarantees that the server does not learn the query. Also, the client learns nothing more than the answer to its request (or the fact that the request has no answer at all). This last remark seems to imply that our protocol actually achieves the security requirements of oblivious transfer, but we will discuss this matter in the following.

The implementation of the protocol will not use instantiations of homomorphic encryption from lattice assumptions, but the so-called homomorphic encryption *over the integers*, stemming from the assumed hardness of the approximate GCD problem (cf. Definition 2.4.6). Specifically, our encryption scheme will be a symmetric version of that presented in [DGHV10], extended to support a message space larger than just $\{0, 1\}$.

This work was done within a European project, and it aimed at showing that homomorphic encryption can be practical for some use-cases.

Contents

6.1 Introduction

The unstoppable diffusion of new technologies has undeniably changed many aspects of our lives, both in the positive and in the negative. Cryptography in particular can be seen as a double-edged tool: on one hand, it offers effective ways of protecting one's privacy from malicious eavesdroppers, whereas on the other hand, it can help shielding criminals from law enforcement agencies, by making their communications secure and untraceable.

Organized crime (OC) is becoming increasingly diverse in its methods, group structures and impact on society. A new criminal landscape is emerging, marked increasingly by highly mobile and flexible groups operating in multiple jurisdictions and criminal sectors. Internet and mobile technologies have emerged as key facilitators for organized crime, because of the possibility that they offer to communicate in a rapid, efficient, and covert way, thus allowing members of criminal organizations to exchange messages without meeting in person and without incurring the risk of being intercepted. Internet essentially underpins criminal activities like high-tech cybercrime, payment card fraud, audio and visual piracy, drug smuggling, THB (trafficking in human beings), facilitation of illegal immigration, supply of counterfeit commodities, and many other illicit activities.

Especially after the 9/11 terrorist attacks, the old-fashioned manual work of analyzing links between actors and putting them in a graph was no longer suited to the amount of data that had to be processed. A need for more sophisticated techniques to navigate large datasets emerged, and a key role in this sense is played by social network analysis (SNA). This kind of techniques aims at extracting information about social structures through the use of network and graph theories. Concretely, it means that law enforcement agencies have the possibility to collect data about private individuals, analyze the relationships that link them, and investigate their social behavior in order to establish if they pose any threat to the society. In fact, although electronic communications have made organized crime activities less visible to authorities targeting criminal assets, the increasing usage of the Internet and of mobile communications offers new opportunities to investigators to detect signals and to pre-empt organized crime activities, as well as to co-operate effectively and efficiently. For example, new technologies can be used for scanning "weak OC signals", in order to search, fuse, and interpret data collected from several diverse sources. These sources typically include databases of call detail records of many (potentially all) telephone companies in a country, databases of financial transactions (banks, credit cards, etc.), databases of vehicle registrations (linking car plates to the vehicle's owner), databases of biometric features (DNA, fingerprints, etc.), and national databases that contain many details about each citizen (like date and place of birth, job, annual income, etc.).

However, while carrying out their duty, police forces have to comply with the law, and democratic systems protect the rights of citizens, the privacy of their personal data, and that of their communications. Wide-range scanning for weak organized crime signals is typically incompatible with the legal constraints, because it would unduly give power to the executive arm and thereby limit personal freedom. This kind of aggressive behavior from national agencies has already been reported, e.g., by the famous revelations made by former CIA employee and NSA contractor Edward Snowden [Gre13], who exposed pervasive mass surveillance programs. These revelations led to thorough discussions on investigation policies and, to some extent, to their revision. Generally speaking, currently standing laws prescribe that interference with the right to privacy and to the protection of personal data should be *necessary* and *proportionate* to the legitimate goal they are aimed at, and not excessive, arbitrary, or discriminatory. Also, it is worth noting that failing to comply with laws and regulations would expose police forces to legal sanctions and public protests, but could also have the disastrous side effect of totally invalidating legal cases against OC members, who are known to benefit from top-quality legal assistance.

Extensive surveillance: an example from the news. As an example of how extensive investigations can lead to issues with freedom and privacy, let us consider the following case, which happened in the United States [And10]. Between 2009 and 2010, 16 robberies were carried out in rural locations of Arizona by two men, known as "the High Country bandits". Although bank surveillance footage was available to investigators, this proved to be of little use, since both men were using jackets, ski masks, and gloves. After bringing the FBI into the case, the investigators found a witness who said that he had noticed a suspicious man hanging out by the bank a couple of hours before the robbery, while talking on a cell phone. The FBI then asked a judge to approve a full *cell tower dump*, which is a procedure in which wireless operators turn over the records of every cell phone that registered with a particular tower at a particular time. This procedure was approved and executed for the four most rural locations where robberies had taken place, in order to minimize the number of extraneous telephone numbers that would show up in the tower dumps. It should be noticed at this point that tower dumps are obtained *without a warrant*: they use a "court order", which still implies some judicial supervision, but to a much lesser extent than an ordinary warrant (there is no need for a "probable cause"). This implies that the government could potentially access location information for a vast number of people, without having any warrant. In particular, during the investigations on the robberies, the FBI received more than 150 000 telephone numbers from these tower dumps. Intersecting the records that had been registered in the locations of the robberies gave one specific phone number that, through more interaction with a judge, finally gave a name. Subsequent investigations led to the capture of both perpetrators, thus solving the crime. Although the conclusion of the investigation was positive, this story leads to some privacy-related questions. In fact, several judges (e.g., [Ows13]) state that these tower dumps are to be considered "searches" under the Fourth Amendment, and thus require a full warrant backed by evidence of "probable cause". Also, the Supreme Court has ruled [Uni12] that warrantless GPS tracking of a suspect is not allowed. In the case of tower dumps, it is true that they do not provide the precision of GPS tracking, but it is equally true that they compromise the privacy of many people, and not just one. Furthermore, those whose records are collected in the process are never notified of the fact.

From this story, it is clear that there is a potential conflict between the need to conduct thorough investigations and data privacy rules, also because the rules and their interpretation are not always clear and agreed upon. Such data privacy rules are even more stringent when it comes to the collaboration between investigators from different agencies and countries, because of organizational issues, bureaucratic hurdles, differences in legislations, different political views of the respective governments, ... There is then a conflict between safety and privacy, and the fundamental paradox that arises from this picture is that protecting some rights (privacy) makes it more difficult to protect other rights (security).

Use-case: cross-border cooperation between law-enforcement agencies. A typical case of how privacy related regulations can get in the way of police investigations is represented by cross-border collaboration between law enforcement agencies, i.e., a situation where two agencies, based in two different countries, want to exchange information, or to make the information they possess available to partner agencies. Note that here we are not concerned with how a certain set of records was obtained; in this use case we focus on how these records can be *shared* while respecting the citizens' privacy to the greatest possible extent. This use case is already regulated by a EU framework that was introduced in 2008 by EU Council Decision 2008/615/JHA [08a] and EU Council Decision 2008/616/JHA [08b], which applied the previously established Prüm Convention [Uni05] to all the member states. These decisions describe a framework in which member states can grant one another access rights to their automated DNA analysis files, automated dactyloscopic identification systems or vehicle registration data via a two-step process: the first step is a hit/no-hit system, that returns whether the query has a match in the target database (and whose result should be available in less than 15 minutes); in case of a positive answer, the second step is a request for specific personal data related to the result of the query.

The framework currently in place still presents issues regarding the protection of citizens' privacy; for example, the country that hosts the database receives queries in the clear. This amounts to knowing what the other country is looking for, which, in some cases, might be undesirable. Also, the cryptographic standards which are used (AES 256, RSA 1024, SHA1) are not sufficiently adequate and do not take into account the most recent developments in research, as far as constructions and cryptanalysis are concerned. Moreover, at the time the decisions were produced, several cryptographic primitives like Fully Homomorphic Encryption (FHE) were not known to exist, so the results that can be achieved now are largely more sophisticated than what was imaginable in 2008. Thanks to the new possibilities opened by progresses in cryptographic research, the privacy model can be considerably strengthened, in order to achieve a framework that allows for the exchange of information without compromising the citizens' right to privacy.

Problem statement and overview of the results. We now proceed to outlining the problem we consider and the goals we want to achieve. As in [CLT11], we assume a party, say France (FR), wants to query a database held by another party, say Germany (DE), on a certain input XYZ; furthermore, we assume that both countries recognize the authority of a trusted party, say a Judge (JU). The basic property that we want to achieve is that DE should not learn FR's query, i.e., XYZ. The solution that we propose actually achieves more than this, namely

- FR learns no more than whether there is a hit/no-hit on the data XYZ and, if authorized by JU, which records match XYZ;
- DE does not learn anything (not even XYZ), but the possible data it would be legally obliged to provide after a successful match;
- JU learns the query XYZ but not the associated data (i.e., to whom the profile corresponds, etc.), even if he allows the query.

As already stated, the hit/no-hit part should complete in less than 15 minutes. The rest of the protocol is not strictly time-bounded by any regulation, and this kind of application is usually not time-critical. Nevertheless, we consider that the protocol should complete in a reasonable amount of time, in order to provide necessary information in a timely and usable manner.

We now introduce two fundamental cryptographical primitives, called *Private information retrieval* and *Oblivious transfer*, which are tightly linked to the problem at hand.

6.1.1 Private information retrieval

In cryptography, a private information retrieval (PIR) protocol is a protocol that allows a user to retrieve an item from a database hosted on a server, without revealing to the server which item is retrieved.

It is trivial to notice that there is an immediate protocol that provides perfect security for the user: the server simply sends the entire database to the user, who is then able to read the information it needs. Obviously, this reveals nothing about the user's query, since nothing is submitted to the server. However, it is clear that this kind of protocol is likely to be totally impractical for any realistic use-case because of the communication costs. The goal then becomes to design a protocol with a lower communication complexity; in this sense, [DMO00] defines as *non-trivial* any protocol whose communication complexity is less than the size of the entire database. The problem of PIR was introduced by Chor *et al.* in 1995 [CGKS95] in the information-theoretic setting, and by Kushilevitz and Ostrovsky in 1997 [KO97] in the computational setting. Several protocol were presented in subsequent works, e.g., [Ste98; Cha04; Lip04].

Note, however, that PIR protocols do not ensure server security, meaning that the user can retrieve more than what it queries for. This enhanced security guarantee is captured by the definition of oblivious transfer, which is presented next.

6.1.2 Oblivious transfer

In private information retrieval protocols, the security requirement is that the server should not learn the user's query. However, nothing is said about what the user is allowed to learn, which could be more than what he queries for. In fact, in the most trivial protocol where the entire database is sent from the server to the user, it is clear that the user can learn all the records. Instead, oblivious transfer (OT) introduces a new security requirement. Let us consider the case where a server is holding a database with records $x_1, x_2, \ldots, x_n$ and a user wants to learn x_i for some i. At the end of the OT protocol, the server should not learn i, whereas the user should not learn x_j for any $j \neq i$. This is called 1-out-of-n oblivious transfer, the simplest case being $n = 2$, i.e., the server has a database with two records and the user obliviously obtains one of the two. An important result by Crepeau [Cré88] showed that 1-out-of-2 OT and 1-out-of-n OT are equivalent. In the literature, several OT protocols have been presented, culminating in a very simple and efficient proposal by Chou and Orlandi [CO15], based on the hardness of the well-established DDH assumption [Bon98].

6.1.3 Our contributions

In this part of the manuscript, we present a protocol that allows a user to retrieve a record from a database held by a server, without revealing which record has been queried. Furthermore, the user learns only the record (or records, in case of multiple matches), that corresponds to its query. This means that our protocol respects the security requirements of oblivious transfer. However, the specific application for which the protocol is designed takes into account that the user can submit several queries (provided they are authorized by the trusted party, referred to as "the Judge"). This means that, through several executions, the querying party can learn up to the entire database. For this reason, we can say that this is an oblivious transfer protocol if a single execution is considered, whereas the definition is not clear in case of multiple executions. In any case, the protocol always achieves the PIR

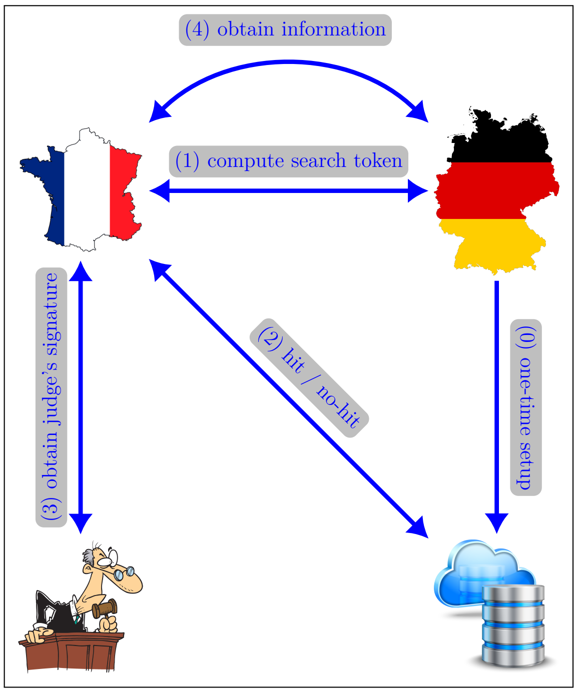


Figure 6.1: Outline of the steps of the ADOC protocol.

requirement, meaning that the server never learns the user's query. This protocol is inspired by [BGH+13].

The basic principle of this use case consists in using homomorphic encryption (HE) to encrypt databases, while enabling data aggregation in the Cloud (for authorized users). Figure 6.1 gives a high-level intuition on the steps of the protocol.

6.2 Our protocol

We now describe our protocol for private information retrieval. Initially, all the operations will be described at a high level; then we will provide implementation details regarding each one of them.

In our protocol, we consider three actors: the querying party (France, or FR), the responding party (Germany, or DE), and the trusted party (Judge, or JU). We assume that the database held by DE is structured as follows: each record is composed of a unique ID (w.l.o.g., we can assume it is numerical), and by a number of fields corresponding to some features (e.g., the person's name, surname, car plate, phone number, address, ...). The protocol then goes through a one-time setup phase as follows:

1. As a first step of the protocol, DE generates the inverted database, which is a dataset where each occurrence of each feature is associated to all the IDs that match. For example, consider the toy-example of database in Table 6.1; then the inverted database will look like that presented in Table 6.2.

2. DE encrypts the inverted database under some secret key sk which is never revealed

ID	**Name**	**Surname**
1	John	Doe
2	Marc	Smith
3	John	Smith

Table 6.1: Toy-example of database.

Feature	**ID**
Name=John	1, 3
Name=Marc	2
Surname=Doe	1
Surname=Smith	2, 3

Table 6.2: Inverted database corresponding to the toy example in Table 6.1.

to any other party. In particular, the part "Feature=..." is converted into a *search token* via a pseudo-random function (PRF), whereas the IDs are considered as roots of a polynomial $p(x)$, which is subsequently encrypted coefficient by coefficient. For example, an entry in the encrypted inverted database might look like

$$\left(\quad \mathsf{PRF}_{k_{\mathsf{DE}}}\left(\text{“DNA}_{\mathsf{D19S433}} = 7.8\text{”}\right), \quad \mathsf{Enc}_{k_{\mathsf{DE}}}\left(\mathcal{I}\right) \quad\right)$$

where $\mathcal{I}$ is the set of indices in the database of the individuals for which the DNA marker D19S433 is equal to 7.8, encrypted with a key k_{DE} only known to DE.

The so-computed encrypted inverted database can then be safely published onto the Cloud and made accessible to anyone. The underlying assumption is that it reveals no information to anyone who does not hold the secret key or does not interact "properly" with DE to obtain the disclosure of relevant information.

These steps have to be performed at the beginning of the protocol (setup time), and every time the database needs to be updated. Note that, in this latter case, it is possible to update only parts of the encrypted inverted database, thus saving bandwidth.

After the setup has been completed, FR (i.e., the querying party) can interact with the other actors of the protocol in order to obtain an answer to a certain query. The steps are different for different types of queries. In details, we consider three possible requests, called *simple query* (FR wants all the records that match a certain condition ξ), *disjunction query* (all the records that match at least one of two conditions, i.e., $\xi_1 \vee \xi_2$), and *conjunction query* (all the records that match simultaneously two conditions, i.e., $\xi_1 \wedge \xi_2$). We now describe the steps of the protocol for each of these cases.

Simple query. In the case of a simple query, the protocol proceeds as follows:

3. FR and DE engage in a simple sub-protocol that allows them to *obliviously* compute the search token Θ corresponding to FR's query ξ. This means that, on private inputs ξ for FR and sk for DE, FR will be able to compute a search token which is a function of ξ and sk, without learning anything on sk, and without revealing anything on ξ.

Referring to the example presented before, FR and DE can obliviously compute

$$\mathsf{PRF}_{k_{\mathsf{DE}}}\left(\text{“DNA}_{\mathsf{D19S433}} = 7.8\text{”}\right)$$

4. After obtaining the search token Θ, FR can check whether there is a match for it in the inverted database, which is publicly available. Notice that this will give FR no information on the records that match (if any), because the inverted database is encrypted. If there is no match, the protocol terminates.

5. After verifying that there is a match, FR sends relevant information to the the trusted party JU. This will include the query itself and the search token, together with some supporting evidence that demonstrates why the query is legal and should be allowed.

6. The trusted JU verifies that the supporting material is indeed acceptable from a legal point of view and, if satisfied, fetches from the inverted database the encrypted polynomial corresponding to the search token, signs it with his secret key, and hands it back to FR. Again, with reference to the example shown before, JU will send

$$\Big(\quad \mathsf{Enc}_{k_{\mathsf{DE}}}(\mathcal{I}), \quad \mathsf{Sig}_{k_{\mathsf{JU}}}\left(\mathsf{Enc}_{k_{\mathsf{DE}}}(\mathcal{I})\right) \quad \Big),$$

   where k_{JU} is JU's secret key.

7. FR forwards this signed encrypted polynomial to DE.

8. DE verifies JU's signature and, if everything is correct, decrypts the polynomial, calculates its roots (i.e., the IDs), and sends to FR the information on those records.

These steps are summarized and depicted in Figure 6.2.

An attentive reader might have noticed that there is a simple cheating strategy for FR: when interacting with JU, it could present a legally valid query, together with a random search token. This way, the trusted JU might be tricked into authorizing something different from what it thinks. In reality, it will be clear in the section about implementing the protocol that JU has an easy way of verifying that the search token indeed corresponds to the query, thus preventing this kind of attack.

Disjunction query. In the case of a disjunction query with conditions ξ_1 and ξ_2, the protocol proceeds as follows:

3. FR and DE repeat twice the simple sub-protocol for oblivious computation of search tokens. At the end of this phase, FR will have search tokens Θ_1 and Θ_2, corresponding to conditions ξ_1 and ξ_2.

4. After obtaining the tokens, FR checks in the inverted database. If neither of the tokens gives a match, then the protocol terminates, because the union of the two results is guaranteed to be empty.

5. After verifying that there is a match, FR sends relevant information to JU. This will include both queries and both search tokens, together with material to support the case.

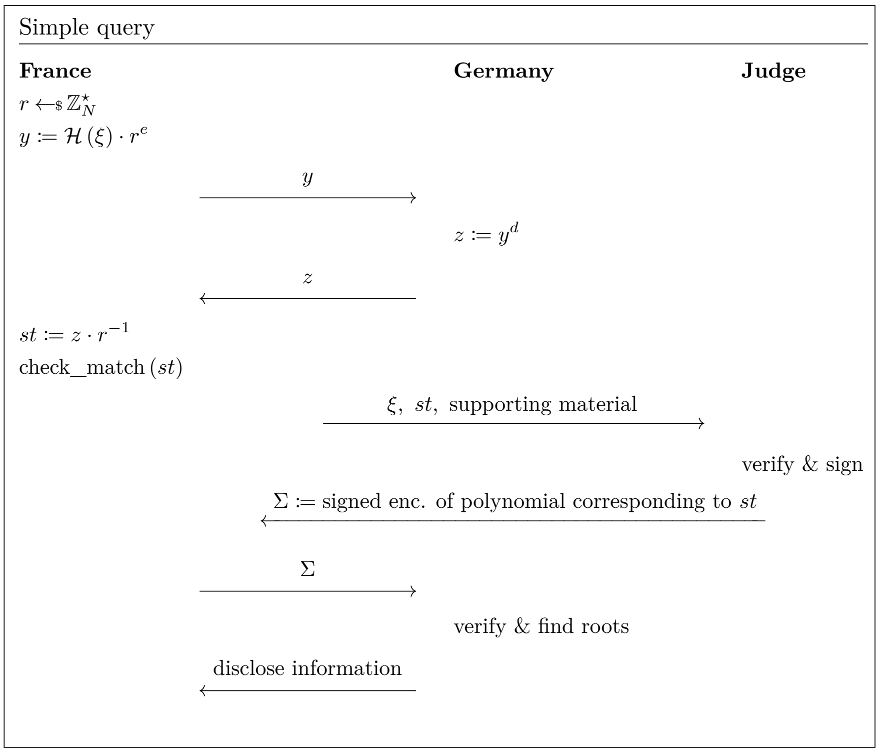

Simple query
France
Germany
Judge
$r \leftarrow_{\$} \mathbb{Z}_N^\star$
$y := \mathcal{H}(\xi) \cdot r^e$
y
$z := y^d$
z
$st := z \cdot r^{-1}$
check_match (st)
$\xi,\ st,$ supporting material
verify & sign
$\Sigma :=$ signed enc. of polynomial corresponding to st
Σ
verify & find roots
disclose information

Figure 6.2: Protocol for the case of a simple query.

6. JU verifies the supporting material and proceeds if it is satisfied. If this is the case, JU fetches the two encrypted polynomials $p_1(x)$ and $p_2(x)$ corresponding to search tokens Θ_1 and Θ_2, homomorphically multiplies them together, and signs the resulting encryption of the polynomial $p_1(x) \cdot p_2(x)$. Notice that the roots of $p_1(x) \cdot p_2(x)$ will be the union of the roots of $p_1(x)$ and $p_2(x)$. Finally, JU hands the signed polynomial back to FR.

7. FR forwards this signed encrypted polynomial to DE.

8. DE verifies JU's signature and, if everything is correct, decrypts the polynomial, calculates its roots (i.e., the IDs), and sends to FR the information on those records.

These steps are presented in Figure 6.3.

In order for these steps to go through, the encryption scheme used for implementing the protocol must support at least one homomorphic multiplication.

Conjunction query. This is the most problematic case, because it turns out that FR is not able to verify on its own whether there is a match for conditions $\xi_1 \wedge \xi_2$. In fact, verifying that there is a match for each of the two conditions is necessary but not sufficient to guarantee that the intersection of the results is non-empty. In order to solve this issue, we have to complicate the protocol and introduce one more round of interaction between FR and DE. The steps are as follows:

3. FR and DE repeat twice the simple sub-protocol for oblivious computation of search

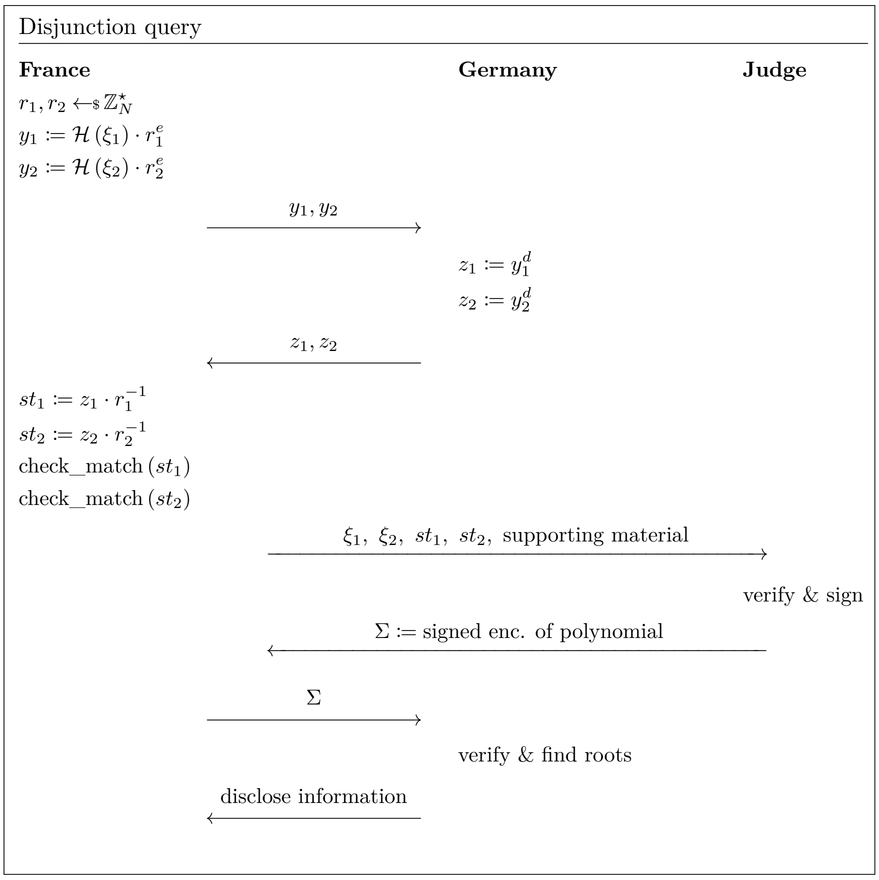

Disjunction query
France
Germany
Judge
$r_1, r_2 \leftarrow\!\!\$\, \mathbb{Z}_N^\star$
$y_1 \coloneqq \mathcal{H}(\xi_1) \cdot r_1^e$
$y_2 \coloneqq \mathcal{H}(\xi_2) \cdot r_2^e$
y_1, y_2
$z_1 \coloneqq y_1^d$
$z_2 \coloneqq y_2^d$
z_1, z_2
$st_1 \coloneqq z_1 \cdot r_1^{-1}$
$st_2 \coloneqq z_2 \cdot r_2^{-1}$
check_match (st_1)
check_match (st_2)
$\xi_1,\ \xi_2,\ st_1,\ st_2,$ supporting material
verify & sign
$\Sigma \coloneqq$ signed enc. of polynomial
Σ
verify & find roots
disclose information

Figure 6.3: Protocol for the case of a disjunction query.

tokens. At the end of this phase, FR will have search tokens Θ_1 and Θ_2, corresponding to conditions ξ_1 and ξ_2.

4. After obtaining the tokens, FR checks in the inverted database. If at least one the tokens gives no match, then the protocol terminates, because the intersection of the two results is guaranteed to be empty. If this is not the case, then it is possible that the intersection is non-empty.

5. FR and DE engage in a sub-protocol to verify whether the intersection is non-empty. First, FR picks two random (not encrypted) polynomials $r_1(x)$ and $r_2(x)$, under some conditions that will be explained in the part regarding the implementation. Then, it homomorphically computes the polynomial $q(x) \coloneqq p_1(x) \cdot r_1(x) + p_2(x) \cdot r_2(x)$. Notice that (a) the polynomial $q(x)$ is computed from both encrypted and not encrypted polynomials, but the result is an encrypted polynomial, and (b) the set of roots of $q(x)$ is *at least* composed of the intersection of those of $p_1(x)$ and $p_2(x)$, but could potentially be larger. In fact, for example it might happen that $r_1(x)$ and $r_2(x)$ have a root in common, which is not a root of either $p_1(x)$ or $p_2(x)$: if this happens, then this operation has introduced a spurious root, that needs to be handled with great care. The reason is that introducing such a root amounts to adding an unrelated ID to the

list of results, thus creating a false positive. Also, it can happen that a spurious root is introduced even if there is no common root among the polynomials. We address the issue and propose a solution in Section 6.4.2. Finally, FR sends $q(x)$ to DE.

6. DE decrypts $q(x)$ with its secret key, computes its roots and sends 1 to FR if there is at least one root (cf. Sections 6.4.1 and 6.4.2 for more details); otherwise DE sends 0.

7. If FR receives 0, then the protocol terminates, as the intersection is empty. If this is not the case, FR sends relevant information to JU. This will include both queries and both search tokens, together with material to support the case.

8. JU verifies the supporting material and proceeds if it is satisfied. If this is the case, JU fetches the two encrypted polynomials $p_1(x)$ and $p_2(x)$ corresponding to search tokens Θ_1 and Θ_2, then picks fresh random polynomials $r'_1(x)$ and $r'_2(x)$, computes $q'(x) \coloneqq p_1(x) \cdot r'_1(x) + p_2(x) \cdot r'_2(x)$, signs it, and hands it back to FR.

9. FR forwards this signed encrypted polynomial to DE.

10. DE verifies JU's signature and, if everything is correct, decrypts the polynomial, calculates its roots (i.e., the IDs), and sends to FR the information on those records.

These steps are presented in Figure 6.4.

Since the polynomials $r_1(x)$ and $r_2(x)$ need not be encrypted, a linearly homomorphic encryption scheme is sufficient for these steps to go through.

In conclusion, we need an encryption scheme that can encrypt integers modulo some prime q and can support at least one homomorphic multiplication between ciphertexts. Also, there is no need for a public-key encryption scheme, as the polynomials will be encrypted under DE's secret key. In the following, we present the DGHV encryption scheme [DGHV10], that we extend and use to instantiate our protocol.

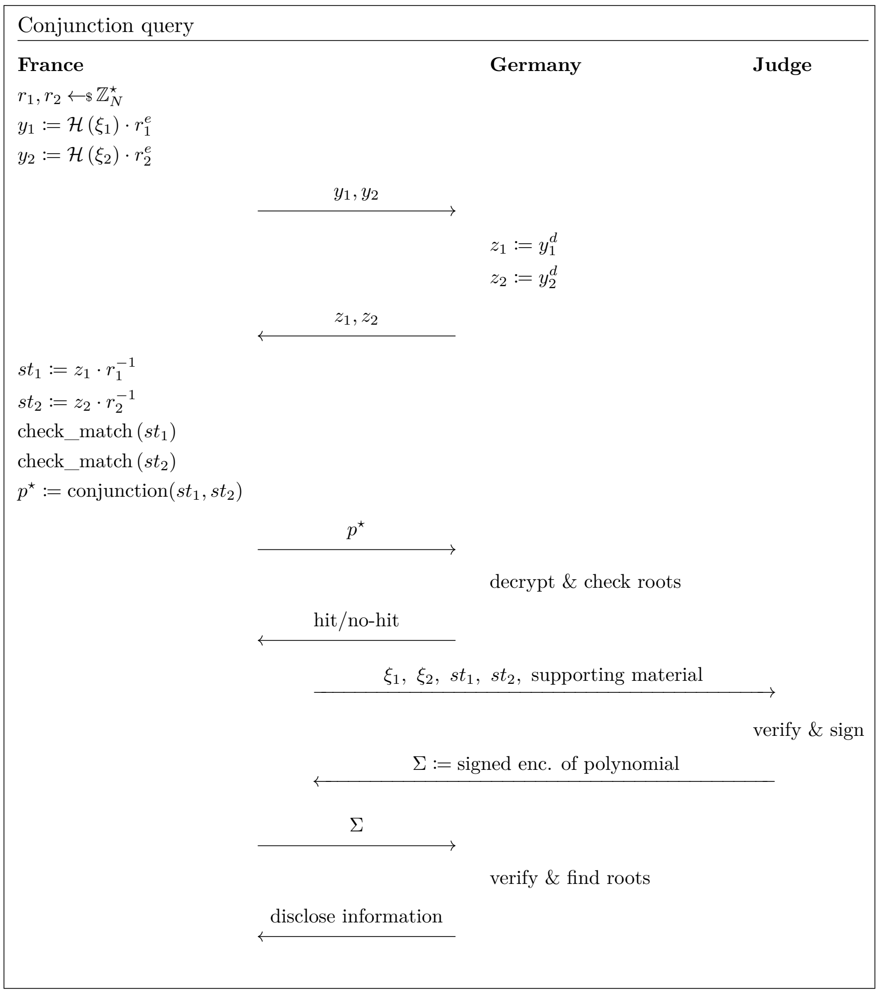

Conjunction query
France
Germany
Judge
$r_1, r_2 \leftarrow_{\$} \mathbb{Z}_N^\star$
$y_1 := \mathcal{H}(\xi_1) \cdot r_1^e$
$y_2 := \mathcal{H}(\xi_2) \cdot r_2^e$
y_1, y_2
$z_1 := y_1^d$
$z_2 := y_2^d$
z_1, z_2
$st_1 := z_1 \cdot r_1^{-1}$
$st_2 := z_2 \cdot r_2^{-1}$
check_match (st_1)
check_match (st_2)
$p^\star :=$ conjunction(st_1, st_2)
$p^\star$
decrypt & check roots
hit/no-hit
$\xi_1,\ \xi_2,\ st_1,\ st_2,$ supporting material
verify & sign
$\Sigma :=$ signed enc. of polynomial
Σ
verify & find roots
disclose information

Figure 6.4: Protocol for the case of a conjunction query.

6.3 The DGHV encryption scheme and its extension

Here we present a basic secret-key version of the DGHV encryption scheme with message space $\mathcal{M} = \{0, 1\}$ (as given in [DGHV10]), and we show how it can be simply extended to support a larger message space. We remind the reader that we are not interested in building a *fully* homomorphic encryption scheme, so we will not focus on the bootstrapping procedure.

For a security parameter κ, the encryption scheme is parametrized by integers $\eta = \eta(\kappa), \gamma = \gamma(\kappa), \rho = \rho(\kappa)$. The scheme is then composed of the following algorithms:

$\mathsf{KGen}(1^\kappa) \to (\mathsf{sk}, x_0)$ **:** generate a random prime p of size η bits. Generate a random prime s_0 of size $\gamma - \eta$ bits, and let $x_0 = s_0 p$ be a public parameter. Return $\mathsf{sk} = p$ and x_0.

$\mathsf{Enc}(\mathsf{sk}, \mu \in \{0, 1\}) \to c$ **:** generate a random positive integer s of $\gamma - \eta$ bits, a random integer r in $(-2^\rho, 2^\rho)$, and output the ciphertext

$$c = ps + 2r + \mu.$$

$\mathsf{Dec}(\mathsf{sk}, c) \to \mu$ **:** output $\mu = (c \mod p) \mod 2$.

For the choice of parameters, [DGHV10] suggests the following:

- $\rho = \omega(\log\kappa)$ to protect against brute-force attacks on the noise;
- $\eta \geq \rho \cdot \Theta\left(\kappa \cdot \log^2 \kappa\right)$ in order to support homomorphisms for deep enough circuits to evaluate the squashed decryption circuit;
- $\gamma = \omega(\eta^2 \cdot \log\kappa)$ to thwart various lattice-based attacks on the underlying approximate GCD problem.

Extending the message space. It turns out that extending the message space to $\mathbb{Z}_q$ for some integer q, is immediate[1]. It is sufficient to change the encryption and decryption algorithms as follows:

$\mathsf{Enc}(\mathsf{sk}, \mu \in \mathbb{Z}_q) \to c$ **:** generate a random positive integer s of $\gamma - \eta$ bits, a random integer r in $(-2^\rho, 2^\rho)$, and output the ciphertext

$$c = ps + qr + \mu.$$

$\mathsf{Dec}(\mathsf{sk}, c) \to \mu$ **:** output $\mu = (c \mod p) \mod q$.

In order to ensure correct decryption, we need $qr < p$. This essentially guarantees that the term qr remains untouched after the reduction modulo p. If this condition does not hold, we can write $qr = kp + t$, with $k \geq 1$ and $t < p$. The ciphertext can then be written as $c = (s + k)p + t + \mu$. When reducing modulo p, we are left with $t + \mu$, which is not guaranteed to be congruent to μ modulo q.

We now analyze how to perform homomorphic operations on DGHV ciphertexts. In this part, the role of the term x_0 output by the KGen procedure will finally become clear. Let $(\mathsf{sk}, x_0) \leftarrow \mathsf{KGen}(1^\kappa)$.

Homomorphic addition. Given two messages $\mu_1, \mu_2 \in \mathbb{Z}_q$, and two DGHV ciphertexts c_1, c_2 such that $c_i \leftarrow \mathsf{Enc}(\mathsf{sk}, \mu_i)$, the ciphertext $c^+ := (c_1 + c_2) \mod x_0$ is a DGHV ciphertext of $\mu_1 + \mu_2$.

Showing that this is the case just requires performing the calculations:

$$\begin{aligned} c^+ \mod x_0 &= ps_1 + qr_1 + \mu_1 + ps_2 + qr_2 + \mu_2 \mod s_0 p \\ &= p(s_1 + s_2) + (r_1 + r_2)q + (\mu_1 + \mu_2) \mod s_0 p \\ &= p[(s_1 + s_2) \mod s_0] + (r_1 + r_2)q + \mu_1 + \mu_2. \end{aligned}$$

It then follows that

$$\left(c^+ \mod p\right) \mod q = \mu_1 + \mu_2,$$

provided that $(r_1 + r_2)q < p$.

Homomorphic multiplication. Unlike most lattice-based homomorphic schemes, in DGHV the multiplication follows the same procedure as the addition. Given two messages $\mu_1, \mu_2 \in$

[1]It should be noted, however, that a way to perform the bootstrapping operation and, thus, turn the DGHV into an FHE scheme is known only for $q = 2$.

Chapter 6

$\mathbb{Z}_q$, and two DGHV ciphertexts c_1, c_2 such that $c_i \leftarrow \mathsf{Enc}(\mathsf{sk}, \mu_i)$, the ciphertext $c^\times := (c_1 \cdot c_2) \mod x_0$ is a DGHV ciphertext of $\mu_1 \cdot \mu_2$.

Once again, showing that this holds is rather easy:

$$\begin{aligned} c^\times \mod x_0 &= p\left(s_1 s_2 p + s_1\mu_2 + s_2\mu_1 + s_1 r_2 q + s_2 r_1 q\right) \\ &\quad + q\left(r_1 r_2 q + r_1\mu_2 + r_2\mu_1\right) + \mu_1\mu_2 \mod s_0 p \\ &= p\left[\left(s_1 s_2 p + s_1\mu_2 + s_2\mu_1 + s_1 r_2 q + s_2 r_1 q\right) \mod s_0\right] + q\left(\dots\right) + \mu_1\mu_2. \end{aligned}$$

It then follows that

$$\left(c^\times \mod p\right) \mod q = \mu_1\mu_2,$$

provided that $r_1 r_2 q^2 < p$.

Remark 6.3.1 (On reduction modulo x_0 (cf.[DGHV10, Section 3.3.2]))**.** *Note that reducing modulo x_0 is not necessary for correctness or security to hold. This operation is merely useful for efficiency, i.e., for reducing the size of the ciphertext. In the original paper, the authors remark that publishing an exact multiple of the secret p makes the approximate GCD problem "clearly easier". However, they also note that no attack is known to work for this "easier case" and not for the general one.*

6.4 Implementing our protocol

The components that had to be developed in order to fully implement the protocol are:

- An instantiation of the DGHV encryption scheme for encrypting and decrypting integers, and for performing the required homomorphic operations on ciphertexts;
- A hash function, that will be used for the part related to search token computation and signatures;
- A sub-protocol for oblivious computation of the search tokens that, starting from FR's and DE's private inputs (i.e., the query ξ and the secret exponent d, respectively), allows FR to compute the search token without revealing anything on ξ, and without learning anything on d;
- A signature scheme for JU to sign encrypted polynomials and other parties to verify the signatures;
- A way to allow all the actors to communicate with each other.

As a coding language, we chose C++, in order to benefit from both the speed that is usually associated to C implementations, and from higher level constructs that are usually linked to object-oriented languages.

Since the DGHV encryption scheme requires handling large integers, we implemented it from scratch by taking advantage of GMP (the GNU multiple precision arithmetic library) [Gt12]. This library also provides several remarkably useful features, such as the generation of (probably) prime numbers through the Miller-Rabin primality test, the computation of the modular inverse of an integer, etc.

For the hash function component, we exploited the library Crypto++ [DP18] which, among others, contains an implementation of the hash functions SHA-2 and SHA-3. Particular attention should be given to how we hash strings and polynomials, keeping in mind that the result must be numerical, in order for the protocol to go through. For the string case, we proceed as follows: through the functions provided by Crypto++, we calculate the hash of a string as a raw sequence of bytes. Then, via a simple C++ function, we convert this byte sequence to hexadecimal form, and we use the resulting string to initialize an integer in base 16. The result is then `mpz_class(hash::sha3_256(myString), 16)`, where `mpz_class` is the type that GMP associates with integers of arbitrary size. Providing a sensible and secure hash of an encrypted polynomial can result in a tricky task. For example, one could provide a separate hash for each encrypted coefficient. Although very simple, this approach leads to an immediate attack: if the encrypted coefficients are hashed and signed one by one, a simple shuffle will result in a validly signed encrypted polynomial, which is different from the initial one. This amounts to forging a signed encrypted polynomial. An alternative approach would be that of simply concatenating all the encrypted coefficients (which, as DGHV ciphertexts, are simple integers) into a single string and hash it. The drawback of this approach is that, in case of very long polynomials, this string can exceed the maximum string length permitted by the C++ standard. Also, the string would have to be stored in the RAM memory, thus consuming a considerable amount of resources. A viable approach seems the following: hash the first coefficient, concatenate the second coefficient, hash the result, concatenate the third coefficient, hash the result, and so on. Although there is apparently no clear reason not to follow this way, we decided to take full advantage of the interface provided by Crypto++ to do the following: given an encrypted polynomial, described as a sequence of encrypted coefficients, we construct a "SHA object", which is updated with all the coefficients one by one. In the end, when all the encrypted coefficients have been processed, we produce the final digest as a raw sequence of bytes. Then, analogously to what described before, we convert it to an hexadecimal string, and we use the result to initialize an integer in base 16.

For search token calculations, as in [CLT11], we use RSA decryption algorithm as a PRF. Given a string ξ and a hash function $\mathcal{H}$, the search token associated to ξ will be $(\mathcal{H}(\xi))^d \mod N$, where d is the secret RSA exponent and N is the RSA modulus. In Figure 6.5 we show a simple protocol between Alice (that holds ξ) and Bob (that holds d), that allows Alice to recover $(\mathcal{H}(\xi))^d \mod N$ without learning anything about d and without revealing anything about ξ. We naturally assume that the public RSA exponent e and the RSA modulus N are known to Alice (and to everybody else).

As a signing mechanism, we employ RSA signatures: we then assume JU is equipped with a private key (i.e., a secret exponent), whereas its public key (i.e., a public exponent) is known to all the parties. Then, signing any message m simply consists in taking the hash of m and raising it to the secret exponent. Verification is as immediate as raising this signature to JU's public exponent and checking that the result corresponds to the hash of the message m.

In order to provide the actors with a way to communicate with each other, we implemented a series of simple web servers, that listen for requests and serve them accordingly. For the demo that we coded and successfully demonstrated during several review meetings, everything was running locally on a simple laptop computer. However, the modular structure of the application makes it easy to deploy its different components to remote hosting services. The structure of the system is the following:

- A web server for the party that own the database (DE). This server will expose a service that receives requests and performs several actions, like raising the input to the secret exponent d, or verifying signatures and disclosing information.

- A browser application for the party that wants to execute a query (FR). This application will take some input (e.g., through a keyboard), and perform all the operations that the querying party has to go through during the protocol (blinding the query, unblinding the answer for calculating the search token, calculating the conjunction of two polynomials, submitting requests to the trusted party, ...). This component will also take care of rendering the information that is received from the party that owns the database, i.e., it will provide a visual representation of the records that match the query.

- A web server for the trusted party (JU). This server will expose a service that receives queries from the querying party and, after verifying that the case has valid legal grounds (this part is obviously omitted), proceeds with signing encrypted polynomials and sends back the result to the querying party. This signed material will allow for the disclosure of relevant information.

- A web server for the encrypted inverted database. This server is actually performing no "active operation". Everything it does is exposing a list of encrypted records in the form of
$$\text{searchToken},\ \text{encCoeff}_0,\ \text{encCoeff}_1,\ \ldots$$
In the demo, this component was not properly implemented: the encrypted inverted database simply resides on the local machine.

For setting up and running web servers, we used Flask [10], a Python framework that allows one to simply set up a web server that listens on a predefined port and, upon receiving requests, runs a Python script in order to serve them. The basic architectural concept is that the C++ executable provides all the functions that are necessary to all the parties to engage in the protocol, and the Python scripts interface with this executable (e.g., by passing command line arguments) to perform the actions that are required. Messages between the parties are encoded in JSON format, which makes it easy to serialize and deserialize any kind of object, in several coding languages. For this part, we used "JSON for modern C++" [Loh13] which, through a single header file, provides all the functions that are needed to serialize and deserialize objects in a fast and simple way. In order to mimic the communication process, i.e., sending and receiving files containing messages, JSON files were placed in specific folders that acted like communication channels between the parties. Naturally, this means that a real deployment of the protocol should take into account elements like network latency and failures. However, given the current state of the art for internet connections (e.g., extremely fast connections through optical fiber), we do not expect this to be a major hurdle. Finally, in Figures 6.6 to 6.8 we present screenshots of the web interface we developed, that show the perspective of the querying party, the trusted party, and the responding party, during the execution of a disjunctive query.

6.4.1 How to choose the random polynomials for conjunction queries

When performing a conjunctive search query $\xi_1 \wedge \xi_2$, the protocol requires choosing two random polynomials $r_1(x), r_2(x)$, and then computing $p'(x) := p_1(x) \cdot r_1(x) + p_2(x) \cdot r_2(x)$, where $p_1(x), p_2(x)$ are the polynomials whose roots are the indices that match queries ξ_1, ξ_2.

Search token computation

Alice **Bob**

$r \leftarrow_{\$} \mathbb{Z}_N^*$

$y := \mathcal{H}(\xi) \cdot r^e \mod N$

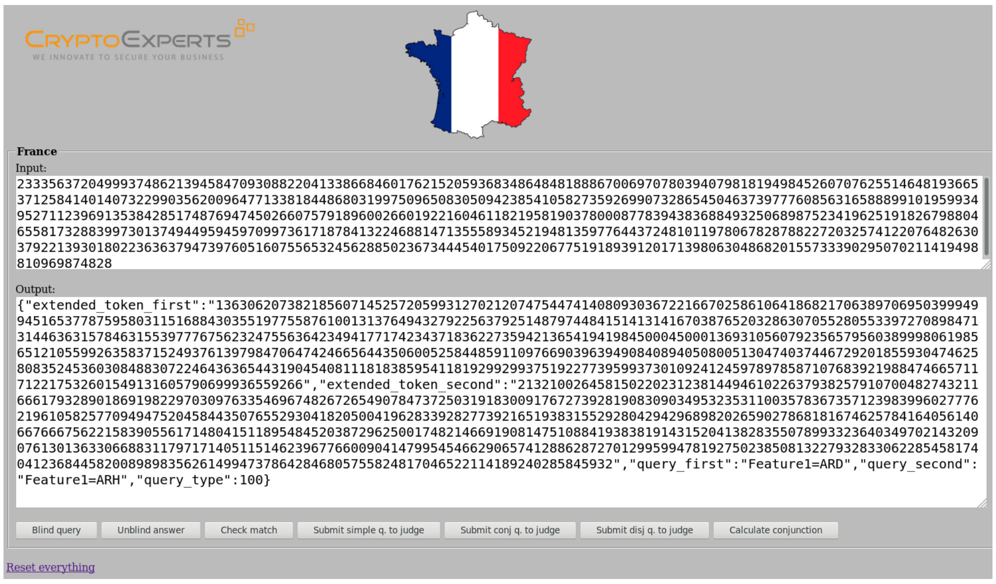
$\xrightarrow{\quad y \quad}$

$z := y^d \mod N$

$\xleftarrow{\quad z \quad}$

$st = z \cdot r^{-1} \mod N$

Figure 6.5: Protocol for oblivious computation of a search token.



France

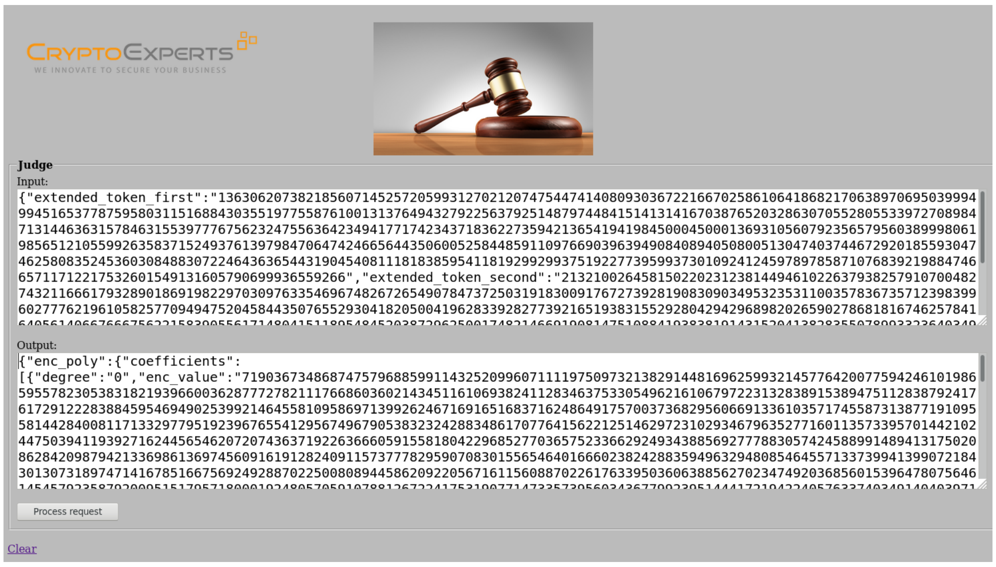
Input:

```
2333563720499937486213945847093088220413386684601762152059368348648481888670069707803940798181949845260707625514648193665
3712584140140732299035620096477133818448680319975096508305094238541058273592699073286545046373977760856316588899101959934
9527112396913538428517487694745026607579189600266019221604611821958190378000877839438368849325068987523419625191826798804
6558173288399730137494495945970997361718784132246881471355589345219481359776443724810119780678287882272032574122076482630
3792213930180223636379473976051607556532456288502367344454017509220677519189391201713980630486820155733390295070211419498
810969874828
```

Output:

```
{"extended_token_first":"1363062073821856071452572059931270212074754474140809303672216670258610641868217063897069503999949
9451653778759580311516884303551977558761001313764943279225637925148797448415141314167038765203286307055280553397270898471
3144636315784631553977767562324755636423494177174234371836227359421365419419845000450001369310560792356579560389998061985
6512105599263583715249376139798470647424665644350600525844859110976690396394908408940508005130474037446729201855930474625
8083524536030848830722464363654431904540811181838595411819299299375192277395993730109241245978978587107683921988474665711
712217532601549131605790699936559266","extended_token_second":"2132100264581502202312381449461022637938257910700482743211
6661793289018691982297030976335469674826726549078473725031918300917672739281908309034953235311003578367357123983996027776
2196105825770949475204584435076552930418205004196283392827739216519383155292804294296898202659027868181674625784164056140
6676667562215839055617148041511895484520387296250017482146691908147510884193838191431520413828355078993323640349702143209
0761301363306688311797171405115146239677660090414799545466290657412886287270129959947819275023850813227932833062285458174
041236844582008989835626149947378642846805755824817046522114189240285845932","query_first":"Feature1=ARD","query_second":
"Feature1=ARH","query_type":100}
```

Figure 6.6: Screenshot of the querying party when performing a disjunctive query.



Judge

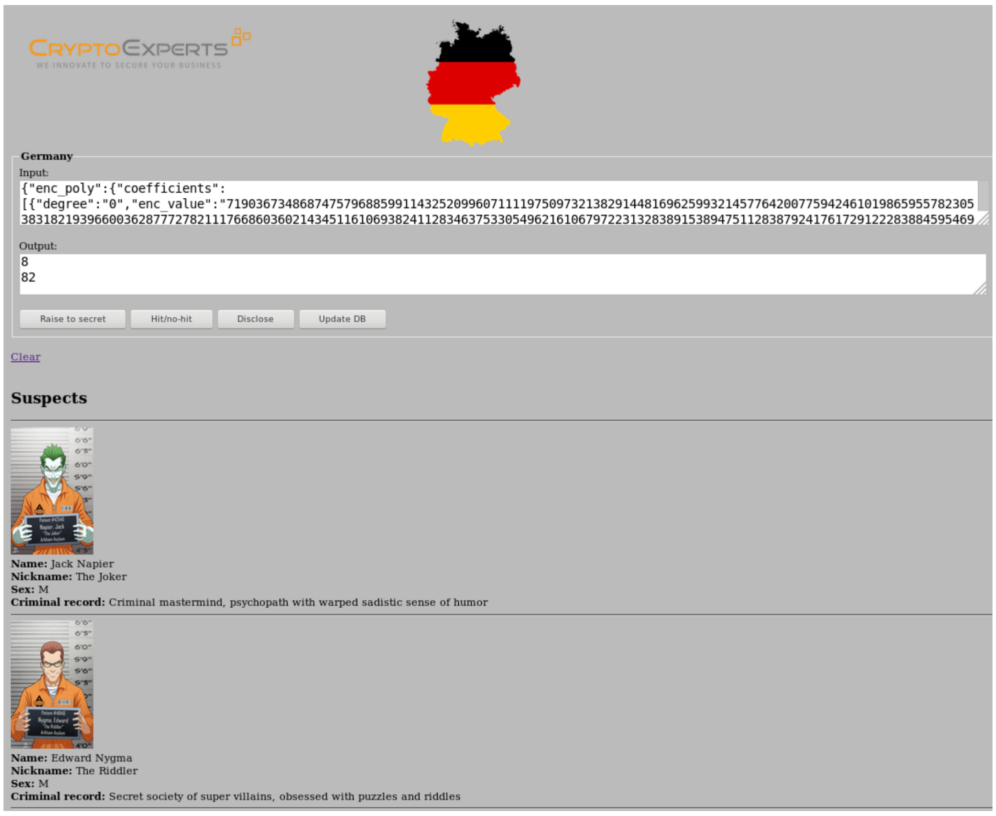
Input:

```
{"extended_token_first":"13630620738218560714525720599312702120747544741408093036722166702586106418682170638970695039994
994516537787595803115168843035519775587610013137649432792256379251487974484151413141670387652032863070552805533972708984
713144636315784631553977767562324755636423494177174234371836227359421365419419845000450001369310560792356579560389998061
985651210559926358371524937613979847064742466564435060052584485911097669039639490840894050800513047403744672920185593047
462580835245360308488307224643636544319045408111818385954118192992993751922773959937301092412459789785871076839219884746
65711712217532601549131605790699936559266","extended_token_second":"2132100264581502202312381449461022637938257910700482
743211666179328901869198229703097633546967482672654907847372503191830091767273928190830903495323531100357836735712398399
602777621961058257709494752045844350765529304182050041962833928277392165193831552928042942968982026590278681816746257841
640561406676667562215839055617148041511895484520387296250017482146691908147510884193838191431520413828355078993323640349
```

Output:

```
{"enc_poly":{"coefficients":
[{"degree":"0","enc_value":"71903673486874757968859911432520996071111975097321382914481696259932145776420077594246101986
595578230538318219396600362877727821117668603602143451161069382411283463753305496216106797223132838915389475112838792417
617291222838845954694902539921464558109586971399262467169165168371624864917570037368295606691336103571745587313877191095
581442840081171332977951923967655412956749679053832324288348617077641562212514629723102934679635277160113573395701442102
447503941193927162445654620720743637192263666059155818042296852770365752336629249343885692777883057424588991489413175020
862842098794213369861369745609161912824091157377782959070830155654640166602382428835949632948085464557133739941399072184
301307318974714167851667569249288702250080894458620922056716115608870226176339503606388562702347492036856015396478075646
145457923587920009515179571800001924805705910788126722417531907714733573956034367799239514441721942240576337403491404039 71
```

Figure 6.7: Screenshot of the trusted party (Judge) when processing a disjunctive query.



Germany

Input:

```
{"enc_poly":{"coefficients":
[{"degree":"0","enc_value":"71903673486874757968859911432520996071111975097321382914481696259932145776420077594246101986595578230 5
383182193966003628777278211176686036021434511610693824112834637533054962161067972231328389153894751128387924176172912228388459546 9
```

Output:

```
8
82
```

Suspects

Name: Jack Napier
Nickname: The Joker
Sex: M
Criminal record: Criminal mastermind, psychopath with warped sadistic sense of humor

Name: Edward Nygma
Nickname: The Riddler
Sex: M
Criminal record: Secret society of super villains, obsessed with puzzles and riddles

Figure 6.8: Screenshot of the responding party when disclosing the information after a disjunctive query.

The random polynomials $r_i(x)$ are chosen such that their degrees respect $\deg(r_1(x)) = \deg(p_2(x)) - 1$ and $\deg(r_2(x)) = \deg(p_1(x)) - 1$. This ensures that $p'(x)$ is distributed uniformly among all the polynomials of the same degree, as per [BGH+13, Lemma 2 and Corollary 3]. Another important observation is about the roots of these random polynomials. In order to decrease the probabilities of having spurious roots (cf. Section 6.4.2), we define a set of "good roots" $\mathcal{G}$ and a set of "fake roots" $\mathcal{F}$. We assume that the description of these sets is public, meaning that any party can distinguish if a root belongs to $\mathcal{G}$ or to $\mathcal{F}$. Good roots are those that correspond to indices which appear in the database, whereas fake roots do not. When choosing the random polynomials, we will take them so that their roots are in $\mathcal{F}$. Any potential collision between the roots will introduce an additional root in the polynomial $p'(x)$, but will be automatically ignored because it belongs to $\mathcal{F}$. However, it turns out this is not enough to prevent false positives.

6.4.2 Handling the "false positives"

As already mentioned in Sections 6.2 and 6.4.1, the way we handle conjunction queries might introduce spurious roots. In details, let $p_1(x)$ and $p_2(x)$ be two polynomials whose roots are the elements of the set $\mathcal{I}_1$ and $\mathcal{I}_2$, respectively. Then, if we pick two random polynomials $r_1(x)$ and $r_2(x)$, we have that the set of roots of $p'(x) \coloneqq p_1(x) \cdot r_1(x) + p_2(x) \cdot r_2(x)$ is at least $\mathcal{I}_1 \cap \mathcal{I}_2$. This is obvious for the following reason: $\forall y \in \mathcal{I}_1 \cap \mathcal{I}_2$, we have

$$q(y) = p_1(y) \cdot r_1(y) + p_2(y) \cdot r_2(y) = 0 \cdot r_1(y) + 0 \cdot r_2(y) = 0.$$

Moreover, following an analogous reasoning, let $\mathcal{R}_1, \mathcal{R}_2$ be the set of roots of random polynomials $r_1(x), r_2(x)$, respectively. It is obvious that any root that belongs to $\mathcal{R}_1 \cap \mathcal{R}_2$ will also be a root of $p'(x)$. However, as already mentioned in Section 6.4.1, this will not be a problem since $\mathcal{R}_1 \subseteq \mathcal{F}$ and $\mathcal{R}_2 \subseteq \mathcal{F}$, meaning that this kind of root will be rejected because it belongs to the set of "fake roots".

However, it can also happen that there exists a y such that $p'(y) = 0$, $y \in \mathcal{G}$, $y \notin \mathcal{I}_1 \cap \mathcal{I}_2$. This means that the root is accepted as corresponding to a record in the database (because it belongs to $\mathcal{G}$), but it represents a false positive, because it does not belong to $\mathcal{I}_1 \cap \mathcal{I}_2$. Avoiding this situation is extremely important, since allowing this would be the equivalent of "framing an innocent". In order to prevent this from happening, we chose to do the following: we repeat the entire procedure several times and, in the end, we take the intersection of the results. This means that we fix a parameter N_REPS and then we pick random polynomials

$$\left(r_1^{(1)}(x), r_2^{(1)}(x)\right), \quad \left(r_1^{(2)}(x), r_2^{(2)}(x)\right), \quad \ldots, \quad \left(r_1^{(\mathsf{N\_REPS})}(x), r_2^{(\mathsf{N\_REPS})}(x)\right),$$

compute

$$p'^{(1)}(x), \quad p'^{(2)}(x), \quad \ldots, \quad p'^{(\mathsf{N\_REPS})}(x)$$

as shown before, and then take the intersection of their roots (that belong to the set of good roots $\mathcal{G}$). It is easy to see that the probability of introducing spurious roots decreases rapidly as N_REPS increases. In the implementation, we take N_REPS $= 3$.

6.4.3 Concrete parameters and benchmarks

We now give the concrete parameters used in the implementation, and an analysis of the communication complexity of the protocol. The timings clearly depend on the environment in which the protocol is deployed, i.e., on the distance between the parties, and especially on the speed of the network. Beyond the notation already used in this chapter, we also denote by N the number of records in the database, by ν the number of features per record (e.g., DNA markers, fingerprint details, car plate numbers, ...), by n the number of records associated to each tag (in the inverted database), by k the number of linear combinations used for conjunctive queries (previously called N_REPS), and by f the maximum probability of false positives that we are willing to tolerate.

The parameters must satisfy the following relationships:

- $q \geq \frac{N}{f^{1/k}}$ (see [BGH+13, Lemma 1])
- $\rho = 2\kappa$, to prevent the improved gcd attack by Chen and Nguyen [CN12], which has complexity $\mathcal{O}\left(2^{\rho/2}\right)$
- $\eta > 2\rho + 2\log q$, to ensure correct decryption after a single multiplication
- $\gamma = \eta^2\omega(\log\kappa)$, to prevent the orthogonal lattice attack. To be conservative, one can take $\gamma = 8\eta^2$

In the implementation, we choose the following settings:

- $N = 10^4$
- $f = 2^{-32} \approx 10^{-9.63}$
- $\kappa = 128$

This leads to the following choice of parameters:

- $k = 3$
- $q \approx 10^{7.21}$
- $\rho = 256$
- $\eta = 560$
- $\gamma = 2508800$

In the polynomials representing the list of indices, each (encrypted) coefficient is approximately 2508800 bit long. Since JSON encodes all the values as text characters (using base 10 representation), the size of each encrypted coefficient is approximately 755 KBytes.

Assuming each record in the database has ν features, the total size of the inverted database is thus

$$N \cdot \nu \left(1 + \frac{1}{n_{\text{average}}}\right) \cdot (\text{size of enc. coefficient}) \approx 7.55 \cdot \nu \left(1 + \frac{1}{n_{\text{average}}}\right) \text{ GBytes},$$

where n_{average} is the average number of records associated to a tag.

	Total bandwidth
Simple query	$\approx 1510\,d$ KB
Conjunction query	$\approx 6795\,(d_1 + d_2)$ KB
Disjunction query	$\approx 1510\,(d_1 + d_2)$ KB

Table 6.3: Communication complexity of our PIR protocol.

6.4.3.1 Bandwidth requirements

Given the description of the protocol presented above, it is now easy to compute the communication complexity of the protocol, for each possible type of query.

Simple query

$$3 \cdot 2048 \text{ bits} + 2 \cdot ((755\,d) \text{ KB} + 2048 \text{ bits})$$

where $d = n + 1$ is the degree of the polynomial associated to the search token.

Conjunctive query

$$4{\cdot}2048 \text{ bits}{+}3\,(755\,(d_1 + d_2) \text{ KB}){+}1 \text{ bit}{+}4096 \text{ bits}{+}2\,(3 \cdot 755\,(d_1 + d_2) \text{ KB} + 2048 \text{ bits})$$

where $d_1 = n_1 + 1, d_2 = n_2 + 1$ are the decrees of the polynomials associated to the search tokens.

Disjunctive query

$$4 \cdot 2048 \text{ bits} + 4096 \text{ bits} + 2\,(755\,(d_1 + d_2) \text{ KB} + 2048 \text{ bits})$$

where $d_1 = n_1 + 1, d_2 = n_2 + 1$ are the decrees of the polynomials associated to the search tokens.

The final results are presented in Table 6.3.

6.5 Discussions

The protocol we presented improves on the protocol currently in place for cross-border cooperation between law enforcement agencies, since it preserves the privacy of the querying party's request. However, our protocol still presents issues and missing features, that we address in the following.

First of all, giving the querying party the ability of knowing whether the intersection is empty or not without the trusted party's authorization, is potentially dangerous because it might cause unwanted leakages. The idea behind this was to limit the need for trusted party's involvement in the protocol to the actual information disclosure moment, in order to limit the workload of the "Judge". A possible fix for this potential issue is that of not allowing any decryption on the responding partner's side without a trusted signature. This would mean that the querying partner has first to obtain a "preliminary authorization" for knowing whether the intersection is empty or not, and then a "final authorization" to obtain the database records, in case the intersection is non-empty. These modifications clearly depend on the legal constraints that one wants to enforce. An alternative solution is that of discarding the step that aims at computing whether there is a match, and execute the protocol until the end in any case. There is, of course, a trade-off: on one hand, this limits the workload of the Judge to having to compute just one signature, and it also solves the problem of the leakage that we described above; on the other hand, this might lead to performing "useless" computations, that would have been avoided, had the querying party known that the intersection is empty.

Another potential flaw in the protocol that we presented is that, after computing the search token, the querying partner is able to determine *how many* matches there are just by looking at the encrypted inverted database. In fact, it is sufficient to count the number of (encrypted) coefficients to determine the degree of the polynomial and, thus, the number of roots, i.e., the number of record IDs. This clearly does not reveal anything on the records themselves, but for some reasons one might want to avoid leaking this piece of information. In order to achieve this goal, the simplest strategy appears that of padding the degree of the polynomials by adding roots in $\mathcal{F}$ to the real IDs. This way, we can enforce the constraint that all the polynomials have the same degree, thus leaking no information about the number or real database records.

Finally, one might want to combine more than two queries, in any potential way. For example, given requests ξ_1, ξ_2, ξ_3, one might want to obtain the records that match $\xi_1 \wedge \xi_2 \wedge \xi_3$, or $(\xi_1 \vee \xi_2) \wedge \xi_3$. This is clearly possible by executing several times the protocol that we presented, but intermediate queries might not be allowed by the trusted Judge. For example, in order to obtain the records that match $(\xi_1 \vee \xi_2) \wedge \xi_3$, one might execute the protocol to obtain the matches for $(\xi_1 \vee \xi_2)$, then execute it again to obtain the records that match ξ_3, and then manually compute the intersection. However, this reveals more than computing the final result directly, and this might break some constraints. The solution is to tweak the choice of parameters so that more homomorphic operations are allowed (for example, in order to support more multiplications). If one wants to enable an unbounded number of unions and intersections, then bootstrapping appears to be necessary.

Chapter 7

Conclusions and open questions

In this chapter we summarize the contributions presented in this manuscript, we draw some conclusions, and we outline some questions that remain open. We also try to put the contributions in perspective and to analyze their impact on the field of cryptology.

Contents

7.1 Conclusions

If introduced on a large scale and in real-world scenarios, fully homomorphic encryption (FHE) would have enormous consequences for people's privacy, allowing them to use remote services hosted on untrusted servers, without disclosing personal information. As of today, the main obstacle towards this goal is efficiency: existing solutions provide all the necessary features, but they remain generally cumbersome when trying to achieve full-fledged FHE, that allows for unbounded computations. A more viable alternative is instantiating application-specific somewhat homomorphic encryption (SHE) schemes, that do not rely on bootstrapping, and that allow for the computations required by that particular application. Naturally, this depends on the application itself, and on how complex the computational tasks are.

In this manuscript, we first surveyed the area of FHE (cf. Chapter 3), and then we presented some works aimed at improving FHE both from a theoretical and a practical point of view.

In Chapter 4, we presented a new framework for evaluating a neural network on encrypted inputs, thus making sure that the user's data is not accessible to the server, while still allowing the user to obtain predictions based on previously-trained and remotely-held cognitive models.

In Chapter 5, we focused on the problem of circuit privacy. We improved on previous works from the point of view of simplicity and efficiency, in order to provide servers with an easy way to ensure that the results of homomorphic computations do not leak information about the algorithm that has been applied. This is particularly interesting from the viewpoint of a company, wanting to provide users with data processing services, without revealing potentially critical and proprietary algorithms. The result that we obtain, with some caveats, is that the final result of the computation looks (almost) like a fresh ciphertext of the result, thus making sure that no information on how this result was derived (i.e., the algorithm) is contained therein.

Finally, in Chapter 6, we presented and implemented a protocol for private information retrieval. The project's goals are mainly to (1) improve upon the protocol currently in place at European level for cross-border cooperation between law enforcement agencies, and (2) to demonstrate that homomorphic encryption can be practical for certain applications. In this work, we do not take advantage of homomorphic encryption schemes stemming from the assumed hardness of lattice problems, but we focus on a different class of schemes, arising from the assumed hardness of the approximate GCD problem (cf. Definition 2.4.6).

7.2 Open questions

Given the double nature (both theoretical and practical) of the works proposed, there are a number of questions that remain open, and that can serve as guidelines for future research. Some of these questions were already mentioned in the related chapters of this manuscript: we report them here for better clarity.

7.2.1 Homomorphic evaluation of neural networks

Question 7.1. *In the framework that we proposed, called* FHE-DiNN*, we are able to homomorphically evaluate a neural network only after a severe simplification of the model.*

Namely, we have to discretize the inputs, the weights and biases. Moreover, we are limited to considering the sign as an activation function. Although it has been shown in the literature that this is already enough to achieve near state-of-the-art results, this is indeed an important limitation. The natural question is then whether we can avoid some (potentially all) of these constraints, towards the goal of homomorphically evaluating any neural network, with real inputs, reals weights and biases, and any activation function.

Question 7.2. *As a complement (or an alternative) to the previous question, from a practical point of view, can we design an automated "compiler" that, given as input an already trained neural network, returns a neural network that can be evaluated on encrypted inputs? That is, designing a complete ecosystem of softwares that takes care of all the steps of the interaction: encrypting the inputs (on the user's side), evaluating the network (on the server's side), and then decrypting the results (back on the user's side).*

Question 7.3. *In the approach that we proposed, we apply the bootstrapping function after processing every neuron. Since this operation is heavy and time-consuming, is there a way to reduce the number of bootstrappings needed to evaluate an entire network? Also, another strong optimization regards parallelization: all the bootstrappings are independent, so we could take advantage of powerful GPUs in order to perform several of these operations together, thus saving in execution time.*

Question 7.4. *Another open question regarding bootstrapping is the following: can we batch several of these operations together, so that the cost for bootstrapping N neurons is less than N times the cost of a single bootstrapping?*

7.2.2 Circuit privacy

Question 7.5. *The first open question that comes to mind is whether (and how) our technique based on a Gaussian leftover hash lemma can be extended to FHE schemes other than GSW. For example, the so-called second generation FHE is usually more efficient for specific applications: can our approach be extended to such schemes? More importantly, can we blend the results on homomorphic evaluation of deep neural networks presented in Chapter 4 together with the technique for circuit privacy showed in Chapter 5?*

Question 7.6. *The threat model that we consider in our work is that of a honest-but-curious adversary. Can we extend our technique to be secure against malicious adversaries, that can deviate from the prescribed protocol, send malformed inputs, . . . ? There appear to be essentially two roads to this goal: we can either (1) have a way to prove or to check that the input values are well formed, and reject them if they are not, or (2) enhance our technique so that the result leaks no information on the computation, regardless of the input that has been submitted.*

Question 7.7. *Can we achieve CCA1-secure FHE? We note that this is not possible for those approaches that rely on bootstrapping, since publishing the bootstrapping key violates the CCA1 security requirements.*

7.2.3 Private information retrieval

Question 7.8. *What exactly are the legal constraints that the protocol has to satisfy?*

Given the goal of the project, i.e., to improve upon the way information is currently exchanged between law enforcement agencies across borders, it is important to pinpoint exactly the legal constraints. For example, is it acceptable to be able to check whether the intersection of two queries is empty or not, without the trusted party's authorization? As it was underlined in Chapter 6, there is a trade-off between how strict the security model is, and how heavy the workload for the trusted party is. Defining this and other constraints can help understand how the protocol should behave.

Question 7.9. *Would the protocol be more efficient if implemented with another homomorphic encryption scheme?*

The implementation that we realized was based on the DGHV encryption scheme [DGHV10], but there are plenty of other possibilities, like lattice-based encryption schemes that rely on the LWE assumption (cf. Assumption 2.4.4).

Notation

General mathematical notation	
$:=$	"Is defined as"
$\mathbb{N}$	The set of natural numbers
$\mathbb{Z}$	The set of integer numbers
$\mathbb{Z}_q$	The set of integer numbers modulo q
$\mathbb{R}$	The set of real numbers
$\mathbb{T}$	The torus, i.e., the set of integers modulo 1
$\log$	Base-2 logarithm
$\mathbf{v}$	A (column) vector
$\mathbf{v}^{\mathsf{T}}$	A transposed vector (i.e., a row vector)
$\mathbf{A}$	A matrix
$\mathcal{R}$	A ring
$\mathcal{R}[X]$	The set of polynomials in X with coefficients in $\mathcal{R}$
$\langle \cdot, \cdot \rangle$	The inner product
$\lvert \mathcal{S} \rvert$	The size (or cardinality) of a set $\mathcal{S}$
$\lceil y \rceil$	The smallest integer z such that $z \geq y$
$\lfloor y \rfloor$	The biggest integer z such that $z \leq y$
$\lfloor y \rceil$	The closest integer to y, with ties broken upward
$\lVert \cdot \rVert_{p,\infty}$	Norms
$x \leftarrow_{\$} \mathcal{S}$	x is sampled uniformly at random from $\mathcal{S}$
$\Delta(X, Y)$	The statistical distance between X and Y
$\mathcal{N}$	A neural network
$\mathcal{O}(\cdot), \widetilde{\mathcal{O}}(\cdot), \Theta(\cdot), \omega(\cdot)$	Asymptotic notation
$\Pr[\mathcal{E}]$	The probability of the event $\mathcal{E}$
$H_\infty(X)$	The min entropy of X
General cryptographic notation	
κ	The security parameter
$\overset{c}{\approx}$	Computationally indistinguishable
$\approx_s$	Statistically indistinguishable
$\mathcal{H}$	A hash function
$\mathcal{D}$	A discrete Gaussian
pk	A public key
sk	A secret key
evk	An evaluation key
ksk	A key-switching key
bk	A bootstrapping key
Notation specific to Chapter 5	
$\mathbf{G}$	The gadget matrix
$\mathbf{G}^{-1}_{\mathrm{det}}(\cdot)$	Bit decomposition
$\mathbf{G}^{-1}(\cdot)$	Randomized bit decomposition
ρ	The Gaussian function
Λ	A lattice
$\lambda_i(\Lambda)$	The i-th minimum of Λ
$\det(\Lambda)$	The determinant of Λ
$\eta_\varepsilon(\Lambda)$	The smoothing parameter of Λ

Abbreviations

FHE	Fully Homomorphic Encryption
SHE	Somewhat Homomorphic Encryption
IND-CPA	Indistinguishability under Chosen Plaintext Attack
SVP	Shortest Vector Problem
SIVP	Shortest Independent Vector Problem
CVP	Closest Vector Problem
BDD	Bounded Distance Decoding
LWE	Learning With Errors
SIS	Short Integer Solution
GCD	Greatest Common Divisor
NN	Neural Network
MNIST	Modified National Institute of Standards and Technology
TFHE	Torus FHE
TLWE	Torus LWE
TGSW	Torus GSW
DiNN	Discretized Neural Network
OC	Organized Crime

List of Illustrations

Figures

Tables

Bibliography

[08a] *Council Decision 2008/615/JHA of 23 June 2008 on the stepping up of cross-border cooperation, particularly in combating terrorism and cross-border crime.* http://eur-lex.europa.eu/legal-content/EN/TXT/?uri=celex:32008D0615. 2008 (cit. on p. 96).

[08b] *Council Decision 2008/616/JHA of 23 June 2008 on the implementation of Decision 2008/615/JHA on the stepping up of cross-border cooperation, particularly in combating terrorism and cross-border crime.* http://eur-lex.europa.eu/legal-content/EN/TXT/?uri=celex:32008D0616. 2008 (cit. on p. 96).

[10] *Flask (a Python microframework): web development, one drop at a time.* http://flask.pocoo.org/. 2010 (cit. on p. 108).

[ABDP15] Michel Abdalla, Florian Bourse, Angelo De Caro, and David Pointcheval. "Simple Functional Encryption Schemes for Inner Products". In: *PKC 2015.* Ed. by Jonathan Katz. Vol. 9020. LNCS. Springer, Heidelberg, Mar. 2015, pp. 733–751. DOI: 10.1007/978-3-662-46447-2_33 (cit. on p. 57).

[ACPS09] Benny Applebaum, David Cash, Chris Peikert, and Amit Sahai. "Fast Cryptographic Primitives and Circular-Secure Encryption Based on Hard Learning Problems". In: *CRYPTO 2009.* Ed. by Shai Halevi. Vol. 5677. LNCS. Springer, Heidelberg, Aug. 2009, pp. 595–618 (cit. on pp. 74, 81).

[AGHS13] Shweta Agrawal, Craig Gentry, Shai Halevi, and Amit Sahai. "Discrete Gaussian Leftover Hash Lemma over Infinite Domains". In: *ASIACRYPT 2013, Part I.* Ed. by Kazue Sako and Palash Sarkar. Vol. 8269. LNCS. Springer, Heidelberg, Dec. 2013, pp. 97–116. DOI: 10.1007/978-3-642-42033-7_6 (cit. on pp. 73, 79).

[AJL+12] Gilad Asharov, Abhishek Jain, Adriana López-Alt, Eran Tromer, Vinod Vaikuntanathan, and Daniel Wichs. "Multiparty Computation with Low Communication, Computation and Interaction via Threshold FHE". In: *EUROCRYPT 2012.* Ed. by David Pointcheval and Thomas Johansson. Vol. 7237. LNCS. Springer, Heidelberg, Apr. 2012, pp. 483–501 (cit. on p. 37).

[Ajt96] Miklós Ajtai. "Generating Hard Instances of Lattice Problems (Extended Abstract)". In: *28th ACM STOC.* ACM Press, May 1996, pp. 99–108 (cit. on pp. 19, 20).

[ALS16] Shweta Agrawal, Benoît Libert, and Damien Stehlé. "Fully Secure Functional Encryption for Inner Products, from Standard Assumptions". In: *CRYPTO 2016, Part III.* Ed. by Matthew Robshaw and Jonathan Katz. Vol. 9816. LNCS. Springer, Heidelberg, Aug. 2016, pp. 333–362. DOI: 10.1007/978-3-662-53015-3_12 (cit. on p. 57).

[And10] Nate Anderson. *How "cell tower dumps" caught the High Country Bandits — and why it matters.* Ars Technica. https://arstechnica.com/tech-policy/2013/08/how-cell-tower-dumps-caught-the-high-country-bandits-and-why-it-matters/. 2010 (cit. on p. 95).

[AP13] Jacob Alperin-Sheriff and Chris Peikert. "Practical Bootstrapping in Quasilinear Time". In: *CRYPTO 2013, Part I.* Ed. by Ran Canetti and Juan A. Garay. Vol. 8042. LNCS. Springer, Heidelberg, Aug. 2013, pp. 1–20. DOI: 10.1007/978-3-642-40041-4_1 (cit. on p. 30).

[AP14] Jacob Alperin-Sheriff and Chris Peikert. "Faster Bootstrapping with Polynomial Error". In: *CRYPTO 2014, Part I.* Ed. by Juan A. Garay and Rosario Gennaro. Vol. 8616. LNCS. Springer, Heidelberg, Aug. 2014, pp. 297–314. DOI: 10.1007/978-3-662-44371-2_17 (cit. on pp. 27, 30, 33, 34, 38, 53, 72–77, 84).

[APS15] Martin R. Albrecht, Rachel Player, and Sam Scott. *On The Concrete Hardness Of Learning With Errors.* Cryptology ePrint Archive, Report 2015/046. http://eprint.iacr.org/2015/046. 2015 (cit. on pp. 21, 65).

[AR13] Divesh Aggarwal and Oded Regev. "A Note on Discrete Gaussian Combinations of Lattice Vectors". In: *CoRR* abs/1308.2405 (2013). URL: http://arxiv.org/abs/1308.2405 (cit. on pp. 73, 79).

[Avn16] Amir Avni. *Two Weeks of Colorizebot - Conclusions and Statistics.* http://whatimade.today/two-weeks-of-colorizebot-conclusions-and-statistics/. 2016 (cit. on p. 47).

[BBB+17] Anthony Barnett, Charlotte Bonte, Carl Bootland, Joppe W. Bos, Wouter Castryck, Anamaria Costache, Louis Goubin, Ilia Iliashenko, Tancrède Lepoint, Michele Minelli, Pascal Paillier, Nigel P. Smart, Frederik Vercauteren, Srinivas Vivek, and Adrian Waller. *Processing Encrypted Data Using Homomorphic Encryption.* Workshop on Data Mining with Secure Computation, SODA project. 2017 (cit. on p. 9).

[BEHZ16] Jean-Claude Bajard, Julien Eynard, Anwar Hasan, and Vincent Zucca. *A Full RNS Variant of FV like Somewhat Homomorphic Encryption Schemes.* Cryptology ePrint Archive, Report 2016/510. http://eprint.iacr.org/2016/510. 2016 (cit. on p. 38).

[Ben16] Fabrice Ben Hamouda–Guichoux. "Diverse modules and zero-knowledge". Theses. PSL Research University, July 2016. URL: https://tel.archives-ouvertes.fr/tel-01492851 (cit. on p. 13).

[Ber14] Daniel J. Bernstein. *A subfield-logarithm attack against ideal lattices.* Blog post. http://blog.cr.yp.to/20140213-ideal.html. 2014 (cit. on p. 21).

[BF17] Guillaume Bonnoron and Caroline Fontaine. "A Note on Ring-LWE Security in the Case of Fully Homomorphic Encryption". In: *INDOCRYPT 2017.* Ed. by Arpita Patra and Nigel P. Smart. Vol. 10698. LNCS. Springer, Heidelberg, Dec. 2017, pp. 27–43 (cit. on p. 21).

[BFM88] Manuel Blum, Paul Feldman, and Silvio Micali. "Non-Interactive Zero-Knowledge and Its Applications (Extended Abstract)". In: *20th ACM STOC.* ACM Press, May 1988, pp. 103–112 (cit. on p. 10).

[BGG+14] Dan Boneh, Craig Gentry, Sergey Gorbunov, Shai Halevi, Valeria Nikolaenko, Gil Segev, Vinod Vaikuntanathan, and Dhinakaran Vinayagamurthy. “Fully Key-Homomorphic Encryption, Arithmetic Circuit ABE and Compact Garbled Circuits”. In: *EUROCRYPT 2014*. Ed. by Phong Q. Nguyen and Elisabeth Oswald. Vol. 8441. LNCS. Springer, Heidelberg, May 2014, pp. 533–556. DOI: 10.1007/978-3-642-55220-5_30 (cit. on p. 91).

[BGH+13] Dan Boneh, Craig Gentry, Shai Halevi, Frank Wang, and David J. Wu. “Private Database Queries Using Somewhat Homomorphic Encryption”. In: *ACNS 13*. Ed. by Michael J. Jacobson Jr., Michael E. Locasto, Payman Mohassel, and Reihaneh Safavi-Naini. Vol. 7954. LNCS. Springer, Heidelberg, June 2013, pp. 102–118. DOI: 10.1007/978-3-642-38980-1_7 (cit. on pp. 98, 111, 112).

[BGN05] Dan Boneh, Eu-Jin Goh, and Kobbi Nissim. “Evaluating 2-DNF Formulas on Ciphertexts”. In: *TCC 2005*. Ed. by Joe Kilian. Vol. 3378. LNCS. Springer, Heidelberg, Feb. 2005, pp. 325–341 (cit. on p. 26).

[BGV12] Zvika Brakerski, Craig Gentry, and Vinod Vaikuntanathan. “(Leveled) fully homomorphic encryption without bootstrapping”. In: *ITCS 2012*. Ed. by Shafi Goldwasser. ACM, Jan. 2012, pp. 309–325 (cit. on pp. 27, 29, 31, 33, 38, 52, 72).

[BISW17] Dan Boneh, Yuval Ishai, Amit Sahai, and David J. Wu. “Lattice-Based SNARGs and Their Application to More Efficient Obfuscation”. In: *EUROCRYPT 2017, Part III*. Ed. by Jean-Sébastien Coron and Jesper Buus Nielsen. Vol. 10212. LNCS. Springer, Heidelberg, Apr. 2017, pp. 247–277 (cit. on p. 10).

[BISW18] Dan Boneh, Yuval Ishai, Amit Sahai, and David J. Wu. “Quasi-Optimal SNARGs via Linear Multi-Prover Interactive Proofs”. In: *EUROCRYPT 2018, Part III*. Ed. by Jesper Buus Nielsen and Vincent Rijmen. Vol. 10822. LNCS. Springer, Heidelberg, Apr. 2018, pp. 222–255. DOI: 10.1007/978-3-319-78372-7_8 (cit. on p. 10).

[BLMZ17] Fabrice Benhamouda, Tancrède Lepoint, Claire Mathieu, and Hang Zhou. “Optimization of Bootstrapping in Circuits”. In: *28th SODA*. Ed. by Philip N. Klein. ACM-SIAM, Jan. 2017, pp. 2423–2433 (cit. on p. 30).

[BLP+13] Zvika Brakerski, Adeline Langlois, Chris Peikert, Oded Regev, and Damien Stehlé. “Classical hardness of learning with errors”. In: *45th ACM STOC*. Ed. by Dan Boneh, Tim Roughgarden, and Joan Feigenbaum. ACM Press, June 2013, pp. 575–584 (cit. on p. 76).

[BMMP18] Florian Bourse, Michele Minelli, Matthias Minihold, and Pascal Paillier. *Fast Homomorphic Evaluation of Deep Discretized Neural Networks*. CRYPTO 2018. https://eprint.iacr.org/2017/1114. 2018 (cit. on p. 8).

[Bon98] Dan Boneh. “The decision Diffie-Hellman problem”. In: *Third Algorithmic Number Theory Symposium (ANTS)*. Vol. 1423. LNCS. Invited paper. Springer, Heidelberg, 1998 (cit. on p. 97).

[Bou17] Florian Bourse. “Functional Encryption for Inner-Product Evaluations”. Theses. Université de recherche Paris Sciences et Lettres, Dec. 2017. URL: https://hal.archives-ouvertes.fr/tel-01665276 (cit. on p. 13).

[BPDS16] Daniel J. Bernstein, Chris Peikert, Léo Ducas, and Damien Stehlé. *Ideal-SVP attacks?* Discussion on Google Groups. https://groups.google.com/forum/#!topic/cryptanalytic-algorithms/y-wAnhmGsIo. 2016 (cit. on p. 21).

[BPL+15] Daniel J. Bernstein, Chris Peikert, Vadim Lyubashevsky, Thomas Wunderer, and Jean-François Biasse. *Soliloquy.* Discussion on Google Groups. https://groups.google.com/forum/#!topic/cryptanalytic-algorithms/GdVfp5Kbdb8. 2015 (cit. on p. 21).

[BPMW16] Florian Bourse, Rafaël del Pino, Michele Minelli, and Hoeteck Wee. "FHE Circuit Privacy Almost for Free". In: *CRYPTO 2016, Part II.* Ed. by Matthew Robshaw and Jonathan Katz. Vol. 9815. LNCS. Springer, Heidelberg, Aug. 2016, pp. 62–89. DOI: 10.1007/978-3-662-53008-5_3 (cit. on p. 9).

[BPTG15] Raphael Bost, Raluca Ada Popa, Stephen Tu, and Shafi Goldwasser. "Machine Learning Classification over Encrypted Data". In: *NDSS 2015.* The Internet Society, Feb. 2015 (cit. on p. 51).

[BV11a] Zvika Brakerski and Vinod Vaikuntanathan. "Efficient Fully Homomorphic Encryption from (Standard) LWE". In: *52nd FOCS.* Ed. by Rafail Ostrovsky. IEEE Computer Society Press, Oct. 2011, pp. 97–106 (cit. on pp. 27, 31, 33, 72, 91).

[BV11b] Zvika Brakerski and Vinod Vaikuntanathan. "Fully Homomorphic Encryption from Ring-LWE and Security for Key Dependent Messages". In: *CRYPTO 2011.* Ed. by Phillip Rogaway. Vol. 6841. LNCS. Springer, Heidelberg, Aug. 2011, pp. 505–524 (cit. on pp. 27, 31, 72).

[BV14] Zvika Brakerski and Vinod Vaikuntanathan. "Lattice-based FHE as secure as PKE". In: *ITCS 2014.* Ed. by Moni Naor. ACM, Jan. 2014, pp. 1–12 (cit. on pp. 27, 72–75, 78, 84).

[CB16] Matthieu Courbariaux and Yoshua Bengio. "BinaryNet: Training Deep Neural Networks with Weights and Activations Constrained to +1 or -1". In: *CoRR* abs/1602.02830 (2016). URL: http://arxiv.org/abs/1602.02830 (cit. on p. 69).

[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: *EUROCRYPT 2013.* Ed. by Thomas Johansson and Phong Q. Nguyen. Vol. 7881. LNCS. Springer, Heidelberg, May 2013, pp. 315–335. DOI: 10.1007/978-3-642-38348-9_20 (cit. on p. 39).

[CGGI16a] I. Chillotti, N. Gama, M. Georgieva, and M. Izabachène. *TFHE: Fast Fully Homomorphic Encryption Library over the Torus.* https://github.com/tfhe/tfhe. 2016 (cit. on pp. 38, 65).

[CGGI16b] Ilaria Chillotti, Nicolas Gama, Mariya Georgieva, and Malika Izabachène. "Faster Fully Homomorphic Encryption: Bootstrapping in Less Than 0.1 Seconds". In: *ASIACRYPT 2016, Part I.* Ed. by Jung Hee Cheon and Tsuyoshi Takagi. Vol. 10031. LNCS. Springer, Heidelberg, Dec. 2016, pp. 3–33. DOI: 10.1007/978-3-662-53887-6_1 (cit. on pp. 30, 38, 41, 52–54, 57, 58, 65, 67).

[CGGI17] Ilaria Chillotti, Nicolas Gama, Mariya Georgieva, and Malika Izabachène. "Faster Packed Homomorphic Operations and Efficient Circuit Bootstrapping for TFHE". In: *ASIACRYPT 2017, Part I*. Ed. by Tsuyoshi Takagi and Thomas Peyrin. Vol. 10624. LNCS. Springer, Heidelberg, Dec. 2017, pp. 377–408 (cit. on pp. 30, 38, 41, 52, 63).

[CGKS95] Benny Chor, Oded Goldreich, Eyal Kushilevitz, and Madhu Sudan. "Private Information Retrieval". In: *36th FOCS*. IEEE Computer Society Press, Oct. 1995, pp. 41–50 (cit. on p. 97).

[CH11] Henry Cohn and Nadia Heninger. *Approximate common divisors via lattices*. Cryptology ePrint Archive, Report 2011/437. http://eprint.iacr.org/2011/437. 2011 (cit. on p. 23).

[Cha04] Yan-Cheng Chang. "Single Database Private Information Retrieval with Logarithmic Communication". In: *ACISP 04*. Ed. by Huaxiong Wang, Josef Pieprzyk, and Vijay Varadharajan. Vol. 3108. LNCS. Springer, Heidelberg, July 2004, pp. 50–61. DOI: 10.1007/978-3-540-27800-9_5 (cit. on p. 97).

[CHH+16] Hao Chen, Kyoohyung Han, Zhicong Huang, Amir Jalali, and Kim Laine. *Simple Encrypted Arithmetic Library (SEAL) v2.3.0*. https://www.microsoft.com/en-us/research/project/simple-encrypted-arithmetic-library/. 2016 (cit. on p. 38).

[Cho+15] François Chollet et al. *Keras*. https://github.com/keras-team/keras. 2015 (cit. on p. 64).

[CLT11] Emiliano De Cristofaro, Yanbin Lu, and Gene Tsudik. "Efficient Techniques for Privacy-Preserving Sharing of Sensitive Information". In: *Trust and Trustworthy Computing - 4th International Conference, TRUST 2011, Pittsburgh, PA, USA, June 22-24, 2011. Proceedings*. Ed. by Jonathan M. McCune, Boris Balacheff, Adrian Perrig, Ahmad-Reza Sadeghi, M. Angela Sasse, and Yolanta Beres. Vol. 6740. Lecture Notes in Computer Science. Springer, 2011, pp. 239–253. DOI: 10.1007/978-3-642-21599-5_18. URL: http://dx.doi.org/10.1007/978-3-642-21599-5_18 (cit. on pp. 96, 107).

[CM15] Michael Clear and Ciaran McGoldrick. "Multi-identity and Multi-key Leveled FHE from Learning with Errors". In: *CRYPTO 2015, Part II*. Ed. by Rosario Gennaro and Matthew J. B. Robshaw. Vol. 9216. LNCS. Springer, Heidelberg, Aug. 2015, pp. 630–656. DOI: 10.1007/978-3-662-48000-7_31 (cit. on p. 37).

[CMNT11] Jean-Sébastien Coron, Avradip Mandal, David Naccache, and Mehdi Tibouchi. "Fully Homomorphic Encryption over the Integers with Shorter Public Keys". In: *CRYPTO 2011*. Ed. by Phillip Rogaway. Vol. 6841. LNCS. Springer, Heidelberg, Aug. 2011, pp. 487–504 (cit. on pp. 23, 39).

[CMS12] D. Cireşan, U. Meier, and J. Schmidhuber. "Multi-column Deep Neural Networks for Image Classification". In: *ArXiv e-prints* (Feb. 2012). arXiv: 1202.2745 [cs.CV] (cit. on p. 51).

[CN12] Yuanmi Chen and Phong Q. Nguyen. “Faster Algorithms for Approximate Common Divisors: Breaking Fully-Homomorphic-Encryption Challenges over the Integers”. In: *EUROCRYPT 2012*. Ed. by David Pointcheval and Thomas Johansson. Vol. 7237. LNCS. Springer, Heidelberg, Apr. 2012, pp. 502–519 (cit. on pp. 23, 112).

[CNT12] Jean-Sébastien Coron, David Naccache, and Mehdi Tibouchi. “Public Key Compression and Modulus Switching for Fully Homomorphic Encryption over the Integers”. In: *EUROCRYPT 2012*. Ed. by David Pointcheval and Thomas Johansson. Vol. 7237. LNCS. Springer, Heidelberg, Apr. 2012, pp. 446–464 (cit. on p. 23).

[CO15] Tung Chou and Claudio Orlandi. “The Simplest Protocol for Oblivious Transfer”. In: *LATINCRYPT 2015*. Ed. by Kristin E. Lauter and Francisco Rodríguez-Henríquez. Vol. 9230. LNCS. Springer, Heidelberg, Aug. 2015, pp. 40–58. DOI: 10.1007/978-3-319-22174-8_3 (cit. on p. 97).

[Cou17] Geoffroy Couteau. “Zero-Knowledge Proofs for Secure Computation”. Theses. PSL research University, Nov. 2017. URL: https://hal.inria.fr/tel-01668125 (cit. on p. 13).

[Cré88] Claude Crépeau. “Equivalence Between Two Flavours of Oblivious Transfers”. In: *CRYPTO’87*. Ed. by Carl Pomerance. Vol. 293. LNCS. Springer, Heidelberg, Aug. 1988, pp. 350–354 (cit. on p. 97).

[CWM+17] Hervé Chabanne, Amaury de Wargny, Jonathan Milgram, Constance Morel, and Emmanuel Prouff. “Privacy-Preserving Classification on Deep Neural Network”. In: *IACR Cryptology ePrint Archive* 2017 (2017), p. 35 (cit. on p. 52).

[Cyb89] G. Cybenko. “Approximation by superpositions of a sigmoidal function”. In: *Mathematics of Control, Signals and Systems* 2.4 (Dec. 1989), pp. 303–314. ISSN: 1435-568X. DOI: 10.1007/BF02551274. URL: https://doi.org/10.1007/BF02551274 (cit. on p. 47).

[DFGK14] George Danezis, Cédric Fournet, Jens Groth, and Markulf Kohlweiss. “Square Span Programs with Applications to Succinct NIZK Arguments”. In: *ASIACRYPT 2014, Part I*. Ed. by Palash Sarkar and Tetsu Iwata. Vol. 8873. LNCS. Springer, Heidelberg, Dec. 2014, pp. 532–550. DOI: 10.1007/978-3-662-45611-8_28 (cit. on p. 10).

[DGHV10] Marten van Dijk, Craig Gentry, Shai Halevi, and Vinod Vaikuntanathan. “Fully Homomorphic Encryption over the Integers”. In: *EUROCRYPT 2010*. Ed. by Henri Gilbert. Vol. 6110. LNCS. Springer, Heidelberg, May 2010, pp. 24–43 (cit. on pp. 5, 27, 39, 72, 93, 103–106, 118).

[DGL+16] Nathan Dowlin, Ran Gilad-Bachrach, Kim Laine, Kristin Lauter, Michael Naehrig, and John Wernsing. *CryptoNets: Applying Neural Networks to Encrypted Data with High Throughput and Accuracy*. Tech. rep. Feb. 2016. URL: https://www.microsoft.com/en-us/research/publication/cryptonets-applying-neural-networks-to-encrypted-data-with-high-throughput-and-accuracy/ (cit. on pp. 45, 52, 69).

[DH76] Whitfield Diffie and Martin E. Hellman. “New Directions in Cryptography”. In: *IEEE Transactions on Information Theory* 22.6 (1976), pp. 644–654 (cit. on p. 3).

[DM15] Léo Ducas and Daniele Micciancio. “FHEW: Bootstrapping Homomorphic Encryption in Less Than a Second”. In: *EUROCRYPT 2015, Part I*. Ed. by Elisabeth Oswald and Marc Fischlin. Vol. 9056. LNCS. Springer, Heidelberg, Apr. 2015, pp. 617–640. DOI: 10.1007/978-3-662-46800-5_24 (cit. on pp. 30, 73, 91).

[DMO00] Giovanni Di Crescenzo, Tal Malkin, and Rafail Ostrovsky. “Single Database Private Information Retrieval Implies Oblivious Transfer”. In: *EUROCRYPT 2000*. Ed. by Bart Preneel. Vol. 1807. LNCS. Springer, Heidelberg, May 2000, pp. 122–138 (cit. on p. 97).

[DP18] Wei Dai and Crypto++ Project. *Crypto++ Library 7.0*. https://www.cryptopp.com/. 2018 (cit. on p. 106).

[DRS04] Yevgeniy Dodis, Leonid Reyzin, and Adam Smith. “Fuzzy Extractors: How to Generate Strong Keys from Biometrics and Other Noisy Data”. In: *EUROCRYPT 2004*. Ed. by Christian Cachin and Jan Camenisch. Vol. 3027. LNCS. Springer, Heidelberg, May 2004, pp. 523–540 (cit. on p. 77).

[DS16] Léo Ducas and Damien Stehlé. “Sanitization of FHE Ciphertexts”. In: *EUROCRYPT 2016, Part I*. Ed. by Marc Fischlin and Jean-Sébastien Coron. Vol. 9665. LNCS. Springer, Heidelberg, May 2016, pp. 294–310. DOI: 10.1007/978-3-662-49890-3_12 (cit. on pp. 72, 91).

[ElG84] Taher ElGamal. “A Public Key Cryptosystem and a Signature Scheme Based on Discrete Logarithms”. In: *CRYPTO’84*. Ed. by G. R. Blakley and David Chaum. Vol. 196. LNCS. Springer, Heidelberg, Aug. 1984, pp. 10–18 (cit. on p. 5).

[Elm90] Jeffrey L. Elman. “Finding Structure in Time”. In: *Cognitive Science* 14.2 (1990), pp. 179–211. DOI: 10.1207/s15516709cog1402_1. URL: https://onlinelibrary.wiley.com/doi/abs/10.1207/s15516709cog1402_1 (cit. on p. 43).

[FV12] Junfeng Fan and Frederik Vercauteren. *Somewhat Practical Fully Homomorphic Encryption*. Cryptology ePrint Archive, Report 2012/144. http://eprint.iacr.org/2012/144. 2012 (cit. on p. 38).

[Gen09a] Craig Gentry. “A fully homomorphic encryption scheme”. crypto.stanford.edu/craig. PhD thesis. Stanford University, 2009 (cit. on pp. 6, 30, 83).

[Gen09b] Craig Gentry. “Fully homomorphic encryption using ideal lattices”. In: *41st ACM STOC*. Ed. by Michael Mitzenmacher. ACM Press, May 2009, pp. 169–178 (cit. on pp. 26–28, 30, 72).

[GGH97] Oded Goldreich, Shafi Goldwasser, and Shai Halevi. “Public-Key Cryptosystems from Lattice Reduction Problems”. In: *CRYPTO’97*. Ed. by Burton S. Kaliski Jr. Vol. 1294. LNCS. Springer, Heidelberg, Aug. 1997, pp. 112–131 (cit. on p. 31).

[GH10] Craig Gentry and Shai Halevi. *Implementing Gentry's Fully-Homomorphic Encryption Scheme*. Cryptology ePrint Archive, Report 2010/520. http://eprint.iacr.org/2010/520. 2010 (cit. on p. 31).

[GHS12] Craig Gentry, Shai Halevi, and Nigel P. Smart. "Homomorphic Evaluation of the AES Circuit". In: *CRYPTO 2012*. Ed. by Reihaneh Safavi-Naini and Ran Canetti. Vol. 7417. LNCS. Springer, Heidelberg, Aug. 2012, pp. 850–867 (cit. on pp. 27, 72).

[GHV10] Craig Gentry, Shai Halevi, and Vinod Vaikuntanathan. "i-Hop Homomorphic Encryption and Rerandomizable Yao Circuits". In: *CRYPTO 2010*. Ed. by Tal Rabin. Vol. 6223. LNCS. Springer, Heidelberg, Aug. 2010, pp. 155–172 (cit. on pp. 72, 84).

[GM82] Shafi Goldwasser and Silvio Micali. "Probabilistic Encryption and How to Play Mental Poker Keeping Secret All Partial Information". In: *14th ACM STOC*. ACM Press, May 1982, pp. 365–377 (cit. on p. 26).

[GMNO18] Rosario Gennaro, Michele Minelli, Anca Nitulescu, and Michele Orrù. *Lattice-Based zk-SNARKs from Square Span Programs*. ACM CCS 2018. https://eprint.iacr.org/2018/275. 2018 (cit. on p. 10).

[GMR89] Shafi Goldwasser, Silvio Micali, and Charles Rackoff. "The Knowledge Complexity of Interactive Proof Systems". In: *SIAM Journal on Computing* 18.1 (1989), pp. 186–208 (cit. on p. 10).

[GPV08] Craig Gentry, Chris Peikert, and Vinod Vaikuntanathan. "Trapdoors for hard lattices and new cryptographic constructions". In: *40th ACM STOC*. Ed. by Richard E. Ladner and Cynthia Dwork. ACM Press, May 2008, pp. 197–206 (cit. on pp. 74, 76, 77, 81).

[Gre13] Glenn Greenwald. *NSA collecting phone records of millions of Verizon customers daily*. The Guardian. https://www.theguardian.com/world/2013/jun/06/nsa-phone-records-verizon-court-order. 2013 (cit. on p. 94).

[GSW13] Craig Gentry, Amit Sahai, and Brent Waters. "Homomorphic Encryption from Learning with Errors: Conceptually-Simpler, Asymptotically-Faster, Attribute-Based". In: *CRYPTO 2013, Part I*. Ed. by Ran Canetti and Juan A. Garay. Vol. 8042. LNCS. Springer, Heidelberg, Aug. 2013, pp. 75–92. DOI: 10.1007/978-3-642-40041-4_5 (cit. on pp. 27, 33, 38, 53, 72, 74).

[Gt12] Torbjörn Granlund and the GMP development team. *GNU MP: The GNU Multiple Precision Arithmetic Library*. 5.0.5. http://gmplib.org/. 2012 (cit. on p. 106).

[GVW15] Sergey Gorbunov, Vinod Vaikuntanathan, and Daniel Wichs. "Leveled Fully Homomorphic Signatures from Standard Lattices". In: *47th ACM STOC*. Ed. by Rocco A. Servedio and Ronitt Rubinfeld. ACM Press, June 2015, pp. 469–477 (cit. on pp. 38, 91).

[HAO15] Ryo Hiromasa, Masayuki Abe, and Tatsuaki Okamoto. "Packing Messages and Optimizing Bootstrapping in GSW-FHE". In: *PKC 2015*. Ed. by Jonathan Katz. Vol. 9020. LNCS. Springer, Heidelberg, Mar. 2015, pp. 699–715. DOI: 10.1007/978-3-662-46447-2_31 (cit. on pp. 27, 33, 37).

[HDY+12] G. Hinton, L. Deng, D. Yu, G. E. Dahl, A. r. Mohamed, N. Jaitly, A. Senior, V. Vanhoucke, P. Nguyen, T. N. Sainath, and B. Kingsbury. “Deep Neural Networks for Acoustic Modeling in Speech Recognition: The Shared Views of Four Research Groups”. In: *IEEE Signal Processing Magazine* 29.6 (Nov. 2012), pp. 82–97. ISSN: 1053-5888. DOI: 10.1109/MSP.2012.2205597 (cit. on p. 47).

[Hop88] J. J. Hopfield. “Neurocomputing: Foundations of Research”. In: ed. by James A. Anderson and Edward Rosenfeld. Cambridge, MA, USA: MIT Press, 1988. Chap. Neural Networks and Physical Systems with Emergent Collective Computational Abilities, pp. 457–464. ISBN: 0-262-01097-6. URL: http://dl.acm.org/citation.cfm?id=65669.104422 (cit. on p. 43).

[Hor91] Kurt Hornik. “Approximation Capabilities of Multilayer Feedforward Networks”. In: *Neural Netw.* 4.2 (Mar. 1991), pp. 251–257. ISSN: 0893-6080. DOI: 10.1016/0893-6080(91)90009-T. URL: http://dx.doi.org/10.1016/0893-6080(91)90009-T (cit. on p. 47).

[How01] Nick Howgrave-Graham. “Approximate Integer Common Divisors”. In: *Cryptography and Lattices*. Ed. by Joseph H. Silverman. Berlin, Heidelberg: Springer Berlin Heidelberg, 2001, pp. 51–66. ISBN: 978-3-540-44670-5 (cit. on pp. 22, 23).

[HS06] Geoffrey Hinton and Ruslan Salakhutdinov. “Reducing the Dimensionality of Data with Neural Networks”. In: *Science* 313.5786 (2006), pp. 504–507 (cit. on p. 47).

[HS14a] Shai Halevi and Victor Shoup. “Algorithms in HElib”. In: *CRYPTO 2014, Part I*. Ed. by Juan A. Garay and Rosario Gennaro. Vol. 8616. LNCS. Springer, Heidelberg, Aug. 2014, pp. 554–571. DOI: 10.1007/978-3-662-44371-2_31 (cit. on pp. 38, 60).

[HS14b] Shai Halevi and Victor Shoup. *HElib - An implementation of homomorphic encryption*. https://github.com/shaih/HElib. 2014 (cit. on p. 38).

[HS15] Shai Halevi and Victor Shoup. “Bootstrapping for HElib”. In: *EUROCRYPT 2015, Part I*. Ed. by Elisabeth Oswald and Marc Fischlin. Vol. 9056. LNCS. Springer, Heidelberg, Apr. 2015, pp. 641–670. DOI: 10.1007/978-3-662-46800-5_25 (cit. on pp. 38, 60, 73, 91).

[HZRS15] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. “Deep Residual Learning for Image Recognition”. In: *CoRR* abs/1512.03385 (2015). URL: http://arxiv.org/abs/1512.03385 (cit. on p. 52).

[ILL89] Russell Impagliazzo, Leonid A. Levin, and Michael Luby. “Pseudo-random Generation from one-way functions (Extended Abstracts)”. In: *21st ACM STOC*. ACM Press, May 1989, pp. 12–24 (cit. on p. 77).

[IP07] Yuval Ishai and Anat Paskin. “Evaluating Branching Programs on Encrypted Data”. In: *TCC 2007*. Ed. by Salil P. Vadhan. Vol. 4392. LNCS. Springer, Heidelberg, Feb. 2007, pp. 575–594 (cit. on pp. 8, 26, 72, 78, 83).

[Jor89] Michael I. Jordan. “Serial Order: A Parallel, Distributed Processing Approach”. In: *Advances in Connectionist Theory: Speech*. Ed. by Jeffrey L. Elman and David E. Rumelhart. Hillsdale, NJ: Erlbaum, 1989 (cit. on p. 43).

[JRS17] Aayush Jain, Peter M. R. Rasmussen, and Amit Sahai. *Threshold Fully Homomorphic Encryption*. Cryptology ePrint Archive, Report 2017/257. http://eprint.iacr.org/2017/257. 2017 (cit. on p. 37).

[JVC18] Chiraag Juvekar, Vinod Vaikuntanathan, and Anantha Chandrakasan. "Gazelle: A Low Latency Framework for Secure Neural Network Inference". In: *CoRR* abs/1801.05507 (2018). arXiv: 1801.05507. URL: http://arxiv.org/abs/1801.05507 (cit. on p. 51).

[Kag] Kaggle. *Digit recognizer*. https://www.kaggle.com/c/digit-recognizer (cit. on p. 47).

[KGV14] Alhassan Khedr, Glenn Gulak, and Vinod Vaikuntanathan. *SHIELD: Scalable Homomorphic Implementation of Encrypted Data-Classifiers*. Cryptology ePrint Archive, Report 2014/838. http://eprint.iacr.org/2014/838. 2014 (cit. on p. 33).

[Kil92] Joe Kilian. "A Note on Efficient Zero-Knowledge Proofs and Arguments (Extended Abstract)". In: *24th ACM STOC*. ACM Press, May 1992, pp. 723–732 (cit. on p. 10).

[KO97] Eyal Kushilevitz and Rafail Ostrovsky. "Replication is NOT Needed: SINGLE Database, Computationally-Private Information Retrieval". In: *38th FOCS*. IEEE Computer Society Press, Oct. 1997, pp. 364–373 (cit. on p. 97).

[KTX08] Akinori Kawachi, Keisuke Tanaka, and Keita Xagawa. "Concurrently Secure Identification Schemes Based on the Worst-Case Hardness of Lattice Problems". In: *ASIACRYPT 2008*. Ed. by Josef Pieprzyk. Vol. 5350. LNCS. Springer, Heidelberg, Dec. 2008, pp. 372–389 (cit. on p. 57).

[LBBH98] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner. "Gradient-Based Learning Applied to Document Recognition". In: *Proceedings of the IEEE* 86.11 (Nov. 1998), pp. 2278–2324 (cit. on pp. 47, 50, 51, 69).

[LCB98] Y. LeCun, C. Cortes, and C.J.C. Burges. *THE MNIST DATABASE of handwritten digits*. http://yann.lecun.com/exdb/mnist/. 1998 (cit. on p. 51).

[Lep14] Tancrède Lepoint. "Design and Implementation of Lattice-Based Cryptography". Theses. Ecole Normale Supérieure de Paris - ENS Paris, June 2014. URL: https://tel.archives-ouvertes.fr/tel-01069864 (cit. on p. 25).

[Lep16] T. Lepoint. *FV-NFLlib*. GitHub repository. https://github.com/CryptoExperts/FV-NFLlib. 2016 (cit. on p. 38).

[Lip04] Helger Lipmaa. *An Oblivious Transfer Protocol with Log-Squared Communication*. Cryptology ePrint Archive, Report 2004/063. http://eprint.iacr.org/2004/063. 2004 (cit. on p. 97).

[LJLA17] Jian Liu, Mika Juuti, Yao Lu, and N. Asokan. "Oblivious Neural Network Predictions via MiniONN Transformations". In: *ACM CCS 17*. Ed. by Bhavani M. Thuraisingham, David Evans, Tal Malkin, and Dongyan Xu. ACM Press, Oct. 2017, pp. 619–631 (cit. on p. 51).

[LLL82] A. K. Lenstra, H. W. Lenstra, and L. Lovász. "Factoring polynomials with rational coefficients". In: *MATH. ANN* 261 (1982), pp. 515–534 (cit. on p. 22).

[LMSV12] Jake Loftus, Alexander May, Nigel P. Smart, and Frederik Vercauteren. “On CCA-Secure Somewhat Homomorphic Encryption”. In: *SAC 2011*. Ed. by Ali Miri and Serge Vaudenay. Vol. 7118. LNCS. Springer, Heidelberg, Aug. 2012, pp. 55–72 (cit. on p. 91).

[Loh13] Niels Lohmann. *JSON for modern C++*. `https://github.com/nlohmann/json`. 2013 (cit. on p. 108).

[LPR10] Vadim Lyubashevsky, Chris Peikert, and Oded Regev. “On Ideal Lattices and Learning with Errors over Rings”. In: *EUROCRYPT 2010*. Ed. by Henri Gilbert. Vol. 6110. LNCS. Springer, Heidelberg, May 2010, pp. 1–23 (cit. on pp. 21, 53, 60).

[LTV12] Adriana López-Alt, Eran Tromer, and Vinod Vaikuntanathan. “On-the-fly multiparty computation on the cloud via multikey fully homomorphic encryption”. In: *44th ACM STOC*. Ed. by Howard J. Karloff and Toniann Pitassi. ACM Press, May 2012, pp. 1219–1234 (cit. on p. 37).

[MAP+15] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dan Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. *TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems*. Software available from tensorflow.org. 2015. URL: `https://www.tensorflow.org/` (cit. on p. 64).

[MBG+16] Carlos Aguilar Melchor, Joris Barrier, Serge Guelton, Adrien Guinet, Marc-Olivier Killijian, and Tancrède Lepoint. “NFLlib: NTT-Based Fast Lattice Library”. In: *CT-RSA 2016*. Ed. by Kazue Sako. Vol. 9610. LNCS. Springer, Heidelberg, Feb. 2016, pp. 341–356. DOI: `10.1007/978-3-319-29485-8_20` (cit. on p. 38).

[Méa17] Pierrick Méaux. “Hybrid fully homomorphic framework”. Theses. Université de recherche Paris Sciences et Lettres, Dec. 2017. URL: `https://hal.archives-ouvertes.fr/tel-01665358` (cit. on p. 13).

[MHN13] Andrew L. Maas, Awni Y. Hannun, and Andrew Y. Ng. “Rectifier nonlinearities improve neural network acoustic models”. In: *in ICML Workshop on Deep Learning for Audio, Speech and Language Processing*. 2013 (cit. on p. 46).

[Mic10] Daniele Micciancio. “A first glimpse of cryptography’s Holy Grail”. In: *Commun. ACM* 53.3 (2010), p. 96. DOI: `10.1145/1666420.1666445`. URL: `http://doi.acm.org/10.1145/1666420.1666445` (cit. on p. 26).

[Mic94] Silvio Micali. “CS Proofs (Extended Abstracts)”. In: *35th FOCS*. IEEE Computer Society Press, Nov. 1994, pp. 436–453 (cit. on p. 10).

[MOT15a] Alexander Mordvintsev, Christopher Olah, and Mike Tyka. *DeepDream - A code example for visualizing Neural Networks*. https://ai.googleblog.com/2015/07/deepdream-code-example-for-visualizing.html. Google Research. 2015 (cit. on p. 47).

[MOT15b] Alexander Mordvintsev, Christopher Olah, and Mike Tyka. *Inceptionism: Going Deeper into Neural Networks*. https://ai.googleblog.com/2015/06/inceptionism-going-deeper-into-neural.html. Google Research. 2015 (cit. on p. 47).

[MP12] Daniele Micciancio and Chris Peikert. "Trapdoors for Lattices: Simpler, Tighter, Faster, Smaller". In: *EUROCRYPT 2012*. Ed. by David Pointcheval and Thomas Johansson. Vol. 7237. LNCS. Springer, Heidelberg, Apr. 2012, pp. 700–718 (cit. on pp. 34, 76).

[MR04] Daniele Micciancio and Oded Regev. "Worst-Case to Average-Case Reductions Based on Gaussian Measures". In: *45th FOCS*. IEEE Computer Society Press, Oct. 2004, pp. 372–381 (cit. on p. 19).

[MR07] Daniele Micciancio and Oded Regev. "Worst-Case to Average-Case Reductions Based on Gaussian Measures". In: *SIAM J. Comput.* 37.1 (Apr. 2007), pp. 267–302. ISSN: 0097-5397. DOI: 10.1137/S0097539705447360. URL: http://dx.doi.org/10.1137/S0097539705447360 (cit. on p. 77).

[MRSV17] Eleftheria Makri, Dragos Rotaru, Nigel P. Smart, and Frederik Vercauteren. *PICS: Private Image Classification with SVM*. Cryptology ePrint Archive, Report 2017/1190. https://eprint.iacr.org/2017/1190. 2017 (cit. on p. 51).

[MW16] Pratyay Mukherjee and Daniel Wichs. "Two Round Multiparty Computation via Multi-key FHE". In: *EUROCRYPT 2016, Part II*. Ed. by Marc Fischlin and Jean-Sébastien Coron. Vol. 9666. LNCS. Springer, Heidelberg, May 2016, pp. 735–763. DOI: 10.1007/978-3-662-49896-5_26 (cit. on p. 37).

[MZ17] Payman Mohassel and Yupeng Zhang. "SecureML: A System for Scalable Privacy-Preserving Machine Learning". In: *2017 IEEE Symposium on Security and Privacy*. IEEE Computer Society Press, May 2017, pp. 19–38 (cit. on p. 51).

[NH10] Vinod Nair and Geoffrey E. Hinton. "Rectified Linear Units Improve Restricted Boltzmann Machines". In: *Proceedings of the 27th International Conference on International Conference on Machine Learning*. ICML'10. Haifa, Israel: Omnipress, 2010, pp. 807–814. ISBN: 978-1-60558-907-7. URL: http://dl.acm.org/citation.cfm?id=3104322.3104425 (cit. on p. 46).

[NK15] Koji Nuida and Kaoru Kurosawa. "(Batch) Fully Homomorphic Encryption over Integers for Non-Binary Message Spaces". In: *EUROCRYPT 2015, Part I*. Ed. by Elisabeth Oswald and Marc Fischlin. Vol. 9056. LNCS. Springer, Heidelberg, Apr. 2015, pp. 537–555. DOI: 10.1007/978-3-662-46800-5_21 (cit. on p. 39).

[OPP14] Rafail Ostrovsky, Anat Paskin-Cherniavsky, and Beni Paskin-Cherniavsky. "Maliciously Circuit-Private FHE". In: *CRYPTO 2014, Part I*. Ed. by Juan A. Garay and Rosario Gennaro. Vol. 8616. LNCS. Springer, Heidelberg, Aug. 2014, pp. 536–553. DOI: 10.1007/978-3-662-44371-2_30 (cit. on pp. 72, 91).

[org] Various organizations. *Homomorphic Encryption Standardization - An Open Industry / Government / Academic Consortium to Advance Secure Computation.* http://homomorphicencryption.org/introduction/ (cit. on p. 38).

[Ows13] Brian Owsley. *The Fourth Amendment Implications of the Government's Use of Cell Tower Dumps in its Electronic Surveillance.* University of Pennsylvania Journal of Constitutional Law, Vol. 16, 2013. 2013 (cit. on p. 95).

[Pai99] Pascal Paillier. "Public-Key Cryptosystems Based on Composite Degree Residuosity Classes". In: *EUROCRYPT'99.* Ed. by Jacques Stern. Vol. 1592. LNCS. Springer, Heidelberg, May 1999, pp. 223–238 (cit. on pp. 5, 26).

[Pas16] Alain Passelègue. "Algebraic Frameworks for Pseudorandom Functions". Theses. PSL Research University, Dec. 2016. URL: https://hal.inria.fr/tel-01422093 (cit. on p. 13).

[Pei10] Chris Peikert. "An Efficient and Parallel Gaussian Sampler for Lattices". In: *CRYPTO 2010.* Ed. by Tal Rabin. Vol. 6223. LNCS. Springer, Heidelberg, Aug. 2010, pp. 80–97 (cit. on p. 77).

[Pei15a] Chris Peikert. *A Decade of Lattice Cryptography.* Cryptology ePrint Archive, Report 2015/939. http://eprint.iacr.org/2015/939. 2015 (cit. on p. 13).

[Pei15b] Chris Peikert. *What does GCHQ's "cautionary tale" mean for lattice cryptography?* Blog post. https://web.eecs.umich.edu/~cpeikert/soliloquy.html. 2015 (cit. on p. 21).

[PHGR13] Bryan Parno, Jon Howell, Craig Gentry, and Mariana Raykova. "Pinocchio: Nearly Practical Verifiable Computation". In: *2013 IEEE Symposium on Security and Privacy.* IEEE Computer Society Press, May 2013, pp. 238–252 (cit. on p. 10).

[PW08] Chris Peikert and Brent Waters. "Lossy trapdoor functions and their applications". In: *40th ACM STOC.* Ed. by Richard E. Ladner and Cynthia Dwork. ACM Press, May 2008, pp. 187–196 (cit. on p. 57).

[RAD78] R. L. Rivest, L. Adleman, and M. L. Dertouzos. "On Data Banks and Privacy Homomorphisms". In: *Foundations of Secure Computation, Academia Press* (1978), pp. 169–179 (cit. on pp. 6, 26).

[Rav17] Mirco Ravanelli. "Deep Learning for Distant Speech Recognition". In: *CoRR* abs/1712.06086 (2017). arXiv: 1712.06086. URL: http://arxiv.org/abs/1712.06086 (cit. on p. 47).

[Reg05] Oded Regev. "On lattices, learning with errors, random linear codes, and cryptography". In: *37th ACM STOC.* Ed. by Harold N. Gabow and Ronald Fagin. ACM Press, May 2005, pp. 84–93 (cit. on pp. 20, 22, 30, 52, 74, 77, 81).

[Ros57] F. Rosenblatt. *The Perceptron, a Perceiving and Recognizing Automaton Project Para.* Report: Cornell Aeronautical Laboratory. Cornell Aeronautical Laboratory, 1957. URL: https://books.google.fr/books?id=P%5C_XGPgAACAAJ (cit. on p. 46).

[RRK17] Bita Darvish Rouhani, M. Sadegh Riazi, and Farinaz Koushanfar. *DeepSecure: Scalable Provably-Secure Deep Learning.* Cryptology ePrint Archive, Report 2017/502. http://eprint.iacr.org/2017/502. 2017 (cit. on p. 51).

[RSA78] Ronald L. Rivest, Adi Shamir, and Leonard M. Adleman. “A Method for Obtaining Digital Signature and Public-Key Cryptosystems”. In: *Communications of the Association for Computing Machinery* 21.2 (1978), pp. 120–126 (cit. on pp. 3, 5).

[Sch87] C.P. Schnorr. “A hierarchy of polynomial time lattice basis reduction algorithms”. In: *Theoretical Computer Science* 53.2 (1987), pp. 201–224. ISSN: 0304-3975. DOI: https://doi.org/10.1016/0304-3975(87)90064-8. URL: http://www.sciencedirect.com/science/article/pii/0304397587900648 (cit. on p. 22).

[SS10] Damien Stehlé and Ron Steinfeld. “Faster Fully Homomorphic Encryption”. In: *ASIACRYPT 2010*. Ed. by Masayuki Abe. Vol. 6477. LNCS. Springer, Heidelberg, Dec. 2010, pp. 377–394 (cit. on pp. 27, 72).

[SSTX09] Damien Stehlé, Ron Steinfeld, Keisuke Tanaka, and Keita Xagawa. “Efficient Public Key Encryption Based on Ideal Lattices”. In: *ASIACRYPT 2009*. Ed. by Mitsuru Matsui. Vol. 5912. LNCS. Springer, Heidelberg, Dec. 2009, pp. 617–635 (cit. on p. 21).

[Ste98] Julien P. Stern. “A New Efficient All-Or-Nothing Disclosure of Secrets Protocol”. In: *ASIACRYPT’98*. Ed. by Kazuo Ohta and Dingyi Pei. Vol. 1514. LNCS. Springer, Heidelberg, Oct. 1998, pp. 357–371 (cit. on p. 97).

[SV10] Nigel P. Smart and Frederik Vercauteren. “Fully Homomorphic Encryption with Relatively Small Key and Ciphertext Sizes”. In: *PKC 2010*. Ed. by Phong Q. Nguyen and David Pointcheval. Vol. 6056. LNCS. Springer, Heidelberg, May 2010, pp. 420–443 (cit. on pp. 27, 72).

[SYY99] Tomas Sander, Adam Young, and Moti Yung. “Non-Interactive CryptoComputing For NC1”. In: *40th FOCS*. IEEE Computer Society Press, Oct. 1999, pp. 554–567 (cit. on pp. 8, 72).

[Uni05] European Union. *Prüm Convention*. http://register.consilium.europa.eu/doc/srv?l=EN&f=ST%2010900%202005%20INIT. 2005 (cit. on p. 96).

[Uni12] Supreme Court of the United States. *United States v. Jones*. https://www.supremecourt.gov/opinions/11pdf/10-1259.pdf. 2012 (cit. on p. 95).

[Yao86] Andrew Chi-Chih Yao. “How to Generate and Exchange Secrets (Extended Abstract)”. In: *27th FOCS*. IEEE Computer Society Press, Oct. 1986, pp. 162–167 (cit. on p. 51).

[ZK16] Sergey Zagoruyko and Nikos Komodakis. “Wide Residual Networks”. In: *CoRR* abs/1605.07146 (2016). URL: http://arxiv.org/abs/1605.07146 (cit. on p. 52).

[ZPB+17] Ying Zhang, Mohammad Pezeshki, Philemon Brakel, Saizheng Zhang, César Laurent, Yoshua Bengio, and Aaron C. Courville. “Towards End-to-End Speech Recognition with Deep Convolutional Neural Networks”. In: *CoRR* abs/1701.02720 (2017). arXiv: 1701.02720. URL: http://arxiv.org/abs/1701.02720 (cit. on p. 47).

[ZRM+13] M.D. Zeiler, M. Ranzato, R. Monga, M. Mao, K. Yang, Q.V. Le, P. Nguyen, A. Senior, V. Vanhoucke, J. Dean, and G.E. Hinton. “On Rectified Linear Units For Speech Processing”. In: *38th International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. Vancouver, 2013 (cit. on p. 46).

[ZYC16] Qingchen Zhang, Laurence T. Yang, and Zhikui Chen. “Privacy Preserving Deep Computation Model on Cloud for Big Data Feature Learning”. In: *IEEE Transactions on Computers* 65.5 (2016), pp. 1351–1362. ISSN: 0018-9340. DOI: doi.ieeecomputersociety.org/10.1109/TC.2015.2470255 (cit. on p. 52).

[ZYL+17] TanPing ZHOU, XiaoYuan YANG, LongFei LIU, Wei ZHANG, and Yi-Tao DING. *Faster Bootstrapping with Multiple Addends*. Cryptology ePrint Archive, Report 2017/735. http://eprint.iacr.org/2017/735. 2017 (cit. on pp. 30, 61, 62).

Résumé

Le chiffrement totalement homomorphe permet d'effectuer des calculs sur des données chiffrées sans fuite d'information sur celles-ci. Pour résumer, un utilisateur peut chiffrer des données, tandis qu'un serveur, qui n'a pas accès à la clé de déchiffrement, peut appliquer à l'aveugle un algorithme sur ces entrées. Le résultat final est lui aussi chiffré, et il ne peut être lu que par l'utilisateur qui possède la clé secrète.

Dans cette thèse, nous présentons des nouvelles techniques et constructions pour le chiffrement totalement homomorphe qui sont motivées par des applications en apprentissage automatique, en portant une attention particulière au problème de l'inférence homomorphe, c'est-à-dire l'évaluation de modèles cognitifs déjà entrainé sur des données chiffrées.

Premièrement, nous proposons un nouveau schéma de chiffrement totalement homomorphe adapté à l'évaluation de réseaux de neurones artificiels sur des données chiffrées. Notre schéma atteint une complexité qui est essentiellement indépendante du nombre de couches dans le réseau, alors que l'efficacité des schéma proposés précédemment dépend fortement de la topologie du réseau.

Ensuite, nous présentons une nouvelle technique pour préserver la confidentialité du circuit pour le chiffrement totalement homomorphe. Ceci permet de cacher l'algorithme qui a été exécuté sur les données chiffrées, comme nécessaire pour protéger les modèles propriétaires d'apprentissage automatique. Notre mécanisme rajoute un coût supplémentaire très faible pour un niveau de sécurité égal. Ensemble, ces résultats renforcent les fondations du chiffrement totalement homomorphe efficace pour l'apprentissage automatique, et représentent un pas en avant vers l'apprentissage profond pratique préservant la confidentialité.

Enfin, nous présentons et implémentons un protocole basé sur le chiffrement totalement homomorphe pour le problème de recherche d'information confidentielle, c'est-à-dire un scénario où un utilisateur envoie une requête à une base de donnée tenue par un serveur sans révéler cette requête.

Mots Clés

chiffrement à base de réseaux euclidiens, chiffrement totalement homomorphe, informatique en nuage, apprentissage automatique, confidentialité du circuit, recherche d'information confidentielle.

Abstract

Fully homomorphic encryption enables computation on encrypted data without leaking any information about the underlying data. In short, a party can encrypt some input data, while another party, that does not have access to the decryption key, can blindly perform some computation on this encrypted input. The final result is also encrypted, and it can be recovered only by the party that possesses the secret key.

In this thesis, we present new techniques/designs for FHE that are motivated by applications to machine learning, with a particular attention to the problem of homomorphic inference, i.e., the evaluation of already trained cognitive models on encrypted data.

First, we propose a novel FHE scheme that is tailored to evaluating neural networks on encrypted inputs. Our scheme achieves complexity that is essentially independent of the number of layers in the network, whereas the efficiency of previously proposed schemes strongly depends on the topology of the network.

Second, we present a new technique for achieving circuit privacy for FHE. This allows us to hide the computation that is performed on the encrypted data, as is necessary to protect proprietary machine learning algorithms. Our mechanism incurs very small computational overhead while keeping the same security parameters.

Together, these results strengthen the foundations of efficient FHE for machine learning, and pave the way towards practical privacy-preserving deep learning.

Finally, we present and implement a protocol based on homomorphic encryption for the problem of private information retrieval, i.e., the scenario where a party wants to query a database held by another party without revealing the query itself.

Keywords

lattice based cryptography, fully homomorphic encryption, cloud computing, machine learning, circuit privacy, private information retrieval.