Fully homomorphic encryption in JCrypTool

Ramaekers, C.F.W.

Award date:
2011

Fully Homomorphic Encryption in JCrypTool

Coen Ramaekers
c.f.w.ramaekers@student.tue.nl

August 4, 2011

Abstract

This thesis provides an overview of the recent achievements on Fully Homomorphic Encyrption (FHE) schemes and also provides a tool demonstrating that FHE allows computations with ciphertexts while preserving the ability to correctly decrypt the result.

The general construction as presented by Gentry is explained, along with several schemes based on this construction. These schemes range from the first theoretic scheme to the first practically implementable scheme. The main ideas of bootstrapping and squashing needed to obtain a Fully Homomorphic scheme are also described.

A major part of the final project was the implementation of the first practical fully homomorphic scheme as of Gentry and Halevi in the software package JCrypTool. The parameters are chosen for educational purposes: they do not offer sufficient security for real-world use, but they give a sufficiently fast implementation that runs on average PCs. This implementation is the first implementation with which users can actually see that calculations can be performed on ciphertexts. The security was sacrificed to reach better performance.

Contents

List of Figures

1 Introduction

The idea of homomorphic encryption is known for quite some time now. Shortly after the introduction of RSA in 1978 [RSA78], which contains a multiplicative homomorphism, the notion of "privacy homomorphisms" was introduced [RAD78]. There are two possible homomorphisms, namely the multiplicative and additive homomorphism. This implies that there exists a group structure, that is preserved by the encryption and decryption. An easy to grasp definition is as follows (we will formalize it later on), where π and ψ denote the plaintext and ciphertext, respectively. Suppose we have a cryptosystem where $\text{Enc}(\pi)$ denotes an encryption and $\text{Dec}(\psi)$ denotes the corresponding decryption. In a homomorphic encryption scheme this then yields $(\pi_1 \times \pi_2) = \text{Dec}(\text{Enc}(\pi_1) \otimes \text{Enc}(\pi_2))$ and $(\pi_1 + \pi_2) = \text{Dec}(\text{Enc}(\pi_1) \oplus \text{Enc}(\pi_2))$, where $\otimes$ and $\oplus$ denote certain operations on the ciphertext.
If an encryption scheme allows either one of these operations, but only one, it is called a partially homomorphic encryption scheme. These schemes generally allow an indefinite amount of operations to be performed. In section 2.5 some examples will be given.
The multiplicative homomorphism in RSA raised some natural questions; does this imply extra cryptographic properties? Does this somehow compromise the security of such a scheme? Do there exist schemes which contain a ring structure instead of only a group structure, i.e. can we construct a scheme which allows both addition and multiplication on the ciphertext?

1.1 Applications of Partially and Fully Homomorphic Encryption Schemes

Several interesting applications using the multiplicative homomorphism of RSA and the additive homomorphism of Paillier have arisen over time, some of which will be briefly explained in section 2.5.3. It soon became clear that the group structure in RSA does indeed compromise its security; it allows an efficient chosen ciphertext attack. For this reason it was required in the RSA Encryption Standard that the scheme uses a random padding of messages before encrypting [PKC91].
But the question if it is possible to construct a fully homomorphic scheme, i.e. a scheme with a ring structure that is preserved by encryption and decryption, remained an open question for years. The problem is interesting because of the numerous possibilities. Cryptographers have come up with various applications of a fully homomorphic scheme. Several interesting applications include untrusted storage, "outsourcing" computation and cloud computing. It goes without saying that one would like private data stored on the web to be encrypted, but as the amount of data increases it becomes harder to manage an untrusted storage. Fully homomorphic encryption would allow a user to perform for instance search queries on the encrypted data to find the right files, without needing to decrypt. Under outsourcing computing we understand distributing computationally intensive operations among large amounts of untrusted computers, in which fully homomorphic

encryption plays a vital role [GGP10, CKV10]. Perhaps the most popular application is secure computations in the cloud. It would allow companies to buy computation time to outsource their computationally expensive tasks to an untrusted cloud, but with the use of fully homomorphic encryption, data, computation and answer all remain encrypted.

1.2 Somewhat and Fully Homomorphic Encryption Schemes

In 2009, Gentry finally made the breakthrough and presented a first plausible fully homomorphic encryption scheme and described a general construction to create fully homomorphic encryption schemes out of schemes which allow both operations, but only a limited amount of them [Gen09a]. He dubs these schemes "somewhat" homomorphic schemes. After the breakthrough, a sequence of schemes appeared, which ultimately led to an implementable scheme. The part of the sequence that is covered in this thesis is as follows.

Gentry's Construction
In [Gen09a, chap. 2-4, pp. 27-56] Gentry shows that the key idea to the construction is to have a somewhat homomorphic encryption scheme with a noise parameter, i.e. encryption adds some random noise to the ciphertext. Such a scheme would have algorithms that allow ciphertexts to be multiplied and added at the expense of an increase in noise. Decryption only works if the noise is less then some threshold. Clearly this scheme is not yet fully homomorphic, since if enough operations are performed, the noise grows larger than the threshold resulting in incorrect decryption.
If in addition the scheme would have an algorithm which reduces this noise, then we would be back in business. Gentry's main contribution is the idea of an algorithm that can reduce this noise. This is roughly possible when the encryption scheme is able to evaluate its own decryption algorithm homomorphically. In that case the algorithm decrypts the ciphertexts and re-encrypts it homomorphically, so that the amount of noise is reduced. More details are given in section 3.

Gentry's Scheme
Together with this general construction, Gentry also described a scheme which is proven to be fully homomorphic. This scheme is based on lattices and builds on some well known and thoroughly analyzed hard problems. The scheme is introduced with some detail in section 4.1. Unfortunately the scheme is not yet practical, but improvements have already been made.

The Smart-Vercauteren Variant
Smart and Vercauteren presented in [SV10] a variant of the original fully homomorphic encryption scheme by Gentry. They use a special type of lattice with the property that the basis can be represented by only two integers. From that they show that the ciphertext can be represented by a single integer. This drastically reduces the public key and message size in comparison to the original scheme. See section 4.2.

The Gentry-Halevi Variant
The variant by Smart and Vercauteren requires the lattice to have prime determinant. This is hard to achieve for lattices with a high dimension. Gentry and Halevi present a variant in which this requirement is dropped [GH11]. Thanks to this and to some other tweaks to the scheme, they managed to implement the first functional fully homomorphic encryption scheme. More details in sections 4.3 and 5.

Fully Homomorphic Encryption over the Integers
After the appearance of the fully homomorphic schemes mentioned before, van Dijk, Gentry, Halevi and Vaikuntanathan achieved the goal of creating a fully homomorphic encryption scheme using only elementary modular arithmetic [vDGHV10]. As usual, first a somewhat homomorphic encryption scheme is created and after some optimizing, Gentry's general construction is used to turn it into a fully homomorphic one. This will be described in more detail in section 4.4.

1.3 Goals

The goals for this project are to give a summary of the core developments on fully homomorphic encryption and to create a practical implementation of a fully homomorphic encryption scheme as a plugin in JCrypTool, which demonstrates the additive and multiplicative homomorphisms on integers so that it is understandable by laymen.

1.4 JCrypTool

As mentioned, some of the homomorphic encryption schemes as described in this thesis are implemented as a plug-in for JCrypTool. JCrypTool is a successor of the CrypTool package [Cry11] and is a modular cryptography e-learning program. The CrypTool project started in 1998 as a private learning tool, but was made public in 2000. As of 2003 it is an open source project. CrypTool was created to teach people the various aspects of cryptography and allow them to try out various schemes, reaching from the ancient Caeser cipher, to elliptic curve cryptography. Additionally various cryptanalysis techniques are implemented, so that users can understand that nothing is ever 100% secure.
JCrypTool is programmed in Java with the main purpose of being platform independent. The software package is plugin-based, i.e. everyone can write a plugin as extension to use in JCrypTool. A large advantage of this approach is that one can focus solely on the cryptographic algorithm which is to be implemented, instead of also on creating some sort of user interface. Writing a plugin for JCrypTool, or for the other CrypTool packages, has the great advantage that once work is accepted into the program it will be automatically included in future versions and enduringly maintained as teaching tool.

2 Foundation

In this section the necessary foundations will be given. First, we will define the notation that is used throughout this thesis. Next, circuits and gates are introduced. After this, general lattices are discussed, along with several known hard problems. Then, ideal lattices are explained and finally, some partially homomorphic encryption schemes will be explained, along with some of the possible applications.
But before we jump into all this, we will recap on what an encryption scheme actually is. The encryption schemes that we consider in this thesis are public key encryption schemes. These schemes consist of three algorithms, key generation, encryption and decryption. The key generation algorithm creates both a public and a private key. The public key is needed to perform encryption, the private key for decryption.
For such a scheme to make sense, decryption must be the inverse of encryption. Furthermore, with the knowledge of the needed key, both encryption and decryption should be easy to compute. Without the knowledge of the private key, it should be hard to perform decryption. The hardness of decryption without knowledge of the key is related to a security parameter.
As mentioned in the introduction, these schemes can also have more structure. If a scheme has a group structure, it is called partially homomorphic, and if a scheme has a ring structure, it is called fully homomorphic. We will present some examples of schemes with both of these structures.

2.1 Notation

Throughout this thesis, we will use the following notation.

M bold capital letters represent matrices.

v bold small letters denote vectors.

n**,** M scalars will be represented by normal faced letters, either capital or small.

$\mathcal{E}$ calligraphic letters will denote algorithms, encryption schemes, or collections of some kind; which will be clear from the context.

λ always denotes the security parameter of the scheme. This parameter is defined such that it takes time at least 2^λ to break the scheme.

$\lceil z \rceil$**,** $\lfloor z \rfloor$ **and** $\lfloor z \rceil$ respectively denote rounding up, down or to the nearest integer.

$[z]_d$ **and** $\langle z \rangle_d$ both denote $z \mod d$, but the former maps to the interval $[-d/2, d/2)$ and the latter to the interval $[0, d)$.

$q_p(z)$ **and** $r_p(z)$ denote the quotient and the remainder of z with respect to p, respectively. For the remainder, we take $r_p(z) \in (-p/2, p/2]$.

$\mathcal{B}_{n,d}(r)$ denotes the ball with radius r around the origin, with respect to the norm n. The dimension of the space is given by d. If n or d is omitted, it is clear what norm or dimension is used.

$\mathbf{e}_i$ denotes the vector with 1 at position i and 0 at all other positions.

π **and** ψ respectively denote plaintexts and ciphertexts.

Multiplication will be denoted by $\times$ in the case of ring multiplication. When denote by $\cdot$ or nothing, it denotes multiplications of polynomials or integers.

2.2 Circuits

The construction of fully homomorphic encryption schemes is based on circuits. A circuit generally consists of a set of gates, which operate on the inputs to give a certain output. Without further specification, one can consider a circuit as a black box, which evaluates a function on the inputs. Every computable function can be represented by a circuit.
A gate in a circuit is a part which performs a single operation. It has one or more inputs and a single output. Basically, there are three different gates which are commonly used to construct circuits, namely the AND, OR and NOT gate. To specify the functionality of a gate, one often uses a truth table. In such a table, all possible inputs are given along with the output. For the AND, OR and NOT gate, these tables are given in Table 1.

a	b	a AND b	a OR b	NOT a
0	0	0	0	1
0	1	0	1	1
1	0	0	1	0
1	1	1	1	0

Table 1: Truth table for AND, OR and NOT gate.

Besides these gates, we also consider the NAND and XOR gates. The NAND gate is named after NOT-AND, its output is the negation of the AND gate. XOR stands for eXclusive OR, it only outputs a 1 whenever exactly one of its inputs is 1. The truth tables are given in Table 2.

a	b	a NAND b	a XOR b
0	0	1	0
0	1	1	1
1	0	1	1
1	1	0	0

Table 2: Truth table for NAND and XOR gate.

It is a well known fact that one can construct any circuit from only NAND gates [Mar05, p. 76]. So if we can construct a NAND gate, we can evaluate any circuit and thus compute any function. As an example, in figure 1 we construct an AND gate from two NAND gates. The corresponding truth table is given in table 3.

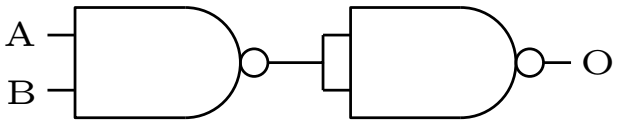


Figure 1: AND gate from two NAND gates.

A	B	A NAND B	(A NAND B) NAND (A NAND B)
0	0	1	0
0	1	1	0
1	0	1	0
1	1	0	1

Table 3: Truth table for AND gate from two NAND gates.

The complexity of a circuit is expressed as its depth. One can view a circuit as a directed acyclic graph, in which each node represents a gate. The circuit depth then is defined as the maximum length of a path from any input to any output. Each node in a directed acyclic graph has an in-degree and an out-degree. These terms are analogous to the terms fan-in and fan-out used for circuits.

2.3 Lattices

Some of the fully homomorphic encryption schemes presented are based on lattices. A lattice can be seen as discrete subgroup of $\mathbb{R}^d$. It consists of all integer combinations of n linearly independent vectors, called the basis:

$$\mathcal{L}(\mathbf{b}_1, \ldots, \mathbf{b}_n) = \left\{ \sum_{i=1}^{n} x_i \mathbf{b}_i : x_i \in \mathbb{Z} \right\}.$$

In other words, $\mathbf{b}_1, \ldots, \mathbf{b}_n$ generate the lattice. The rank of a lattice is the size of a maximal set of independent vectors in the lattice. If this rank equals the dimension of the vector space, we say the lattice has full rank. The basis can also be represented as a matrix $\mathbf{B} \in \mathbb{R}^{d \times n}$, in which case we take the basis vectors as the columns of this matrix. If we use the matrix notation, we can represent the lattice as $\mathcal{L}(\mathbf{B}) = \{\mathbf{B}x : x \in \mathbb{Z}^n\}$. From now on we will only consider lattices with full rank.
From this representation, it is easy to see that different bases can generate the same lattice.

In fact, every lattice has infinitely many bases. See Figure 2 for an example. In the matrix representation, we can multiply one basis with a unimodular matrix (an integer square matrix with determinant ± 1) to obtain another basis for the same lattice.

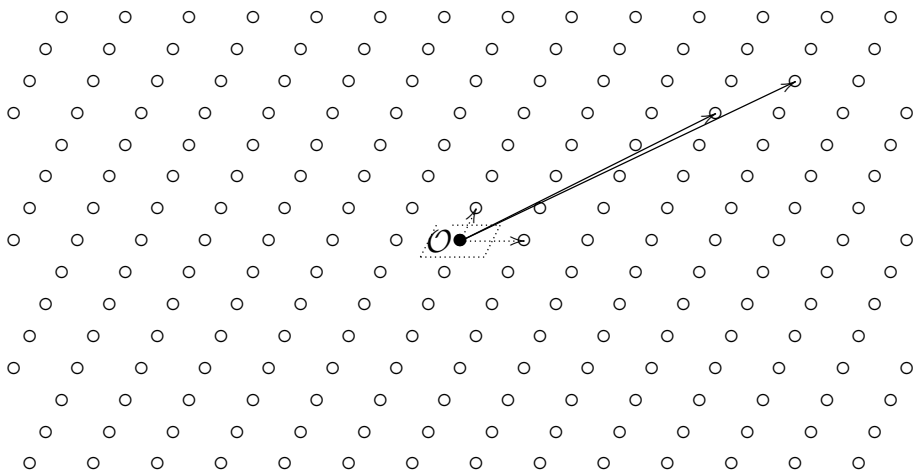


Figure 2: Lattice with two different bases, one with parallelepiped

Even though a lattice has infinitely many bases, finding a particular basis is a hard problem. This is the fact that an encryption scheme using lattices builds on. The same lattice can have a public and private basis, both with different properties, so that encryption and decryption can be performed. It then is a hard problem to compute the private basis given only the public basis.

The determinant of a lattice is defined as the absolute value of the basis matrix. Since all basis matrices have the same determinant up to the sign, this value is well defined for a lattice. We state $\det(\mathcal{L}(\mathbf{B})) = |\det(\mathbf{B})|$.

Using the matrix representation, one can take the Hermite Normal Form (HNF) of this matrix. The HNF of any (square) matrix is a matrix $\mathbf{B}$ where $b_{i,j} = 0$ for $i < j$, $b_{j,j} > 0$ for all j and for all $i > j$ we have $b_{i,j} \in [-b_{j,j}/2, b_{j,j}/2)$. This form can be computed efficiently from any basis via Gaussian elimination [GH11, p. 4].

Given a basis in vector form, we can define a half-open parallelepiped $\mathcal{P}(\mathbf{b}_1, \ldots, \mathbf{b}_n) = \{\sum_{i=1}^{n} x_i \mathbf{b}_i : x \in [-1/2, 1/2)\}$, see Figure 2. The volume of this parallelepiped is equal to the determinant of the lattice [GH11, p. 4]. For any vector $\mathbf{t} \in \mathbb{R}^n$, define $\mathbf{t} \mod \mathcal{L}(\mathbf{B})$ to be the unique vector $\mathbf{t}' \in \mathcal{P}(\mathbf{B})$ such that $\mathbf{t} - \mathbf{t}' \in \mathcal{L}(\mathbf{B})$.

Though this volume is the same for each basis, having an infinite amount of possible bases yields problems which are conjectured to be hard. Some of these are the following [MR09, p. 5]:

- Shortest Vector Problem (SVP): Given a lattice basis $\mathbf{B}$ and a norm $||\cdot||$, find the shortest nonzero vector in $\mathcal{L}(\mathbf{B})$.
- Closest Vector Problem (CVP): Given a lattice basis $\mathbf{B}$, a target vector $\mathbf{t}$ and a norm

$||\cdot||$, not necessarily in the lattice, find the lattice point $\mathbf{v} \in \mathcal{L}(\mathbf{B})$ which is closest to $\mathbf{t}$.

- Shortest Independent Vector Problem (SIVP): Given a lattice basis $\mathbf{B} \in \mathbb{Z}^{n \times n}$ and a norm $||\cdot||$, find n linearly independent lattice vectors $\mathbf{S} = \{\mathbf{s}_1, \ldots, \mathbf{s}_n\}$ with $\mathbf{s}_i \in \mathcal{L}(\mathbf{B})$ for all i, minimizing the quantity $||\mathbf{S}|| = \max_i ||\mathbf{s}_i||$.

Instead of solving these problems, often one approximates them. In this case, the approximation factor usually is stated as well. Several approximation algorithms for lattice problems are known. The best known is LLL [LLL82], which runs in polynomial time and approximates SVP with an approximation factor of $2^{O(n)}$, with n the lattice dimension. All known algorithms either run in exponential time or have an exponential approximation factor [MR09, p. 2].

A lattice $\mathcal{L}$ also has a dual. The dual lattice is denoted by $\mathcal{L}^*$ and is defined as $\mathcal{L}^* = \{\mathbf{u} : \forall \mathbf{v} \in \mathcal{L}, \langle \mathbf{u}, \mathbf{v} \rangle \in \mathbb{Z}\}$ (here $\langle \mathbf{u}, \mathbf{v} \rangle$ denotes the scalar product). For a lattice and its dual, it holds that $\det(\mathcal{L}) \cdot \det(\mathcal{L}^*) = 1$. Since we only consider full rank lattices, we have that $(\mathbf{B}^{-1})^T$ is a basis for $\mathcal{L}^*$ if $\mathbf{B}$ is a basis for $\mathcal{L}$.

2.4 Ideal Lattices

Ideal lattices are lattices with some additional structure. Lattices have a group structure and as the term suggests, ideal lattices have an ideal structure. Let $f(x) \in \mathbb{Z}[x]$ be a polynomial of degree n. Let $R = \mathbb{Z}[x]/(f)$ denote the ring of all polynomials modulo f, and define the isomorphism $\Phi : \mathbb{Z}^n \to R$ as the mapping $(v_0, v_1, \ldots, v_{n-1}) \mapsto v_0 + v_1 x + v_2 x^2 + \cdots + v_{n-1} x^{n-1} + f(x)\mathbb{Z}[x]$. Then we have the following definition for an ideal lattice [DL07, p.3].

Definition 2.4.1. *Let $\mathcal{L}$ be a lattice in $\mathbb{Z}^n$. If there exists a monic polynomial $f \in \mathbb{Z}[x]$ of degree n, such that $\Phi(\mathcal{L})$ is an ideal in R, then $\mathcal{L}$ is an ideal lattice.*

As mentioned in the definition, the polynomial should be monic and of degree n. If in addition the polynomial f is irreducible, we find that for a nonzero polynomial $v \in \mathbb{Z}[x]/(f)$, the coefficient vectors of $v, vx, \ldots, vx^{n-1}$ are linearly independent, and thus span an ideal lattice of rank n in $\mathbb{Z}^n$. Ideals that are generated by a single element v are called principal ideals. This need not be the case in general.
One can also consider the fractional ideal I^{-1}, related to an ideal I. It is defined as the elements in R, for which it holds that the product with any element in the ideal I is again in R: $I^{-1} = \{\mathbf{u} \in R : \forall \mathbf{v} \in I, \mathbf{u} \times \mathbf{v} \in R\}$. This definition has a natural extension to the ring $\mathbb{Q}[x]/(f(x))$. In this case, the fractional ideal has the same properties as a normal ideal, except that it is not necessarily a subset of R.
For a principal ideal $(\mathbf{v})$, we also find that the fractional ideal $(\mathbf{v})^{-1}$ is generated by $1/\mathbf{v}$, which yields that it holds that the determinant of $(\mathbf{v})$ is the inverse of the determinant of $(\mathbf{v})^{-1}$. We assume here that we have $f(x)$ irreducible. Then there is some relation between

a principal ideal J, its dual J^* and its inverse J^{-1}. Suppose J is generated by $(\mathbf{v})$, and denote the rotation basis by $\mathbf{B}_J$. Now for the inverse J^{-1}, being generated by $(1/\mathbf{v})$, we find as basis the rotation basis of $1/\mathbf{v} \in \mathbb{Q}[x]/(f(x))$, which are exactly the columns of $\mathbf{B}_J^{-1}$. Recall that the dual J^* has basis $(\mathbf{B}_J^{-1})^T$.

2.5 Partially Homomorphic Encryption Schemes

There exist several partially homomorphic encryption schemes, some of which are widely used today. Two of these schemes are RSA and Paillier. Both schemes are described below and it is shown that these schemes are partially homomorphic.

2.5.1 RSA

The RSA cryptosystem as originally introduced in [RSA78] works as follows. We have public key (e, n) and private key (d, n), which are all positive integers. Furthermore, n is the product of two distinct random primes, $n = p \cdot q$. The bitlength of these primes is determined by the security parameter λ. Furthermore, d is the multiplicative inverse of $e \bmod \phi(n)$, i.e. $d \cdot e \equiv 1 \bmod \phi(n)$, where $\phi(n) = (p-1)(q-1)$. Here ϕ denotes Euler's totient function.
To encrypt a message $\pi \in \{0, \ldots, n-1\}$ one simply computes $\psi = \mathrm{Enc}(\pi) = \pi^e \mod n$, and to decrypt this message compute $\pi' = \mathrm{Dec}(\psi) = \psi^d \mod n$. For the scheme to work, we thus require $\pi \equiv (\pi^e)^d \mod n$. This can be seen as follows;

$$\begin{aligned}(\pi^e)^d \equiv& \pi^{d \cdot e} \mod n \\ \equiv& \pi^{1+k\phi(n)} \mod n \\ \equiv& \pi \cdot (\pi^{\phi(n)})^k \mod n \\ \equiv& \pi \cdot 1^k \mod n.\end{aligned}$$

Here the equivalence $\pi^{\phi(n)} = 1$ follows from Euler's theorem. To be precise this only works if π and n are coprime, but this happens with overwhelming probability (and otherwise the scheme is broken).
To see that this system is homomorphic, consider two messages π_1 and π_2. We then find that

$$\begin{aligned}\mathrm{Dec}(\mathrm{Enc}(\pi_1)\mathrm{Enc}(\pi_2) \mod n) \equiv& \mathrm{Dec}((\pi_1^e \mod n)(\pi_2^e \mod n) \mod n) \\ \equiv& \mathrm{Dec}((\pi_1 \cdot \pi_2)^e \mod n) \\ \equiv& \mathrm{Dec}(\mathrm{Enc}(\pi_1 \cdot \pi_2)).\end{aligned} \tag{1}$$

Note that the multiplications are all taken modulo n. Also note that we could show this equivalence using only encryption, but we include decryption for consistency. Because of this, the ciphertext remains within the same range, even if it is multiplied with another ciphertext, and the decryption remains possible and correct. As a result of this, the multiplication of the plaintexts is also the result modulo n.

2.5.2 Paillier

Paillier introduced the Paillier Cryptosystem in [Pai99], which we will briefly explain here. It is based on n-th residues in $\mathbb{Z}_{n^2}^*$. A number $x \in \mathbb{Z}_{n^2}^*$ is defined to be an n-th residue modulo n^2 if there exists a $y \in \mathbb{Z}_{n^2}^*$, for which it holds that $x = y^n \mod n^2$.
The cryptosystem uses n as the product of two primes, p and q. The bitlength of these primes is defined to be the security parameter λ. We use Carmichael's function, $\Lambda(n^2) = \text{lcm}(p-1, q-1)$. For convenience, we will denote this value by Λ. Now define $\mathcal{G} = \{u \in \mathbb{Z}_{n^2}^* | \text{ord}(u) = kn, k \in \{1, \ldots, \Lambda\}\}$, and select a number $g \in \mathcal{G}$. The public key of an instantiation of Paillier consists of n and g. Now we will first introduce the encryption function, the private key and decryption will be defined later.
Encryption is given by the function $\text{Enc}_g : \mathbb{Z}_n \times \mathbb{Z}_n^* \to \mathbb{Z}_{n^2}^*$, which is defined as $(\pi, r) \mapsto g^\pi r^n \mod n^2$. Here $\pi \in \mathbb{Z}_n$ is the message and the masking factor $r \in \mathbb{Z}_n^*$ is chosen at random. We will show that this encryption function is bijective, which will aid in the definition of the decryption function.

Theorem 2.5.1. *The function* Enc_g *as defined above is bijective for any* $g \in \mathcal{G}$.

Proof. First, we note that $|\mathbb{Z}_n \times \mathbb{Z}_n^*| = n\varphi(n) = \varphi(n^2) = |\mathbb{Z}_{n^2}^*|$, where $\varphi(n)$ is Euler's totient function. The second equality follows from the property $\varphi(mn) = \varphi(m)\varphi(n)d/\varphi(d)$, with $d = \gcd(m, n)$. Now it suffices to show that Enc_g is injective.
Suppose we have (π_1, r_1) and (π_2, r_2) with $\text{Enc}_g(\pi_1, r_1) \equiv \text{Enc}_g(\pi_2, r_2) \mod n^2$. For Enc_g to be injective, we must show that $(\pi_1, r_1) = (\pi_2, r_2)$. We find the following:

$$\begin{aligned}
\text{Enc}_g(m_1, r_1) \equiv & \text{Enc}_g(m_2, r_2) \mod n^2 \\
\Rightarrow & g^{(\pi_1 - \pi_2)} r_1^n \equiv r_2^n \mod n^2 \\
\Rightarrow & g^{\Lambda(\pi_1 - \pi_2)} r_1^{n\Lambda} \equiv r_2^{n\Lambda} \mod n^2 \\
\Rightarrow & g^{\Lambda(\pi_1 - \pi_2)} \equiv 1 \mod n^2
\end{aligned}$$

Where the last implication holds since by Carmichael's theorem we have $x^{n\Lambda} \equiv 1 \mod n^2$ for every $x \in \mathbb{Z}_{n^2}^*$. But then we know that $\text{ord}(g) | \Lambda(\pi_1 - \pi_2)$ and since $g \in \mathcal{G}$, also $n | \Lambda(\pi_1 - \pi_2)$. By the definition of Carmichael's function, we have $\gcd(\Lambda, n^2) = 1$, so $\gcd(\Lambda, n) = 1$. This implies $n | (\pi_1 - \pi_2)$ and since $0 \leq \pi_1, \pi_2 < n$, we find $\pi_1 = \pi_2$. From this we find that $r_1^n \equiv r_2^n \mod n^2$. Now since $r_1, r_2 \in \mathbb{Z}_n^*$ and $(r_1/r_2)^n \equiv 1 \mod n^2$, we find that $r_1 = r_2$. □

Now we define the n-th residuosity class of ψ with respect to g to be the unique (follows from the above theorem) integer $\pi \in \mathbb{Z}_n$ such that there exists an $r \in \mathbb{Z}_n^*$ such that $\psi = \text{Enc}_g(\pi, r)$. Let $[\psi]_g$ denote the n-th residuosity class of ψ with respect to g. We find that the following holds.

Lemma 2.5.2. *Let* $\psi \in \mathbb{Z}_{n^2}^*$ *and distinct* $g_1, g_2 \in \mathcal{G}$ *be given. Then the following equations hold.*

$$\begin{aligned}
[\psi]_{g_1} \equiv & [\psi]_{g_2} [g_2]_{g_1} \mod n \\
[\psi]_{g_2} \equiv & [\psi]_{g_1} [g_2]_{g_1}^{-1} \mod n
\end{aligned}$$

Proof. We write $\psi \equiv g_2^{[\psi]_{g_2}} r_1^n \mod n^2$ and $g_2 \equiv g_1^{[g_2]_{g_1}} r_2^n \mod n^2$. Substituting the second into the first equation yields

$$\psi \equiv (g_1^{[g_2]_{g_1}} r_2^n)^{[\psi]_{g_2}} r_1^n \mod n^2 \equiv \mathrm{Enc}_{g_1}([\psi]_{g_2}[g_2]_{g_1}, r_1 r_2^{[\psi]_{g_2}}).$$

But clearly also we have $\psi = \mathrm{Enc}_{g_1}([\psi]_{g_1}, r_3)$, for some $r_3 \in \mathbb{Z}_n^*$. So we find that

$$\mathrm{Enc}_{g_1}([\psi]_{g_2}[g_2]_{g_1}, r_1 r_2^{[\psi]_{g_2}}) = \mathrm{Enc}_{g_1}([\psi]_{g_1}, r_3)$$

and by injectivity of Enc_g, we find

$$[\psi]_{g_1} \equiv [\psi]_{g_2}[g_2]_{g_1} \mod n.$$

The same follows if the role of g_1 and g_2 are interchanged, so also we have

$$[\psi]_{g_2} \equiv [\psi]_{g_1}[g_1]_{g_2} \mod n.$$

From this it follows that $[g_1]_{g_2} \equiv [g_2]_{g_1}^{-1} \mod n$ and thus $[\psi]_{g_2} \equiv [\psi]_{g_1}[g_2]_{g_1}^{-1} \mod n$. □

We are now ready to define the decryption function. First, let $\mathcal{S}_n = \{u \in \mathbb{Z}_{n^2} | u = 1 \mod n\}$ and define the function $L : \mathcal{S}_n \to \mathbb{Z}_n$ by $L(u) = \frac{u-1}{n}$. Now we find the following.

Lemma 2.5.3. *Let $\psi \in \mathbb{Z}_{n^2}^*$ be given. Then*

$$L(\psi^\Lambda \mod n^2) \equiv \Lambda[\psi]_{n+1} \mod n.$$

Proof. We have that $n+1$ has order n, so $(n+1) \in \mathcal{G}$. Therefore for some r we have the following.

$$\begin{aligned} \psi \equiv& \mathrm{Enc}_{n+1}([\psi]_{n+1}, r) \\ \equiv& (n+1)^{[\psi]_{n+1}} r^n \mod n^2 \\ \Rightarrow \psi^\Lambda \equiv& (n+1)^{\Lambda[\psi]_{n+1}} \mod n^2 \\ \equiv& 1 + \Lambda[\psi]_{n+1} n \mod n^2, \end{aligned}$$

where the last equivalence holds since $(1+n)^x \equiv 1 + xn \mod n^2$. This clearly implies the claim. □

Decryption now is defined to be $\mathrm{Dec}_g(\psi) = \frac{L(\psi^\Lambda \mod n^2)}{L(g^\Lambda \mod n^2)} \mod n$. The private key thus consists of Λ. For computational convenience, often $\mu = L(g^\Lambda \mod n^2)^{-1}$ is also included. Correctness of decryption can be shown as follows.

$$\begin{aligned} \mathrm{Dec}_g(\mathrm{Enc}_g(\pi, r)) \equiv& L(\psi^\Lambda \mod n^2) \cdot \mu \mod n \\ \equiv& \frac{L(\psi^\Lambda \mod n^2)}{L(g^\Lambda \mod n^2)} \mod n \\ \equiv& \frac{\Lambda[\psi]_{n+1}}{\Lambda[g]_{n+1}} \mod n \\ \equiv& [\psi]_g \equiv \pi \mod n \end{aligned}$$

And the homomorphic property can be seen as follows. Consider two messages π_1 and π_2. Then we find that

$$\begin{aligned}\text{Dec}(\text{Enc}(\pi_1)\text{Enc}(\pi_2)) =& \text{Dec}(g^{\pi_1+\pi_2}(r_1r_2)^n \mod n^2)\\ =& L((g^{\pi_1+\pi_2}(r_1r_2)^n)^\Lambda \mod n^2)\cdot\mu\\ =& L((g^{(\pi_1+\pi_2 \mod n)}(r_1r_2)^n)^\Lambda \mod n^2)\cdot\mu\\ =& \text{Dec}(\text{Enc}(\pi_1+\pi_2 \mod n)).\end{aligned}$$

Here the third equality holds since

$$\pi_1+\pi_2 \mod n = \begin{cases}\pi_1+\pi_2 & \pi_1+\pi_2 < n\\ \pi_1+\pi_2-n & \pi_1+\pi_2 \geq n\end{cases}$$

Note that here the modulo operation maps to $[0, n-1]$. Also note that here we must include decryption for equality to hold, since the Paillier cryptosystem has a random encryption factor. Again, since the decryption works modulo n, the ciphertext will always be correctly decrypted independent of the amount of multiplications performed.

2.5.3 Applications: Secure Multiparty Computation

One class of the possible applications of partially homomorphic encryption schemes is secure multiparty computation. The idea is that we have n parties $P_1, \dots P_n$, and each party holds a specific value, say P_i holds x_i. These parties share the desire to compute a function $f(x_1, \dots, x_n)$, but all do not want to share their secret (except for of course information that follows directly from the outcome). As long as f is a computable function, a protocol which evaluates the function can be found [Sch11, p. 66].
A known example of secure multiparty computation is for $n = 2$ and is often referred to as Yao's Millionaire Problem [Yao82]. The problem is that of two millionaires who both want to know which of them is richer. But of course, they do not want to tell the other their exact wealth. The function corresponding to the problem is $f(x_1, x_2) = (x_1 > x_2)$. Yao presents several solutions to this problem in [Yao82].
Another example of an application which is used today is electronic voting. It is clear that this is a specialization of secure multiparty computation, since in the end one would like to know the sum of all votes given to a certain party. But there is a bit more to it, since one would like to verify if a voter is eligible and honest (i.e. only cast a single vote), and guard the privacy of the voter. It is possible to define a scheme that achieves all of these properties, one such scheme is presented in [CGS97].

2.6 Somewhat Homomorphic Encryption

Now that we have seen that partially homomorphic encryption schemes exist, we want to work towards fully homomorphic encryption schemes. Since this step is far from trivial, we will first define something in between. As we have seen, partially homomorphic schemes

have a group structure; only a single operation is defined, which is preserved by the homomorphism. There is no limit to the amount of times this operation is performed.
Now a logical intermediate step would be an encryption scheme which allows two operations, i.e. addition and multiplication, but only a finite amount. In other words, both operations are preserved by the homomorphism, but correct decryption is only guaranteed for a limited amount of operations. Gentry calls such a scheme "somewhat homomorphic". It turns out that this intermediate step will eventually lead to a fully homomorphic scheme. For this we need several new definitions, which will be presented in the next chapter.

3 Gentry's Construction

In this section we present Gentry's general construction to create fully homomorphic encryption schemes [Gen09a, chap. 2-4, pp. 27-56]. Before we can properly introduce Gentry's general construction we first provide some motivations of what we can consider a somewhat homomorphic scheme or a fully homomorphic scheme. These will be given below. After this, we will show how to construct a fully homomorphic encryption scheme from a somewhat homomorphic encryption scheme. The ability to do this is called "**bootstrappability**". Finally, the security of these schemes is discussed.

3.1 Definitions

Gentry [Gen09a] introduced the following definitions, in order to define his general construction. To use this construction, we assume that we have a somewhat homomorphic encryption scheme $\mathcal{E}$. Like every public-key encryption scheme, this scheme contains the three algorithms $\text{KeyGen}_{\mathcal{E}}$, $\text{Encrypt}_{\mathcal{E}}$ and $\text{Decrypt}_{\mathcal{E}}$, but in addition also the algorithm $\text{Evaluate}_{\mathcal{E}}$. This algorithm is able to evaluate circuits on the ciphertexts. A circuit is called permitted when the ciphertext output by $\text{Evaluate}_{\mathcal{E}}$ can be decrypted correctly. As input, this algorithm requires the public key, a permitted circuit, and a tuple of ciphertexts of size equal to the circuit input.
The first definition provides a criterium to judge when a somewhat homomorphic encryption scheme is correct [Gen09a, p. 28].

Definition 3.1.1. *Let $\mathcal{C}_{\mathcal{E}}$ denote the set of* permitted circuits *on the plaintexts. Now for a somewhat homomorphic encryption scheme $\mathcal{E}$ we define the scheme to be* correct *if, for any key-pair (sk,pk) generated by* $\text{KeyGen}_{\mathcal{E}}$*, any circuit $C \in \mathcal{C}_{\mathcal{E}}$, any plaintexts $\pi_1, \ldots, \pi_t$, and any ciphertexts $\Psi = \langle \psi_1, \ldots, \psi_t \rangle$, with $\psi_i = \text{Encrypt}_{\mathcal{E}}(\text{pk}, \pi_i)$, the following holds:*

$$\psi = \text{Evaluate}_{\mathcal{E}}(\text{pk}, C, \Psi) \Rightarrow \text{Decrypt}_{\mathcal{E}}(\text{sk}, \psi) = C(\pi_1, \ldots, \pi_t).$$

This ensures that the circuit is correctly evaluated, but this definition by itself is not sufficient to introduce a fully homomorphic encryption scheme, since one could define $\text{Evaluate}_{\mathcal{E}}$ to do nothing at all and just output (C, Ψ). Then $\text{Decrypt}_{\mathcal{E}}$ would decrypt the ciphertexts and apply the circuit to the plaintexts. To rule out this possibility, Gentry limits the size of the circuit defined by the $\text{Decrypt}_{\mathcal{E}}$ algorithm. This bound should only depend on the security parameter. A scheme following this definition is called **compact**.

Definition 3.1.2. *A somewhat homomorphic encryption scheme $\mathcal{E}$ is considered* compact *if there exists a polynomial f in the security parameter λ such that* $\text{Decrypt}_{\mathcal{E}}$ *can be performed by a circuit of size at most $f(\lambda)$.*

Note that this definition prevents that one defines $\text{Evaluate}_{\mathcal{E}}$ to do nothing but output the ciphertext along with the circuit, since in this case after every evaluation the size of the circuit required to correctly decrypt the ciphertext would increase by the size of the circuit to be evaluated. After sufficiently many evaluations, the decryption circuit will grow beyond the bound $f(\lambda)$. Now fully homomorphic encryption is defined as follows:

Definition 3.1.3. *A homomorphic encryption scheme is considered to be* fully homomorphic *if it is* compact *and* correct *for all permitted circuits.*

But Gentry introduces two more notions. One could require only a scheme to be able to evaluate circuits up to a certain fixed depth d. He names such schemes **leveled** fully homomorphic encryption schemes. The other property that one might require is that the evaluated circuit is hidden from any eavesdroppers. This property will be called **circuit privacy**.
These two notions are defined below [Gen09a, p. 29].

Definition 3.1.4. *A collection of homomorphic encryption schemes* $\{\mathcal{E}^{(d)} : d \in \mathbb{Z}^+\}$ *is considered to be* leveled fully homomorphic *if:*

- *for all* $d \in \mathbb{Z}^+$ *all schemes in the collection use the same decryption circuit,*
- $\mathcal{E}^{(d)}$ *is* compact *and* correct *for all circuits of depth at most* d*,*
- *the complexity of the algorithms in* $\mathcal{E}^{(d)}$ *is polynomial in* λ*,* d *and the size of the circuit* C*.*

Definition 3.1.5. *A homomorphic encryption scheme* $\mathcal{E}$ *is called* circuit private *if the distributions of* $\text{Encrypt}_{\mathcal{E}}(\text{pk}, C(\pi_1, \ldots, \pi_t))$ *and* $\text{Evaluate}_{\mathcal{E}}(\text{pk}, C, \Psi)$ *are statistically indistinguishable for any key-pair (sk,pk) generated by* $\text{KeyGen}_{\mathcal{E}}$*, any circuit* $C \in C_{\mathcal{E}}$ *and any fixed ciphertexts* $\Psi = \langle \psi_1, \ldots, \psi_t \rangle$ *in the image of* $\text{Encrypt}_{\mathcal{E}}$ *for plaintexts* $\pi_1, \ldots, \pi_t$*.*

Building on these definitions, Gentry defines a construction which allows to create fully homomorphic encryption schemes from somewhat homomorphic encryption schemes.

3.2 From Somewhat to Fully Homomorphic Encryption

It will be shown in this section that if we have a somewhat homomorphic encryption scheme $\mathcal{E}$ which compactly evaluates augmentations of its own decryption circuit, we can use this scheme to create a fully homomorphic encryption scheme. For now, we will assume that the somewhat homomorphic scheme introduces an error term upon encryption. It is easy to remove this error with use of the secret key, but hard without this knowledge.
The idea then is as follows. We have an encryption $\psi_1, \ldots, \psi_t$ of a set of plaintexts $\pi_1, \ldots, \pi_t$ under some public key pk_i. If $\text{Evaluate}_{\mathcal{E}}$ is applied to the set of ciphertexts, we have an increased error parameter on the resulting ciphertext ψ. The scheme is able to correctly decrypt the ciphertext as long as the error term is not too large. So we want to be able to reduce the error as otherwise circuit depth is limited.
It is clear that if we would decrypt a ciphertext with a large error term (but within the schemes ability to decrypt) and then encrypt it again, the error again is as small as possible for a ciphertext. But of course, not everyone has access to the secret key. We can solve this by decrypting homomorphically. This is done by encrypting ψ under pk_{i+1} and then homomorphically evaluating the decryption circuit corresponding to $\text{Decrypt}_{\mathcal{E}}$, where sk_i is

encrypted under pk_{i+1}. If one homomorphically evaluates a permitted circuit, one obtains the encryption of the result of the evaluation of this circuit. So if the decryption circuit is a permitted circuit, the result ψ' is a fresh encryption under pk_{i+1} and the corresponding plaintext is never revealed.
Furthermore, nobody has direct access to the corresponding secret key, as it is only available in encrypted form. Whether or not this is a security issue will be discussed in section 3.3.
More formally, Gentry introduces the algorithm $\mathrm{Recrypt}_{\mathcal{E}}$. It is assumed that the plaintext space is $\mathcal{M} = \{0, 1\}$ and $D_{\mathcal{E}}$ is a boolean circuit in $C_{\mathcal{E}}$, which is the translation of decryption into a circuit. Note that this is possible since decryption obviously is a computable function and we can represent every computable function as a circuit. Furthermore, Gentry uses two instances of $\mathcal{E}$ to create a single system. Suppose we have two $\mathcal{E}$ key-pairs $(\mathrm{sk}_1, \mathrm{pk}_1)$ and $(\mathrm{sk}_2, \mathrm{pk}_2)$. Let ψ_1 be an encryption of $\pi \in \mathcal{M}$ under pk_1. Let $\overline{\mathrm{sk}_{1j}}$ denote an encryption of the j-th bit of sk_1 under pk_2. The algorithm then is as follows.

$\mathrm{Recrypt}_{\mathcal{E}}(\mathrm{pk}_2, D_{\mathcal{E}}, \langle\overline{\mathrm{sk}_{1j}}\rangle, \psi_1)$:
$\overline{\psi_{1j}} = \mathrm{Encrypt}_{\mathcal{E}}(\mathrm{pk}_2, \psi_{1\mathrm{j}})$
$\psi_2 = \mathrm{Evaluate}_{\mathcal{E}}(\mathrm{pk}_2, D_{\mathcal{E}}, \langle\langle\overline{\mathrm{sk}_{1j}}\rangle, \langle\overline{\psi_{1j}}\rangle\rangle)$
Output ψ_2

Clearly $D_{\mathcal{E}}$ requires the secret key as input along with the ciphertext which is to be decrypted. Since it is a circuit, these inputs are required as bits. The circuit is evaluated homomorphically, so we need these bits in encrypted form. The encrypted bits of the secret key are given in the public key, and we simply encrypt the ciphertext bits to obtain them in the correct form.
Now since $\mathrm{Recrypt}_{\mathcal{E}}$ decrypts and then encrypts the ciphertext again, it allows us to reduce the error in a ciphertext. But to make this useful, we must also be able to evaluate some circuit before refreshing the ciphertext. Hence Gentry requires that the encryption scheme can compactly evaluate augmentations of its own decryption circuit. Let Γ denote a set of gates. Now for a gate $g \in \Gamma$, define multiple copies of $D_{\mathcal{E}}$, connected by g, to be called a g-augmented decryption circuit. Denote the set containing these circuits by $D_{\mathcal{E}}(\Gamma)$. This leads to the following definition of bootstrappability.

Definition 3.2.1. *Let $C_{\mathcal{E}}$ denote the set of circuits that $\mathcal{E}$ compactly evaluates. We call $\mathcal{E}$* bootstrappable *with respect to a set of gates Γ if*

$$D_{\mathcal{E}}(\Gamma) \subseteq C_{\mathcal{E}}.$$

It will become clear that, if Γ consists of at least the trivial gate and the NAND gate, if we have a scheme which is bootstrappable with respect to Γ, then this scheme can be turned into a leveled fully homomorphic scheme.

3.2.1 The Construction of a Leveled Scheme

Below we will give a construction that, using an encryption scheme $\mathcal{E}$ which is bootstrappable with respect to the before mentioned Γ (consisting of the trivial and NAND gate),

will create a leveled fully homomorphic encryption scheme $\mathcal{E}^{(d)}$ for any integer $d \geq 1$. Again, let $\mathcal{M} = \{0,1\}$ be the plaintext space. Let $\ell = \ell(\lambda)$ be such that $\mathcal{E}$'s secret key can be expressed as an element of $\mathcal{M}^\ell$ and let $D_\mathcal{E}(\Gamma, \delta)$ denote the set of circuits which equal a circuit of depth δ and gates in Γ. Note that if a circuit has depth smaller than δ, we can add trivial gates to make it have depth δ. So from now on we will fix the depth to δ, even though the circuits could be smaller. The algorithms of $\mathcal{E}^{(d)}$ are as follows.

$\text{KeyGen}_{\mathcal{E}^{(d)}}(\lambda, d)$:

To obtain d levels, we need d keypairs. As mentioned before, we also need the secret keys, bitwise encrypted.
$(\text{sk}_i, \text{pk}_i) = \text{KeyGen}_\mathcal{E}(\lambda) \qquad \text{for } i \in [0, d]$
$\overline{\text{sk}_{ij}} = \text{Encrypt}_\mathcal{E}(\text{pk}_{i-1}, \text{sk}_{ij}) \quad \text{for } i \in [1, d], j \in [1, \ell]$
Now we have the secret key $\text{sk}^{(d)} = \text{sk}_0$ and the public key $\text{pk}^{(d)} = (\langle \text{pk}_i \rangle, \langle \overline{\text{sk}_{ij}} \rangle)$. Denote by $\mathcal{E}^{(\delta)}$ the subsystem with secret and public key respectively $\text{sk}^{(\delta)} = \text{sk}_0$ and $\text{pk}^{(\delta)} = (\langle \text{pk}_i \rangle, \langle \overline{\text{sk}_{ij}} \rangle_{i \in [1,\delta]})$, for $\delta \leq d$.
The algorithm outputs $\text{sk}^{(d)}$ and $\text{pk}^{(d)}$.

$\text{Encrypt}_{\mathcal{E}^{(d)}}(\text{pk}^{(d)}, \pi)$:

Outputs a ciphertext $\psi = \text{Encrypt}_\mathcal{E}(\text{pk}_d, \pi)$.

$\text{Decrypt}_{\mathcal{E}^{(d)}}(\text{sk}^{(d)}, \psi)$:

The ciphertext ψ should be an encryption under pk_0. Output is a plaintext $\pi' = \text{Decrypt}_\mathcal{E}(\text{sk}_0, \psi)$.

$\text{Evaluate}_{\mathcal{E}^{(d)}}(\text{pk}^{(\delta)}, C_\delta, \Psi_\delta)$:

We have $C_\delta \in D_\mathcal{E}(\Gamma, \delta)$ and each element of Ψ_δ is an encryption under pk_δ. If $\delta = 0$, it outputs Ψ_0. Otherwise it does the following:

- $(C^\dagger_{\delta-1}, \Psi^\dagger_{\delta-1}) = \text{Augment}_{\mathcal{E}^{(\delta)}}(\text{pk}^{(\delta)}, C_\delta, \Psi_\delta)$
- $(C_{\delta-1}, \Psi_{\delta-1}) = \text{Reduce}_{\mathcal{E}^{(\delta-1)}}(\text{pk}^{(\delta-1)}, C^\dagger_{\delta-1}, \Psi^\dagger_{\delta-1})$
- Call $\text{Evaluate}_{\mathcal{E}^{(\delta-1)}}(\text{pk}^{(\delta-1)}, C_{\delta-1}, \Psi_{\delta-1})$

$\text{Augment}_{\mathcal{E}^{(\delta)}}(\text{pk}^{(\delta)}, C_\delta, \Psi_\delta)$:

Again, $C_\delta \in D_\mathcal{E}(\Gamma, \delta)$, and each element of Ψ_δ is an encryption under pk_δ. Define $C^\dagger_{\delta-1}$ to be the augmentation of C_δ and $D_\mathcal{E}$. This is a circuit which takes as input the encrypted secret key $\langle \overline{\text{sk}_{\delta j}} \rangle$ and $\Psi^\dagger_{\delta-1} = \{\langle \langle \overline{\text{sk}_{\delta j}} \rangle, \langle \overline{\psi_j} \rangle \rangle | \psi_j \in \Psi_\delta\}$, where $\overline{\psi_j} = \text{Encrypt}_{\mathcal{E}^{(\delta-1)}}(\text{pk}^{(\delta-1)}, \psi_j)$ for all $\psi_j \in \Psi_\delta$.
Level δ of the circuit is taken and every gate g is replaced with the g-augmented decryption circuit. When the circuit is evaluated, the output bits are fresh encryptions of the bits which would have resulted by evaluating level δ in the clear.
The algorithm outputs $(C^\dagger_{\delta-1}, \Psi^\dagger_{\delta-1})$.

$\text{Reduce}_{\mathcal{E}^{(\delta)}}(\text{pk}^{(\delta)}, C^\dagger_\delta, \Psi^\dagger_\delta)$:

Here each element of $\Psi^\dagger_\delta$ must be in the image of $\text{Encrypt}_{\mathcal{E}^{(\delta)}}$ and we have $C^\dagger_\delta \in$

$D_{\mathcal{E}}(\Gamma, \delta+1)$. Now we define by C_δ the sub-circuit of $C_\delta^\dagger$, containing the first δ levels. Furthermore, if w denotes a wire at level δ (i.e. an input wire for C_δ) define by $C_\delta^{(w)}$ the sub-circuit of $C_\delta^\dagger$ with output wire w and by $\psi_\delta^{(w)}$ the ciphertext output by this sub-circuit upon input by $\Psi_\delta^{(w)}$, the corresponding subset of $\Psi_\delta^\dagger$.
This sub-circuit thus consists of g-augmentations of the decryption circuit. The circuit can be evaluated and will output a ciphertext encrypted under pk_δ with small error. Note that now the depth of the entire circuit is reduced by one. This process is repeated until the entire circuit has been evaluated at this level.
So we have $\psi_\delta^{(w)} = \mathrm{Evaluate}_{\mathcal{E}}(\mathrm{pk}_\delta, C_\delta^{(w)}, \Psi_\delta^{(w)})$. Now define $\Psi_\delta = \{\psi_\delta^{(w)} | w \text{ wire at level } \delta\}$ and return (C_δ, Ψ_δ).

3.2.2 Correctness of the Construction

We must show that the construction is correct, i.e. if $\mathcal{E}$ is a correct bootstrappable encryption scheme, then $\mathcal{E}^{(d)}$ is leveled fully homomorphic. This is shown in the following theorem.

Theorem 3.2.2. *Suppose $\mathcal{E}$ is a correct bootstrappable encryption scheme with respect to a universal set of gates Γ (i.e. a set of gates with which all other gates can be constructed), then $\mathcal{E}^{(d)}$ as defined above is leveled fully homomorphic.*

Proof. First, define the following notation: $D(\delta, w, C, \Psi)$ denotes the plaintext value at the end of a wire w in the circuit C, for $C \in D_{\mathcal{E}}(\Gamma, \delta)$. This plaintext value is obtained by decrypting the ciphertexts in Ψ (which are encrypted under sk_δ) that are associated with the input wires of C, and then evaluating the circuit up to the end of w. This definition means that for each $\delta \leq d$, the levels before level δ are evaluated as ciphertexts and the levels starting at δ are evaluated in the clear.
Now we have a circuit C_d and input Ψ_d for this circuit. To prove correctness of the above given construction, it suffices to show that

$$D(d, w_{out}, C_d, \Psi_d) = D(0, w_{out}, C_0, \Psi_0), \tag{2}$$

for each output wire w_{out} of the circuit at level 0. Clearly, if this holds then the output value of each wire w_{out} is the same, regardless of how much of the circuit is evaluated in the clear. There are only two algorithms which affect the circuit depth, so we will look into these.
First, consider $(C_{\delta-1}^\dagger, \Psi_{\delta-1}^\dagger) = \mathrm{Augment}_{\mathcal{E}^{(\delta)}}(\mathrm{pk}^{(\delta)}, C_\delta, \Psi_\delta)$. Now we claim that $D(\delta, w, C_\delta, \Psi_\delta) = D(\delta-1, w, C_{\delta-1}^\dagger, \Psi_{\delta-1}^\dagger)$ for all wires w up to level at most $\delta - 1$. To validate this claim, note that the circuits C_δ and $C_{\delta-1}^\dagger$ are equivalent in the first $\delta - 1$ levels. $\mathrm{Augment}_{\mathcal{E}^{(\delta)}}$ will evaluate the circuit at level δ and output a fresh ciphertext which is the input for the remaining circuit of depth $\delta - 1$.
Second, consider $(C_\delta, \Psi_\delta) = \mathrm{Reduce}_{\mathcal{E}^{(\delta)}}(\mathrm{pk}^{(\delta)}, C_\delta^\dagger, \Psi_\delta^\dagger)$. We claim that $D(\delta, w, C_\delta^\dagger, \Psi_\delta^\dagger) = D(\delta, w, C_\delta, \Psi_\delta)$ for all wires up to level δ. This claim is correct, since the circuits $C_\delta^\dagger$ and C_δ are equivalent in the first δ levels and $\mathrm{Evaluate}_{\mathcal{E}}$ is correct for circuits in $\mathrm{D}_{\mathcal{E}}(\Gamma)$.

Since these two claims result in $D(\delta, w, C_\delta, \Psi_\delta) = D(\delta - 1, w, C^\dagger_{\delta-1}, \Psi^\dagger_{\delta-1}) = D(\delta-1, w, C_{\delta-1}, \Psi_{\delta-1})$, for all wires w in level at most $\delta-1$, this suffices to prove Equation 2. □

3.2.3 Making the Scheme Fully Homomorphic

The obvious way to make the construction as mentioned before fully homomorphic is by making this construction cyclic. In the scheme, for each level the secret key is encrypted with the public key of the next level. Now if one would only have one keypair and encrypt the secret key under its corresponding public key, this would result in a leveled fully homomorphic scheme with infinite levels, which is exactly a fully homomorphic encryption scheme.

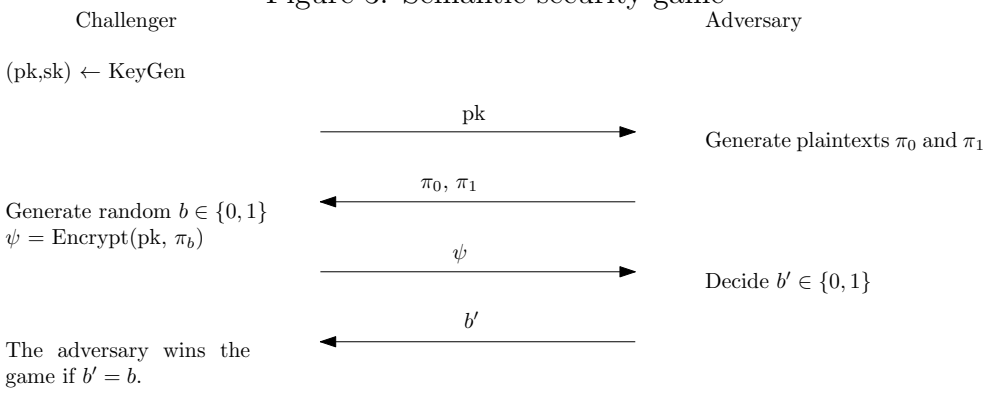
3.3 Security

With the general construction in place, we must analyze its security. First we will show that if the underlying scheme $\mathcal{E}$ is semantically secure, then also the leveled fully homomorphic scheme $\mathcal{E}^{(d)}$ is semantically secure. Then we will look into security with respect to Key Dependent Messages (KDM-security) and finally we will argue that the fully homomorphic scheme is secure in the random oracle model.

3.3.1 Semantic Security

An encryption scheme is called semantically secure if an adversary cannot deduce any information about the plaintext given just the ciphertext and the public key.
This notion is often explained using a game, which we will call the Semantic Security Game. The picture below depicts this game.

Figure 3: Semantic security game



The definition of semantic security now is as follows.

Definition 3.3.1. *Consider the setting as depicted above. A scheme is considered to be* $(t, \varepsilon)-$semantically secure *if there exists no adversary running in time t, that has non-negligible advantage ε in the semantic security game, i.e.* $\mathbb{P}[b' = b] = \frac{1}{2} + \varepsilon$.

We now proceed to show that if $\mathcal{E}$ is semantically secure than also $\mathcal{E}^{(d)}$ is semantically secure, this is done by contradiction.

Theorem 3.3.2. *Suppose there exists an adversary $\mathcal{A}$ which (t, ε)-breaks $\mathcal{E}^{(d)}$, then there exists an adversary $\mathcal{A}^*$ that (t', ε')-breaks $\mathcal{E}$, with $t' \approx t \cdot \ell$ and $\varepsilon' \approx \varepsilon / 2\ell(d+1)$, where ℓ is such that a secret key in $\mathcal{E}$ can be expressed in $\mathcal{M}^\ell$.*

Proof. For $k \in [0, d]$, we define game k as follows. The challenger generates the public and secret key for $\mathcal{E}^{(d)}$ as usual, but for $i \in [1, k]$ generates random $\mathcal{E}$ keypairs $(\mathrm{sk}'_i, \mathrm{pk}'_i) = \mathrm{KeyGen}_{\mathcal{E}}$ and replaces the encryption of sk_i under pk_{i-1} in the public key with an encryption of sk'_i under pk_{i-1}. Thus the public key contains for $i \in [1, k]$ encryptions of secret keys which have nothing to do with the paired public key.
Note that game 0 is the standard semantic security game against $\mathcal{E}^{(d)}$. Now we also define game $d+1$ to be as game d, but the challenger ignores the given plaintexts and returns an encryption of a random generated plaintext of correct size. It is clear that the advantage of $\mathcal{A}$ in game 0 is $\varepsilon_0 = \varepsilon$ and in game $d+1$ it is $\varepsilon_{d+1} = 0$, since the given ciphertext is independent of the bit b.
Now since we have $\varepsilon_0 = \varepsilon$ and $\varepsilon_{d+1} = 0$, there must exist some $k \in [0, d]$ such that $\varepsilon_k - \varepsilon_{k+1} \geq \varepsilon/(d+1)$. Fix this value of k.
We will now show that $\mathcal{A}^*$ can use $\mathcal{A}$ to break $(\mathcal{E})^\ell$, i.e. ℓ-fold repetition of $\mathcal{E}$. First the case that $k < d$:
$\mathcal{A}^*$ receives an $\mathcal{E}$-public key from the challenger. It then generates the public and secret key for $\mathcal{E}^{(d)}$ as in game k, but where pk_k is replaced with pk, and generates an additional dummy keypair $(\mathrm{sk}'_{k+1}, \mathrm{pk}'_{k+1})$. Now it sets $\pi_0 = \mathrm{sk}_{k+1}$ and $\pi_1 = \mathrm{sk}'_{k+1}$ and sends these to the challenger.
The challenger randomly selects $\beta \in \{0, 1\}$ and replies with $\mathbf{\Psi}$, which is the bitwise encryption of π_β. Now $\mathcal{A}^*$ replaces the encryption of sk_{k+1} with $\mathbf{\Psi}$ and forwards the public values to $\mathcal{A}$. Note that these public values exactly correspond to game $k + \beta$.
$\mathcal{A}$ generates two plaintexts $\tilde{\pi}_0$ and $\tilde{\pi}_1$ which it sends to $\mathcal{A}^*$. $\mathcal{A}^*$ randomly generates $b \in \{0, 1\}$ and replies with the bitwise encryption of π_b under pk_d. Now $\mathcal{A}$ sends guess b' to $\mathcal{A}^*$ and $\mathcal{A}^*$ sends $\beta' = b \oplus b'$ to the challenger.
We now have that $\mathcal{A}^*$ has advantage as follows:

$$\mathbb{P}[\beta' = \beta] = \mathbb{P}[\beta = 1]\mathbb{P}[\beta' = 1] + \mathbb{P}[\beta = 0]\mathbb{P}[\beta' = 0] = \frac{1}{2}(1 + \varepsilon_k - \varepsilon_{k+1}) \geq \frac{1}{2} + \varepsilon/2(d+1).$$

In the case that $k = d$ the game is as follows. $\mathcal{A}^*$ receives the $\mathcal{E}$ public key pk from the challenger. It generates the secret and public key values as in game d, with pk_d replaced with pk, and forwards the public values to $\mathcal{A}$. Now $\mathcal{A}$ generates two plaintexts $\tilde{\pi}_0$ and $\tilde{\pi}_1$ and sends these to $\mathcal{A}^*$. Then $\mathcal{A}^*$ randomly generates $b \in \{0, 1\}$ and $\pi \in \mathcal{M}^\ell$ and forwards

$\pi_0 = \tilde{\pi}_b$ and $\pi_1 = \pi$ to the challenger.
The challenger randomly generates $\beta \in \{0, 1\}$ and replies with the bitwise encryption $\mathbf{\Psi}$ of π_β under pk. $\mathcal{A}^*$ forwards these values to $\mathcal{A}$. Note that again the game between $\mathcal{A}^*$ and $\mathcal{A}$ is game $d + \beta$. $\mathcal{A}$ responds with a guess b' and $\mathcal{A}^*$ sends $\beta' = b' \oplus b$ to the challenger.
Again we find that the advantage of $\mathcal{A}^*$ is at least $\varepsilon/2(d+1)$. So this shows that $\mathcal{A}^*$ breaks $(\mathcal{E})^\ell$ with time $t'' \approx t$ and advantage $\varepsilon'' \geq \varepsilon/2(d+1)$. This translates to $\mathcal{A}^*$ breaking $\mathcal{E}$ with time $t' \approx t \cdot \ell$ and $\varepsilon \geq \varepsilon/2\ell(d+1)$ as required. □

3.3.2 KDM-Security

We now have shown that the semantic security of the leveled fully homomorphic scheme follows from the semantic security of the underlying scheme. But this unfortunately proves nothing about the semantic security of the fully homomorphic scheme, since in the leveled version a secret key always is encrypted under a different public key. But in the fully homomorphic scheme, if we were to introduce a cycle in the bootstrapping, one can compute via the homomorphism an encryption of the secret key sk_0 under the corresponding public key pk_0.
The security of a fully homomorphic scheme requires KDM-security, i.e. key dependent messages, in particular the secret key itself, are indistinguishable from encryptions of random secret keys unrelated to the public key under which they are encrypted.
It is clear that a scheme which is bootstrappable and KDM-secure, can be used to create a semantically secure fully homomorphic scheme.

3.3.3 Random Oracle Model

Let $\mathcal{E}^*$ denote a fully homomorphic encryption scheme created by adding a self-loop. In the Random Oracle Model, the semantic security of $\mathcal{E}^*$ can be proven. Given a leveled fully homomorphic scheme $\mathcal{E}^{(d)}$ and a hash function $H : \mathcal{M}^{\ell'} \to \mathcal{M}^\ell$, define the scheme $\mathcal{E}^{(d)^\dagger}$ to be the same as $\mathcal{E}^{(d)}$, except for the following. During key generation, one generates a random vector $r \in \mathcal{M}^{\ell'}$, encrypts it bitwise under $\mathrm{pk}^{(d)}$ to $\bar{r}$, sets $\sigma = H(r) \star \mathrm{sk}_0$ and includes $(\langle \bar{r_j} \rangle, \sigma)$ in the public key. Here $\star$ is some invertible operation which would completely hide $b \in \mathcal{M}^\ell$ in $a \star b$ if $a \in \mathcal{M}^\ell$ were a one-time pad. The hash function is treated as random oracle in analysis. Then we have the following two theorems.

Theorem 3.3.3. [Gen09a, p. 53] *If $\mathcal{E}^{(d)}$ is semantically secure, then $\mathcal{E}^{(d)^\dagger}$ is semantically secure in the random oracle model.*

Theorem 3.3.4. [Gen09a, p. 54] *Suppose $\mathcal{E}$ is leveled circuit-private and let $\mathcal{E}^{(d)^\dagger}$ and $\mathcal{E}^*$ be constructed as described above. Now if $\mathcal{E}^{(d)^\dagger}$ is semantically secure, and the circuit required to compute H and invert $\star$ is at most d levels, then $\mathcal{E}^*$ is semantically secure.*

We do not give the proofs here, but just the main idea. The surprising result here is that $\mathcal{E}^*$ is proven to be semantically secure in the random oracle model, even though it does

not contain a hash function. This is a simple consequence of the homomorphic property, which allows us to create an encryption of sk_0 under pk_0. We can do this by using the encryption of r and a circuit corresponding to the hash function H, to find the encryption of $H(r)$. Since we have $\sigma = H(r) \star \mathrm{sk}_0$ in the public key, we can find the encryption sk_0 under pk_0 (if the encryption is under a different public key, we can simply use the cycle to find it under pk_0. This gives an encryption without using the hash function. And thus we find that $\mathcal{E}^*$ is semantically secure.
Unfortunately, this does not imply that $\mathcal{E}^*$ is semantically secure in the standard model.

4 Fully Homomorphic Encryption Schemes

As mentioned in the introduction, this section shows the first fully homomorphic scheme as presented by Gentry [Gen09a, chap. 2-4, pp. 69-114]. Thereafter we present two slight improvements on his scheme [SV10, GH11]. Finally we present a scheme which was created with the goal to only use integer arithmetic [vDGHV10].

4.1 Gentry's Scheme

In addition to the general construction, Gentry also provided a first fully homomorphic encryption scheme using ideal lattices [Gen09b]. This scheme is presented in two steps. First an initial somewhat homomorphic scheme is given. It will become clear that the scheme was not yet bootstrappable, so Gentry introduced another idea, he "squashed" the decryption circuit. This finally lead to a fully homomorphic scheme. After the complete introduction of the scheme, its security and performance will be analyzed.

4.1.1 The Somewhat Homomorphic Scheme

The scheme $\mathcal{E}$ is based on a polynomial ring and ideal lattices. We take as polynomial ring the ring $R = \mathbb{Z}[x]/f(x)$ with $f(x) \in \mathbb{Z}[x]$ monic and of degree n, and a fixed basis $\mathbf{B}_I$ of an ideal $I \subset R$. Let $\text{IdealGen}_{\mathcal{E}}(R, \mathbf{B}_I)$ be an algorithm which efficiently returns public and secret bases $(\mathbf{B}_J^{sk}, \mathbf{B}_J^{pk})$ of some ideal $J \subset R$, such that $I + J = R$, i.e. I and J are relatively prime. The public basis is chosen to be a least revealing basis, i.e. a basis of skewed long vectors. The secret basis consists of short nearly orthogonal vectors. Why this is chosen in this way will become clear soon. Let $\text{Samp}_{\mathcal{E}}(\mathbf{x}, \mathbf{B}_I, R, \mathbf{B}_J)$ be an algorithm which efficiently samples from the coset $\mathbf{x} + I$. We will leave these algorithms abstract for now, as only their outputs are interesting for the understanding. The scheme then consists of the following algorithms.

$\text{KeyGen}_{\mathcal{E}}(R, \mathbf{B}_I)$:
set $(\mathbf{B}_J^{sk}, \mathbf{B}_J^{pk}) = \text{IdealGen}_{\mathcal{E}}(R, \mathbf{B}_I)$. Now the plaintext space $\mathcal{M}$ of the scheme becomes (a subset of) $R \mod \mathbf{B}_I$. The public key is set to contain R, $\mathbf{B}_I$, $\mathbf{B}_J^{pk}$, and $\text{Samp}_{\mathcal{E}}$. The secret key contains all these as well and in addition $\mathbf{B}_J^{sk}$.

$\text{Encrypt}_{\mathcal{E}}(\text{pk}, \pi)$:
for plaintext $\pi \in \mathcal{M}$, take $\psi' = \text{Samp}_{\mathcal{E}}(\pi, \mathbf{B}_I, R, \mathbf{B}_J^{pk})$ and output $\psi = \psi' \mod \mathbf{B}_J^{pk}$.

$\text{Decrypt}_{\mathcal{E}}(\text{sk}, \psi)$:
for a ciphertext ψ output $\pi = (\psi \mod \mathbf{B}_J^{sk}) \mod \mathbf{B}_I$.

$\text{Evaluate}_{\mathcal{E}}(\text{pk}, C, \Psi)$:
for a permitted circuit C (see below) and a set of ciphertexts Ψ, run $\text{Add}_{\mathcal{E}}$ and $\text{Mult}_{\mathcal{E}}$ in the sequence as implied by C.

$\text{Add}_{\mathcal{E}}(\text{pk}, \psi_1, \psi_2)$:
output $\psi_1 + \psi_2 \mod \mathbf{B}_J^{pk}$.

$\text{Mult}_{\mathcal{E}}(\text{pk}, \psi_1, \psi_2)$:
output $\psi_1 \times \psi_2 \mod \mathbf{B}_J^{pk}$.

Next we will show that this scheme is indeed correct. Note that in the scheme, C actually is a circuit modulo $\mathbf{B}_I$, but its operations are replaced by ring operations. This is clarified in the following definition.

Definition 4.1.1. *Let C be a modulo $\mathbf{B}_I$ circuit. The* generalized circuit *$g(C)$ is defined to be the circuit C where the $\text{Add}_{\mathbf{B}_I}$ and $\text{Mult}_{\mathbf{B}_I}$ gates in C are replaced by the corresponding ring operations, addition and multiplication in R modulo $\mathbf{B}_I$.*

To prove correctness, Gentry introduces a few more definitions.

Definition 4.1.2. *Let X_{enc} be the image of $\text{Samp}_{\mathcal{E}}$. Then all ciphertexts output by $\text{Encrypt}_{\mathcal{E}}$ are in the coset $X_{enc} + J$. Let X_{dec} equal $R \mod \mathbf{B}_J^{sk}$, the distinguished representatives of the cosets of J with respect to the secret basis $\mathbf{B}_J^{sk}$. We also define r_{enc} to be the smallest value such that $X_{enc} \subseteq \mathcal{B}(r_{enc})$ and r_{dec} to be the largest value such that $X_{dec} \supseteq \mathcal{B}(r_{dec})$.*

Note that since I and J are ideal lattices, both X_{enc} and X_{dec} are subsets of $\mathbb{Z}^n$.

Definition 4.1.3. *Define $\mathcal{C}'_{\mathcal{E}} = \{C : \forall (x_1, \ldots, x_t) \in X_{enc}^t, g(C)(x_1, \ldots, x_t) \in X_{dec}\}$, this is the set of $\mod \mathbf{B}_I$ circuits for which the generalized circuit has output in X_{dec} as long as the inputs are in X_{enc}. For all subsets $\mathcal{C}_{\mathcal{E}} \subseteq \mathcal{C}'_{\mathcal{E}}$ we say that $\mathcal{C}_{\mathcal{E}}$ is a* set of permitted circuits.

Definition 4.1.4. *For r_{dec} and r_{enc} as defined above, define the set of* permitted circuits *$\mathcal{C}_{\mathcal{E}} = \{C : \forall (x_1, \ldots, x_t) \in \mathcal{B}(r_{enc})^t, g(C)(x_1, \ldots, x_t) \in \mathcal{B}(r_{dec})\}$. Clearly $\mathcal{C}_{\mathcal{E}} \subseteq \mathcal{C}'_{\mathcal{E}}$.*

Definition 4.1.5. *A ciphertext ψ' is considered to be* valid *with respect to the scheme $\mathcal{E}$ with public key* pk *and permitted circuits $\mathcal{C}_{\mathcal{E}}$ if it equals $\text{Evaluate}_{\mathcal{E}}(\text{pk}, C, \Psi)$ for some $C \in \mathcal{C}_{\mathcal{E}}$, where each $\psi \in \Psi$ is in the image of $\text{Encrypt}_{\mathcal{E}}$.*

Now we are ready to prove correctness of the above described scheme.

Theorem 4.1.6. *Let $\mathcal{C}_{\mathcal{E}}$ be a set of permitted circuits. We claim that $\mathcal{E}$ is correct for $\mathcal{C}_{\mathcal{E}}$, i.e. $\text{Decrypt}_{\mathcal{E}}$ correctly decrypts valid ciphertexts.*

Proof. Let $\Psi = \{\psi_1, \ldots, \psi_t\}$, $\psi_k = \pi_k + i_k + j_k$, with $\pi_k \in \mathcal{M}$, $i_k \in I$, $j_k \in J$ and $\pi_k + i_k \in \mathcal{B}(r_{enc})$, the following holds:

$$\begin{aligned} \text{Evaluate}_{\mathcal{E}}(\text{pk}, C, \Psi) &= g(C)(\Psi) \mod \mathbf{B}_J^{pk} \\ &\in g(C)(\pi_1 + i_1, \ldots, \pi_t + i_t) + J \end{aligned} \tag{3}$$

Since we have $\pi_k + i_k \in \mathcal{B}(r_{enc})$ and if $C \in \mathcal{C}_\mathcal{E}$, we find $g(C)(\pi_1 + i_1, \ldots, \pi_t + i_t) \in \mathcal{B}(r_{dec})$. And thus:

$$\begin{aligned} \text{Decrypt}_\mathcal{E}(\text{sk}, \text{Evaluate}_\mathcal{E}(\text{pk}, C, \Psi)) &= g(C)(\pi_1 + i_1, \ldots, \pi_t + i_t) \mod \mathbf{B}_I \\ &= g(C)(\pi_1, \ldots, \pi_t) \mod \mathbf{B}_I \\ &= C(\pi_1, \ldots, \pi_t). \end{aligned} \tag{4}$$

Which proves that $\mathcal{E}$ is correct for $\mathcal{C}_\mathcal{E}$. □

It is clear that the scheme is correct and can evaluate circuits in the permitted set, but it is not yet clear what this permitted set concretely is. With the triangle inequality, we have $||\mathbf{u} + \mathbf{v}|| \leq ||\mathbf{u}|| + ||\mathbf{v}||$, for $\mathbf{u}, \mathbf{v} \in R$. For multiplication we have that $||\mathbf{u} \times \mathbf{v}|| \leq \gamma_{mult}(R) \cdot ||\mathbf{u}|| \cdot ||\mathbf{v}||$, where $\gamma_{mult}(R)$ is some value which depends only on R.
The following theorem relates the circuit depth to the set of permitted circuits.

Theorem 4.1.7. *Let $r_{enc} \geq 1$, let C's additive fan-in be $\gamma_{mult}(R)$, let C's multiplicative fan-in be 2, and let the depth be at most*

$$\log_2 \log_2 r_{dec} - \log_2 \log_2(\gamma_{mult}(R) \cdot r_{enc}). \tag{5}$$

Then we have $C(\mathbf{x}_1, \ldots, \mathbf{x}_t) \in \mathcal{B}(r_{dec})$ for all $\mathbf{x}_1, \ldots, \mathbf{x}_t \in \mathcal{B}(r_{enc})$.

Proof. For a circuit C with d levels, let r_i be an upper bound on the Euclidean norm at level i and let $r_d = r_{enc}$. It follows that an addition gate at level i outputs a $\mathbf{v} \in R$ with $||\mathbf{v}|| \leq \gamma_{mult}(R) \cdot r_i$. For a multiplication gate at level i, we have $||\mathbf{v}|| \leq \gamma_{mult}(R) \cdot r_i^2$. So we find that always $||\mathbf{v}|| \leq \gamma_{mult}(R) \cdot r_i^2$ holds, which leads to $r_{dec} = r_0 \leq (\gamma_{mult}(R) \cdot r_{enc})^{2^d}$. This completes the proof. □

To put it simple, one wants to minimize r_{enc} and $\gamma_{mult}(R)$ and maximize r_{dec}, which will clearly increase the maximum depth of permitted circuits. Minimizing r_{enc} is a security issue and will be dealt with in section 4.1.3. Minimizing $\gamma_{mult}(R)$ depends on the chosen ring R. It is possible to choose f so that $\gamma_{mult}(R)$ becomes polynomial in n:

Theorem 4.1.8. *Let $R = \mathbb{Z}[x]/f(x)$ and suppose $f(x) = x^n - h(x)$ where $h(x)$ has degree at most $n - (n-1)/k$ for $k \geq 2$. Then, $\gamma_{mult}(R) \leq \sqrt{2n}(1 + 2n(\sqrt{(k-1)n} \cdot ||f||)^k)$.*

Before we can prove this theorem, we first need the following theorem.

Theorem 4.1.9. *Let $f(x) \in \mathbb{Z}[x]$ be monic, with degree n. Define $F(x) = x^n f(1/x)$ and $g(x) = F(x)^{-1} \mod x^{n-1}$. Then for two integer polynomials u and v of degree at most $n-1$ we have that*

$$||u \times v|| \leq \gamma_{mult}(R)||u|| \cdot ||v||$$

for

$$\gamma_{mult}(R) \leq \sqrt{2n}(1 + 2n||f|| \cdot ||g||).$$

Proof. (Thm 4.1.9) Let $t(x) = u(x)v(x)$, which has degree at most $2n-2$. Then for $w(x) = t(x) \mod f(x)$ we find $t(x) = q(x)f(x) + w(x)$, where $\deg(q(x)) \leq n-2$ and $\deg(w(x)) \leq n-1$. It is clear that $||t|| = ||u \times v||$, and for each coefficient of t we find by the Cauchy Schwarz inequality that its absolute value is bounded by $||u|| \cdot ||v||$. This implies that $||t|| \leq \sqrt{2n}||u|| \cdot ||v||$.
Define $T(x) = x^{2n-2}t(1/x)$, $Q(x) = x^{n-2}q(1/x)$, and $W(x) = x^{2n-2}w(1/x)$. Then $T(x) = Q(x)F(x) + W(x)$ and T, Q and F are all integer polynomials of the same norm as respectively t, q and f. The polynomial W with norm equal to the norm of p is divisible by x^{n-1}, which implies that $Q(x) = T(x)g(x) \mod x^{n-1}$.
We now find that $||Q|| \leq \sqrt{2n}||T|| \cdot ||g||$ for the same argument as holds for the norm of t. We ultimately find that

$$\begin{aligned} ||u \times v|| = ||r|| = ||W|| &\leq ||T|| + ||Q \cdot F|| \\ &\leq ||T|| + 2n||T|| \cdot ||g|| \cdot ||F|| \\ &= ||t||(1 + 2n||f|| \cdot ||g||) \\ &\leq ||u|| \cdot ||v||\sqrt{2n}(1 + 2n||f|| \cdot ||g||). \end{aligned}$$

□

Proof. (Thm 4.1.8) Define $F(x) = x^n f(1/x) = 1 - x^n h(1/x)$ and $H(x) = x^n h(1/x)$. Note that $H(x)$ is divisible by x^m for $m \geq (n-1)/k$. This implies that $1 - H(x)^k = 1 \mod x^{n-1}$ and $g(x) = F(x)^{-1} = \frac{1}{1-H(x)} = \frac{1-H(x)^k}{1-H(x)} \mod x^{n-1}$.
This now leads to

$$\begin{aligned} ||g(x)|| &\leq 1 + ||H|| + ||H^2|| + \ldots + ||H^{k-1}|| \\ &\leq 1 + ||H|| + \sqrt{2n}||H||^2 + \ldots + ((k-1)n)^{(k-1)/2}||H||^{k-1} \\ &\leq 1 + ||f|| + \sqrt{2n}||f||^2 + \ldots + ((k-1)n)^{(k-1)/2}||f||^{k-1} \\ &\leq ((\sqrt{(k-1)n}||f||)^k - 1)/((\sqrt{(k-1)n}||f||) - 1). \end{aligned}$$

Now since $||f|| < (\sqrt{(k-1)n}||f||) - 1$ together with Thm 4.1.9 we find $\gamma_{mult}(R) \leq \sqrt{2n}(1 + 2n(\sqrt{(k-1)n}||f||)^k)$. □

There are suitable polynomials $f(x)$ for which these theorems cannot be used to bound $\gamma_{mult}(R)$, because they do not meet the conditions of the theorem [Gen09a, p. 74], but it is generally a lot harder to determine an upper bound for $\gamma_{mult}(R)$. One easy example is $f(x) = x^n - 1$, which has $\gamma_{mult}(R) \leq \sqrt{n}$, but it is preferred to have $f(x)$ irreducible so that fractional ideals and inverses in $\mathbb{Q}[x]/f(x)$ can be used.
Concerning r_{dec}, maximizing is related to maximizing the parallelepiped $\mathcal{P}(\mathbf{B}_J^{sk})$, since r_{dec} is the radius of the largest sphere centered at the origin. So this depends strongly on the shape of $\mathbf{B}_J^{sk}$, if the basis is more orthogonal, the radius becomes larger. The following theorem clarifies this.

Recall the matrix norm; $||\mathbf{B}^*|| = \max\{||(\mathbf{B}^*)\mathbf{x}|| : ||\mathbf{x}|| = 1\}$. We then have the following theorem.

Theorem 4.1.10. *Let* $\mathbf{B}$ *be a lattice basis (represented by a matrix) and* $\mathbf{B}^* = (\mathbf{B}^{-1})^T$. *Let* r *be the radius of the largest sphere, centered at the origin, which can be circumscribed with* $\mathcal{P}(\mathbf{B})$. *Then* $r = 1/(2 \cdot ||\mathbf{B}^*||)$. *If* $||\mathbf{t}|| < r$; *then each coefficient of* $\mathbf{B}^{-1} \cdot \mathbf{t}$ *has magnitude at most 1/2.*

Proof. Since $||\mathbf{t}|| < r = 1/(2 \cdot ||\mathbf{B}^*||)$, we have $||\mathbf{t}|| \cdot ||\mathbf{B}^*|| < 1/2$. Each coefficient of $\mathbf{B}^{-1} \cdot \mathbf{t}$ is an inner product of $\mathbf{t}$ with a column vector of $\mathbf{B}^*$, and thus has magnitude at most $1/2$. This implies $\lfloor \mathbf{B}^{-1} \cdot \mathbf{t} \rceil = 0$ (rounding is performed entry-wise) thus $\mathbf{t} = \mathbf{t} \mod \mathbf{B}$, and thus $\mathbf{t} \in \mathcal{P}(\mathbf{B})$. Now suppose we have $||\mathbf{x}|| > 1/(2 \cdot ||\mathbf{B}^*||)$, for $\mathbf{x}$ parallel to $\mathbf{v}$, the longest vector in $\mathbf{B}^*$. Then $|(\mathbf{v}, \mathbf{x})| > 1/2$ and thus $\lfloor \mathbf{B}^{-1} \cdot \mathbf{x} \rceil \neq 0$. So then clearly $x \notin \mathcal{P}(\mathbf{B})$. □

This theorem implies that $r_{dec} = 1/(2 \cdot ||((\mathbf{B}_J^{sk})^{-1})^T||)$. Ideally, one would like to have an instantiation of $\text{IdealGen}_{\mathcal{E}}$ which results in a private basis which circumscribes a sphere with radius only polynomially shorter than the parallelepiped's diameter. In that case, the decrypting ability of the key is as large as possible. This can be achieved with a rotation basis on a vector $\mathbf{v}$ which is nearly parallel to $\mathbf{e}_1$. The following theorem describes this more formally [Gen09a, pp. 76-77].

Theorem 4.1.11. *For some* $s \in \mathbb{R}$, *let* $t \geq 4 \cdot n \cdot \gamma_{mult}(R) \cdot s$. *Suppose* $\mathbf{v} \in t \cdot \mathbf{e}_1 + \mathcal{B}(s)$ *and let* $\mathbf{B}$ *be the rotation basis of* $\mathbf{v}$. *Then* $\mathcal{P}(\mathbf{B})$ *circumscribes a ball of radius at least* $t/4$.

Proof. For $i = 0, \ldots, n-1$, define $\mathbf{v}_i = \mathbf{v} \times x^i$ and $\mathbf{z}_i = \mathbf{v}_i - t \cdot \mathbf{e}_i$. Now we have $||\mathbf{z}_i|| \leq ||\mathbf{z}_0 \times x^i|| \leq \gamma_{mult}(R) \cdot ||\mathbf{z}_0|| \leq \gamma_{mult}(R) \cdot s$. The last inequality follows from the fact that $\mathbf{z}_0$ is in the ball with radius s around $t \cdot \mathbf{e}_1$.
Now for every point $\mathbf{a}$ on the surface of the parallelepiped $\mathcal{P}(\mathbf{B})$, there must exist an i such that

$$\mathbf{a} = (\pm 1/2) \cdot \mathbf{v}_i + \sum_{j \neq i} x_j \cdot \mathbf{v}_j \tag{6}$$

holds for $x_j \in [-1/2, 1/2]$. So then we find that

$$|(\mathbf{a}, \mathbf{e}_i)| \geq t/2 - n \cdot \gamma_{mult}(R) \cdot s. \tag{7}$$

This implies in particular that $||\mathbf{a}|| \geq t/2 - n \cdot \gamma_{mult}(R) \cdot s = n \cdot \gamma_{mult}(R) \cdot s \geq t/4$. So every point on the surface has distance more than $t/4$ to the origin implying that the circumscribed ball has radius at least $t/4$. □

The above theorem yields an instantiation of $\text{IdealGen}_{\mathcal{E}}$ which provides a private basis which circumscribes a sphere with radius only polynomially shorter than the parallelepiped's diameter.

4.1.2 Bootstrappable Scheme

Unfortunately the scheme as presented in the previous section is not yet bootstrappable, as will become clear soon. To prepare for bootstrapping, first two small tweaks are performed on the scheme to reduce complexity. When the decryption complexity is analyzed it is still too large. Finally, the decryption circuit is squashed to achieve the bootstrappability.

Tweaks to the Initial Scheme

The first tweak lowers the precision that is needed for correct decryption at the expense of the amount of permitted circuits. Tweak two modifies the decryption step from matrix multiplications to ring multiplications. This reduces the public key size and the computation that is needed per gate during bootstrapping.

Tweak 1: Redefine the set of permitted ciphertexts $\mathcal{C}_\mathcal{E}$ by replacing r_{dec} with $r_{dec}/2$.

Purpose: This ensures that the ciphertexts are closer to the lattice J, so that less precision is needed for correct decryption.

The following theorem shows the effect of Tweak 1.

Theorem 4.1.12. *Let ψ be a valid ciphertext after Tweak 1, then each coefficient of $(\mathbf{B}_J^{sk})^{-1} \cdot \psi$ is within 1/4 of an integer.*

Proof. Since ψ is a valid ciphertext, we have that $\psi = \mathbf{x} + \mathbf{j}$ for some $\mathbf{x} \in \mathcal{B}(r_{dec}/2)$ and $\mathbf{j} \in J$. Then it follows that $(\mathbf{B}_J^{sk})^{-1} \cdot \psi = (\mathbf{B}_J^{sk})^{-1} \cdot \mathbf{x} + (\mathbf{B}_J^{sk})^{-1} \cdot \mathbf{j}$, where, by analogous arguments as in the proof of Theorem 4.1.10, the former term has coefficients with magnitude at most 1/4, and the latter is an integer vector since it is an element of R. □

It is clear that correct rounding is much more obvious with the tweak in place. The maximum depth of the evaluation circuit has been lowered to achieve this; by Theorem 4.1.7 the depth after tweak 1 is $\log\log(r_{dec}/2) - \log\log(\gamma_{mult}(R) \cdot r_{enc})$, which is $\log\log 2$ less than the original, which is not really significant.

Tweak 2: From $\mathbf{B}_I$ and $\mathbf{B}_J^{sk}$, compute a short vector $\mathbf{v}_J^{sk} \in J^{-1}$, such that there exists $\mathbf{u} \in 1 + I$ with $\mathbf{u} \times (\mathbf{v}_J^{sk})^{-1} \in 1 + I$. Also redefine the set of permitted ciphertexts $\mathcal{C}_\mathcal{E}$ by replacing r_{dec} with $2 \cdot r_{dec}/(n^{1.5} \cdot \gamma_{mult}(R)^2 \cdot ||\mathbf{B}_I||)$.

Purpose: With this tweak $\text{Decrypt}_\mathcal{E}$ changes from $\pi = (\psi \mod \mathbf{B}_J^{sk}) \mod \mathbf{B}_I = \psi - \mathbf{B}_J^{sk} \cdot \lfloor (\mathbf{B}_J^{sk})^{-1} \cdot \psi \rceil \mod \mathbf{B}_I$ to $\psi - \lfloor \mathbf{v}_J^{sk} \times \psi \rceil \mod \mathbf{B}_I$, where $\mathbf{v}_J^{sk} \in \mathbb{Q}[x]/f(x)$.

Again the depth of the evaluation circuit is lowered, but the change is still not significant. The following two theorems state the effect of Tweak 2.

Theorem 4.1.13. *Let $\mathbf{B}_I$ and $\mathbf{B}_J^{sk}$ defined as above be given. Then a short vector $\mathbf{v}_J^{sk} \in J^{-1}$ and a vector $\mathbf{u} \in 1 + I$, such that $\mathbf{u} \times (\mathbf{v}_J^{sk})^{-1} \in 1 + I$, can be computed in polynomial time. Furthermore, $||\mathbf{v}_J^{sk}|| \leq (n/2) \cdot \gamma_{mult}(R) \cdot ||((\mathbf{B}_J^{sk})^{-1})^T|| \cdot ||\mathbf{B}_I||$.*

To prove this Theorem, we first use the following ideas. Given a short vector in J^{-1} (resp. J^*), can we compute a short basis of J^* (resp. J^{-1})? This turns out to be the case. For a basis $\mathbf{B}_J$ with columns $\mathbf{u}_i$, we find that for $\mathbf{v} \in J^{-1}$, $\mathbf{v} \times \mathbf{u}_i \in R$ and thus $\mathbf{B}_\mathbf{v} \cdot \mathbf{B}_J$ is an integer basis. This implies that the rows of $\mathbf{B}_\mathbf{v}$ are a basis for J^*. To go from J^* to J^{-1}, we have the following Lemma [Gen09a, pp. 83-85].

Lemma 4.1.14. *Let $\mathbf{w} \in J^*$, where J^* is the dual of the ideal lattice J. Let*

$$\mathbf{v} = \sum_{i=0}^{n-1} x^i \sum_{j=i+1}^{n} f_j \cdot w_{j-i-1},$$

then $\mathbf{v} \in \mathbf{J}^{-1}$. Let $\mathbf{B}_\mathbf{v}$ be the rotation basis of $\mathbf{v}$. Then $||\mathbf{B}_\mathbf{v}|| \leq \sqrt{n} \cdot ||\mathbf{f}|| \cdot ||\mathbf{w}||$. This applies even when J is a fractional ideal.

Proof. (Lemma 4.1.14) Let the matrix $\mathbf{B}$ be the rotation basis of $\mathbf{v}$. We will show that the lowest row of $\mathbf{B}$ equals $\mathbf{w} = \{w_0, w_1, \ldots, w_{n-1}\}$. Denote the columns of $\mathbf{B}$ by $\mathbf{b}^{(k)} = \mathbf{v} \cdot x^k \mod f(x)$. Now we claim that

$$\mathbf{b}^{(k)} = \sum_{i=k}^{n-1} x^i \sum_{j=i+1}^{n} f_j \cdot w_{j-i-1+k} - \sum_{i=0}^{k-1} x^i \sum_{j=0}^{i} f_j \cdot w_{j-i-1+k},$$

which implies that the coefficient of x^{n-1} in $\mathbf{b}^{(k)}$ is w_k, i.e. the lowest row of $\mathbf{B}$ is $\mathbf{w}$. This claim clearly holds for $k = 0$. Now assume it holds for $k' - 1$. We have the following result.

$$\begin{aligned}
\mathbf{b}^{(k')} &= x \left(\sum_{i=k'-1}^{n-1} x^i \sum_{j=i+1}^{n} f_j \cdot w_{j-i-1+k'-1} - \sum_{i=0}^{k'-2} x^i \sum_{j=0}^{i} f_j \cdot w_{j-i-1+k'-1} \right) \mod f(x) \\
&= \sum_{i=k'}^{n} x^i \sum_{j=i}^{n} f_j \cdot w_{j-i-1+k'} - \sum_{i=1}^{k'-1} x^i \sum_{j=0}^{i-1} f_j \cdot w_{j-i-1+k'} \mod f(x) \\
&= x^n f_n w_{k'-1} + \sum_{i=k'}^{n-1} x^i \sum_{j=i}^{n} f_j \cdot w_{j-i-1+k'} - \sum_{i=1}^{k'-1} x^i \sum_{j=0}^{i-1} f_j \cdot w_{j-i-1+k'} \mod f(x) \\
&= x^n f_n w_{k'-1} + \sum_{i=k'}^{n-1} x^i \sum_{j=i}^{n} f_j \cdot w_{j-i-1+k'} - \sum_{i=1}^{k'-1} x^i \sum_{j=0}^{i-1} f_j \cdot w_{j-i-1+k'} - \sum_{i=0}^{n} x^i \cdot w_{k'-1} \cdot f_i
\end{aligned}$$

$$
\begin{aligned}
&= \sum_{i=k'}^{n-1} x^i \sum_{j=i}^{n} f_j \cdot w_{j-i-1+k'} - \sum_{i=1}^{k'-1} x^i \sum_{j=0}^{i-1} f_j \cdot w_{j-i-1+k'} - \sum_{i=0}^{n-1} x^i \cdot w_{k'-1} \cdot f_i \\
&= \sum_{i=k'}^{n-1} x^i \left(-f_i \cdot w_{k'-1} + \sum_{j=i}^{n} f_j \cdot w_{j-i-1+k'} \right) \\
&- \sum_{i=1}^{k'-1} x^i \left(f_i \cdot w_{k'-1} + \sum_{j=0}^{i-1} f_j \cdot w_{j-i-1+k'} \right) - w_{k'-1} \cdot f_0 \\
&= \sum_{i=k'}^{n-1} x^i \sum_{j=i+1}^{n} f_j \cdot w_{j-i-1+k'} - \sum_{i=0}^{k'-1} x^i \sum_{j=0}^{i} f_j \cdot w_{j-i-1+k'}
\end{aligned}
$$

This verifies the claim. Next we show that $\mathbf{v} \in J^{-1}$, by proving that $\mathbf{z} = \mathbf{v} \times \mathbf{x} \in R$ for any $\mathbf{x} \in J$. Let such an $\mathbf{x} \in J$ be given, then take the rotation bases $\mathbf{B_z}$, $\mathbf{B_v}$ and $\mathbf{B_x}$ of $\mathbf{z}$, $\mathbf{v}$ and $\mathbf{x}$ respectively. We know that $\mathbf{B_z} = \mathbf{B_v} \cdot \mathbf{B_x}$ and that lowest row of $\mathbf{B_z}$ which is $\mathbf{w} \cdot \mathbf{B_x}$ is an integer vector, since $\mathbf{w} \in J^*$.
Now assume that $\mathbf{z}$ is not an integer vector. Then there exists a largest index i such that $\mathbf{z}_i$ is not an integer. Consider $\mathbf{z}^{(n-i-1)} = x^{n-i-1} \cdot \mathbf{z} \mod f(x)$, which is one of the columns in $\mathbf{B_z}$. Now for $x^{n-i-1} \cdot \mathbf{z}$, the coefficients for $x^n, x^{n+1}, \ldots, x^{2n-i-1}$ are all integers, since i was the largest non-integer index. Since $f(x)$ is monic, we find that $x^{n-i-1} \cdot \mathbf{z} \mod f(x) = x^{n-i-1} \cdot \mathbf{z} - a(x)f(x)$, for some integer polynomial $a(x)$. This implies that the coefficient for x^{n-1} in $\mathbf{z}^{(n-i-1)}$ is also not integer, but this coefficient is in the bottom row of $\mathbf{B_z}$ which consists only of integers, a contradiction. So $\mathbf{z}$ is integral and thus in R.
It remains to be shown that $||\mathbf{B_v}|| \leq \sqrt{n} \cdot ||f|| \cdot ||\mathbf{w}||$. Since every coefficient of $\mathbf{v}$ is an inner product of two vectors with coefficients respectively in $\{f_0, \ldots, f_n\}$ and $\{w_0, \ldots, w_{n-1}\}$, each coefficient in $\mathbf{B_v}$ has magnitude at most $||\mathbf{f}|| \cdot ||\mathbf{w}||$ and thus $||\mathbf{B_v}|| \leq \sqrt{n} \cdot ||\mathbf{f}|| \cdot ||\mathbf{w}||$. □

Proof. (Theorem 4.1.13)[Gen09a, pp. 87 - 88] Let $\mathbf{w} \in J^*$ be a vector in the basis $\mathbf{B}_J^{sk}$. Then by Lemma 4.1.14 we can generate a vector $\mathbf{x} \in J^{-1}$ with corresponding rotation basis $\mathbf{B_x}$ with length at most $\sqrt{n} \cdot ||\mathbf{f}|| \cdot ||\mathbf{w}|| \leq \sqrt{n} \cdot ||\mathbf{f}|| \cdot ||((\mathbf{B}_J^{sk})^{-1})^T||$. Now from $\mathbf{B_x}$ and a vector in I of length at most $||\mathbf{B}_I||$, an independent set $\mathbf{B}_{J^{-1}I}$ of $(\mathbf{x}) \cdot I \subset J^{-1}I$ of length at most $n \cdot ||\mathbf{f}|| \cdot ||\mathbf{B}_I|| \cdot ||((\mathbf{B}_J^{sk})^{-1})^T||$ can be generated. Now take $\mathbf{v}_J^{sk} = \mathbf{e}_1 \mod \mathbf{B}_{J^{-1}I}$, which has length at most $(n/2) \cdot ||\mathbf{f}|| \cdot ||\mathbf{B}_I|| \cdot ||((\mathbf{B}_J^{sk})^{-1})^T|| \leq (n/2) \cdot \gamma_{mult}(R) \cdot ||((\mathbf{B}_J^{sk})^{-1})^T|| \cdot ||\mathbf{B}_I||$. Now since I and J are relatively prime (by definition), there exists a vector $\mathbf{r} \in J \cap (1+I)$. Let $\mathbf{u} = \mathbf{r} \times \mathbf{v}_J^{sk}$. Now since $\mathbf{v}_J^{sk} \in 1 + J^{-1}I$, we have $\mathbf{u} \in 1 + I$ and clearly also $\mathbf{u} \times (\mathbf{v}_J^{sk})^{-1} \in 1 + I$. □

Theorem 4.1.15. *Let ψ be a valid ciphertext after Tweak 2 that decrypts to π. Then $\pi = \psi - \lfloor \mathbf{v}_J^{sk} \times \psi \rceil \mod \mathbf{B}_I$. To decrypt correctly in all cases, the radius r'_{dec} of the sphere, circumscribed by the parallelepiped of the rotation basis of $\mathbf{v}_J^{sk}$, should be at least $2r_{dec}/n^{1.5}\gamma_{mult}(R)||\mathbf{B}_I||$ (by redefinition of r_{dec}).*

Proof. [Gen09a, pp. 87 - 88] We know that $\pi = (\psi \mod \mathbf{B}_J^{sk}) \mod \mathbf{B}_I = \psi - \mathbf{B}_J^{sk} \cdot \lfloor (\mathbf{B}_J^{\mathrm{sk}})^{-1} \cdot \psi \rceil \mod \mathbf{B}_I$. Now let $\mathbf{B}_J^{\dagger}$ be the rotation basis of $(\mathbf{v}_J^{\mathrm{sk}})^{-1}$. Since $\mathbf{v}_J^{\mathrm{sk}}$ generates a sub-lattice of J^{-1}, we have that $\mathbf{B}_J^{\dagger}$ generates a super-lattice of J. So now it remains to show that the sphere circumscribed by $\mathbf{B}_J^{\dagger}$ has large enough radius, denote this radius by r'_{dec}.
We find that

$$r'_{dec} \geq 1/(2\sqrt{n}||(\mathbf{B}_J^{\dagger})^{-1}||) = 1/(2\sqrt{n}||\mathbf{v}_J^{\mathrm{sk}}||) \geq 2r_{dec}/n^{1.5}\gamma_{mult}(R)||\mathbf{B}_I||,$$

where the first inequality follows from theorem 4.1.10. So we find that $\pi = \psi - (\mathbf{v}_J^{\mathrm{sk}})^{-1} \times \lfloor \mathbf{v}_J^{\mathrm{sk}} \cdot \psi \rceil$ decrypts correctly.
Now for $\mathbf{r}$ as in the proof of Theorem 4.1.13, we find that

$$\begin{aligned}(\mathbf{v}_J^{\mathrm{sk}})^{-1} \times \lfloor \mathbf{v}_J^{\mathrm{sk}} \times \psi \rceil &= \mathbf{r} \times (\mathbf{v}_J^{\mathrm{sk}})^{-1} \times \lfloor \mathbf{v}_J^{\mathrm{sk}} \times \psi \rceil \mod \mathbf{B}_I \\ &= \mathbf{u} \times \lfloor \mathbf{v}_J^{\mathrm{sk}} \times \psi \rceil \mod \mathbf{B}_I \\ &= \lfloor \mathbf{v}_J^{\mathrm{sk}} \times \psi \rceil \mod \mathbf{B}_I.\end{aligned}$$

□

To determine whether the scheme is bootstrappable yet, we will analyze the decryption complexity of the tweaked scheme.

Decryption Complexity of the Tweaked Scheme

We would like to express the decryption algorithm as a circuit and analyze its complexity. For decryption, one computes $(\psi - \mathbf{B}_J^{sk} \cdot \lfloor (\mathbf{B}_J^{sk})^{-1} \cdot \psi \rceil) \mod \mathbf{B}_I$. But as we have seen, after applying tweak 2, we can compute this as $(\psi - \lfloor \mathbf{B}_J^{sk2} \cdot \psi \rceil) \mod \mathbf{B}_I$, where $\mathbf{B}_J^{sk2}$ is the rotation basis of $\mathbf{v}_J^{sk}$. Also, we know from tweak 1 that the coefficients of $\mathbf{B}_J^{sk2} \cdot \psi$ are all within 1/4 of an integer.
First, we split this computation up into the following steps:

Step 1: Generate n vectors $\mathbf{x}_1, \ldots, \mathbf{x}_n$ with sum $\mathbf{B}_J^{sk2} \cdot \psi \bmod f$,

Step 2: From the vectors $\mathbf{x}_1, \ldots, \mathbf{x}_n$, generate integer vectors $\mathbf{y}_1, \ldots, \mathbf{y}_{n+1}$ with sum $\lfloor \sum \mathbf{x}_i \rceil$,

Step 3: Compute $\pi = \psi - \sum \mathbf{y}_i \mod \mathbf{B}_I$.

Later we will find that step 1 can be eliminated by the method with which the decryption circuit is squashed, so we will not look further into that. Concerning step 2, it is straightforward to select for $\mathbf{y}_1, \ldots, \mathbf{y}_n$ the integer parts of the $\mathbf{x}_i$ and for $\mathbf{y}_{n+1}$ the sum of the fractional parts of the $\mathbf{x}_i$ and then rounded. The problem now is the precision which is needed to give the correct result, i.e. the amount of significant fractional digits that are required to yield the correct result. This can be avoided by restricting the plaintext space to $\mathcal{M} = \{0, 1\}$ so that the operations become boolean operations. In this case, we find the following lemma.

Lemma 4.1.16. [Gen09a, pp. 93-94] *For $i \in [1,t]$, let $a_i = (\ldots, a_{i,1}, a_{i,0}, a_{i,-1}, \ldots)$ be a real number given in binary representation* mod $\mathbf{B}_I$ *with in addition the promise that $\sum_i a_i \mod 1 \in [-1/4, 1/4]$. There exists a* mod-$\mathbf{B}_I$ *circuit C that generates $t+1$ integers $z_1, \ldots, z_{t+1}$ whose sum is $\lfloor \sum_i a_i \rceil$, such that if the generalized circuit $g(C)$'s inputs are in $\mathcal{B}(r_{in})$, then its outputs are in $\mathcal{B}(r_{out})$ for:*

$$r_{out} \leq (\gamma_{mult}(R) \cdot n \cdot ||\mathbf{B}_I|| \cdot (1 + \gamma_{mult}(R) \cdot r_{in})^t \cdot t)^{\text{polylog}(t)}.$$

Now for $||\mathbf{B}_I|| \leq r_{in}$, $t \leq n$, and $\gamma_{mult}(R) = n^{\Omega(1)}$, we have:

$$r_{out} \leq (\gamma_{mult}(R) \cdot r_{in})^{t+\text{polylog}(t)}.$$

Proof. We will only give a sketch of the proof. The details are not important for understanding, and are available in [Gen09a, pp. 93-94] for the interested reader.
The idea of the proof is to take for the integer parts of the a_i's as the first t integers. Then we note that since all the a_i's are within $1/4$ of an integer, it suffices to take the first $T = \lceil \log_2 t \rceil + 2 \rceil$ bits of the fractional part. Denote these truncations by b_i. The last integer is taken to be the sum of these truncations, rounded to the nearest integer. These $t+1$ integers indeed sum up to the correct result.
To sum up the b_i, it is easier to compute the Hamming weight c_j of $(b_{1,-j}, b_{2,-j}, \ldots, b_{t,-j})$ for each $j \in [1,T]$. Then the rounded sum of the b_i's can be computed by computing $\lfloor \sum_{j=1}^{T} 2^{-j} c_j \rceil$. This sum only consists of T elements, instead of t.
The norm of the values output by the circuit that computes these Hamming weights is bounded by $n \cdot ||\mathbf{B}_I|| \cdot (1 + \gamma_{mult}(R) \cdot r_{in})^t \cdot t$, where $n \cdot ||\mathbf{B}_I||$ is used as upper bound for the length of elements in $R \bmod \mathbf{B}_I$. The sum of these T Hamming weights (each with $O(\log T)$ bits) then can be computed by a $O(\log T)$-depth circuit. This, together with the results in the proof of theorem 4.1.7 yields the result. □

Since for step 2 we have $t = n$, we find that $\log\log(r_{out}) - \log\log(\gamma_{mult}(R) \cdot r_{in}) \leq \log(n + \text{polylog}(n))$, which is not enough to show that the scheme is bootstrappable, since our somewhat homomorphic scheme can only evaluate circuits with depth $O(\log n)$. This requires some changes to the scheme, which will be presented in the next section. But first, we will look at step 3 using the following lemma:

Lemma 4.1.17. [Gen09a, pp. 95-96] *The term $\psi - \sum \mathbf{y}_i \mod \mathbf{B}_I$ can be computed from a binary representation of the terms of the expression using only a constant depth circuit with polynomial fan-in* $\text{Add}_{\mathbf{B}_I}$ *gates and constant fan-in* $\text{Mult}_{\mathbf{B}_I}$ *gates.*

Proof. We will only give the main idea behind the proof, the details can be found in [Gen09a, pp. 95-96]. The idea is to use the mod-$\mathbf{B}_I$ gates for multiplication and addition. If the integers are given in mod-$\mathbf{B}_I$ representation, the summation can clearly be performed by constant depth circuit with polynomial fan-in $\text{Add}_{\mathbf{B}_I}$ gates.
To do this, we first need to translate binary representation to mod-$\mathbf{B}_I$ representation. The advantage here is that the largest part of this translation can be done by pre-computation, which does not yield extra circuit depth.

This translation is computed as follows. Consider $\mathbf{y} = \mathbf{y}_1$. Represented as binary number, the ith coefficient is given as $y_{ix} \times \mathbf{e}_1, \ldots, y_{i0} \times \mathbf{e}_1 \mod \mathbf{B}_I$, where $y_i = \sum_{j=0}^{x} 2^j \cdot y_{ij}$ (and thus $y_{ij} \in \{0, 1\}$). So we find that

$$\mathbf{y} = \sum_{i \in [1,n], j \in [0,x]} 2^j \times (y_{ij} \times \mathbf{e}_1) \times \mathbf{e}_i \mod \mathbf{B}_I$$

is the mod-$\mathbf{B}_I$ representation of $\mathbf{y}_1$.
Now we can pre-compute $\mathbf{a}_j = 2^j \mod \mathbf{B}_I$ for each $j \in [0, x]$, so that the computation of the mod-$\mathbf{B}_I$ representation can be done by using a constant depth circuit of polynomial fan-in $\text{Add}_{\mathbf{B}_I}$ gates and constant fan-in $\text{Mult}_{\mathbf{B}_I}$ gates. □

Squashing the Decryption Circuit

We have seen that the decryption circuit of our scheme has a too high complexity to allow bootstrapping. We can fix this by squashing the decryption circuit. Abstractly, this is done by delegating the computationally intensive part of the decryption to the encrypter and leaving only a low complexity decryption circuit for the decrypter. This requires an additional hardness assumption, which will be covered in section 4.1.3.
Concretely, we add two more algorithms to the scheme, namely $\text{SplitKey}_{\mathcal{E}}$ and $\text{ExpandCT}_{\mathcal{E}}$. Let $\mathcal{E}^*$ denote the original scheme and $\mathcal{E}$ the new, squashed scheme. Here is an overview of the algorithms in our new scheme:

$\text{KeyGen}_{\mathcal{E}}(\lambda)$**:** Runs $(\text{pk}^*, \text{sk}^*) = \text{KeyGen}_{\mathcal{E}^*}(\lambda)$ and $(\text{sk}, \tau) = \text{SplitKey}_{\mathcal{E}}(\text{sk}^*, \text{pk}^*)$. The secret key is sk and the public key is (pk^*, τ).

$\text{SplitKey}_{\mathcal{E}}(\text{sk}^*, \text{pk}^*)$**:** Extracts the vector $\mathbf{v}_J^{sk^*}$ from sk^* as defined in Theorem 4.1.13. Now let $\gamma_{set}(n)$ and $\gamma_{subset}(n)$ be functions, where the former is $\omega(n)$ (as $n \to \infty$, $\gamma_{set}(n) \geq n \cdot k$, for every $k > 0$) and $\text{poly}(n)$ and the latter is $\omega(1)$ and $o(n)$. It now outputs (sk, τ) where:

- τ is a set of $\gamma_{set}(n)$ vectors $\mathbf{t}_1, \ldots, \mathbf{t}_{\gamma_{set}(n)}$ chosen uniformly at random in $J^{-1} \mod \mathbf{B}_I$, except that there exists a subset S of cardinality $\gamma_{subset}(n)$ that sums up to $\mathbf{v}_J^{sk^*} + I$.
- sk includes $\gamma_{subset}(n)$ binary vectors s_i of dimension $\gamma_{set}(n)$, where each s_i has all coefficients equal to zero except for the j_ith coefficient, which is equal to one. Here j_i is the ith member of S. So these vectors encode the secret subset S.

$\text{ExpandCT}_{\mathcal{E}}(\text{pk}, \psi^*)$**:** Outputs $\mathbf{c}_i = \mathbf{t}_i \times \psi^* \mod \mathbf{B}_I$ for $i \in [1, \gamma_{set}(n)]$. These terms are represented in binary, as explained in Lemma 4.1.17.

$\text{Encrypt}_{\mathcal{E}}(\text{pk}, \pi)$**:** Runs $\psi^* = \text{Encrypt}_{\mathcal{E}^*}(\text{pk}^*, \pi)$ and $\mathbf{c} = \text{ExpandCT}_{\mathcal{E}}(\text{pk}, \psi^*)$, then outputs $\psi = (\psi^*, \mathbf{c})$.

$\mathrm{Add}_{\mathcal{E}}(\mathrm{pk}, \psi_1, \psi_2)$**:** Extracts (ψ_1^*, ψ_2^*) from (ψ_1, ψ_2), computes $\psi^* = \mathrm{Add}_{\mathcal{E}^*}(\mathrm{pk}^*, \psi_1^*, \psi_2^*)$ and $\mathbf{c} = \mathrm{ExpandCT}_{\mathcal{E}}(\mathrm{pk}, \psi^*)$, then outputs $\psi = (\psi^*, \mathbf{c})$. The modification of $\mathrm{Mult}_{\mathcal{E}^*}$ is analogous.

$\mathrm{Decrypt}_{\mathcal{E}}(\mathrm{sk}, \psi)$**:** Performs the following steps:

Step 1: Set the vectors $\mathbf{x}_i = \sum_{j=1}^{\gamma_{set}(n)} s_{ij} \cdot \mathbf{c}_j$, where the s_{ij} are the binary vectors from the secret key and $\mathbf{c}_j$ the vectors included in the ciphertext.

Step 2: Generate integer vectors $\mathbf{y}_1, \ldots, \mathbf{y}_{\gamma_{subset}(n)+1}$ with sum $\lfloor \sum \mathbf{x}_i \rceil$.

Step 3: Compute $\pi = \psi - (\sum \mathbf{y}_i) \mod \mathbf{B}_I$.

It then outputs π.

The decryption step is more or less the same as before, but with the difference that it only adds up $\gamma_{subset}(n)$ numbers instead of n, which is a sub-linear quantity. This change is sufficient to achieve bootstrappability as shown in the following theorem.

Theorem 4.1.18. [Gen09a, p. 102] *The scheme $\mathcal{E}$ as presented above, is bootstrappable when*

$$\gamma_{subset}(n) \cdot \log^{c_1} \gamma_{subset}(n) \leq \left(\frac{\log(r_{dec}/m)}{2^{c_2} \cdot \log(\gamma_{mult}(R) \cdot r_{enc})} \right),$$

where $\log^{c_1} \gamma_{subset}(n)$ is the polylog term in Lemma 4.1.16, m depends on the use of the tweaks, and c_2 is a constant representing the depth needed in a circuit having $\mathrm{Add}_{\mathbf{B}_I}$ gates with $\gamma_{mult}(R)$ fan-in and $\mathrm{Mult}_{\mathbf{B}_I}$ gates with constant fan-in to sequentially perform $\mathrm{Decrypt}_{\mathcal{E}}$ Steps 1 and 3 and a NAND gate.

Proof. We know that, as shown in the proof of Theorem 4.1.7, for a circuit of depth c with inputs in $\mathcal{B}(r)$, the outputs are in $\mathcal{B}((\gamma_{mult}(R) \cdot r)^{2^c})$. Now from Lemma 4.1.16 it follows that the if the inputs to the NAND-augmented circuit are in $\mathcal{B}(r_{enc})$, then the output is in

$$(\gamma_{mult}(R) \cdot r_{enc})^{2^{c_2} \cdot (\gamma_{subset}(n) \operatorname{polylog}(\gamma_{subset}(n)))}.$$

The result follows when this value is at most r_{dec}/m. □

Obviously now that we have a bootstrappable somewhat homomorphic encryption scheme, we can use the construction as explained in Section 3. This leads to a fully homomorphic encryption scheme.

4.1.3 Security

Now the scheme is given in its concrete form, we will analyze its security. But first there are some comments to be taken care of, namely minimizing r_{enc} and the additional hardness assumption required for the squashing of the decryption circuit.

Minimizing r_{enc}
As stated in section 4.1.1, we want to minimize r_{enc} as far as possible without compromising security to enlarge the depth of the allowed circuits. Recall that we defined r_{enc} to be the smallest radius such that $X_{enc} \subseteq \mathcal{B}(r_{enc})$, where X_{enc} is the image of $\text{Samp}_{\mathcal{E}}$. So to state something about r_{enc}, we must first make $\text{Samp}_{\mathcal{E}}$ more concrete.
In $\text{Encrypt}_{\mathcal{E}}$, first a sample is taken from the coset $\pi + I$, which is then reduced modulo the public basis of J. It is important for security that $\pi + I \mod J$ seems to be taken uniformly in J and not according to some distribution. This can be related to the Bounded Distance Decoding Problem (BDDP), which is defined as follows.

Definition 4.1.19. *(BDDP)* [Gen09a, pp. 77-78] *Fix a polynomial ring $R = \mathbb{Z}[x]/(f(x))$, an algorithm* IdealGen *that samples a basis of an ideal in R and an algorithm* Samp_1 *that efficiently samples $\mathbb{Z}^n$. The challenger sets $b \overset{R}{\in} \{0,1\}$ and $(\mathbf{B}+J^{\text{sk}}, \mathbf{B}_J^{\text{sk}}) \overset{R}{=} \text{IdealGen}(R, \mathbf{B}_I)$. If $b = 0$, it sets $\mathbf{r} \overset{R}{=} \text{Samp}_1(R)$ and $\mathbf{t} = \mathbf{r} \mod \mathbf{B}_J^{\text{pk}}$. If $b = 1$, it samples $\mathbf{t}$ uniformly from $R \mod \mathbf{B}_J^{\text{pk}}$. The challenge: guess b given $(\mathbf{t}, \mathbf{B}_J^{\text{pk}})$.*

Now let $\mathbf{s}$ be a generator of I. We then define $\text{Samp}_{\mathcal{E}}$ as follows.

$\text{Samp}_{\mathcal{E}}(\mathbf{B}_I, \mathbf{x})$:
    $\mathbf{r} = \text{Samp}_1(R)$
    Output $\mathbf{x} + \mathbf{r} \times \mathbf{s}$.

We then have the following theorem.

Theorem 4.1.20. *Suppose we have an algorithm $\mathcal{A}$ that (t, ε)-breaks $\mathcal{E}$ with* $\text{Samp}_{\mathcal{E}}$ *as given above. Then there exists an algorithm $\mathcal{B}$ which, using $\mathcal{A}$, (t', ε')-breaks BDDP with $t' \approx t$, and $\varepsilon' = \varepsilon/2$.*

Proof. The challenger randomly generates $b \in \{0,1\}$ and generates a keypair. It generates $\mathbf{t}$ according to b and sends $(\mathbf{t}, \mathbf{B}_J^{\text{pk}})$ to $\mathcal{B}$. $\mathcal{B}$ sends $\mathcal{A}$ the public key corresponding to $\mathcal{E}$ composed form the instance the challenger generated.
$\mathcal{A}$ requests a challenge ciphertext on π_0, π_1, $\mathcal{B}$ sets $\beta \overset{R}{\in} \{0,1\}$ and answers with $\psi = \pi_\beta + \mathbf{t} \times \mathbf{s} \mod \mathbf{B}_J^{\text{pk}}$. $\mathcal{A}$ sends back a guess β' and $\mathcal{B}$ sends guess $b' = \beta \oplus \beta'$ to the challenger.
If $b = 0$, $\mathbf{r}$ was chosen according to Samp_1, so the ciphertext ψ was of the correct form. In this case $\mathcal{A}$ has an advantage of ε, which gives $\mathcal{B}$ an advantage of ε.
If $b = 1$, $\mathbf{t}$ is chosen uniformly random modulo $\mathbf{B}_J^{\text{pk}}$ and since I and J are coprime, $\mathbf{t} \times \mathbf{s}$ is also uniformly random modulo $\mathbf{B}_J^{\text{pk}}$. Thus ψ is a uniformly random element of $R \mod \mathbf{B}_J^{\text{pk}}$, independent of the choice of β. It is clear that $\mathcal{A}$ now has advantage 0, which gives $\mathcal{B}$ also advantage 0.
Overall, we find that $\mathcal{B}$ has advantage $\varepsilon/2$. □

Let $||\mathbf{r}|| < \ell_{samp}$, for some number ℓ_{samp}, with $\mathbf{r}$ drawn as according to Samp_1. Then one has

$$r_{enc} = \max\{||\mathbf{x} + \mathbf{r} \times \mathbf{s}||\} \leq n||\mathbf{B}_I|| + \sqrt{n}\ell_{samp}||\mathbf{B}_I||.$$

Now we can choose $\mathbf{s}$ short as for instance $\mathbf{s} = 2\mathbf{e}_1$ (as is the case in the Smart-Vercauteren and Gentry-Halevi variants). The hardness of BDDP depends on both ℓ_{samp} and on $\lambda_1(J)$ (the shortest vector in J) and in particular its size compared to ℓ_{samp}. For instance for $\lambda_1(J)/\ell_{samp} \geq 2^n$ one can find the closest vector to $\mathbf{t}$ in polynomial time with the LLL reduction algorithm.
For $\lambda_1(J) = 2^{O(\sqrt{n})}$ and $\ell_{samp} \approx n$, no known attacks run in polynomial time.

Additional Hardness Assumption
As stated, including τ in the public key requires an additional hardness assumption. This relies on the following abstract problem.

Definition 4.1.21. *(Splitkey Distinguishing Problem)* [Gen09a, pp. 104-105] *The challenger sets* $(\mathrm{sk}^*, \mathrm{pk}^*) = \mathrm{KeyGen}_{\mathcal{E}^*}$ *and randomly chooses* $b \in \{0, 1\}$. *We have that* sk^* *includes the secret vector* $\mathbf{v}_J^{\mathrm{sk}}$. *Now if* $b = 0$, *it then sets* τ *to be a set of* $\gamma_{set}(n)$ *vectors uniformly random in* $J^{-1} \mod \mathbf{B}_I$ *such that a subset of cardinality* $\gamma_{subset}(n)$ *of these vectors sums up to* $\mathbf{v}_J^{\mathrm{sk}} + I$. *If* $b = 1$, *it sets* τ *in the same way, except that a subset of cardinality* $\gamma_{subset}(n)$ *sums up to* $0 + I$. *The challenge is to guess* b *given* $(\tau, \mathrm{sk}^*, \mathrm{pk}^*)$.

Theorem 4.1.22. *Suppose there exists an algorithm* $\mathcal{A}$ *that* (t, ε)*-breaks the semantic security of* $\mathcal{E}$. *Then there exists algorithms* $\mathcal{B}_0$ *and* $\mathcal{B}_1$, *such that either* $\mathcal{B}_0$ $(t', \varepsilon/3)$*-solves the* SplitKey *Distinguishing problem, or* $\mathcal{B}_1$ $(t', \varepsilon/3)$*-breaks the semantic security of* $\mathcal{E}^*$.

Proof. Denote by Game 0 the semantic security game against $\mathcal{E}$ and let Game 1 be like Game 0, except that the challenger runs SplitKey with $\mathbf{v}_J^{\mathrm{sk}} = 0$ in sk^* instead of the normal sk^*. $\mathcal{A}$'s advantage in Game 0 is ε, let $\mathcal{A}$'s advantage in Game 1 be ε'.
Now $\mathcal{B}_0$ runs as follows. The challenger randomly generates a bit b and sends a SplitKey Distinguishing Problem instance $(\tau, \mathrm{sk}^*, \mathrm{pk}^*)$ to $\mathcal{B}_0$. $\mathcal{B}_0$ then sends $\mathrm{pk} = (\mathrm{pk}^*, \tau)$ to $\mathcal{A}$. Now when $\mathcal{A}$ asks for a challenge ciphertext on π_0, π_1, $\mathcal{B}_0$ generates a bit β and replies with $\psi = \mathrm{Encrypt}_{\mathcal{E}}(\mathrm{pk}, \pi_\beta)$. $\mathcal{A}$ answers with a guess bit β' and $\mathcal{B}_0$ forwards $b' = \beta' \oplus \beta$ to the challenger.
This game is exactly distributed as Game b, so $\mathcal{B}_0$'s advantage is at least $|\varepsilon - \varepsilon'|/2$.
$\mathcal{B}_1$ does the following. It obtains an $\mathcal{E}^*$ public key from the challenger. Then it sets τ to be a set of $\gamma_{set}(n)$ vectors uniformly random in $J^{-1} \mod \mathbf{B}_I$ such that a subset of cardinality $\gamma_{subset}(n)$ of these vectors sums up to $0 + I$ and sends $pk = (\mathrm{pk}^*, \tau)$ to $\mathcal{A}$. $\mathcal{A}$ asks for a challenge ciphertext on π_0, π_1 and $\mathcal{B}_1$ forwards this to the challenger. The challenger sends back ψ^*, $\mathcal{B}_1$ sets π to include ψ^* and $\mathrm{ExpandCT}_{\mathcal{E}}(\mathrm{pk}, \psi^*)$ and sends this to $\mathcal{A}$. When $\mathcal{A}$ guesses bit b', $\mathcal{B}_1$ simply forwards this guess to the challenger.
The distributions in this game are exactly as in Game 1 and $\mathcal{B}_1$ has the same advantage as $\mathcal{A}$, namely ε'.
Now we either have $\varepsilon' < \varepsilon/3$ or $\varepsilon' \geq \varepsilon/3$. In the former case this leads to $\mathcal{B}_0$ having advantage at least $\varepsilon/3$ and the latter case leads to $\mathcal{B}_1$ having advantage at least $\varepsilon/3$. □

The Splitkey Distinguishing Problem is not very intuitive, but it can be related to the Sparse Subset Sum Problem, which is well researched [Gen09a, chap. 11]. The details

are omitted here. The best known attack on SSSP is exponential in $\gamma_{subset}(n)$, as long as $\gamma_{set}(n)$ is large enough, for instance about $2\log(\det(IJ))$.

Choice of Parameters
As shown in Theorem 4.1.18, the scheme is bootstrappable when

$$\gamma_{subset}(n) \cdot \log^{c_1} \gamma_{subset}(n) \leq \left(\frac{\log(r_{dec}/m)}{2^{c_2} \cdot \log(\gamma_{mult}(R) \cdot r_{enc})} \right).$$

We want to choose $\gamma_{subset}(n)$ as large as possible, so we take $\gamma_{mult}(R)$, r_{enc} and m as small as possible (i.e. polynomial in n), and r_{dec} is then approximately $2^{\gamma_{subset}(n)}$. The approximation factor of BDDP is at least r_{dec}/r_{enc}, which thus is about $2^{\gamma_{subset}(n)}$. Solving this problem takes about $2^{n/\gamma_{subset}(n)}$ time with currently known attacks [Gen09a, pp. 113]. So if we choose $\gamma_{subset}(n) \approx \lambda$, the security parameter, we obtain about 2^{λ} security for both problems against known attacks. This does however require the lattice dimension to be $n \approx \lambda^2$.

4.2 The Smart-Vercauteren Variant

The scheme as presented by Smart and Vercauteren [SV10] is based on the scheme by Gentry as presented in Section 4.1, but is optimized with respect to the size of both the public and private key, as well as the size of the ciphertext. Using the construction by Gentry as presented in Section 3, we will first introduce the somewhat homomorphic scheme and then present the fully homomorphic scheme. Finally, we will compare the scheme to the original scheme and analyze its security.

4.2.1 The Somewhat Homomorphic Scheme

For simplicity, the scheme is first presented with plaintext space $\mathcal{M} = \{0, 1\}$. As usual, the scheme consists of five algorithms, namely KeyGen, Encrypt, Decrypt, Add, Mult. The scheme contains three parameters, (n, η, μ), the settings of these parameters will be discussed later. As it is based on Gentry's scheme, also this scheme is defined on the polynomial ring $R = \mathbb{Z}[x]/f(x)$, for $f(x) \in \mathbb{Z}[x]$ monic and of degree n.
Like the scheme by Gentry, this variant is also based on ideal lattices defined by a polynomial ring. The algorithms will first be described and are discussed thereafter.

KeyGen :

- Repeat until p is prime:
  - $S(x) \xleftarrow{R} \mathcal{B}_{\infty,n}(\eta/2)$
  - $g(x) = 1 + 2 \cdot S(x)$
  - $p = \mathrm{res}(g(x), f(x))$
- $D(x) = \gcd(g(x), f(x))$ over $\mathbb{F}_p[x]$
- Let $\alpha \in \mathbb{F}_p$ be the unique root of $D(x)$
- Find $Z(x) = \sum_{i=0}^{n-1} z_i x^i \in \mathbb{Z}[x]$ for which $Z(x)g(x) = p \mod f(x)$ holds
- $B = \langle z_0 \rangle_{2p}$
- Now we have public key $\mathrm{pk} = (p, \alpha)$ and private key $\mathrm{sk} = (p, B)$.

Encrypt(π, pk) :

- $R(x) \xleftarrow{R} \mathcal{B}_{\infty,n}(\mu/2)$
- $\Psi(x) = \pi + 2 \cdot R(x)$
- $\psi = \langle \Psi(\alpha) \rangle_p$
- Output ψ.

Decrypt(ψ, sk) :

- $\pi = \langle (\psi - \lfloor \psi \cdot B/p \rceil) \rangle_2$

- Output π.

Add $s(\psi_1, \psi_2, \text{pk})$:

- $\psi_3 = \langle \psi_1 + \psi_2 \rangle_p$
- Output ψ_3.

Mult$(\psi_1, \psi_2, \text{pk})$:

- $\psi_3 = \langle \psi_1 \cdot \psi_2 \rangle_p$
- Output ψ_3.

These algorithms require some analysis to determine how the parameters should be chosen and the amount of operations which can be performed while decryption remains correct. Per algorithm we will give a more detailed description.

KeyGen: the algorithm produces a small generator $\gamma = g(\theta)$ for a residue degree one prime ideal in the number field K defined by the monic irreducible polynomial $f(x)$.
Given our prime ideal $\mathfrak{p} = \gamma \cdot \mathbb{Z}[\theta]$ we need to find the two element presentation, which forms the public key. The two element presentation is chosen because it is significantly smaller than storing the entire HNF.
To achieve this, we must find the correct root α of $f(x)$. It is obvious that $\gamma \in \mathfrak{p}$, since it generates it, so $\gamma \mod \mathfrak{p} \equiv 0$ and thus we have $g(\alpha) \equiv 0 \mod p$. This implies that $f(x)$ and $g(x)$ have at least one root in common. Since $\mathfrak{p}$ is a residue degree one prime ideal, there can only exist one common root, otherwise γ would generate two different prime ideals $\langle p, \theta - \alpha_1 \rangle$ and $\langle p, \theta - \alpha_2 \rangle$, which is clearly impossible.
Now that we have established that f and g only have a single root in common, the greatest common divisor $D(x)$ of these polynomials suffices to find this root, since it will be of degree one and have α as only root.
So we have found the two element representation $\mathfrak{p} = p \cdot \mathbb{Z}[\theta] + (\theta - \alpha) \cdot \mathbb{Z}[\theta]$. KeyGen also computes $Z(x) \in \mathbb{Z}[x]$, for which it holds that $Z(x)g(x) \equiv p \mod f(x)$. This value is of importance for decryption, as we will see soon.

Encrypt: to encrypt a message π, i.e. a single bit, we generate a random polynomial $R(x)$ with $||R(x)||_\infty \leq \mu/2$ and $\deg(R(x)) = n-1$. Now the ciphertext is computed by adding π to $R(x)$ to form $\Psi(x)$, which is then reduced modulo $\mathfrak{p}$, i.e. $\Psi(\theta) \mod \mathfrak{p} \equiv \Psi(\alpha) \mod p$. We denote this value, which is the ciphertext, by ψ. Note that since we have $\Psi(\alpha) - \psi \equiv 0 \mod p$, we know $(\Psi(\theta) - \psi) \in \mathfrak{p}$.

Decrypt: we are given a ciphertext ψ for which it holds that $\Psi(\theta) - \psi \in \mathfrak{p}$, for some unknown $\Psi(\theta)$. Note that recovering the message is trivial after we have found $\Psi(\theta)$.
Now we know that $\mathfrak{p}$ is a principal ideal, so it has a single generator. Therefore, we can find a $q(\theta) \in \mathbb{Z}[\theta]$ such that $\Psi(\theta) - \psi = q(\theta)\gamma = q(\theta)g(\theta)$. Remember that in KeyGen we

computed $Z(x)$ such that $Z(x)g(x) \equiv p \mod f(x)$. So the inverse of $g(\theta)$ equals $Z(\theta)/p$. Hence we find

$$-\psi Z(\theta)/p = q(\theta) - \Psi(\theta)Z(\theta)/p. \tag{8}$$

Now if $||\Psi(\theta)Z(\theta)/p||_\infty < 1/2$, simply rounding $-\psi Z(\theta)/p$ will yield $q(\theta)$. Since we know $||\Psi(\theta)||_\infty < \mu/2$, we will have to find a bound for $Z(\theta)$. To do this, we first introduce some terminology.

Sylvester matrix For two polynomials $f(x)$ and $g(x)$, of degree n and $m < n$ respectively, for which it holds that $\gcd(f(x), g(x)) = 1$ over $\mathbb{Q}[x]$, we can find two polynomials $t(x), s(x) \in \mathbb{Q}[x]$ such that $t(x)f(x) + s(x)g(x) = 1$. We will show that we can find these polynomials easily using the Sylvester Matrix, which is defined as follows.

Definition 4.2.1. *For two polynomials $f(x) = \sum_{i=0}^{n-1} f_i x^i$ and $g(x) = \sum_{i=0}^{m-1} g_i x^i$, we define the* Sylvester matrix *to be the following matrix.*

$$Syl(g,f) = \begin{pmatrix} \overbrace{\begin{matrix} g_{m-1}z & g_{m-2} & g_{m-3} & \cdots & g_1 & g_0 \\ 0 & g_{m-1} & g_{m-2} & \cdots & g_2 & g_1 \\ \vdots & \ddots & \ddots & \ddots & & \ddots \\ 0 & \cdots & 0 & g_{m-1} & g_{m-2} & \cdots \\ 0 & 0 & \cdots & 0 & g_{m-1} & g_{m-2} \end{matrix}}^{n} \\ \hline \underbrace{\begin{matrix} f_{n-1} & f_{n-2} & f_{n-3} & \cdots & f_1 & f_0 \\ 0 & f_{n-1} & f_{n-2} & \cdots & f_2 & f_1 \\ \vdots & \ddots & \ddots & \ddots & & \ddots \\ 0 & \cdots & 0 & f_{n-1} & f_{n-2} & \cdots \\ 0 & 0 & \cdots & 0 & f_{n-1} & f_{n-2} \end{matrix}}_{m} \end{pmatrix} \begin{matrix} \overbrace{\begin{matrix} 0 & 0 & \cdots & 0 \\ g_0 & 0 & \cdots & 0 \\ \ddots & \ddots & \ddots & \vdots \\ g_2 & g_1 & g_0 & 0 \\ \cdots & g_2 & g_1 & g_0 \end{matrix}}^{n} \\ \begin{matrix} 0 & 0 & \cdots & 0 \\ f_0 & 0 & \cdots & 0 \\ \ddots & \ddots & \ddots & \vdots \\ f_2 & f_1 & f_0 & 0 \\ \cdots & f_2 & f_1 & f_0 \end{matrix} \end{matrix}$$

With the Sylvester matrix, we have the following property:

$$Syl(g,f)^T \cdot \begin{pmatrix} s_{n-1} \\ \vdots \\ s_0 \\ t_{m-1} \\ \vdots \\ t_0 \end{pmatrix} = \begin{pmatrix} 0 \\ \\ \vdots \\ \\ 0 \\ 1 \end{pmatrix}, \tag{9}$$

where s_i and t_i are the coefficients of $s(x)$ and $t(x)$ respectively. This property can be shown with a straightforward induction proof. We are now ready for the following lemma.

Lemma 4.2.2. *Let $f(x), g(x) \in \mathbb{Z}[x]$, with $f(x)$ monic,* $\deg(f) = n$, $\deg(g) = m < n$, $\mathrm{res}(f,g) = p$ *prime. Then there exists a polynomial $Z(x) \in \mathbb{Z}[x]$ with $Z(x)g(x) = p \mod f(x)$ and $||Z(x)||_\infty \leq ||g(x)||_2^{n-1} ||f(x)||_2^m$.*

Proof. We have $\gcd(f(x), g(x)) = 1$ over $\mathbb{Q}[x]$, so there exists polynomials $s(x), t(x) \in \mathbb{Q}[x]$ with $\deg(s) < n$, $\deg(t) < m$ such that $s(x)g(x) + t(x)f(x) = 1$. We are only interested in the coefficients for $s(x)$. These can be found using Cramer's rule, i.e.

$$s_i = \frac{\det((Syl(g,f)^T)_i)}{\det(Syl(g,f)^T)} = \frac{\det((Syl(g,f)^T)_i)}{p},$$

where $(Syl(g,f)^T)_i$ denotes the Sylvester matrix of f and g, with the ith column replaced by $\begin{pmatrix}0 & \dots & 0 & 1\end{pmatrix}^T$. Now since $s(x)g(x) \equiv 1 \mod f(x)$, we find that $Z(x) = p \cdot s(x)$. In order to bound the norm of $Z(x)$, we use Hadamard's inequality for determinants:

$$||Z(x)||_\infty \leq \max_i \prod_{j=1}^{m+n} ||((Syl(g,f)^T)_i)_j||_2 \leq ||g||_2^{n-1}||f||_2^m.$$

Here $((Syl(g,f)^T)_i)_j$ denotes the jth column of $(Syl(g,f)^T)_i$. The latter inequality holds because the first n columns have norm $||g||_2$, the latter m columns have $||f||_2$ and the ith column is replaced by $\begin{pmatrix}0 & \dots & 0 & 1\end{pmatrix}^T$, for $1 \leq i \leq n$. Integrality of $Z(x)$ follows from the fact that the determinant of an integral matrix is itself integral. □

Since with very high probability we have that $m = n - 1$ we will assume from now on that this is the case. Define

$$\delta_\infty := \sup\left\{ \frac{||g(x)h(x) \mod f(x)_\infty}{||g(x)||_\infty \cdot ||h(x)||_\infty},\ \deg(g), \deg(h) < n \right\}.$$

Then we have

$$||g(\theta)h(\theta)||_\infty \leq \delta_\infty \cdot ||g(\theta)||_\infty \cdot ||h(\theta)||_\infty.$$

For different polynomials $f(x)$, δ_∞ can range from exponential in n to linear in n, as is shown in the following lemma.

Lemma 4.2.3. *Let $f_1(x) = x^n - a$ and $f_2(x) = x^n - ax^{n-1}$, then $\delta_\infty(f_1) \leq |a|n$ and $\delta_\infty(f_2) \leq |a|^{n-1}n^2$.*

Proof. Let $g = \sum_{i=0}^{n-1} g_i x^i$ and $h = \sum_{i=0}^{n-1} h_i x^i$, then

$$g \cdot h \mod f_1 \equiv \sum_{k=0}^{n-1} \left(\sum_{0 \leq i \leq k} g_i h_{k-i} + a \sum_{k < i < n} g_i h_{n+k-i} \right) x^k.$$

So we find that $||g \cdot h \mod f_1||_\infty \leq |a|n||g||_\infty \cdot ||h||_\infty$ from which the bound follows. Now if we write $g \cdot h = \sum_{k=0}^{2n-2} c_k x^k$, then $g \cdot h \mod f_2 = \sum_{k=0}^{n-1} d_k x^k$ with $d_k = c_k$ for $0 \leq k \leq n-2$ and $d_k = \sum_{i=0}^{n-1} c_{n-1+i} a^i$ for $k = n-1$. Clearly we have $c_i \leq n||g||_\infty \cdot ||h||_\infty$ for all i, so for d_k we find $d_k \leq |a|^{n-1}n^2||g||_\infty \cdot ||h||_\infty$ from which the bound follows. □

Returning to our initial goal, we need $||\Psi(\theta)Z(\theta)/p||_\infty < 1/2$ for decryption to work. The lemmas yield that

$$||\Psi(\theta)Z(\theta)/p||_\infty \leq \frac{\delta_\infty ||\Psi||_\infty \cdot ||g||_2^{n-1} \cdot ||f||_2^{n-1}}{p},$$

so decryption will work if

$$||\Psi||_\infty \leq \frac{p}{2\delta_\infty ||g||_2^{n-1} \cdot ||f||_2^{n-1}} =: r_{dec}.$$

Now if $||\Psi||_\infty < r_{dec}$, we find from equation 8 that $q(\theta) = -\lfloor \psi Z(\theta)/p \rceil$ and thus that $\Psi(x) = \psi + q(\theta)\gamma = \psi - \lfloor \psi Z(x)/p \rceil \gamma$. Furthermore, $\pi \equiv \Psi(x) \mod 2$ and $\gamma \equiv 1 \mod 2$, so $\pi \equiv \Psi(x) \mod 2 \equiv \psi - \lfloor Z(x)/p \rceil \mod 2$ and thus we must have $\pi \equiv \psi - \lfloor \psi B/p \rceil \mod 2$. Here $B = z_0 \mod 2p$ suffices.
The resultant p will be close to $||g||_2^n ||f||_2^{n-1}$, so we find $r_{dec} \approx ||g||_2/(2\delta_\infty)$. g is generated such that $||g||_\infty \approx \eta$, which gives

$$r_{dec} \approx \frac{\sqrt{n}\eta}{2\delta_\infty}.$$

If we then select $f(x) = x^n + 1$, this reduces to $r_{dec} \approx \eta/(2\sqrt{n})$.

Add and Mult: since it is obvious that these algorithms are correct, we will only analyze what happens to the error term as the operations are performed on the ciphertext. Let $\pi_1, \pi_2 \in \{0, 1\}$ be two messages and let $n_i \in \mathcal{B}_{\infty,n}(r_i - 1)$ be random polynomials. Then $\Psi_i(x) = \pi_i + n_i(x)$ represent the encryption polynomials, with

$$\Psi_i(x) \in \mathcal{B}_{\infty,n}(r_i).$$

Now for $\Psi_3(x) = \Psi_1(x) + \Psi_2(x)$ and $\Psi_4(x) = \Psi_1(x) \cdot \Psi_2(x)$ we clearly find that

$$\begin{aligned} \Psi_3(x) &\in \mathcal{B}_{\infty,n}(r_1 + r_2) \\ \Psi_4(x) &\in \mathcal{B}_{\infty,n}(\delta_\infty \cdot r_1 \cdot r_2). \end{aligned}$$

Typically we have $\Psi(x) \in \mathcal{B}_{\infty,n}(\mu+1)$, so after a circuit of multiplicative depth d we expect $\Psi'(x) \in \mathcal{B}_{\infty,n}(r)$, where $r \approx (\delta_\infty \cdot \mu)^{2^d}$. Correct decryption is only possible for $r \leq r_{dec}$, so we find the following.

$$\begin{aligned} (\delta_\infty \mu)^{2^d} &\leq r_{dec} \\ 2^d \log(\delta_\infty \mu) &\leq \log(r_{dec}) \\ d\log(2) + \log\log(\delta_\infty \mu) &\leq \log\log(r_{dec}) \\ d\log(2) &\leq \log\log(r_{dec}) - \log\log(\delta_\infty \mu). \end{aligned}$$

This yields a bound on the maximum multiplicative depth of a circuit which we can evaluate for each f.

4.2.2 The Fully Homomorphic Scheme

With the somewhat homomorphic scheme in place it remains to turn this scheme into a fully homomorphic one. Since the scheme is actually a specialization of Gentry's scheme, we will not describe this in too much detail.
We need to define an algorithm which refreshes a given ciphertext, Recrypt. To define this algorithm, we first need to redefine KeyGen. It will be identical to what's given above, with in addition the following operations, where s_1 and s_2 are two given integer parameters:

- Generate s_1 uniformly random integers B_i in $[-p, p]$, such that there exists a bit-vector sk with $w(\text{sk}) = s_2$ for which it holds that

$$\sum_{i=1}^{s_1} \text{sk}_i B_i = B.$$

- Encrypt sk_i under the somewhat homomorphic scheme to obtain $\mathfrak{c}_i = \text{Encrypt}(\text{sk}_i, \text{PK})$.
- The public key now consists of

$$\text{PK} = (p, \alpha, s_1, s_2, \{\mathfrak{c}_i, B_i\}_{i=1}^{s_1}).$$

We can now define the Recrypt procedure. For a detailed description and analysis, we refer to [SV10, Appendix A].

Recrypt(ψ, PK)**:**

- Compute the first t bits of the s_1 floating point numbers $(\psi B_i \mod 2p)/p$ and form an $s_1 \times t$-matrix $(b_{i,j})$ for $i = 1, \ldots, s_1$ and $j = 1, \ldots, t$.
- Encrypt each of the bits $b_{i,j}$ to obtain an $s_1 \times t$-matrix of "clean" ciphertexts $(\psi_{i,j})$.
- Multiply row i of the matrix $(\psi_{i,j})$ with the corresponding $\mathfrak{c}_i$. This matrix now represents the element-wise encryption of a matrix with only s_2 non-zero rows.
- Add the rows of this matrix together and reduce its output to obtain the new encryption of ψ.

This completes the fully homomorphic scheme. Next we will compare the scheme to the scheme as presented by Gentry and analyze the security of this variant.

4.2.3 Comparison and security

Comparison
As before mentioned, the variant by Smart and Vercauteren is a specialization of Gentry's scheme. We have that the generator γ is equivalent to $\mathbf{B}_J^{\text{sk}}$, and $\langle p, \theta - \alpha \rangle$ is equivalent to $\mathbf{B}_J^{\text{pk}}$. Furthermore, the ideal I is simply set to (2). Note that the main difference here is

the size of the public basis.
In the encryption algorithm, one computes $\Psi(\theta) = \pi(\theta) + 2R(\theta)$, which is exactly equal to ψ' as generated in Gentry's scheme. Now in the original scheme this value is reduced modulo J using $\mathbf{B}_J^{\text{pk}}$. But in this variant the reduction modulo $\mathfrak{p}$ is computed by using the two-element presentation, i.e. one replaces θ by α and reduces modulo p. In the end this actually is equivalent, but easier to compute.

Security
For this variant we cover several aspects of security, namely Key Recovery, Onewayness of Encryption and Semantic security.

Concerning Key Recovery we recall that the public key consists of a principal degree one prime ideal in the two element representation, and the secret key consists of the inverse of a small generator of this ideal. This relates recovering the private key from the public key to an instance of the small principal ideal problem, which is defined as follows.

Definition 4.2.4. (Small Principal Ideal Problem) *Given a principal ideal $\mathfrak{a}$ in either two element or HNF presentation, compute a small generator of this ideal.*

This problem is a well researched problem in algebraic number theory for which there are no efficient solutions [SV10, p. 430].

Instead of extracting the private key from the public key, one could also try to recover the message straight from the ciphertext without using the key. It is easy to see that this is equivalent to solving the following problem:
Given p and α as in the public key, and a ciphertext $\psi \in \mathbb{F}_p$, find x_i for $i = 0, \ldots, n-1$ such that

$$\sum_{i=0}^{n-1} x_i \cdot \alpha^i = \psi - k \cdot p,$$

for some integer k and $|x_i| \leq r_{enc}$. Using the HNF-matrix H, induced by the public key value, we can rewrite this to

$$(k, -x_1, \ldots, -x_n) \cdot H = (\psi - x_0, -x_1, \ldots, -x_n).$$

Note that this vector is in the lattice and is very close to the non-lattice vector $(\psi, 0, \ldots 0)$, so finding the plaintext given a ciphertext is related to the shortest vector problem.
As mentioned in section 4.1.3, the problem can also be related to BDDP, in which case solving the problem has difficulty $2^{n/\varepsilon}$ where ε such that $2^{\varepsilon} = r_{dec}/r_{enc}$.

Remaining to address is the semantic security of the scheme, which is related to the following problem, dubbed the Polynomial Coset Problem.

Definition 4.2.5. [SV10, p. 431] (Polynomial Coset Problem) *The challenger first randomly selects $b \in \{0, 1\}$ and runs* KeyGen *to obtain p and α. If $b = 0$ the challenger*

generates a random polynomial $R(x) \in \mathcal{B}_{\infty,n}(r_{enc})$ *and sets* $r = R(\alpha) \mod p$. *But if* $b = 1$, *it randomly selects* $r \in \mathbb{F}_p$.
Now the problem is to guess whether $b = 0$ *or* $b = 1$, *given* (r, pk).

This yields the following theorem.

Theorem 4.2.6. *Suppose there exists an algorithm* $\mathcal{A}$ *which* (t, ε) *breaks the semantic security of our scheme. Then there exists an algorithm* $\mathcal{B}$ *which* (t', ε') *solves PCP, where* $t' \approx t$ *and* $\varepsilon' = \varepsilon/2$.

Proof. The proof is essentially analogous to the proof of theorem 4.1.20 and is given in full detail in [SV10, p. 431]. □

4.3 The Gentry-Halevi Variant

The variant as presented by Gentry and Halevi is in fact an implementation of the variant by Smart and Vercauteren with some clever improvements. Since this is the variant which is used in the JCrypTool plugin which is presented in section 5, we will skip some details in this section. These details are given in 5. Some techniques are used to speed up the different algorithms in the implementation, these will be given separately in the section describing the plugin.

4.3.1 The Somewhat Homomorphic Scheme

As mentioned the scheme strongly resembles the variant by Smart and Vercauteren, so we will jump straight into describing the algorithms of the somewhat homomorphic scheme. We select $f(x) = x^{2^m} + 1$, then the scheme is as follows.

KeyGen**:**

To generate the public and private key, we perform the following steps:

- Select a random integer polynomial v of degree $n-1$, where each coefficient is a t-bit integer. The rotation basis of v, $\mathbf{V}$, now defines an integer lattice. And in addition, we require the HNF of $\mathbf{V}$ to be of the following form:

$$\mathrm{HNF}(\mathbf{V}) = \begin{pmatrix} d & 0 & 0 & \dots & 0 \\ -r & 1 & 0 & \dots & 0 \\ -[r^2]_d & 0 & 1 & \ddots & \vdots \\ \vdots & \vdots & \ddots & \ddots & 0 \\ -[r^{n-1}]_d & 0 & \dots & 0 & 1 \end{pmatrix},$$

  where d is the resultant of $v(x)$ and $f(x)$, and r an integer such that there exists an integer vector $\mathbf{y}$ with $\mathbf{y} \times \mathbf{V} = \langle -r, 1, 0, \dots, 0 \rangle$. We will show later that the existence of such $\mathbf{y}$ and r is required and sufficient for $\mathbf{V}$ to have such an HNF.

- Compute a polynomial $w(x)$ such that

$$w(x)v(x) = d \mod f(x)$$

  holds. Associate with $w(x)$ also its rotation basis $\mathbf{W}$.

We now have the public key $\mathrm{HNF}(\mathbf{V})$, which can be represented by $\mathrm{pk} = (d, r)$ and the private key (v, w), but we will show that it suffices to have one coefficient of w_i, which must be odd. So $\mathrm{sk} = w$.

Encrypt(π, pk)**:** In this scheme, we again take $\pi \in \mathcal{M} = \{0, 1\}$.

- Generate a random n-dimensional vector $\mathbf{u}$ with $u_i = 0$ with probability q and $u_i = \pm 1$ each with probability $(1-q)/2$.

- $\mathbf{a} = 2\mathbf{u} + \pi\mathbf{e_1}$
- $\psi = \mathbf{a} \mod \mathrm{HNF}(\mathbf{V})$.

We will later show that we can represent ψ by a single integer $\psi := [a(r)]_d$.

Decrypt$(\psi, \mathrm{pk}, \mathrm{sk})$**:**

- $\mathbf{a} = \psi \mod \mathbf{V}$
- $\pi = a_0 \mod 2$

We will later show that we can find π with $\pi = [\psi \cdot w]_d \mod 2$, and thus do not need the public key for decryption.

Add, Mult**:** Will be omitted as ψ is later shown to be an integer.

The description of the algorithms requires some additional clarifications. These will be given below.

KeyGen: We claim that the existence of an integer vector $\mathbf{y}$ with $\mathbf{y} \times \mathbf{V} = \langle -r, 1, 0, \ldots, 0\rangle$ is necessary and sufficient for $\mathbf{V}$ to have HNF of the correct form. This is proven in the following lemma.

Lemma 4.3.1. *The Hermite normal form of* $\mathbf{V}$ *is equal to the identity matrix in all but the leftmost column, if and only if the lattice spanned by the rows of* $\mathbf{V}$ *contains a vector of the form* $\mathbf{r} = \langle -r, 1, 0, \ldots, 0\rangle$.

Proof. Denote by $\mathbf{B}$ the Hermite normal form of $\mathbf{V}$. It is clear that the second row of $\mathbf{B}$ must be of the form $\mathbf{r}$, for some integer r, which implies that the condition is necessary. It remains to show that the condition is sufficient.
We know that the vector $\langle d, 0, \ldots, 0\rangle$ is in the lattice $\mathcal{L}(\mathbf{V})$, since we have $\langle w_0, w_1, \ldots, w_{n-1}\rangle \times \mathbf{V} = \langle d, 0, \ldots, 0\rangle$. Now assume that $\mathcal{L}(\mathbf{V})$ also contains the vector $\mathbf{r}$, this implies in addition that we have $\langle [-r]_d, 1, 0, \ldots, 0\rangle \in \mathcal{L}(\mathbf{V})$, since we can subtract multiples of $d \cdot \mathbf{e_1}$. This gives the first two rows of $\mathbf{B}$. For $i = 1, 2, \ldots, n-1$ let $r_i := [r^i]_d$. We will prove by induction over i that for all $i = 1, 2, \ldots, n-1$ the vector

$$\mathbf{r}_i := -r_i\mathbf{e_1} + \mathbf{e_{i+1}}$$

is in the lattice. Note that these vectors form exactly the rows of $\mathbf{B}$. Since all the rows in $\mathbf{B}$ are lattice vectors, they are all independent, and $\mathbf{B}$ has the same determinant as $\mathbf{V}$, we have that they must span $\mathcal{L}(\mathbf{V})$ itself.
The start the induction, note that this is true for $i = 1$ by the assumption that $\mathbf{r} \in \mathcal{L}(\mathbf{V})$. Now assume that $\mathbf{r_i} \in \mathcal{L}(\mathbf{V})$ for some $i \in [1, n-2]$, then we will prove that it also holds for $i+1$. We know that the lattice is closed under rotation, so since $\mathbf{r_i} = -r_i\mathbf{e_1} + \mathbf{e_{i+1}} \in \mathcal{L}(\mathbf{V})$ we also have $\mathbf{s_{i+1}} = -r_i\mathbf{e_2} + \mathbf{e_{i+2}} \in \mathcal{L}(\mathbf{V})$. It is then clear that $\mathcal{L}(\mathbf{V})$ also contains

$$\mathbf{s_{i+1}} + r_i\mathbf{r} = (-r_i\mathbf{e_2} + \mathbf{e_{i+2}}) + r_i(-r\mathbf{e_1} + \mathbf{e_2}) = -r_i r\mathbf{e_1} + \mathbf{e_{i+2}}.$$

And thus by adding or subtracting a multiple of $d\mathbf{e_1}$, we find that $[-rr_i]_d\mathbf{e_1} + \mathbf{e_{i+2}} = -[r^{i+1}]_d\mathbf{e_1} + \mathbf{e_{i+2}} = \mathbf{r_{i+1}}$, which completes the induction.
Now since we know that the HNF is unique and that the given matrix $\mathbf{B}$ is in HNF and spans $\mathcal{L}(\mathbf{V})$, this must be the Hermite normal form of $\mathbf{V}$. □

Encrypt: We claim that $\psi = [a(r)]_d$. In the encryption phase, we compute $\psi = \mathbf{a} \mod \text{HNF}(\mathbf{V})$. Let $\mathbf{B} = \text{HNF}(\mathbf{V})$, then this reduces to $\psi = \mathbf{a} - (\lfloor \mathbf{a} \times \mathbf{B}^{-1} \rceil \times \mathbf{B}) = [\mathbf{a} \times \mathbf{B}^{-1}] \times \mathbf{B}$. We know exactly what $\mathbf{B}$ looks like, and it is easy to find its inverse:

$$\mathbf{B}^{-1} = \frac{1}{d} \begin{pmatrix} 1 & 0 & 0 & 0 & \dots & 0 \\ r & d & 0 & 0 & \dots & 0 \\ [r^2]_d & 0 & d & 0 & \dots & 0 \\ [r^3]_d & 0 & 0 & d & \ddots & \vdots \\ \vdots & \vdots & \vdots & \ddots & \ddots & 0 \\ [r^{n-1}]_d & 0 & 0 & \dots & 0 & d \end{pmatrix}.$$

Now we have $\mathbf{a} \times \mathbf{B}^{-1} = \langle \frac{s}{d}, a_1, \dots, a_{n-1} \rangle$ for an integer $s = a(r) \mod d$. It is clear that the fractional part equals $[\mathbf{a} \times \mathbf{B}^{-1}] = \langle \frac{[a(r)]_d}{d}, 0, \dots, 0 \rangle$ and we thus have $\psi = [\mathbf{a} \times \mathbf{B}^{-1}] \times \mathbf{B} = \langle [a(r)]_d, 0, \dots, 0 \rangle$, which clearly is determined by $\psi = [a(r)]_d$.

Decrypt: It is claimed that decryption can be done with a single odd coefficient of the polynomial w. This is not very obvious, so here is some clarification. The decryption recovers $\mathbf{a}$ with $\mathbf{a} = \psi \mod \mathbf{V}$, after which it is trivial to find π. Now, like in the encryption phase, this translates to $\mathbf{a} = \psi - (\lfloor \psi \times \mathbf{V}^{-1} \rceil \times \mathbf{V}) = [\psi \times \mathbf{W}/d] \times \mathbf{V}$. We know that $\psi - \mathbf{a} \in \mathcal{L}(\mathbf{V})$, so for an integer vector $\mathbf{y}$, we find $\psi = \mathbf{y} \times \mathbf{V} + \mathbf{a}$. This then yields

$$[\psi \times \mathbf{W}/d] \times \mathbf{V} = [\mathbf{y} \times \mathbf{V} \times \mathbf{W}/d + \mathbf{a} \times \mathbf{V} \times \mathbf{W}/d] = [\mathbf{a} \times \mathbf{W}/d] \times \mathbf{V}.$$

Note that $[\mathbf{a} \times \mathbf{W}/d] \times \mathbf{V}$ is supposed to be $\mathbf{a}$, so we must have $[\mathbf{a} \times \mathbf{W}/d] = (\mathbf{a} \times \mathbf{W}/d)$, which means decryption only is correct if every entry in $\mathbf{a} \times \mathbf{W}$ is less than $d/2$ in absolute value.
This now gives us the equation $[\psi \times \mathbf{W}/d] = \mathbf{a} \times \mathbf{W}/d$, and thus

$$[\psi \times \mathbf{W}]_d = \mathbf{a} \times \mathbf{W}.$$

This gives us

$$[\psi \times \mathbf{W}]_d = [\psi \cdot \langle w_0, w_1, \dots, w_{n-1} \rangle]_d = \langle [\psi w_0]_d, [\psi w_1]_d, \dots, [\psi w_{n-1}]_d \rangle,$$

but we also know that

$$[\psi \times \mathbf{W}]_d = \mathbf{a} \times \mathbf{W} = 2\mathbf{u} \times \mathbf{W} + \pi \mathbf{e_1} \times \mathbf{W} = 2\mathbf{u} \times \mathbf{W} + \pi \cdot \langle w_0, w_1, \dots, w_{n-1} \rangle.$$

These two equations together give us that ψ must satisfy the following

$$\langle [\psi w_0]_d, [\psi w_1]_d, \dots, [\psi w_{n-1}]_d \rangle = \pi \cdot \langle w_0, w_1, \dots, w_{n-1} \rangle \mod 2.$$

So now if we have an odd coefficient of $w(x)$, say w, we find that $\pi = [\psi \cdot w]_d \mod 2$.

This wraps up the changes made to the somewhat homomorphic scheme. Now we will make the scheme fully homomorphic.

4.3.2 The Fully Homomorphic Scheme

As before, we only need to introduce the Recrypt procedure. First, some additional steps are added to KeyGen:

- Generate a set of random elements $\mathcal{B} = \{x_i \in \mathbb{Z}_d : i = 1, 2, \ldots, S\}$ such that there exists a bit-vector $\sigma = \langle \sigma_1, \ldots, \sigma_S \rangle$ with $w(\sigma) = s$ for which it holds that

$$\sum_{i=1}^{S} \sigma_i x_i = w \mod d.$$

- We add this set of random elements along with an encrypted version of σ, denoted by $\bar{\sigma}$, to the public key. Later we will see that we can optimize this to reduce the size of the public key. This will be thoroughly covered in section 5.1.4.

Before we can introduce the Recrypt procedure, we will first derive its steps. Given a ciphertext $\psi \in \mathbb{Z}_d$, we first compute $y_i := \langle \psi x_i \rangle_d$. Decryption, denoted as $D_{\psi,d}(\sigma)$, then is as follows

$$D_{\psi,d}(\sigma) := \left[\sum_{i=1}^{S} \sigma_i y_i \right]_d \mod 2.$$

We can now deduce that

$$\left[\sum_{i=1}^{S} \sigma_i y_i \right]_d = \left(\sum_{i=1}^{S} \sigma_i y_i \right) - d \cdot \left\lfloor \sum_{i=1}^{S} \sigma_i \frac{y_i}{d} \right\rfloor,$$

and since we reduce everything modulo 2 and d is odd, this gives us

$$D_{\psi,d}(\sigma) = \left(\bigoplus_{i=1}^{S} \sigma_i \langle y_i \rangle_2 \right) \oplus \left\langle \left\lfloor \sum_{i=1}^{S} \sigma_i \frac{y_i}{d} \right\rfloor \right\rangle_2.$$

Recall that $[w\psi]_d = \pi + [2u(r)]_d$, so if the ciphertext ψ is closer to the lattice than the scheme can correctly decrypt, it follows that $w\psi$ is much closer to a multiple of d than $d/2$. If we keep the noise small so that ψ is within distance $1/(s+1)$ from the nearest lattice point, we have that the distance from $w\psi$ to the nearest multiple of d is smaller than $d/2(s+1)$. This leads to

$$\text{abs}([w\psi]_d) = \text{abs}\left(\left[\sum_{i=1}^{S} \sigma_i y_i \right]_d \right) < \frac{d}{2(s+1)}$$

and thus

$$\mathrm{abs}\left(\left\lceil \sum_{i=1}^{S} \sigma_i \frac{y_i}{d} \right\rceil\right) < \frac{1}{2(s+1)}.$$

Note that $y_i \in \mathbb{Z}_d$, so $y_i/d \in [0,1)$. Define $\xi := \lceil \log_2(s+1) \rceil$ to be the precision parameter, with which we approximate every y_i/d with z_i to within ξ bits after the binary point. This means we have $\mathrm{abs}(z_i - \frac{y_i}{d}) \leq 2^{-(\xi+1)} \leq 1/2(s+1)$. Now suppose we replace one of the y_i/d with z_i, if $\sigma_i = 0$ this makes no difference. If instead $\sigma_i = 1$, we make an error in the total sum of at most $2^{-(\xi+1)}$. Since $w(\sigma) = s$, we find $\mathrm{abs}((\sum_i \sigma_i z_i) - (\sum_j \sigma_j \frac{y_j}{d})) \leq s/2(s+1)$. We know that the latter sum is at distance at most $1/2(s+1)$ from the nearest integer, which means that the former sum is at distance at most $1/2(s+1) + s/2(s+1) = 1/2$ from the same integer. This implies that

$$\left\lfloor \sum_{i=1}^{S} \sigma_i \frac{y_i}{d} \right\rfloor = \left\lfloor \sum_{i=1}^{S} \sigma_i z_i \right\rfloor.$$

So we have established that decryption can be computed as $D_{\psi,d}(\sigma) = \langle \lfloor \sum_i \sigma_i z_i \rceil \rangle_2 \oplus \bigoplus_i \sigma_i \langle y_i \rangle_2$. Now the Recrypt procedure is as follows.

Recrypt(ψ, pk)**:**

- Compute $y_i = \langle x_i \cdot \psi \rangle_d$ and approximate $z_i \overset{\xi}{\approx} y_i/d$. Store these $\xi + 1$ bits in an $S \times (\xi + 1)$-matrix $\mathbf{\Psi}$.
- Encrypt $\mathbf{\Psi}$ elementwise.
- Multiply row i of $\mathbf{\Psi}$ with $\bar{\sigma}_i$.
- Add these rows together and reduce to a new encryption of ψ

4.4 Fully Homomorphic Encryption over the Integers

This scheme is due to van Dijk, Gentry, Halevi and Vaikuntanathan [vDGHV10]. Their aim was to create a fully homomorphic scheme using only basic modular arithmetic. Though working over the integers, also this scheme only encrypts a single bit at a time. The scheme uses several parameters, some of which are the following:

ρ the bit-length of the noise,

ρ' a secondary noise parameter,

η the bit-length of the secret key,

γ the bit-length of the integers in the public key,

τ the number of integers in the public key.

These parameters will be set according to the following:

- $\rho = \omega(\log \lambda)$, to be safe against brute force attacks on the noise,
- $\rho' = \rho + \omega(\log \lambda)$,
- $\eta \geq \rho \cdot \Theta(\lambda \log^2 \lambda)$, necessary to obtain bootstrappability,
- $\gamma = \omega(\eta^2 \log \lambda)$, to protect against lattice-based attacks,
- $\tau \geq \gamma + \omega(\log \lambda)$, so the leftover hash lemma can be used.

Also, the following distribution is used to efficiently sample integers a, for which $[a]_p << p$.

$$\mathcal{D}_{\gamma,\rho}(p) = \Big\{\text{choose uniformly random } q \in \mathbb{Z} \cap [0, 2^\gamma/p), r \in \mathbb{Z} \cap (-2^\rho, 2^\rho) : \text{ output } x = pq + r\Big\}.$$

4.4.1 The Somewhat Homomorphic Scheme

As usual for somewhat homomorphic schemes, this scheme consists of four different algorithms, namely key generation, encryption, evaluation of a circuit and decryption. These are as follows:

KeyGen(λ) The secret key is a uniformly random selected odd η-bit integer, i.e. $p \in_R (2\mathbb{Z} + 1) \cap [2^{\eta-1}, 2^\eta)$. To create the public key, sample $x_i \in \mathcal{D}_{\gamma,\rho}(p)$ for $i = 0, \ldots, \tau$. Relabel so that x_0 is the largest. Now x_0 must be odd and $r_p(x_0)$ even, otherwise re-sample (recall that $q_p(z)$ and $r_p(z)$ respectively denote the quotient and the remainder of z with respect to p). The public key then is the collection of these x_i's, $\langle x_0, \ldots, x_\tau \rangle$.

Encrypt($\text{pk}, \pi \in \{0, 1\}$) Randomly take a subset $S \subseteq \{1, 2, \ldots, \tau\}$ and an integer $r \in (-2^{\rho'}, 2^{\rho'})$, the ciphertext then is $\psi = [\pi + 2r + 2\sum_{i \in S} x_i]_{x_0}$.

Evaluate$(\mathrm{pk}, C, \psi_1, \ldots, \psi_t)$ Apply the circuit C to the ciphertexts $\psi_1, \ldots, \psi_t$ and return the resulting integer.

Decrypt(sk, ψ) The plaintext is $\pi = [\psi]_p \mod 2$. Actually, since p is odd, we find $\pi = [\psi]_p \mod 2 = [c - \lfloor c/p \rceil]_2 = (c \bmod 2) \oplus (\lfloor c/p \rceil \bmod 2)$.

We claim that the scheme is somewhat homomorphic, but this is not entirely obvious. Later, we will see that a ciphertext is of the form $\psi = a \cdot p + 2b + \pi$, for some a and b. Clearly, for addition the noise only doubles. For multiplication, we find $\psi' = a' \cdot p + 2(2b_1b_2 + b_1\pi_2 + b_2\pi_1) + \pi_1\pi_2$, so the noise approximately squares. This will soon be proved formally.
Now consider the generalization of the mod-2 circuits to circuits on the integers. We then define a permitted circuit as follows.

Definition 4.4.1. *A* permitted circuit *is a circuit in which for any $\alpha \geq 1$ and any set of integer inputs, all less than $2^{\alpha(\rho'+2)}$ in absolute value, it holds that the generalized circuit's output integer has absolute value at most $2^{\alpha(\eta-4)}$. Denote this set of circuits by $\mathcal{C}_{\mathcal{E}}$.*

This definition gives us the following lemma.

Lemma 4.4.2. [vDGHV10, p. 6] *The scheme is correct for $\mathcal{C}_{\mathcal{E}}$.*

Proof. To ensure correct decryption, we require that the noise in the ciphertext is at most $p/2$. First consider a ciphertext output by Encrypt. Then we have $\psi = a \cdot p + (2b + \pi)$. Clearly $2b + \pi$ has the same parity as π. We will first bound $|2b + \pi|$. Per definition, we have $\psi = [\pi + 2r + \sum_{i \in S} x_i]_{x_0}$. Since $|x_0| \geq |x_i|$ for $i \in \{1, \ldots, \tau\}$, this yields

$$\psi = \left(\pi + 2r + \sum_{i \in S} x_i\right) + k \cdot x_0,$$

for some $|k| \leq \tau$. Now for every i, we have per definition that there exist integers q_i and r_i with $|r_i| \leq 2^\rho$ such that $x_i = q_i \cdot p + 2r_i$. This gives us

$$\psi = p\left(kq_0 + \sum_{i \in S} q_i\right) + \left(\pi + 2r + k \cdot 2r_0 + \sum_{i \in S} 2r_i\right).$$

The absolute value can be bounded by $|2b + \pi| \leq (4\tau + 3)2^\rho < \tau 2^{\rho+3}$.
Consider a circuit $C \in \mathcal{C}_{\mathcal{E}}$, with t inputs and a single output. Let C' denote the generalized circuit over the integers. Now we have that $C'(\psi_1, \ldots, \psi_t) \in C'(2b_1 + \pi_1, \ldots, 2b_t + \pi_t) + p\mathbb{Z}$. This yields that $[C'(2b_1 + \pi_1, \ldots, 2b_t + \pi_t)]_p$ has the same parity as $C(\pi_1, \ldots, \pi_t)$. Furthermore,

$$|C'(2b_1 + \pi_1, \ldots, 2b_t + \pi_t)| \leq 2^{\eta-4} \leq p/8$$

by the definition of the permitted circuits and since $|2b_i + \pi_i| \leq \tau 2^{\rho+3}$. Which clearly suffices to ensure correct decryption. □

The definition of permitted circuits seems not very intuitive. Now if one considers these circuits, a k-fan-in add gate can only increase the size of the ciphertext by at most a factor k, while a 2-fan-in Mult gate can square this size. So it makes more sense to relate the permitted circuits to the multiplicative depth of the circuits, or the degree of the multivariate polynomial which it actually computes. This is formalized in the following lemma.

Lemma 4.4.3. [vDGHV10, p. 7] *Let C be a boolean circuit with t inputs, and let $C^\dagger$ be the associated integer circuit. Let $f(x_1, \ldots, x_t)$ be the multivariate polynomial which $C^\dagger$ computes. Then for $d = \deg(f)$ and $|\mathbf{f}|$ the ℓ_1 norm of the coefficient vector of f, we have that if $|\mathbf{f}|(2^{\rho'+2})^d \leq 2^{\eta-4}$ then $C \in \mathcal{C}_\mathcal{E}$.*

This in particular implies that $\mathcal{E}$ can correctly evaluate f as long as

$$d \leq \frac{\eta - 4 - \log|\mathbf{f}|}{\rho' + 2}.$$

Denote the polynomials for which this holds, the permitted polynomials, by $\mathcal{P}_\mathcal{E}$ and denote by $C(\mathcal{P}_\mathcal{E})$ the set of circuits that compute these polynomials. We clearly have that $C(\mathcal{P}_\mathcal{E}) \subseteq \mathcal{C}_\mathcal{E}$.

Though we can simply reduce modulo x_0 in the encryption phase, we cannot do this during the evaluation of a circuit, since after a single multiplication, the ciphertext size grows much larger than x_0. In that case the error becomes much larger than $p/2$, introducing possible errors in decryption. So we need another strategy to reduce the ciphertext size.
One option is to add in the public key more elements of the form $x'_i = q'_i \cdot p + 2r'_i$ as before, only with q'_i according to

$$q'_i \overset{R}{\in} \mathbb{Z} \cap [2^{\gamma+i-1}/p, 2^{\gamma+i}/p),$$

such that $x'_i \in [2^{\gamma+i}, 2^{\gamma+i+1}]$. Now as soon as the ciphertext grows larger than 2^γ, we reduce it consecutively modulo $x'_\gamma, x'_{\gamma-1}, \ldots, x'_0$. As a result the ciphertext is bounded in absolute value by 2^γ. It is clear that the bit-length of the ciphertext can at most double with a single operation. This means that the ciphertext is always smaller than $2x'_\gamma$, which means that the reductions only involve small multiples of the x_i's, and thus the error remains small.

4.4.2 Security

The security of this somewhat homomorphic scheme can be related to the approximate-GCD problem. This problem is in that case defined as follows.

Definition 4.4.4. (Approximate-GCD) *The $(\rho, \eta, \gamma)-$approximate-gcd problem is to find p, given polynomially many samples from $\mathcal{D}_{\gamma,\rho}(p)$, for a randomly chosen η-bit odd integer p.*

We have the following theorem, which we will present with only a sketch of the proof. The formal details are in [vDGHV10, pp. 9 - 12].

Theorem 4.4.5. *Fix the parameters $(\rho, \rho', \eta, \gamma, \tau)$ as in the somewhat homomorphic scheme. Then any attack $\mathcal{A}$ with advantage ε on the scheme can be converted into an algorithm $\mathcal{B}$ that solves the (ρ, η, γ)-approximation-gcd problem with success probability at least $\varepsilon/2$. The running time of $\mathcal{B}$ is polynomial in the running time of $\mathcal{A}$, λ and in $1/\varepsilon$.*

Proof. We have that $\mathcal{A}$ has advantage ε on the encryption scheme. Thus given a ciphertext, $\mathcal{A}$ guesses the encrypted bit with probability $\frac{1}{2} + \varepsilon$. We will describe step by step what $\mathcal{B}$ does to solve the approximate-gcd problem.

Step 1: Creating a public key. First $\mathcal{B}$ creates a fake public key by sampling $\tau + 1$ numbers $x_0, \ldots, x_\tau$ from $\mathcal{D}_{\gamma,\rho}(p)$, and relabeling it so that x_0 is the largest. If x_0 is even it repeats this until x_0 is odd. Now if $r_p(x_0)$ happens to be even, then this fake public key has exactly the same distribution as a real public key. This obviously happens with probability $\frac{1}{2}$.

Step 2: An LSB predictor. To find p, and thus solve the approximate-gcd problem, $\mathcal{B}$ uses $\mathcal{A}$ to predict the least significant bit of the quotient of an integer with respect to p. $\mathcal{B}$ can roughly do this by creating a fake ciphertext, using the previously created fake public key. This ciphertext is of the form $[z + \pi + 2r + \sum_{i \in S} x_i]_{x_0}$, resembling a real ciphertext. Then $\mathcal{A}$ can be used to guess the parity of this ciphertext, say a, which implies the parity of $q_p(z)$ by $a \oplus \mathbf{parity}(z) \oplus \pi$.
Now $\mathcal{A}$ only has probability of $\frac{1}{2}$ of guessing correctly, so $\mathcal{B}$ creates $\mathrm{poly}(\lambda)/\varepsilon$ fake ciphertexts using the same public key, and takes the majority vote of the guessed parities for $q_p(z)$, yielding the correct parity with high probability.

Step 3: Binary GCD. Now that $\mathcal{B}$ has the possession of an LSB predictor, it can use a sort of binary-GCD algorithm to recover p. The algorithm is as follows, upon input of two integers z_1, z_2.

1. If $z_2 > z_1$ then swap them.
2. Use the LSB predictor to learn $b_i = [q_p(z_i)]_2$.
3. If both $q_p(z_i)$ are odd then set $z_1 = z_1 - z_2$ and $b_1 = 0$.
4. For each z_i with $b_i = 0$, set $z_i = (z_i - \mathbf{parity}(z_i))/2$.

Repeat this process until $z_2 = 0$. We then have that $q_p(z_1)$ equals the odd part of $\gcd(q_p(z_1), q_p(z_2))$ (for the two initial integers).

Step 4: Recovering p. Recovering p now is straightforward, $\mathcal{B}$ draws z_1^*, z_2^* from $\mathcal{D}_{\gamma,\rho}(p)$ and applies the algorithm. Now if the odd part of $\gcd(q_p(z_1^*), q_p(z_2^*))$ equals one, the

algorithm from above will return $\tilde{z} = p + r$, for some $|r| \leq 2^{\rho}$. Otherwise, repeat this with two new z_i^*.
Finally $\mathcal{B}$ runs the binary-GCD algorithm from above again with z_1^* and $\tilde{z}$. Note that since $q_p(\tilde{z})$ will always remain odd, it will never be changed. Now clearly the sequence of bits b_1 in the algorithm exactly equals the binary representation of $q_p(z_1^*)$ and $\mathcal{B}$ can recover $p = \lfloor z_1^*/q_p(z_1^*) \rceil$. □

4.4.3 The Fully Homomorphic Scheme

As usual, to make the scheme fully homomorphic we need to squash the decryption circuit. Recall that decryption only requires to compute $[\psi - \lfloor \psi/p \rceil]_2$. To make this step easier, we add to the public key a set $\mathbf{y} = \{y_1, y_2, \ldots, y_\xi\}$ where $y_i \in [0, 2)$ rational with κ bits of precision. This set is subject to the constraint that there exists a subset $S \subset \{1, \ldots, \xi\}$, with $|S| = \zeta$, for which it holds that $\sum_{i \in S} y_i \approx 1/p \mod 2$. In addition, the secret key is replaced by the indicator vector of S.
The encryption scheme then is as follows:

KeyGen Let sk^* and pk^* be as before. Now set $x_p = \lfloor 2^\kappa/p \rceil$ and let $\mathbf{s}$ be a random ξ-bit vector with Hamming weight ζ. Now let $S = \{i : s_i = 1\}$ and choose random integers $u_i \in \mathbb{Z} \cap [0, 2^{\kappa+1})$, such that $\sum_{i \in S} u_i = x_p \mod 2^{\kappa+1}$. Set $y_i = u_i/2^\kappa$ and define $\mathbf{y} = \langle y_1, \ldots, y_\xi \rangle$. It follows that these y_i are in the interval $[0, 2)$, have κ bits of precision and $[\sum_{i \in S} y_i]_2 = (1/p) - \Delta_p$ with $|\Delta_p| < 2^{-\kappa}$.

Encrypt **and** Evaluate Compute the ciphertext ψ^* as before and compute $z_i = [\psi^* \cdot y_i]_2$, with only $n = \lceil \log \zeta \rceil + 3$ bits of precision. Output ψ^* and $\mathbf{z} = \langle z_1, \ldots, z_\xi \rangle$.

Decrypt The plaintext is $\pi = [\psi^* - \lfloor \sum_i s_i z_i \rceil]_2$.

It is not obvious that the revised scheme is still correct, therefore the following lemma is presented.

Lemma 4.4.6. [vDGHV10, pp. 16] *The modified scheme is correct for $C(\mathcal{P}_\mathcal{E})$. Moreover, for any ciphertext $(\psi^*, \mathbf{z})$ generated by evaluating a permitted polynomial, it holds that $\sum_i s_i z_i$ is within 1/4 of an integer.*

Now it only remains to show that this scheme is indeed bootstrappable.

Theorem 4.4.7. [vDGHV10, pp. 17] *Let $\mathcal{E}$ be the revised scheme as above, and let $D_\mathcal{E}$ be the set of augmented squashed decryption circuits. Then we have that $D_\mathcal{E} \subset C(\mathcal{P})$.*

Proof. (outline) As we have seen in lemma 4.4.3, we require that the decryption circuit can be written as a polynomial with degree d satisfying $d \leq (\eta - 4 - \log |\mathbf{f}|)/(\rho' + 2)$. To show this, the decryption process is split up into three steps:

1. For $i \in \{1, \ldots, \xi\}$, set $a_i = s_i \cdot z_i$.

2. Generate $n+1$ numbers w_j, each with less than n bits of precision, such that $\sum_j w_j = \sum_i a_i \mod 2$.

3. Output $\psi^* - (\sum_j w_j) \mod 2$.

The first step is easy to express as a circuit with multiplicative depth 1, but the second and third are a bit tougher. To achieve this, a method analogous to the method as presented in lemma 4.1.16 is used. In short, since the a_i's have precision less than n, we can compute the Hamming weights "columnwise", and obtain $n+1$ numbers that sum up to $\sum_i a_i \mod 2$. Since at most ζ of the a_i's are nonzero, these w_j have precision $\lceil \log(\zeta + 1) \rceil < n$. The Hamming weights can be efficiently computed using elementary symmetric polynomials. To perform the final step, the three-for-two trick can be used [KR88, Sec. 4.2.2].
Omitting the further details, this shows the bootstrappability of the scheme when $\eta \geq \rho\xi(\lambda \log^2 \lambda)$. □

5 Implementation in JCrypTool

The plug-in for JCrypTool consists of three separate parts, an implementation of the fully homomorphic Gentry-Halevi variant, an implementation of RSA and an implementation of the Paillier cryptosystem. The implementation of RSA is built on an existing version of RSA which is already available within JCT. It simply uses the built-in encryption and decryption from RSA and shows the homomorphic properties by multiplying the corresponding ciphertexts. This is not very interesting, so it will not be discussed in this section. The implementation of Paillier is straightforward, so it is not discussed either.

5.1 Optimization in the Gentry-Halevi Variant

As mentioned in Section 4.3, several techniques proposed by Gentry and Halevi to speed up the different algorithms are used in the implementation in JCrypTool. These techniques will be described here.

5.1.1 KeyGen

In the KeyGen algorithm, first a random polynomial $v(x)$ is generated with t-bit integer coefficients. Next the scaled inverse $w(x)$ of this polynomial is to be computed. With this inverse it is easy to verify whether the HNF is of the correct form. First we will describe an efficient way of computing the inverse $w(x)$ and then we will show how to compute the HNF.

Inverting $v(x)$
To invert the polynomial $v(x)$, we use a method based on fast fourier transformation which computes a single odd coefficient of $w(x)$, which is all we need. Recall that we use $f_n(x) = x^n + 1$ (see 4.3.1), where n is a power of 2. Let $\rho_0, \rho_1, \ldots, \rho_{n-1}$ denote the roots of $f_n(x)$ over the complex field. We then have that $\rho_i = \rho^{2i+1}$, where ρ is a primitive $2n$'th root. For these roots we have the following properties:

$$\rho_{i+\frac{n}{2}} = \rho^{2i+1+n} = \rho^n \rho_i = -\rho_i,$$

for all i, and

$$\left(\rho_{i+n_j/2}\right)^{2^j} = (\rho^{2i+n_j+1})^{2^j} = (\rho^{2i+1})^{2^j} \cdot \rho^n = -(\rho_i^{2^j}),$$

for all i, $j = 0, 1, \ldots, \log_2 n$ and $n_j := n/2^j$. Furthermore, we define the polynomial

$$g(z) := \prod_{i=0}^{n-1} (v(\rho_i) - z),$$

which, as we will see later, is closely related to w. We only need to compute the free term and the linear term of $g(z)$, through the method as described below.

Computing the free and linear term of $g(z)$**:**
First we will show that all the coefficients g_i of $g(z)$ are integers, for all $n = 2^m$. For $m = 1$

this is easy to see, we have roots $\rho_0 = i$ and $\rho_1 = -i$. Now let $v(x) = ax + b$ for some $a, b \in \mathbb{Z}$, then $g(z) = (v(\rho_0) - z)(v(\rho_1) - z) = (ai + b - z)(b - ai - z) = a^2 + b^2 - 2bz + z^2$. Now suppose all the g_i are integers for some $m = k$, then for $m = k + 1$ we have

$$\begin{aligned} g(z) &= \prod_{i=0}^{2^{k+1}-1} (v(\rho_i) - z) \\ &= \prod_{i=0}^{2^k-1} (v(\rho_i) - z)(v(\rho_{i+2^k}) - z) \\ &= \left(\prod_{i=0}^{2^k-1} (v(\rho_i) - z) \right) \left(\prod_{j=0}^{2^k-1} (v^*(\rho_j) - z) \right), \end{aligned}$$

where $v^*(x) = v(-x)$. So we find that $g_i \in \mathbb{Z}$ for all i and all $n = 2^m$.
Now to compute the free and linear term of $g(z)$, it suffices to compute $g(z) \mod z^2$. We have the following

$$\begin{aligned} g(z) &= \prod_{i=0}^{\frac{n}{2}-1} (v(\rho_i) - z)(v(-\rho_i) - z) \\ &= \prod_{i=0}^{\frac{n}{2}-1} \left(\underbrace{v(\rho_i)v(-\rho_i)}_{a(\rho_i)} - z \underbrace{(v(\rho_i) + v(-\rho_i))}_{b(\rho_i)} + z^2 \right) \\ &= \prod_{i=0}^{\frac{n}{2}-1} (a(\rho_i) - zb(\rho_i)) \mod z^2. \end{aligned}$$

Note that for $a(x)$ and $b(x)$ all the odd powers have zero coefficients. Furthermore, since these polynomials are only evaluated at the roots of $f_n(x)$, reducing these polynomials modulo $f_n(x)$ has no effect. Define the polynomials $\hat{v}$ and $\tilde{v}$ as $\hat{v}(x^2) = a(x) \mod f_n(x)$ and $\tilde{v}(x^2) = b(x) \mod f_n(x)$. Now we have reduced a product of n terms in polynomials of degree n to a product of $n/2$ terms in polynomials of degree $n/2$. Applying this recursively leads to the polynomial $g(z) \mod z^2$.
More concretely, define $U_0(x) \equiv 1$ and $V_0(x) = v(x)$. Now we compute $U_j(x)$ and $V_j(x)$ for $j = 1, 2, \ldots, m = \log_2 n$, of degrees at most $n_j - 1$ such that

$$g_j(z) = \prod_{i=0}^{n_j-1} \left(V_j(\rho_i^{2^j}) - zU_j(\rho_i^{2^j}) \right) = g(z) \mod z^2.$$

This equation clearly holds for $j = 0$. Assume it holds for some $j < m$, then we can

compute U_{j+1} and V_{j+1} as follows:

$$\begin{aligned} g_j(z) &= \prod_{i=0}^{n_j/2-1} \left(V_j(\rho_i^{2^j}) - zU_j(\rho_i^{2^j})\right)\left(V_j(-\rho_i^{2^j}) - zU_j(-\rho_i^{2^j})\right) \\ &= \prod_{i=0}^{n_j/2-1} \left(\underbrace{V_j(\rho_i^{2^j})V_j(-\rho_i^{2^j})}_{=A_j(\rho^{2^j})} - z\underbrace{(U_j(\rho_i^{2^j})V_j(-\rho_i^{2^j}) + U_j(-\rho_i^{2^j})V_j(\rho_i^{2^j}))}_{=B_j(\rho_i^{2^j})} \right) \mod z^2. \end{aligned}$$

Denote $f_{n_j}(x) = x^{n_j} + 1$ and observe that $r_i^{n^j}$ is a root of f_{n_j} for all i. We consider the following polynomials:

$$\begin{aligned} A_j(x) &= V_j(x)V_j(-x) \mod f_{n_j}(x) \\ B_j(x) &= U_j(x)V_j(-x) + U_j(-x)V_j(x) \mod f_{n_j}(x) \end{aligned}$$

and note that since $\rho_i^{2^j}$ is a root of f_{n_j} the reduction modulo f_{n_j} makes no difference when evaluating in $\rho_i^{2^j}$. Furthermore, all the coefficients of odd powers are zero, and modular reduction does not change this since n_j is a power of two. So we can define

$$U_{j+1}(x) = \sum_{t=0}^{n_j/2-1} b_{2t} \cdot x^t,\ V_{j+1}(x) = \sum_{t=0}^{n_j/2-1} a_{2t} \cdot x^t,$$

which leads to

$$\begin{aligned} g_{j+1}(z) &= \prod_{i=0}^{n_j/2-1} \left(V_{j+1}(\rho_i^{2^{j+1}}) - zU_{j+1}(\rho_i^{2^{j+1}})\right) \\ &= \prod_{i=0}^{n_j/2-1} \left(A_j(\rho_i^{2^j}) - zB_j(\rho_i^{2^j})\right) = g_j(z) \mod z^2. \end{aligned}$$

Now by the induction hypothesis we have $g_j(z) = g(z) \mod z^2$, so we get $g_{j+1}(z) = g(z) \mod z^2$ as needed. Now that we have shown how to compute the two lower terms of $g(z)$, we next show how this helps us in finding $w(x)$.

Finding d and w_0:
We know that if $v(x)$ is square free then resultant$(v, f_n) = \prod_{i=0}^{n-1} v(\rho_i)$, which is exactly the free term of $g(z)$. Next we will show that $w_0 = g_1/n$, where w_0 is the free term in $w(x)$. We define a sequence of polynomials $\hat{W}_0, \hat{W}_1, \ldots, \hat{W}_m$, as

$$\hat{W}_j(x) = \sum_{t=0}^{n_j-1} w_{2^j t} x^t.$$

Now we have that $\hat{W}_0 = w$, $\hat{W}_m = w_0$ and in particular $\hat{W}_j(x) + \hat{W}_j(-x) = 2\hat{W}_{j+1}(x^2)$. We will show that

$$\sum_{i=0}^{n-1} w(\rho_i) = 2^j \sum_{i=0}^{n_j-1} \hat{W}_j(\rho_i^{2^j}),$$

for $j = 0, 1, \ldots, m$. Again it clearly holds for $j = 0$, so assume it holds for some $j < m$. Now since $\left(\rho_{i+n_j/2}\right)^{2^j} = -\rho_i^{2^j}$, we find

$$\sum_{i=0}^{n-1} w(\rho_i) = 2^j \sum_{i=0}^{n_j-1} \hat{W}_j(\rho_i^{2^j}) = 2^j \sum_{i=0}^{n_j/2-1} \hat{W}_j(\rho_i^{2^j}) + \hat{W}_j(-\rho_i^{2^j}) = 2^{j+1} \sum_{i=0}^{n_{j+1}-1} \hat{W}_{j+1}(\rho_i^{2^{j+1}})$$

which completes the induction proof. This now implies that $\sum_{i=0}^{n-1} w(\rho_i) = n\hat{W}_n(-1) = nw_0$, and since $w(\rho_i) = d/v(\rho_i)$ we find

$$\begin{aligned} w_0 = \frac{1}{n}\sum_{i=0}^{n-1} w(\rho_i) = \frac{1}{n}\sum_{i=0}^{n-1} \frac{d}{v(\rho_i)} &= \frac{1}{n}\sum_{i=0}^{n-1} \frac{\prod_{j=0}^{n-1} v(\rho_j)}{v(\rho_i)} \\ &= \frac{1}{n}\sum_{i=0}^{n=1} \prod_{j\neq i} v(\rho_j) = g_1/n. \end{aligned}$$

So we indeed find that $d = g_0$ and $w_0 = g_1/n$.

Recovering w:
Recall that we only require the lattice to contain a vector of the form $\mathbf{y}\times\mathbf{V} = \langle -r, 1, 0, \ldots, 0\rangle$ for $\mathbf{V}$ to have the correct HNF. Now if we multiply this vector on the right with $\mathbf{W}$, we find that

$$\mathbf{y} \times \mathbf{V} \times \mathbf{W} = d\mathbf{y} = -r \cdot \langle w_0, w_1, \ldots, w_{n-1}\rangle + \langle -w_{n-1}, w_0, \ldots, w_{n-2}\rangle.$$

This leads to the condition

$$-r \cdot \langle w_0, w_1, \ldots, w_{n-1}\rangle = \langle w_{n-1}, -w_0, \ldots, -w_{n-2}\rangle \mod d.$$

So we find $r = w_0/w_1 \mod d$ and $w_{i+1} = w_i/r \mod d$ for all $i = 1, \ldots, n-2$ and $w_{n-1} = r \cdot w_0 \mod d$. This in particular means that $r^n = 1 \mod d$. Now we have found w_0 and w_1, so we compute $r = w_0/w_1 \mod d$ and verify that $r^n = 1 \mod d$. Now we can compute all the coefficients of w until we have found an odd integer, which will be our secret key.

5.1.2 Encrypt

As shown in the previous section, encryption in this scheme is simply $\psi = [a(r)]_d$, for $a(x) = \pi + 2u(x)$. Now to evaluate the polynomial u at x, we could simply calculate all the powers of r and then add up to values where $u_i = 1$, this would take $n-1$ multiplications.

But this can be optimized, as we will show in this section.
Note that we can split the u into two polynomials with half the degree, $u^{low}(x) = \sum_{i=0}^{n/2-1} u_i x^i$ and $u^{up}(x) = \sum_{i=0}^{n/2-1} u_{i+n/2} x^i$. Now the evaluation is given by $u(r) = u^{low}(r) + r^{n/2} u^{up}(r)$, which now only requires $n/2 - 1 + 1 + 1$ multiplications, $n/2 - 1$ for the powers of r, one additional to compute the $n/2$'th power of r and one final to multiply $u^{up}(r)$ with $r^{n/2}$. We can apply this recursively. Let $M(k, n)$ denote the number of multiplications needed to evaluate k polynomials of degree $(n-1)$. Now it is clear that this gives us

$$M(k, n) = \min(n - 1, M(2k, n/2) + k + 1).$$

It is also clear that cutting the polynomial in half is more efficient as long as $n - 1 > (n/2 - 1) + k + 1$. This yields the recursive formula

$$M(k, n) = \begin{cases} M(2k, n/2) + k + 1 & \text{when } n/2 > k + 1 \\ n - 1 & \text{otherwise.} \end{cases}$$

Analysis of this formula yields $M(k, n) \leq \min(n - 1, \sqrt{2kn})$. So the number of multiplications this method uses for the evaluation of a single degree $n - 1$ polynomial is at most $\sqrt{2n}$.
This recursive method will be more efficient in calculation time, but requires more space. Since the dimensions used in the plugin are reasonably small, this is not an issue.

5.1.3 Decrypt

Decryption is already reduced to the multiplication of two integers, which can not be optimized any further. The only thing one could do is implement an efficient modular multiplication algorithm, but it is assumed that Java has these implemented by default.

5.1.4 Recrypt

At the moment there are three optimizations in place for the Recrypt operation. Namely, the addition of the S numbers $\sigma_i z_i$, the way the set of x_i's is stored and the way in which σ is encrypted.

Addition of the z_i's
We want to perform the addition of the z_i's with a polynomial of low degree. The achieve this, we first split up the vector σ. Recall that $w(\sigma) = s$, so we split σ into s vectors $\sigma_{\mathbf{k}}$, each with weight 1. This requires that we have s sets $\mathcal{B}_k$, one for each σ_k. Denote the elements of $\mathcal{B}_k$ by $\{x(k, i) : i = 1, 2, \ldots, S\}$, and the bits of σ_k by $\sigma_{k,i}$, $y(k, i)$ and $z(k, i)$ are defined as expected (see section 4.3.2). We now have the decryption function

$$D_{c,d}(\sigma_1, \ldots, \sigma_s) = \left\langle \left\lfloor \sum_{k=1}^{s} \left(\sum_{i=1}^{S} \sigma_{k,i} z(k, i) \right) \right\rceil \right\rangle_2 \oplus \bigoplus_{i,k} \sigma_{k,i} \langle y(k, i) \rangle_2.$$

Denote $q_k = \sum_i \sigma_{k,i} z(k,i)$ and note that this represents a sum of S numbers, of which at most one is nonzero. Hence we can bitwise XOR these S numbers, which homomorphically translates to adding $q_{k,j} \mod d$ for all $j = 0, 1, \ldots, \xi$.
Now that the sum of S $\xi+1$-bit numbers is reduced to a sum of s $\xi+1$-bit numbers, we use a grade-school addition algorithm to add them. This works as follows. Arrange these numbers is s rows, each with $\xi+1$ columns, one for each bit $j = 0, 1, \ldots, \xi$. These columns are processed starting at column ξ and moving down to column 0.
We first compute the carry bits the addition of these columns produce. The carry bit column $j+\delta$ gets from column j is computed by evaluating the elementary symmetric polynomial of degree 2^δ in the bits of column j. The maximum degree δ for $j = \xi, \xi-1, \ldots, 1$ is respectively $\xi, \xi-1, \ldots 1$. And thus the number of bits to be processed in each column is respectively $s, s+1, \ldots, s+\xi-1$. For a column with m bits and degree 2^δ, we need at most $m2^\delta$ multiplications to compute the elementary symmetric polynomial. So we find the total number of multiplications is bounded by $\sum_{k=0}^{\xi-1}(s+k)\cdot 2^{\xi-k} = 2(s(2^\xi-1)+2^\xi-\xi-1) = O(s^2)$.

Storing the x_i's
We have seen that we need to include in the public key s sets of S elements in $\mathbb{Z}_d$. Storing all these elements explicitly requires a lot of space. Instead, we will store s elements x_i, each of which represents a set $\mathcal{B}_i$, through a geometric progression. To be exact $\mathcal{B}_k = \{x(k,i) : i = 0, \ldots, S-1\}$ where $x(k,i) = \langle x_k \cdot R^i \rangle_d$ and R is some predefined parameter.

Encrypting the $\sigma_\mathbf{k}$
We now have s vectors, each with S bits and weight one which must be encrypted and included in the public key. Encrypting every single bit thus would require storing $s \cdot S$ elements of $\mathbb{Z}_d$ in the public key. Instead, we will implicitly represent these bits. This is achieved as follows.
We will store c encrypted bits for every $\mathcal{B}_k$, all but two of these are encryptions of zero. Now $\sigma_{k,i}$ is obtained by multiplying two of these ciphertexts, i.e. for $a, b \in [1, c]$ and $a > b$ we have

$$i(a,b) := (a-1)\cdot c - \binom{a}{2} + (b-a).$$

Note that these are all the pairs of two distinct numbers one can choose in $[1, c]$, numbered lexicographically. Denote these encryptions by

$$\{\eta_m^{(k)} : k \in [1, s], m \in [1, c]\}, \tag{10}$$

such that $\eta_a^{(k)}\eta_b^{(k)}$ is the encryption of $\sigma_{k,i(a,b)}$.
Recall we want to compute $q_k = \sum_{i=0}^{S-1} \sigma_{k,i} z(k,i)$, which now can be achieved using

$$q_k = \sum_{a,b} \eta_a^{(k)}\eta_b^{(k)} z(k, i(a,b)) = \sum_{a=1}^{c} \eta_a^{(k)} \sum_{b=a+1}^{c} \eta_b^{(k)} z(k, i(a,b)).$$

Note that the multiplications with $\eta_b^{(k)}$ are not in fact multiplications, since $z(k, i(a, b))$ are bits that we have in the clear. So the only multiplications in $\mathbb{Z}_d$ are those with $\eta_a^{(k)}$, and there are c of those.
There is a space-time tradeoff here in choosing c. We need at least $c \geq \lceil \sqrt{2S} \rceil$ to be able to encode all the $i \in [0, S-1]$ by a pair $(a, b) \in \binom{c}{2}$. Thus increasing c will increase the public key size, but decrease the amount of extra multiplications needed. In the implementation $c = \lceil 2\sqrt{S} \rceil$ is chosen, since it increases the space requirements only by a $\sqrt{2}$ factor, but halves the amount of extra multiplications needed.

5.2 Practical Implementation of the Gentry-Halevi Variant

In this section we will give an overview of the implementation in JCrypTool of the Gentry-Halevi variant. We will first cover KeyGen, Encrypt and Recrypt and then discuss the appearance of the plug-in and what is done in de background. Decrypt is left out of scope, since it is straightforward to implement. Note that the implementation of KeyGen, Encrypt and Recrypt is based on the C-code by Gentry and Halevi (see `https://researcher.ibm.com/researcher/view_project.php?id=1548`).

5.2.1 KeyGen

Polynomials
First of all, a random polynomial $v(x)$ must be generated. Java does not have a built-in class to handle polynomials, so I built one. The coefficients of these polynomials will have bit-length $t = 384$, so the BigInteger class is used to store these numbers. The polynomial then is stored as an array of BigIntegers. Randomly selecting the coefficients is done by generating the correct amount of random bytes ($t/8$). The constructor of a BigInteger accepts byte-arrays to create the number.
Besides generating a random polynomial, the necessary functions for computation with polynomials were added. These are for instance addition and multiplication, but of course also modular reduction. Since these functions are fairly basic, they are not included here.

Computing the determinant, root and w
Next the determinant and the first odd coefficient of $w(x)$ are computed. Recall $w(x)$ is the scaled inverse of $v(x)$ and an odd coefficient is all that is needed for decryption. As mentioned in section 5.1.1, this is done by computing the free and the linear term of $g(z) = \prod_{i=0}^{n-1}(v(\rho_i) - z)$, where the ρ_i are the roots of $f(x)$. Clearly it thus suffices to compute $g(z) \mod z^2$. This is done in the loop given in figure 4, where N is halved every iteration.

```
while (N>1) {
  V2 = new Polynomial(V.coeffs);
  for (i=1; i<=V2.degree; i+=2) { // set V2(x) := V(-x)
    V2.coeffs[i] = V2.coeffs[i].negate(); // negate odd coefficients
  }
  V = Polynomial.mod(Polynomial.mult(V, V2), F);// V := V(x) * V(-x) mod f(x)
  U = Polynomial.mod(Polynomial.mult(U, V2), F);// U := U(x) * V(-x) mod f(x)

  // Sanity-check: verify that the odd coefficients in V are zero
  for (i=1; i <= V.degree; i+=2)
        if (!V.coeffs[i].equals(new BigInteger("0"))) {
      return null;
  }

  // "Compress" the non-zero coefficients of V
  for (i = 1; i <= V.degree/2; i++) V.coeffs[i] = V.coeffs[2*i];
  for (    ;    i <= V.degree;    i++) V.coeffs[i] = new BigInteger("0");
  V.normalize();

  // Set U to the "compressed" ( U(x) + U(-x) ) /2
  for (i = 0; i <= U.degree/2; i++) U.coeffs[i] = U.coeffs[2*i];
  for (    ; i <= U.degree;    i++) U.coeffs[i] = new BigInteger("0");
  U.normalize();

  // Set N := N/2 and update F accordingly
  F.coeffs[N] = new BigInteger("0");
  N >>= 1;
  F.coeffs[N] = new BigInteger("1");
  F.normalize();
}
```

Figure 4: Computing $g(z) \bmod z^2$

Note that in the description in section 5.1.1 one takes

$$U_{j+1} = U_j(x^2)V_j((-x)^2) + U_j((-x)^2)V_j(x^2) \mod f_{n_j}(x),$$

and in the code we have

$$U_{j+1} = (U_j(x^2)V_j((-x)^2) + U_j((-x)^2)V_j(x^2))/2 \mod f_{n_j}(x).$$

This is more efficient, since in this way we find $w_0 = g_1$ instead of $w_0 = g_1/n$. Also note that this does not influence g_0, and the other terms are not used since the polynomial is taken modulo z^2.
But to generate the entire polynomial $w(x)$, we need both w_0 and w_1. So we repeat the process with $x \cdot v(x) \mod f(x)$ to find w_1. Note that this yields the same determinant, since the lattice is cyclic.
Now to verify that $v(x)$ provides a usable lattice, we just need to check that the determinant is odd and that the root $r = w_0/w_1$ satisfies $r^n \equiv 1 \mod d$. Finally, we compute coefficients w_i, until we find an odd coefficient. These computations are straightforward.

Encrypting the private key
Of course, we still need to encrypt the private key w and add this encryption to the public

key. As explained in section 5.1.4, we will create s numbers x_i, which represent geometric progressions. Along with these numbers, we also need s exponents e_i, such that

$$\sum_{i=0}^{s-1} x_i R^{e_i} \equiv w \mod d.$$

We generate these sets as follows.

- Choose the $x_i \in \mathbb{Z}_d$ and $e_i \in \mathbb{Z}_S$ uniformly at random, for $i = 0, \ldots, s-2$,
- compute $\tilde{x} = w - \sum_{i=0}^{s-2} x_i R^{e_i} \mod d$,
- choose $e_{s-1} \in \mathbb{Z}_S$ uniformly at random until $R^{e_{s-1}}$ is invertible modulo d,
- compute $x_{s-1} = \tilde{x} R^{-e_{s-1}}$.

These x_i will be included in the public key. Finally, the exponents e_i are encoded as in equation 10, then encrypted, and also included in the public key.
How to code these computations is also relatively straightforward, so not included here.

5.2.2 Encrypt

For encryption, the only hard part is evaluating the random polynomial. As described in section 5.1.2, we will split the polynomial into two pieces as long as $m/2 > n + 1$, where n is the number of polynomials of degree $m - 1$. The implementation can thus be considered in two parts, splitting the polynomials and evaluating the final polynomials. We will comment on both parts here.

Splitting the polynomials
To split up the polynomials, a recursive function is used. In the code, we use m to denote the degree of the polynomial and n to denote the number of polynomials. The function returns an array which holds the value r^m in the first entry, and the evaluation of n polynomials of degree $m - 1$. First, we check if we want to split up the polynomial or not. If this is the case, a recursive call is made. If this is not the case, the polynomials are evaluated.
The result of the recursive call thus holds $r^{\lfloor m/2 \rfloor}$ and the $2n$ evaluated polynomials. If m is odd, we have polynomials of degree $\lfloor m/2 \rfloor$, so if we would simply multiply the "top half" by $r^{\lfloor m/2 \rfloor}$, the degree would become $m - 1$ instead of m. Thus we must add another coefficient to the "top half". This also means that if we pass on the power of r to the next level, we must first square the current power and then multiply the result by r. The recursive function is given in figure 5.

```
public static BigInteger[] evalRandPoly(int n, long m, double p,
                                        BigInteger root, BigInteger det) {
  BigInteger[] vals;
  if ((n+1+(m&1) < m/2)) {
    double q;
    vals = evalRandPoly(2*n, m/2, p, root, det); //{root^{m/2},c0,c1,...}
      for (int i = 1; i <= n; i++) {
            // vals[i] += root^{m/2} * vals[i+n] mod det
            // If m is odd then add another random 0/1 coefficient
        if (((m&1) == 1) && ((q = Math.random()) < p)) {
          vals[i+n] = ((q < p/2) ? vals[i+n].add(vals[0]).mod(det)
                                 : vals[i+n].subtract(vals[0]).mod(det));
        }
            // multiply "top half" by root^{d/2}
        BigInteger tmp = vals[i+n].multiply(vals[0]).mod(det);
        vals[i] = vals[i].add(tmp);
      }
      // compute root^m for the next level
      vals[0] = vals[0].modPow(new BigInteger("2"), det);
        // if m is odd, multiply by r again
      if ((m&1) == 1) vals[0] = vals[0].multiply(root).mod(det);
  } else {
    vals = basicRandPoly(n, m, p, root, det);
  }
  return vals;
}
```

Figure 5: Splitting the polynomials

Computing the polynomials

When the polynomials are split up into many polynomials of low degree, it remains to evaluate these polynomials. There is still some room for improvement here. For cases where the polynomials have degree smaller than four, the evaluation is trivial. But if the degree is larger, we make use of the fact that squaring large numbers is faster than multiplying. To make use of this, we first compute r^{2^j}, for $2^j < m$. We continue by computing the odd powers of r, and then all squares of these. To find the next odd power, one simply multiplies the previous with r^2, which is stored in the beginning.

Note that with this method, all powers of r are handled. To see this, suppose we are at odd power $2j+1$, and the power $k < 2j+1$ is not handled. We may assume k is unique, since otherwise it would have turned up before. Then either $k = 2j'+1$ or $k = 2j'$. In the former case, it is an odd power and is thus in the iteration. In the latter case, j' is handled and thus also $2j'$.

The method which evaluates these polynomials is given in figure 6.

```
public static BigInteger[] basicRandPoly(int n, long m, double p,
                                         BigInteger root, BigInteger det) {
  BigInteger[] vals = new BigInteger[n+1];
  int i,j,k;
  if (m <= 0) {
    vals[0] = new BigInteger("1");
    return vals;
  }
  double q;
  for (i = 1; i <= n; i++) vals[i] = new BigInteger(Integer.toString(
              (((q = Math.random()) < p) ? ((q < p/2) ? -1 : 1) : 0 )));
  if (m==1) {
    vals[0] = root;
    return vals;
  }

  BigInteger rSqr = root.modPow(new BigInteger("2"), det); //root^2 mod det
  BigInteger rPowm;
  // Handle the powers 1,2,4,... separately (saves maybe 1-2 mults)
  for (i = 1; i <= n; i++) {
    if ((q = Math.random()) < p) {
      vals[i] = (q < p/2) ? vals[i].add(root) : vals[i].subtract(root).mod(det);
    }
    if (m > 2 && ((q = Math.random()) < p)) {
      vals[i] = (q < p/2) ? vals[i].add(rSqr).mod(det)
                : vals[i].subtract(rSqr).mod(det);
    }
  }
  if (m>4) {
    rPowm = rSqr;
    for (j = 4; j < m; j *= 2) {
      rPowm = rPowm.modPow(new BigInteger("2"), det); // r^j mod det
      for (i = 1; i <= n; i++) {
        if ((q = Math.random()) < p) {
          vals[i] = (q < p/2) ? vals[i].add(rPowm).mod(det)
                              : vals[i].subtract(rPowm).mod(det);
        }
      }
    }
  } else if (m<4) { // if m==2 or 3 we're done, just return the correct r^m
    vals[0] = ((m == 2) ? rSqr : rSqr.multiply(root).mod(det));
    return vals;
  }

  // Compute r^j,r^{2j},r^{4j},..., and add to all values
  BigInteger rOddPow = root;
  for (j = 3; j < m; j += 2) {
    rOddPow = rOddPow.multiply(rSqr).mod(det); // next odd power of r
    rPowm = rOddPow;
    k = j;
    while (true) {
      for (i = 1; i <= n; i++)
        if ((q = Math.random()) < p) {
          vals[i] = (q < p/2) ? vals[i].add(rPowm).mod(det)
                              : vals[i].subtract(rPowm).mod(det);
        }
      k *= 2;
      if (k >= m) break;
            // r^k := (previous-r^k)^2 mod det
      rPowm = rPowm.modPow(new BigInteger("2"), det);
    }
  }

  // r_odd_power is r^{m-1} or r^{m-2}, depending  on whether m is even or odd
  vals[0] = (((m&1) == 1) ? rOddPow.multiply(rSqr).mod(det)
            : rOddPow.multiply(root).mod(det));
  return vals;
}
```

Figure 6: Computing the polynomials

5.2.3 Recrypt

For the Recrypt algorithm, the most interesting part concerning implementation is the evaluation of the decryption function, which has been reduced to

$$D_{c,d}(\sigma_1,\ldots,\sigma_s) = \left\langle \left\lfloor \sum_{k=1}^{s} \left(\sum_{i=1}^{S} \sigma_{k,i} z(k,i) \right) \right\rceil \right\rangle_2 \oplus \bigoplus_{i,k} \sigma_{k,i} \langle y(k,i) \rangle_2.$$

As mentioned before, the $\sigma_{k,i}$ are encoded and encrypted as

$$\{\eta_m^{(k)} : k \in [1,s], m \in [1,c]\},$$

so the decryption function reduces to

$$D_{c,d}(\sigma_1,\ldots,\sigma_s) = \left\langle \left\lfloor \sum_{k=1}^{s} q_k \right\rceil \right\rangle_2 \oplus \bigoplus_{i,k} \sigma_{k,i} \langle y(k,i) \rangle_2,$$

where

$$q_k = \sum_{a=1}^{c} \eta_a^{(k)} \sum_{b=a+1}^{c} \eta_b^{(k)} z(k, i(a,b)).$$

Recall that $z(k,i(a,b))$ is the ξ-bit approximation of $\langle \psi x(k,i(a,b)) \rangle_d / d$. The computation of these q_k is done in the loop given in figure 7, where p is used for the precision instead of ξ.

```
int j,j1,j2;

for (j = j1 =0; j1 < nCtxts-1; j1++) { // sk-bits indexed by (j1,*) pairs
    for (int k = 0; k < psums.length; k++) psums[k] = new BigInteger("0");

    for (j2 = j1 + 1; j2 < nCtxts; j2++) {
      // get the top bits of factor/det. The code below assumes
      // that p+1 bits can fit in one long
      long binary = getBinaryRep(factor, det, vars.length);
      if (factor.testBit(0)) {    // "xor" the LSB to column 0
        binary ^= (1 << fheparams.p);
      }
      // For every 1 bit, add the current ciphertext to the partial sums
      for (int k = 0; k < psums.length; k++) if (((binary>>k)&1) == 1) {
      int k2 = psums.length - k - 1;
      psums[k2] = psums[k2].add(ctxts[baseIdX + j2]).mod(det);

      }
      j++;              // done with this element
      if (j < fheparams.S) { // compute next element = current * R mod det
        factor = factor.shiftLeft(fheparams.logR).mod(det);
      } else break;       // don't add more than S elements
    }

    // multiply partial sums by ctxts[j1], then add to sum
    for (int k = 0; k < vars.length; k++) {
      psums[k] = psums[k].multiply(ctxts[baseIdX + j1]).mod(det);
      vars[k] = vars[k].add(psums[k]).mod(det);
    }

    if (j >= fheparams.S) break;
}
```

Figure 7: Loop to compute q_k

Here the ξ-bit approximation for q_k will be stored in the positions 1 through ξ of the array psums. For every bit in this representation, add $\eta_b^{(k)}$ to the corresponding position of the partial sum. Note that $j1$ and $j2$ in the code represent respectively a and b in the above sum. Since we need to compute $y(k, i(a, b))$ to compute $z(k, i(a, b))$ anyway, we store the least significant bit of this number in position 0 of psums (the rightmost part of the decryption function). After all $j2$ are covered, multiply the partial sums with the right $\eta_a^{(k)}$.

Now that all q_k are computed, it remains to add and round these s numbers of ξ bits. These numbers are thus stored in the last ξ columns of an $s \times (\xi + 1)$-matrix. As explained in section 5.1.4, these numbers are added using a grade school addition (i.e. add the numbers column by column and add the carry to the next column, etc.) , where the carries are found by evaluating the elementary symmetric polynomials. Recall that homomorphic addition corresponds with XOR and multiplication with AND. The columns are put in stacks, so that it is easy to put the carries on top of the specific rows. The evaluation of these polynomials is the most interesting part, this is done in the loop given in figure 8.

```
int i, j;
BigInteger[] out = new BigInteger[deg+1];
out[0] = new BigInteger("1");
for (i = 1; i <= deg; i++) out[i] = new BigInteger("0");

BigInteger tmp;
for (i=1; !vars.empty(); i++) {  // process the next variable, i=1,2,...
  for (j = Math.min(i,deg); j>0; j--) { // compute the j'th elem. sym. poly
    tmp = out[j-1].multiply(vars.peek()).mod(det);
    out[j] = out[j].add(tmp).mod(det); // out[j] += out[j-1] * vars.top() mod M

    // At the end of the inner loop, out[j] holds the
    // j'th symmetric polynomial in the first i variables
  }
  vars.pop();  // done with this variable
}
```

Figure 8: Evaluation of the elementary symmetric polynomials

In this loop, vars denotes the stack of the column to be computed and deg the maximum degree of the polynomial. This is done first for column $\xi + 1$, then column ξ, down to column 2. To see that this loop works, it is easiest to consider a toy example. Suppose that we want to compute the elementary symmetric polynomials in 3 variables, up to degree 2. Denote the elements in vars by $X_1, \ldots, X_4$. We will show how the output array out changes in every step.

$$\begin{pmatrix}1\\0\\0\end{pmatrix} \longrightarrow_{i=1}^{j=1} \begin{pmatrix}1\\X_1\\0\end{pmatrix} \longrightarrow_{i=2}^{j=2} \begin{pmatrix}1\\X_1\\X_1 \cdot X_2\end{pmatrix} \longrightarrow_{i=2}^{j=1} \begin{pmatrix}1\\X_1 + X_2\\X_1 \cdot X_2\end{pmatrix} \longrightarrow_{i=2}^{j=1}$$
$$\begin{pmatrix}1\\X_1 + X_2\\X_1 \cdot X_2\end{pmatrix} \longrightarrow_{i=3}^{j=2} \begin{pmatrix}1\\X_1 + X_2\\X_1 \cdot X_2 + X_1 \cdot X_3 + X_2 \cdot X_3\end{pmatrix} \longrightarrow_{i=3}^{j=1} \begin{pmatrix}1\\X_1 + X_2 + X_3\\X_1 \cdot X_2 + X_1 \cdot X_3 + X_2 \cdot X_3\end{pmatrix}$$

Note that the result of the addition is in position 1, and the carry for i columns to the left is in position 2^i. Once the sums and carries are computed for columns 2 through $\xi + 1$, the result is given in position 1 of the output of the evaluation of column 2. This result then is added to the sum of column 1 and the computation is complete.

5.2.4 Functionality

The schemes as presented in this thesis all work on a single bit. To show the user that this is enough to do computations on the ciphertext, the plug-in encrypts several bits in parallel. These bits then represent integers which the user chooses to perform computations. These parallel bits cause a decrease in performance. To compensate this somewhat, the security is set very low.
To be able to perform computations on these parallel bits, we need algorithms two perform

addition and multiplication. These are as follows, where a and b are assumed to have n bits.

```
ADDITION(a, b)
1  d_0 ← 0, e_0 ← 0
2  for i ← 0, ..., n − 1
3        do c_i ← a_i + b_i
4           d_{i+1} ← a_i · b_i
5           e_{i+1} ← c_i · (d_i + e_i)
6           f_i ← c_i + d_i + e_i
7
8  return f
```

Proof. (correctness of ADDITION) First, note that addition is inherently modulo 2. Now to be correct, there cannot be more carries than the ones given in d and e. We note that there are only two possible levels of carries, namely the carries that follow directly from the sum of a and b, and the carries that follow from the sum of a and b and a previous carry. Clearly, if a_i and b_i both are set, then they give the carry d_{i+1}. And if d_i is set and c_i is set, this gives another carry e_{i+1}.
Now we must show that the above given procedure finds both of these carries. To see this, we must show that d_i and e_i cannot both be set. If this is true, then it is easy to see that the procedure is correct.
We show that d_i and e_i cannot both be set by contradiction. Suppose both d_i and e_i are set. For e_i to be set, we must have that c_{i-1} is set and either d_{i-1} or e_{i-1} is set. But if c_{i-1} is set, we have that either a_{i-1} or b_{i-1} is set, but not both. But then d_i is not set. This is a contradiction.
So we find that d_i and e_i cannot both be set, thus taking $e_i = c_{i-1} \cdot (d_{i-1)} + e_{i-1})$ finds the necessary carry and the procedure is correct. □

To do multiplication we use the standard binary multiplication algorithm; simply multiply a with each bit of b and add these numbers using the addition algorithm.
After the implementation was finished, there was the open question whether a scheme which could encrypt several bits at a time was possible. There was some correspondence with prof. dr. Müller-Quade and Mr. Döttling (Karsruhe Institute of Technology), but before a solution was found one was published by Smart and Vercauteren [SV11].

5.2.5 Appearance

The Homomorphic Encryption plug-in consists of three different tabs, one for each of the implemented schemes. We will describe the tab which shows the Gentry-Halevi scheme, the tabs for RSA and Paillier are similar. When the user opens the plug-in, the plug-in opens as shown in figure 10.

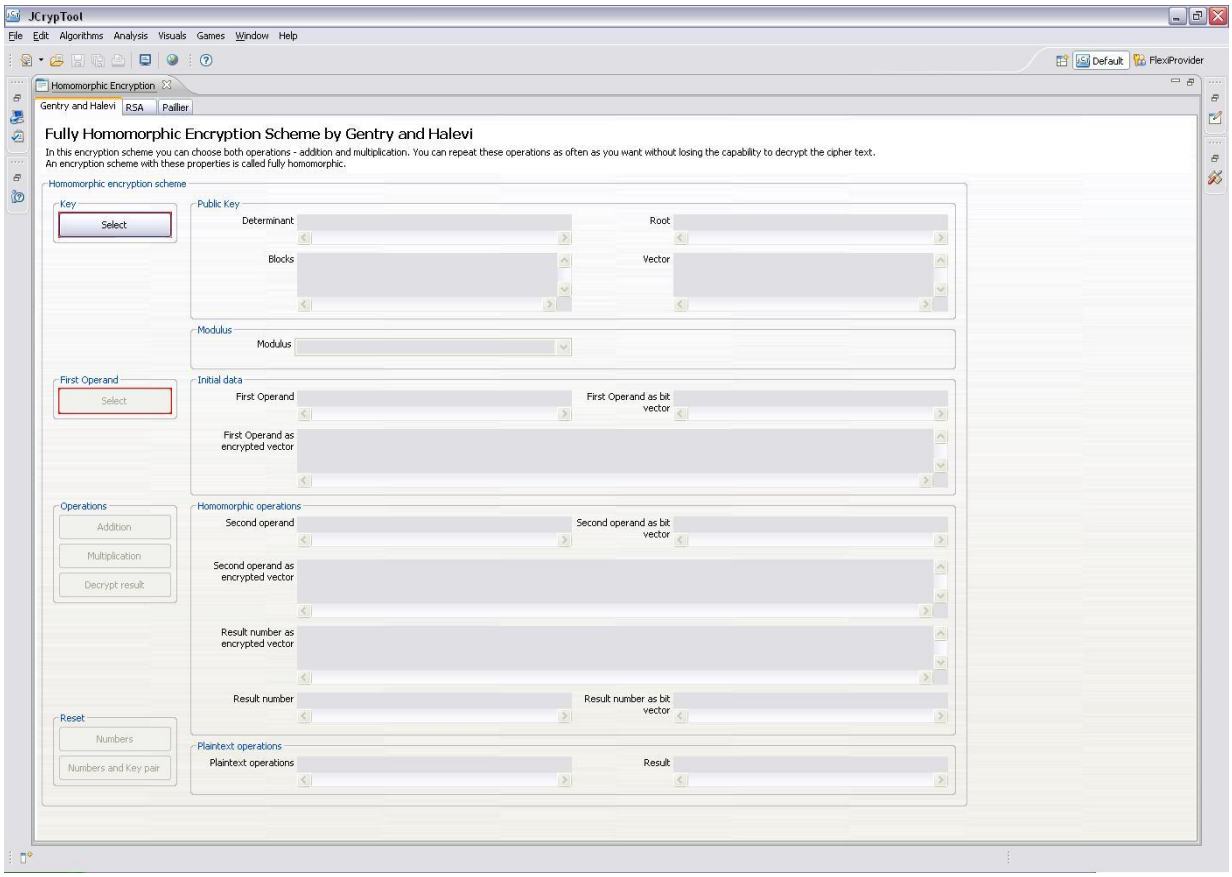


Figure 9: JCrypTool plug-in of the Gentry and Halevi fully homomorphic encryption scheme

To try the scheme, clearly the user first has to have a key-pair. The plug-in can generate key-pairs with lattice dimensions 2^d, for $d = 2, \ldots, 6$. The maximum dimension is limited to reduce the maximum computation time during addition and multiplication. Once a key-pair is generated, the user can choose to save the key-pair. To save a key-pair, one has to enter the name of the owner and choose a password, with which the private key is encrypted. The private key is written to a text file and it is encrypted with DES, using the MD5 hash of the password. This type of encryption is chosen since there exist built-in Java functions to perform these kinds of encryptions.
Having selected a key-pair, the user must choose a modulus for computations. The available moduli are 2^m, for $m = 5, \ldots, 10$. Now when the user enters an initial operand to encrypt, this number will be taken modulo the modulus and the binary representation will be encrypted bitwise.
Now the user can choose an operation to perform on this ciphertext; either add a number, or multiply with a number. In both cases, the second operand will be entered, taken modulo the modulus, and the computation is carried out homomorphically. The encrypted result will be shown as soon as the computation is complete. At the bottom of the screen,

the computations are also shown in plaintext, so that the user can keep track of what has happened so far.

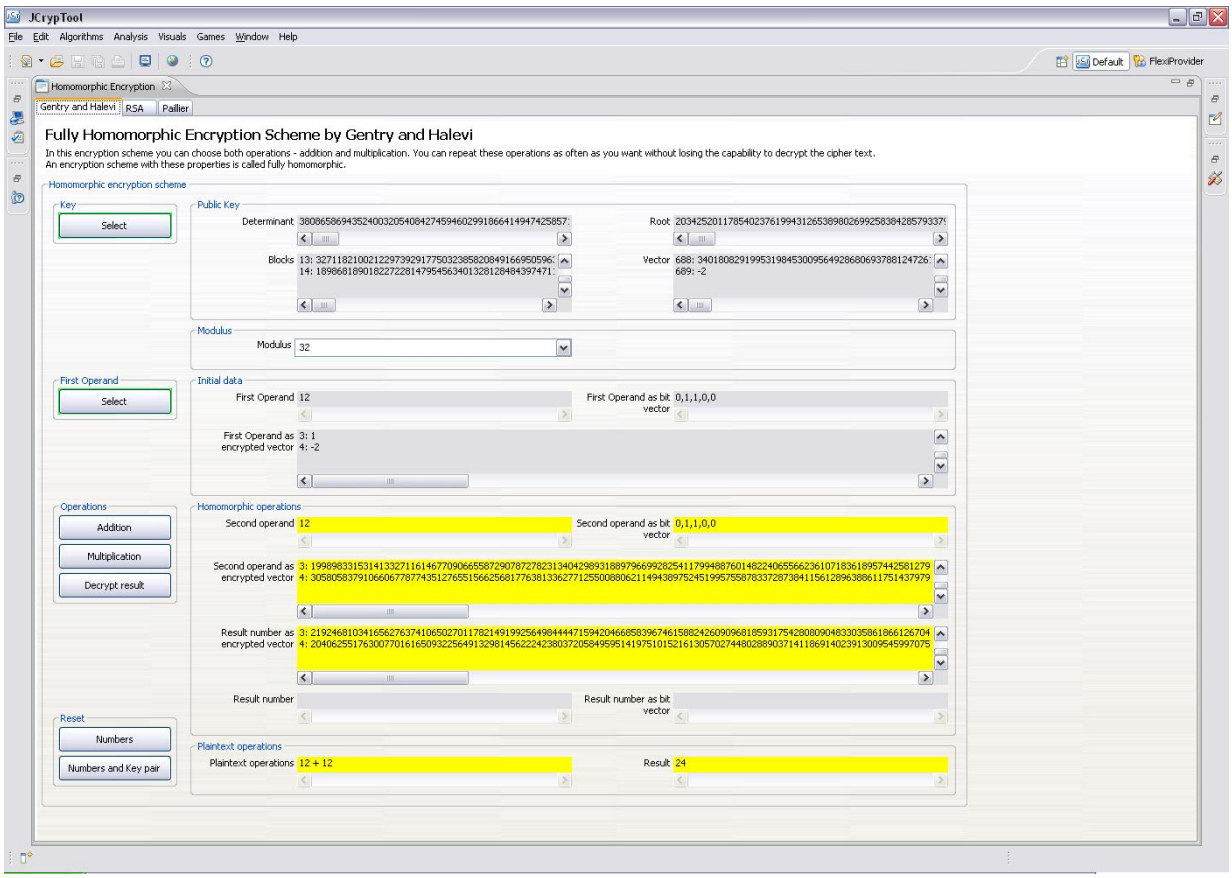


Figure 10: JCrypTool plug-in of the Gentry and Halevi fully homomorphic encryption scheme after computations

When some computations are carried out, the user might want to check if the ciphertext actually encrypts the correct result. To verify this, the user can choose to decrypt the current result and compare the decryption with the plaintext result given below. But of course, this will always be correct. The reader can verify this by trying it, JCrypTool is available through `http://sourceforge.net/projects/jcryptool/`.

5.3 Performance

As mentioned before, the Java implementation of the Gentry-Halevi scheme is based on the C-code which they published. Here we will list some of the performance results of both implementations. Note that the performance results from Gentry and Halevi are gathered on a high-end server (IBM System x3500 server with a 64-bit quad-core Intel Xeon E5450 processor @ 3GHz with 12MB L2 cache and 24GB of RAM), whereas the performance

results of our implementation are gathered on a modern household PC (Intel Core i7 @ 2.93 GHz, 8GB of RAM). This is done because the plug-in is supposed to run on the PC of an average user.
The following settings are used in the plug-in, for all the possible dimensions, and in the C implementation for dimension 2^{11}.

	Java	C
Security parameter	$\lambda = 72$	$\lambda = 80$
Bit-size of coefficients of $v(x)$	$t = 384$	$t = 380$
Sparse subset size	$s = 15$	
Big set size	$S = 512$	
Precision parameter	$\xi = 4$	

Table 4: Parameter settings

These parameters give the following speeds in several dimensions, all times below are given in the notation (hh:)mm:ss.ms. Note that in these dimensions, the scheme is trivial to break. But the purpose of this plug-in is just to visualize that building a fully homomorphic encryption scheme is possible.

Dimension	4	8	16	32	64
KeyGen	0:00.093	0:00.047	0:00.094	0:00.313	0:01.342
Encrypt single bit	-	-	-	0:00.015	0:00.063
Encrypt 690 bits simultaneously	-	-	0:00.048	0:00.328	0:00.485
Decrypt single bit	-	-	-	-	0:00.016
Recrypt single bit	0:00.047	0:00.094	0:00.312	0:01.061	0:03.995

Table 5: Algorithm speeds Java implementation

As a comparison, the C implementation by Gentry and Halevi gives the following speeds in high dimensions. Here dimension 512 is still considered to be a toy example. Dimension 32768 is expected to have security comparable to 1024 bits RSA.

Dimension	512	2048	8192	32768
KeyGen	0:02	0:41	8:24	2:12:00
Encrypt	0:00.19	0:01.8	0:19	3:00
Decrypt	-	0:00.02	0:00.13	0:00.66
Recrypt	0:06	0:32	2:48	31:00

Table 6: Algorithm speeds C implementation

But as mentioned, in the scheme we want the user to be able to multiply and add numbers, not bits. So we encrypt these numbers as several bits and perform addition and multiplication on them. To see why the dimensions are kept below 64, the running times

for addition and multiplication are given. Note that the speeds are not dependent on the numbers, since the scheme always performs the same operations, as the bits are encrypted. There is still some variability, but it is not linked directly to the input.

Modulus	32	64	128	256	512	1024
Addition dim 4	0:00.827	0:00.920	0:01.139	0:01.310	0:01.497	0:01.670
Addition dim 8	0:02.262	0:02.777	0:03.307	0:03.869	0:04.430	0:04.976
Addition dim 16	0:07.254	0:09.064	0:10.873	0:12.761	0:14.633	0:16.380
Addition dim 32	0:26.067	0:32.776	0:38.907	0:45.396	0:52.120	0:58.563
Addition dim 64	1:37.423	2:01.696	2:26.702	2:51.169	3:16.523	3:40.817
Multiplication dim 4	0:04.509	0:06.303	0:08.908	0:12.402	0:15.335	0:19.110
Multiplication dim 8	0:13.322	0:19.937	0:27.675	0:36.801	0:47.299	0:58.859
Multiplication dim 16	0:43.977	1:05.364	1:32.060	2:01.556	2:35.579	3:13.861
Multiplication dim 32	2:36.795	3:56.360	5:29.420	7:12.423	9:14.143	11:29.531
Multiplication dim 64	9:46.007	12:57.595	18:03.075	23:52.620	30:23.06	37:48.470

Table 7: Operation speeds Java implementation

6 Lunchtime Attack on the Gentry-Halevi Variant

In [LMSV10], Loftus et al. show that the Gentry-Halevi variant, in the form as presented above, is susceptible to a so called lunchtime attack. Indeed, a straightforward attack is able to recover the secret key in polynomial time. In this section we will first define the concept of a lunchtime attack and then show how the attack works. After this, we will discuss the concept of CCA1 secure fully homomorphic encryption.

6.1 What is a Lunchtime Attack?

The name lunchtime attack comes from the idea that an adversary can use the computer of the private key owner, which is able to decrypt messages, while the owner is out for lunch. In fact, a lunchtime attack is more formally called a non-adaptive chosen ciphertext attack, also known as CCA1 [BDPR98]. The notion is based on the semantic security game (see 3.3), except that in this case, the adversary has access to a decryption oracle which returns the plaintext corresponding to the given ciphertext [NY90]. The adversary only has access to this oracle when it has not yet received the challenge from the verifier. This clarifies the non-adaptive part; the queries made to the oracle can not depend on the given challenge. Adaptive chosen ciphertext attacks (CCA2) are a natural extension, where the adversary also has acces to this oracle after being given the challenge. The only restriction is that it can not query the oracle on the challenge ciphertext. Clearly, this notion is not interesting for any homomorphic encryption scheme, since the adversary could take the encryption ψ' of a known plaintext π', query the oracle on $\psi + \psi'$ and obtain π from $(\pi + \pi') - \pi'$.

6.2 The Attack

We know that the private key consists only of the (odd) integer w and decryptions follows from $[\psi \cdot w]_d$. Now assume that the adversary has access to an oracle $\mathcal{O}_D(\psi)$ which returns the corresponding plain text b to every cipher text ψ with which it is presented. If the adversary can make polynomially many queries to this adversary, w can be recovered by using an algorithm as presented in [LMSV10, p. 9], which is given below.
The algorithm builds on the idea that we may assume that $z \in [0, d)$. The algorithm starts out with an interval $[L, U]$ which surely contains z. Now since $[\psi \cdot z]_d$ maps to $[-d/2, d/2)$, we can choose a subinterval of $[L, U]$, depending on whether k multiples of d suffice to have $\psi \cdot w - k \cdot d \in [-d/2, d/2)$ or $k + 1$ are needed. The algorithm is as follows.

CCA1-ATTACK(d)

```
1  L ← 0, U ← d − 1
2  while (U − L > 1)
3        do ψ ← ⌊d/(U − L)⌋
4           b ← O_D(ψ)
5           q ← (ψ + b) mod 2
6           k ← ⌊Lψ/d + 1/2⌋
7           B ← (k + 1/2)d/ψ
8           if (k mod 2 = q)
9              then U ← ⌊B⌋
10             else L ← ⌈B⌉
11 return L
```

Proof. (correctness of CCA1-ATTACK(d)) Initially, the algorithm sets $L = 0$ and $U = d-1$. Clearly w is contained in this interval. Now we select a ciphertext $\psi = \lfloor d/(U-L) \rfloor$, such that size of the interval $[\psi L, \psi U]$ is bounded by d.
For $k = \lfloor \psi L/d + 1/2 \rfloor$, we find that $-d/2 \leq \psi L - kd < d/2$. This implies $kd - d/2 \leq \psi L < kd + d/2$, and together with the fact that the size of the interval $[\psi L, \psi U]$ is bounded by d, we find that $d/2 + kd$ is the only value of the form $d/2 + id$, for $i \in \mathbb{Z}$, which is in the interval $[\psi L, \psi U]$. Define $B = (k + 1/2)d/\psi$, from which it then follows that this value is in the interval $[L, U]$.
We have that ψw either is in the interval $[\psi L, (k+1/2)d)$ or in the interval $[(k+1/2)d, \psi U]$. Say we have $-d/2 \leq \psi w - qd < d/2$, then in the former case we have $q = k$ and in the latter case $q = k + 1$. The oracle gives us $b = \psi w - qd \mod 2 = \psi + q \mod 2$. We find $q = b + \psi \mod 2$ and if $q = k \mod 2$, then we have that $\psi w \in [\psi L, (k + 1/2)d)$ and thus $w \in [L, \lfloor B \rfloor]$, otherwise $w \in [\lceil B \rceil, U]$.
This ensures that after every step we have $w \in [L, U]$. Since the position of B in the interval $[L, U]$ and the sub-interval which contains w can be considered to be randomly chosen, the interval is halved on average in every iteration. So after $O(\log d)$ oracle queries, w is found. □

Since the adversary can find the private key with a polynomial amount of oracle queries, it can clearly correctly decrypt the given challenge. This differs from the usual approach, where the adversary only tries to decrypt the challenge and nothing further. The consequences are similar, however.

6.3 CCA1-Secure Fully Homomorphic Encryption

It is shown in [LMSV10] that one can make the somewhat homomorphic part of the Smart-Vercauteren scheme CCA1 secure. In the current version of this paper (uploaded March 10, 2011), it is stated that this result remains unaffected if one extends the CCA1-secure scheme to a fully homomorphic scheme. This however is not true, as we will show here. Correspondence with Frederik Vercauteren, one of the authors, confirmed that this error

was spotted and that it will be removed.
Actually, it is impossible for a fully homomorphic scheme that is based on Gentry's construction to be CCA1-secure. This is due to the fact that the secret key is included in the public key in encrypted form. Clearly, if an adversary is given a public key and access to a decryption oracle for polynomially many queries before being given a challenge, then the adversary could simply query the encryption of the secret key.
The first thought to fix this would be to redefine CCA1-security in this case, and exclude the encryption of the secret key from the decryption power of the oracle. This would be similar to the definition of CCA2-security, where the adversary cannot query the given challenge to the oracle. But then, for the same reasons as that a fully homomorphic scheme cannot be CCA1-secure, the adversary can multiply by an encryption of 1 and query this. That this is possible cannot be prevented, so such a scheme cannot be CCA1-secure.

7 Conclusion

In this thesis, the current development in the area of fully homomorphic encryption is summarized. The main ideas behind the construction, namely bootstrapping and squashing are explained. Along with the construction, it is shown how this construction can be used to turn different kinds of somewhat homomorphic schemes into fully homomorphic schemes.
The scheme by Gentry and Halevi was used to make an implementation in a plug-in demonstrating homomorphic encryption for JCrypTool. This implementation is the first implementation which allows users to see that computations on ciphertexts are possible.
Finally, a lunchtime attack on the implemented scheme is described. This attack is also integrated in a level 2 challenge available on Mystery Twister C3 (see `http://www.mysterytwisterc3.org/`). It is explained that it is impossible for a fully homomorphic encryption scheme that follows the given construction to be resistant to such an attack.

Future work
There are several directions in which future research can head. Most importantly is to improve the speed of the operations in these schemes. In particular the recrypt operation. Also, it would be very interesting to be able to encrypt several bits simultaneously, instead of in parallel.
Though the plugin is fully functional, there are still points to be improved. The plugin could keep track of the amount of noise for each ciphertext, and only recrypt if necessary to compute the next step. In the current implementation, after each multiplication a recrypt operation is performed.

References

[BDPR98] Mihir Bellare, Anand Desai, David Pointcheval, and Phillip Rogaway. Relations among notions of security for public-key encryption schemes. In Hugo Krawczyk, editor, *CRYPTO*, volume 1462 of *Lecture Notes in Computer Science*, pages 26–45. Springer, 1998. http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.80.1577&rep=rep1&type=ps.

[CGS97] Ronald Cramer, Rosario Gennaro, and Berry Schoenmakers. A secure and optimally efficient multi-authority election scheme. In *EUROCRYPT '97, Lecture Notes in Computer Science*, volume 1233, pages 103–118, 1997.

[CKV10] Kai-Min Chung, Yael Kalai, and Salil P. Vadhan. Improved delegation of computation using fully homomorphic encryption. In Rabin [Rab10], pages 483–501. Cryptology ePrint Archive 2010/241, http://eprint.iacr.org/2010/241.

[Cry11] CrypTool. The website of the CrypTool project, July 2011. http://www.cryptool.org.

[DL07] Jintai Ding and Richard Lindner. Identifying ideal lattices. *Cryptology ePrint Archive*, 2007/322, 2007. http://eprint.iacr.org/2007/322.pdf.

[Gen09a] Craig Gentry. *A fully homomorphic encryption scheme*. PhD thesis, Stanford University, 2009. http://crypto.stanford.edu/craig/.

[Gen09b] Craig Gentry. Fully homomorphic encryption using ideal lattices. In Michael Mitzenmacher, editor, *STOC*, pages 169–178. ACM, 2009.

[GGP10] Rosario Gennaro, Craig Gentry, and Bryan Parno. Non-interactive verifiable computing: Outsourcing computation to untrusted workers. In Rabin [Rab10], pages 465–482. Cryptology ePrint Archive 2009/547, http://eprint.iacr.org/2009/547.pdf.

[GH11] Craig Gentry and Shai Halevi. Implementing gentry's fully-homomorphic encryption scheme. In Kenneth G. Paterson, editor, *EUROCRYPT*, volume 6632 of *Lecture Notes in Computer Science*, pages 129–148. Springer, 2011. https://researcher.ibm.com/researcher/view_project.php?id=1579.

[KR88] Richard M. Karp and Vijaya Ramachandran. A survey of parallel algorithms for shared-memory machines. Technical Report CSD-88-408, UC Berkely, 1988. http://www.cs.pitt.edu/~kirk/cs1510/ParallelAlgorithmsSurvey.pdf.

[LLL82] A. K. Lenstra, H. W. Lenstra, and L. Lovsz. Factoring polynomials with rational coefficients. *Mathematische Annalen*, 261:515–534, 1982. 10.1007/BF01457454.

[LMSV10] Jake Loftus, Alexander May, Nigel P. Smart, and Frederik Vercauteren. On CCA-secure fully homomorphic encryption. *To appear at SAC 2011, Cryptology ePrint Archive*, 2010/560, 2010. http://eprint.iacr.org/2010/560.

[Mar05] Alan Marcocitz. *Introduction to Logic Design*. McGraw-Hill, second edition edition, 2005.

[MR09] Daniele Micciancio and Oded Regev. Lattice-based cryptography. *Post Quantum Cryptography*, pages 147–191, Springer, February 2009. http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.142.4862&rep=rep1&type=pdf.

[NY90] Moni Naor and Moti Yung. Public-key cryptosystems provably secure against chosen ciphertext attacks. In *STOC*, pages 427–437. ACM, 1990. http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.26.5883&rep=rep&type=pdf.

[Pai99] Pascal Paillier. Public-key cryptosystems based on composite degree residuosity classes. In *EUROCRYPT '99, Lecture Notes in Computer Science*, volume 1592, pages 223–238, 1999.

[PKC91] PKCS 1: RSA encryption standard, 1991. http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.31.9612.

[Rab10] Tal Rabin, editor. *Advances in Cryptology - CRYPTO 2010, 30th Annual Cryptology Conference, Santa Barbara, CA, USA, August 15-19, 2010. Proceedings*, volume 6223 of *Lecture Notes in Computer Science*. Springer, 2010.

[RAD78] Ronald L. Rivest, Leonard M. Adleman, and Michael L. Dertouzos. On data banks and privacy homomorphisms. *Foundations of Secure Computation*, pages 169–180, 1978. http://people.csail.mit.edu/rivest/RivestAdlemanDertouzos-OnDataBanksAndPrivacyHomomorphisms.pdf.

[RSA78] Ronald L. Rivest, Adi Shamir, and Leonard M. Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Communications of the ACM*, 21:120–126, 1978. http://www.research.rutgers.edu/~pupala/cs352_docs/rsa.pdf.

[Sch11] Berry Schoenmakers. Cryptography 2 (2WC13) / cryptographic protocols (2WC10). Lecture Notes, 2011. http://www.win.tue.nl/~berry/2WC13/LectureNotes.pdf.

[SV10] Nigel P. Smart and Frederik Vercauteren. Fully homomorphic encryption with relatively small key and ciphertext sizes. In Phong Q. Nguyen and David Pointcheval, editors, *Public Key Cryptography*, volume 6056 of *Lecture Notes in Computer Science*, pages 420–443. Springer, 2010.

[SV11] Nigel P. Smart and Frederik Vercauteren. Fully homomorphic simd operations. 2011. http://eprint.iacr.org/2011/133.pdf.

[vDGHV10] Marten van Dijk, Craig Gentry, Shai Halevi, and Vinod Vaikuntanathan. Fully homomorphic encryption over the integers. In Henri Gilbert, editor, *EUROCRYPT*, volume 6110 of *Lecture Notes in Computer Science*, pages 24–43. Springer, 2010. Cryptology ePrint Archive 2009/616, http://eprint.iacr.org/2009/616.

[Yao82] Andrew Chi-Chih Yao. Protocols for secure computations (extended abstract). In *FOCS*, pages 160–164. IEEE, 1982. http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.103.7844&rep=rep1&type=pdf.