# Report Fully Homomorphic Encryption

Elena Fuentes Bongenaar

July 28, 2016

## 1 Introduction

Outsourcing computations can be interesting in many settings, ranging from a client that is not powerful enough to perform certain computations itself, to situations where different large datasets need to be combined in calculations but are stored at various locations. Along with outsourcing comes a privacy concern: should the input and/or output be revealed to the third party which will perform the computations? In case this is not desirable one would like to encrypt the data before sending it to this third party, under a key this party does not know. This is not a straight forward solution: usually computations on ciphertexts will yield nothing meaningful, but fortunately there is an exception: homomorphic encryption. This type of encryption will allow certain operations to be performed on ciphertexts and after decryption give the result as if the operations were performed on the plaintexts.

There are different flavors of this type of encryption; the scheme can support either multiplication or addition or both. Furthermore a distinction can be made between schemes that allow an arbitrary number of operations versus limited ones. The 'holy grail' among them is Fully Homomorphic Encryption (FHE), which allows an arbitrary number of additions and multiplications to be performed on the ciphertexts.

A year after the publication of RSA the notion of homomorphic encryption was introduced by Rivest, Adleman and Dertouzos [18], calling it 'privacy homomorphisms'. According to them one of the requirements for such a system is that encryption and decryption should be easy to compute. In order to make it cost-wise interesting to outsource computations this requirement must be met otherwise it would be just as efficient to do the computation yourself.

It has taken many years before the first FHE scheme was actually realized; this was done by Craig Gentry in 2009 [9] when writing his PhD thesis. Gentry's scheme used lattice-based cryptography and the security is partially based on the hardness of certain problems over ideal lattices. This breakthrough marked the beginning of much research on this topic and the proposal of multiple new schemes, many of which are based on the Learning With Errors (LWE) problem, which reduces to hard problems in lattices. One of those FHE schemes, BGV [4], has been implemented in an open source library: HElib [10].

In 2012 the notion of multi-key FHE was introduced, and realized, by [13]. This is a special type of FHE scheme that allows computations on ciphertexts that are encrypted under different keys. There are various scenarios imaginable in which this can be a powerful tool, for example to calculate health care statistics based on data from different hospitals or performing a test on distance between people without

disclosing exact locations or distance.

The goal of this report is to give an introduction to FHE, multi-key FHE and give a feeling of how some of the schemes work. The focus lies on understanding what techniques and tricks are used, rather than on performance or implementation. A high level overview of the first FHE scheme by Gentry is given, the scheme itself is not very efficient but is has been a breakthrough result and the structure has set the tone for many schemes that followed. Then the focus will be on some (R)LWE schemes which we will dive into further. Still there will be many details that need to be right in order for a scheme to work, which can be found in the original schemes. Finally the notion of multi-key FHE is discussed as well as the few multi-key FHE schemes that exist.

# 2 (Fully) Homomorphic Encryption

The term *homomorphic* encryption is inspired by ring homomorphisms. A map $f : P \to C$ is called a homomorphism if the following holds for all $a, a' \in P$:

1. $f(a \cdot_P a') = f(a) \cdot_C f(a')$
2. $f(a +_P a') = f(a) +_C f(a')$

Where $\cdot_P, +_P$ are operations in $P$ and $\cdot_C, +_C$ in $C$ [7]. This means it doesn't matter if you apply an operation on two elements first in $P$ and then go to $C$ through the homomorphism $f$, or first go to $C$ through the homomorphism with both elements and then perform the operation on the resulting elements in $C$. In the setting of encryption $P$ would be the space of plaintexts and $C$ of ciphertexts, while ideally the homomorphism is encryption. This would allow us to perform operations on the ciphertexts that will still decrypt correctly.

Even though the idea of a fully homomorphic scheme was already introduced in the seventies, the breakthrough came only came in the beginning of this century. Before that only partially homomorphic schemes were known, which allow either multiplication or addition on ciphertexts. An easy example is RSA: given ciphertexts $C_i (= m_i^e \mod n)$ encrypted under key $(n, e)$ we can multiply them, which gives the encryption of the multiplication of the plaintexts: $\prod_i C_i = \prod_i m_i^e \mod n = (\prod_i m_i)^e \mod n$.

A Fully Homomorphic Encryption (FHE) scheme **E** will consist of four algorithms; **E** = (**KeyGen**, **Enc**, **Dec**, **Eval**) [9], where **Eval** is the function that makes this encryption scheme special. We consider the public key setting, thus encryption is done with public key $p_k$ and decryption with secret key $s_k$. Also there will be a set of parameters specific to a scheme which will be implicit inputs to the algorithms. The input of **Eval** will consist at least of a function $f$ that has to be evaluated and a number of ciphertexts where $f$ should be applied to. To allow computations on the ciphertexts we require for every function $f$ and ciphertexts $c_1 = \mathbf{Enc}(m_1, p_k), \ldots, c_n = \mathbf{Enc}(m_n, p_k)$ that the following holds:

$$c_{new} = \mathbf{Eval}(f, c_1, \ldots, c_n) \Rightarrow \mathbf{Dec}(c_{new}, s_k) = f(m_1, \ldots, m_n)$$

Note that the plaintexts have to be encrypted with the same public key and the output of **Eval** is again encrypted under that same key. To make using such a scheme

interesting the complexity of **Enc** and **Dec** shouldn't depend on the functions that will be performed on the encrypted data. Ideally it would be possible to perform arbitrary computations, but in some situations this will not even be necessary. In this case Somewhat Homomorphic Encryption (SHE) will do: the homomorphic properties of the scheme work for a certain range of functions (with a certain complexity).

Next to somewhat and fully homomorphic encryption there is a notion of a leveled FHE scheme, in this case the depth of the circuits that can be evaluated influence how parameters should be set.

# 3 Preliminaries

A vector $(v_1, \ldots, v_n) \in \mathbb{Z}^n$ is represented as $\mathbf{v}$, with $v_i$ being the ith component.

**Arithmetic circuits** So far it has been discussed a function $f$ with ciphertexts as input can be performed by a homomorphic encryption scheme. A more precise way to define such a function can be given by arithmetic circuits [12]. A circuit consists of logical gates which are connected in a directed way. We can relate standard operations to these gates, i.e an AND gate does multiplication and a XOR gate addition. All computations we would like to do on data can be seen as evaluating polynomials and these can be built up by a combination of additions and multiplications. An arithmetic circuit can represent such a polynomial. The depth of the circuit is the longest path from input to output, but more important is the multiplicative depth: the maximum number of multiplication gates from input to output. This notion is used often, because multiplication tends to be the restricting factor in homomorphic schemes before addition. The size of the circuit is the amount of gates and the degree of the circuit is equal to the degree of the polynomial that is evaluated by the circuit.

**Analysis of complexity** To compare execution time of algorithms usually Big-O notation is used. By describing the behavior of the functions it gives a way to express the running time of an algorithm depending on the input given to it. A function $f$, informally, runs in $O(g)$ when it doesn't grow faster than $g$. This can be expressed as follows:

$$f \in O(g) \Leftrightarrow \exists c \in \mathbb{R}_{>0}, \exists n \in \mathbb{N} \text{ s.t } \forall n_i \geq n : 0 \leq f(n_i) \leq c \cdot g(n_i)$$

**Lattices** Gentry's first FHE scheme is based on (ideal) lattices, and as mentioned before, the (R)LWE problem reduces to hard problems on lattices. However, for the purpose of this report it is not needed to know many details about lattices because one can work on (R)LWE schemes without knowing lattices and Gentry's scheme will only be considered on a high level. Thus solely a definition of a lattice and hard problems concerning lattices will be given.

For a set of linearly independent vectors $V = \{\mathbf{v_1}, \ldots, \mathbf{v_n}\}$ in $\mathbb{R}^m$ a lattice in this space is formed by all the linear combinations of these vectors:
$\mathcal{L}(V) = \{\sum_{i=1}^{n} z_i \mathbf{v_i} : z_i \in \mathbb{Z}\}$ [15]. In this case $V$ is called the base of the lattice.

Some lattice problems that are assumed to be hard [15]:

- Shortest Vector Problem (SVP). Given lattice base $V$ find the shortest non-zero lattice vector $v$. Thus it should hold that $\forall w \in \mathcal{L}(V) : ||v|| \leq ||w||$.
- Approximate SVP (GapSVP$_\gamma$). Given lattice base $V$ give a $\gamma$-approximation of the shortest vector. Thus it should hold that $\forall w \in \mathcal{L}(V) : ||v|| \leq \gamma||w||$.

- Closest Vector Problem (CVP). Given lattice base $V$ and a vector $y \in \mathbb{R}^m$, find a vector $v \in \mathcal{L}(V)$ which is closest to $y$. Thus it should hold that $\forall w \in \mathcal{L}(V) : ||v - y|| \leq ||w - y||$.
- Approximate CVP ($\text{GapCVP}_\gamma$). Given lattice base $V$ and a vector $y \in \mathbb{R}^m$, find a vector $v \in \mathcal{L}(V)$ which is a $\gamma$-approximation of the closest vector to $y$. Thus it should hold that $\forall w \in \mathcal{L}(V) : ||v - y|| \leq \gamma||w - y||$.

# 4 The first FHE scheme

The first fully homomorphic scheme was proposed by Gentry and the structure of his solution has been used in various schemes later on as well. Here follows a very general overview.

First, Gentry constructed a SHE scheme using ideal lattices which can evaluate function of a certain (low) complexity. A ciphertext looks like $\mathbf{x} + \mathbf{e}$, where $\mathbf{x}$ is in the ideal lattice while we can see $\mathbf{e}$ as an error to encode this. The error should hide what lattice vector it concern exactly, but can't be too large otherwise decryption will not hold the correct message. It turns out to be possible to perform addition and multiplication on the ciphertexts and still be able to decrypt, however this comes with a cost: the error vector grows with every computation, especially in the case of multiplication. When the error becomes too big decryption will not be possible anymore and the data goes lost. This makes the scheme so far somewhat homomorphic.

If there would be a way to reduce the error in between the computations and thus allowing more complex functions to be applied, the scheme would become fully homomorphic. The best way to 'take away' the error is to decrypt the ciphertext since $\mathbf{e}$ has to be removed completely to reveal the original text. Gentry's ingenious insight was that this can be achieved by making use of the homomorphic properties of the scheme: perform the **Dec** function homomorphically! More precisely, the ciphertext is decrypted with the necessary secret key and once again encrypted under a new secret key; now the error is relatively small again because no computations have been performed on this "fresh" ciphertext. Of course this should all be done without the party performing the computation actually knowing the secret key and thus it is done homomorphically. This process of decryption and again encrypting a ciphertext homomorphically is usually called re-encryption or bootstrapping.

To see how re-encryption works exactly consider public keypairs $(pk_1, sk_1)$, $(pk_2, sk_2)$ and let $\{m\}_{pk_1} = \mathbf{Enc}(m, pk_1)$. We know that for function $f$, **Eval** will perform $f$ on the ciphertexts (encrypted with the same key) that are given as input. Now take $f = \mathbf{Dec}$ and ciphertext inputs $\{sk_1\}_{pk_2} = \mathbf{Enc}(sk_1, pk_2)$ and $\{\{m\}_{pk_1}\}_{pk_2} = \mathbf{Enc}(\{m\}_{pk_1}, pk_2)$. Then we have:

$$\begin{aligned}\mathbf{Eval}(\mathbf{Dec}, \{\{m\}_{pk_1}\}_{pk_2}, \{sk_1\}_{pk_2}) &= \mathbf{Enc}(\mathbf{Dec}(\{m\}_{pk_1}, sk_1), pk_2) \\ &= \mathbf{Enc}(m, pk_2) \\ &= \{m\}_{pk_2}\end{aligned}$$

A scheme can only perform **Dec** homomorphically if the complexity of this function is lower than what the scheme can handle. If this is the case, we call such

a scheme *bootstrappable*. Note that this process of bootstrapping requires an extra assumption of circular security, because we encrypt the new secret key with the encryption scheme itself.

In the case of Gentry's scheme, decryption was a too complex function to evaluate and thus the scheme was not bootstrappable. To achieve this, another important technique was introduced: squashing the decryption circuit. The computationally expensive decryption algorithm is split into an intensive preprocessing phase, ideally performed by the cloud, and a lighter final decryption phase. This idea of splitting the computation comes from server-aided cryptography, where a computationally weak device needs to outsource some part of the computation to a more powerful server. For the second part the secret key is needed, while the first phase will only use certain 'hints' on the secret key, since the party performing this doesn't have any access to secret keys. More specifically a set of elements $S$ is added to the public key, for which it holds that there is a sparse subset $S'$ such that adding the elements in $S'$ gives the secret key. Then the new secret key is the indication of which elements of the larger set are contained in $S'$. The high level idea is that the cloud computes the decryption of a ciphertext taking each element of $S$ as the secret key, resulting in a sequence of 'decryptions'. Then the pieces corresponding with decryptions based on elements in $S'$ have to be put together correctly, resulting in the actual decryption. This makes it possible to bring down the complexity of the decryption circuit and make the scheme bootstrappable. The security of this adjustment is based on the hardness of the sparse subset problem, which means retrieving the secret key from these 'hints' is infeasible.

As mentioned before, the structure of Gentry's solution has been very important and re-used many times in follow-up schemes: first make a SHE scheme, make it bootstrappable by squashing the decryption circuit, and then use bootstrapping to make it a FHE scheme.

# 5 Learning with errors (LWE)

Many recent proposed schemes base their security on the hardness of the learning with errors problem introduced by Regev [17]. A basic definition is given here, for details we refer to Regev's paper. Given an integer $q$, a (Gaussian) distribution $\chi$ on $\{-\lceil \frac{q-1}{2} \rfloor, \ldots, \lceil \frac{q-1}{2} \rfloor\}$, and dimension $n$, the decision LWE problem is to distinguish between the following distributions, for a fixed $\mathbf{s}$ uniformly sampled from $\mathbb{Z}_q^n$:

(1) $(\mathbf{a_i}, b_i)$ sampled uniformly from $\mathbb{Z}_q^{n+1}$

(2) $(\mathbf{a_i}, b_i)$, where $\mathbf{a_i}$ is sampled uniformly from $\mathbb{Z}_q^n$, $e_i$ is sampled from $\chi$ and we set $b_i = \langle \mathbf{a_i}, \mathbf{s} \rangle + e_i$

The search variant of LWE would be to find $\mathbf{s}$ from arbitrarily many pairs $(\mathbf{a_i}, b_i = \langle \mathbf{a_i}, \mathbf{s} \rangle + e_i)$. We can see this as solving a "noisy" linear system of equations $\langle \mathbf{a_i}, \mathbf{s} \rangle \approx b_i$. The error $e_i$ should be small in order to solve this correctly, that is why it is sampled from a Gaussian distribution centered around 0.

For different parameters there exists a quantum and a classical reduction to $\text{GapSVP}_\gamma$ [6, 17].

## 5.1 Regev's standard cryptosystem

In the paper where the LWE problem was presented by Regev [17], he also gave a public key cryptosystem based on the hardness of this problem, which has formed the base of later proposed FHE schemes. With the discussed parameters, $n$ is the security parameter of the cryptosystem. Additionally let integer $m = (1+\epsilon)(n+1)\log p$ for an arbitrary positive constant $\epsilon$, then key generation, encryption and decryption go as follows:

- **KeyGen**: sample private key $\mathbf{s}$ uniformly from $\mathbb{Z}_q^n$. For $i \in \{1, \ldots, m\}$ choose $\mathbf{a_i} \in \mathbb{Z}_q^n$ independently from the uniform distribution. Sample $e_1, \ldots, e_m$ from $\chi$. Now the public key is a vector of entries as described in (2).
  *Output:* $(sk, pk) = (\mathbf{s}, (\mathbf{a_i}, b_i)_{i=1}^m)$
- **Enc**(bit, $pk$): choose a random set $S$ of all subsets of $\{0, \ldots, m\}^m$.
  *Output:* $(\sum_{i \in S} \mathbf{a_i}, \text{bit} \cdot \lfloor \frac{p}{2} \rfloor + \sum_{i \in S} b_i)$
- **Dec**($(\mathbf{x}, y)$, $sk$): calculate $z = y - \langle \mathbf{x}, sk \rangle = y - \langle \mathbf{x}, \mathbf{s} \rangle$.
  *Output:* If $z$ is closer to 0 than to $\lfloor \frac{p}{2} \rfloor$ output 0, otherwise 1.

In de decryption procedure we evaluate the following:

$$\begin{aligned} y - \langle \mathbf{x}, \mathbf{s} \rangle &= y - \sum_{j=1}^{m} x_j \cdot s_j \\ &= y - \sum_{j=1}^{m} (\sum_{i \in S} \mathbf{a}_i)_j \cdot s_j \\ &= y - \sum_{i \in S} \langle \mathbf{a_i}, \mathbf{s} \rangle \\ &= \text{bit} \cdot \lfloor \frac{p}{2} \rfloor + \sum_{i \in S} (\langle \mathbf{a_i}, \mathbf{s} \rangle + e_i) - \sum_{i \in S} \langle \mathbf{a_i}, \mathbf{s} \rangle \\ &= \text{bit} \cdot \lfloor \frac{p}{2} \rfloor + \sum_{i \in S} e_i \end{aligned}$$

This can be decrypted correctly as long as the error $\sum_{i \in S} e_i$ is small enough. Regev proved that his choice of parameters would assure correct decryption with a probability of $1 - \delta(n)$ for some negligible function $\delta(n)$ [17].

## 5.2 A Regev-like FHE scheme

Brakerski and Vaikuntanathan proposed a FHE scheme based on LWE in [3]. Through a technique called relinearization they were able to base the security of the scheme on LWE, instead of on ideal lattices which was usual until then. Furthermore a second technique, dimension-modulus reduction, was introduced, making the scheme bootstrappable without needing to squash the decryption circuit. Using the symmetric key variant of the scheme both new techniques will be explained. The scheme can be turned into a public key scheme easily.
The scheme used is similar to Regev's scheme; a bit $m \in \{0, 1\}$ is encrypted with a secret key $\mathbf{s} \in \mathbb{Z}_q^n$ by choosing $\mathbf{a} \in \mathbb{Z}_q^n$ and error $e \in \chi$ randomly and set the ciphertext to be:

$$(\mathbf{a}, b = \langle \mathbf{a}, \mathbf{s} \rangle + 2e + m)$$

Decryption is done by computing $b - \langle \mathbf{a}, \mathbf{s} \rangle \mod 2$.

Relinearization reduces the size of a ciphertext, for which a price must be paid: the publication of a sequence of so-called evaluation keys at the moment of key generation. We see how this technique works and why it is needed by looking at the homomorphic operations of the scheme:

- **Eval:add**, add entry-wise, i.e addition of $(\mathbf{a}, b)$ and $(\mathbf{a'}, b')$ gives $(\mathbf{a}+\mathbf{a'}, b+b')$.
- **Eval:mult**, like we did in the previous scheme consider the input vectors to be polynomials in some variable $z$, then the encryption of the multiplication of the ciphertexts is equal to the coefficients in the resulting multiplied polynomial.

To see that addition works consider the decryption of 2 added ciphertexts:

$$\begin{aligned} b + b' - \langle \mathbf{a}+\mathbf{a'}, \mathbf{s} \rangle \mod 2 &= b + b' - \sum (\mathbf{a} + \mathbf{a'})[i]\mathbf{s}[i] \mod 2 \\ &= b + b' - \sum \mathbf{a}[i]\mathbf{s}[i] - \sum \mathbf{a'}[i]\mathbf{s}[i] \mod 2 \\ &= (b - \sum \mathbf{a}[i]\mathbf{s}[i]) + (b' - \sum \mathbf{a'}[i]\mathbf{s}[i]) \mod 2 \\ &= m + m' \end{aligned}$$

For two multiplied ciphertexts we need to get the following:

$$\begin{aligned} (b - \sum_i \mathbf{a}[i]\mathbf{s}[i])(b' - \sum_j \mathbf{a'}[j]\mathbf{s}[j]) &= bb' - b\sum_j \mathbf{a'}[j]\mathbf{s}[j] - b'\sum_i \mathbf{a}[i]\mathbf{s}[i] + (\sum_i \mathbf{a}[i]\mathbf{s}[i])(\sum_j \mathbf{a'}[j]\mathbf{s}[j]) \\ &= c_0 + \sum_i c_i \mathbf{s}[i] + \sum_{i,j} c_{ij}\mathbf{s}[i]\mathbf{s}[j] \end{aligned}$$

For certain coefficients $c_0, c_i, c_{ij}$. Thus we see this expression contains a linear and a quadratic part in entries of $\mathbf{s}$ and for each of the terms we need to know the coefficient. These coefficients must be stored in the ciphertext resulting from **Eval:mult**. This means that the ciphertext would grow, which is ideally avoided. But what if we could rewrite this expressions in a linear way, thus requiring us to know only all coefficients for the linear part (the $c_i$'s)? In this way the length of the ciphertext would be reduced. It turns out this can be done with the use of evaluation keys that are encryptions of the secret key $\mathbf{s}$ under a new secret key $\mathbf{t}$. More specifically all linear and quadratic terms $\mathbf{s}[i]$ and $\mathbf{s}[i]\mathbf{s}[j]$ are being published encrypted under the new key. This means for certain $\mathbf{a}^* \in \mathbb{Z}_q^n$, $e^* \in \chi$ we get ciphertexts of the form:

- $(\mathbf{a}^*, b^* = \langle \mathbf{a}^*, \mathbf{t} \rangle + 2e^* + \mathbf{s}[i]\mathbf{s}[j])$ or;
- $(\mathbf{a}^*, b^* = \langle \mathbf{a}^*, \mathbf{t} \rangle + 2e^* + \mathbf{s}[i])$

In the first case the following holds: $\mathbf{s}[\mathrm{i}]\mathbf{s}[\mathrm{j}] \approx b^* - \langle \mathbf{a}^*, \mathbf{t} \rangle$ and the same can be done for the second case, the encryptions of $\mathbf{s}[i]$. The trick is to replace all $\mathbf{s}[i]$ and $\mathbf{s}[i]\mathbf{s}[j]$ in the long ciphertext with the corresponding $b^* - \langle \mathbf{a}^*, \mathbf{t} \rangle$, making it a *linear expression* in $\mathbf{t}$. With this technique a shorter ciphertext is obtained, which is encrypted under the new secret key $\mathbf{t}$, allowing more levels of multiplications to be performed.

To make the SHE scheme bootstrappable it must be able to evaluate its own decryption algorithm. This is made possible with the second new technique is introduced: dimension-modulus reduction.

To reduce parameter $n$ and $\log q$ the change in ciphertext during relinearization is used. First of all the key under which the new ciphertext is encrypted will have a lower dimension $k < n$; thus the ciphertext will have a lower dimension as well. Additionally the ciphertext is scaled down to modulus $p$ instead of modulus $q$, where $p < q$. This is done by scaling the entries of the evaluation key by $\frac{p}{q}$ which will result in a ciphertext that is also scaled down by the same fraction. After these changes the result is a ciphertext in $\mathbb{Z}_p^k \times \mathbb{Z}_p$ instead of $\mathbb{Z}_q^n \times \mathbb{Z}_q$, bringing complexity of decryption down. When the parameters are set correctly, the ciphertext will still decrypt in the right way inspite of the change in dimension and modulus. Also in the paper it was shown also for the new parameters the LWE problem is still hard, which makes it the sole assumption where security is based on. After applying dimension-modulus reduction the complexity of decryption is low enough, making the SHE bootstrappable and thus able to be turned into a FHE scheme.

## 5.3 Approximate eigenvector scheme (GSW)

In the GSW scheme, named after the authors Gentry, Sahai and Waters, the ciphertexts are matrices and the homormorphic operations are simply matrix addition and multiplication. The idea comes from nice properties of eigenvectors for matrices: if $\mathbf{v}$ is an eigenvector of matrix $A$, that means there is a scalar $\lambda$ such that $A\mathbf{v}=\lambda\mathbf{v}$ and in this case $\lambda$ is the eigenvalue. If we have matrices $C_1, C_2$ with the same eigenvector $\mathbf{v}$ for certain eigenvalues respectively $m_1, m_2$ then the following holds:

- $\mathbf{v}$ is an eigenvector for $C_1 + C_2$ for eigenvalue $m_1 + m_2$
- $\mathbf{v}$ is an eigenvector for $C_1 \cdot C_2$ for eigenvalue $m_1 \cdot m_2$

This looks like the homomorphic properties we would want for ciphertexts $C_1, C_2$ and messages $m_1, m_2$ encrypted under $\mathbf{v}$. Unfortunately we can not use exactly this setting because for a given matrix it is easy to find eigenvectors and eigenvalues. For this scheme instead of the secret key being an actual eigenvector, an *approximate eigenvector* is considered, for example $C_1\mathbf{v}=m_1\mathbf{v}+\mathbf{e}$ for a certain small error $\mathbf{e}$ and thus $C_1\mathbf{v}\approx m_1\mathbf{v}$. If ciphertexts $C_1, C_2$ are added we get: $(C_1 + C_2)\mathbf{v} = C_1\mathbf{v}+C_2\mathbf{v}= (m_1 + m_2)\mathbf{v} + \mathbf{e_1}+ \mathbf{e_2}$, which will decrypt correctly if the original errors $\mathbf{e_1}$ and $\mathbf{e_2}$ are small enough. Multiplication of the same ciphertexts gives:

$$(C_1 \cdot C_2)\mathbf{v} = (m_1 \cdot m_2)\mathbf{v} + m_2 \cdot \mathbf{e_1} + C_1 \cdot \mathbf{e_2}$$

This situation is different: the new error also depends on the message and the ciphertexts. The message will be 0 or 1 and thus the part $m_2 \cdot \mathbf{e_1}$ will not be large. This does not necessarily hold for the part of the error that depends on the first ciphertext; the ciphertext might have large entries. To ensure the entries will be small and thus the error doesn't grow too much a gadget matrix is used.

A gadget matrix $G$ (i.e $\in \mathbb{Z}_q^{m\times n}$) and corresponding inverse transformation $G^{-1}$ (i.e $\mathbb{Z}_q^{m\times n} \to \mathbb{Z}_k^{m\times m}$, $n < m$ and $k < q$) have the following properties for a matrix $A \in \mathbb{Z}_q^{m\times n}$:

- $G^{-1}(A)$ has small entries
- $G^{-1}(A) \times G = A$

By applying the transformation $G^{-1}$ the entries are guaranteed to be small, at the cost of a larger dimension for the matrix. The way this is used in GSW is by letting $G$ be a matrix with every column containing the powers of 2, and $G^{-1}$

the transformation that turns a matrix into a larger one with every element in its binary representation. Then if you multiply this larger matrix with $G$, the original matrix is obtained. More formally: let $\mathbf{a}$ be a vector in $\mathbb{Z}_q^m$ for integers $q, m$. Let $l = \lfloor \log_2 q \rfloor + 1$ and $N = m \cdot l$, the number of bits needed to represent an element in $\mathbb{Z}_q$ and a complete vector in $\mathbb{Z}_q^m$ respectively. Let $\mathbf{b} \in \mathbb{Z}_q^N$. Define:

- $\mathbf{BitDecomp}(\mathbf{a}) = (a_{1,0}, \ldots, a_{1,l-1}, a_{2,0}, \ldots, a_{m,l-1})$, the binary representation of vector $\mathbf{a}$ with LSB first. Input has dimension $m$, output $N$.
- $\mathbf{BitDecomp}^{-1}(\mathbf{b}) = (\sum_{j=0}^{l-1} b_{1,j}, \ldots, \sum_{j=0}^{l-1} b_{m,j})$, which is also well defined if input isn't a binary vector. Input has dimension $N$, output $m$.

Thus $G^{-1}$ is **BitDecomp** and to apply $\mathbf{BitDecomp}^{-1}$, one can multiply by gadget matrix $G$ that contains the powers of 2.

Now we have the tools to define the scheme. Let $\chi$ be a $B$-bounded distribution. We consider a message $m \in \{0, 1\}$.

- **KeyGen**: choose $\mathbf{t} \in \mathbb{Z}_q^{n-1}$ and set $\mathbf{s}$=(**-t**,1) $\in \mathbb{Z}_q^n$. Sample uniformly matrix $B$ from $\mathbb{Z}_q^{m \times n-1}$ and $\mathbf{e}$ from $\chi^m$.
  Set $\mathbf{b} = B\mathbf{t}+\mathbf{e}$ and $A = \begin{bmatrix} B \\ \mathbf{b} \end{bmatrix}$. Note that $A\mathbf{s}$=$\mathbf{e}$
  *Output:* $(sk, pk)$ =*(**s**, A)*
- **Enc**$(m, pk)$: sample uniformly $R \in \{0, 1\}^{m \times m}$. Set $C = RA + mG$.
  *Output:* $C$
- **Dec**$(C,\ sk)$: define $\mathbf{w}$=$[0, \ldots, 0, \lceil \frac{q}{2} \rceil]^T \in \mathbb{Z}_q$ and compute $\mu = \langle C\mathbf{s}, G^{-1}(\mathbf{w}) \rangle$.
  *Output:* $\lfloor \frac{\mu}{q/2} \rceil$
- **Eval:add**$(C_1, C_2)$: *Output: $(C_1 + C_2)$*
- **Eval:mult**$(C_1, C_2)$: *Output: $(C_1 G^{-1}(C_2))$*

Because no evaluation key is used in this scheme when applying multiple homomorphic operations (which is necessary for relinearization), GSW gives rise to identity-based FHE (IBFHE) schemes. Only the ID would be needed to perform evaluation on ciphertexts instead of an encryption of the secret key which is exactly what we want in identity-based encryption (IBE). They described a compiler that can turn known LWE-based IBE schemes that have certain properties into IBFHE schemes.

# 6 Ring Learning with Errors (RLWE)

Hardness of the LWE problem made it very interesting for cryptographic applications but efficiency turned out to be a problem. An algebraic variant of this problem also gives hardness guarantees and is a more practical base for such systems [14]. This ring-based variant was proposed by Lyubashevsky, Peikert and Regev. Now we consider the ring $R = \mathbb{Z}_q[x]/\langle x^n + 1 \rangle$ with $n$ a power of 2 and an error distribution $\chi$ over $R$ that is concentrated on 'small' elements. Note that the elements in $R$ are polynomials of degree $(n-1)$ or lower. For $s$ sampled uniformly from $R$ the (decision) RLWE problem is to distinguish between:

- $(a, b)$, with $a, b$ sampled uniformly from $R$
- $(a, b)$, with $a$ again sampled in the same way and $e$ from $\chi$. Set $b = a \cdot s + e$.

The RLWE assumption is that this is a hard problem; in [14] a quantum reduction of this problem to SVP in the worst case on ideal lattices is given. Compared to LWE in most cases $n$ noisy LWE equations can be replaced by 1 noisy RLWE equation which obviously improves in terms of efficiency.

## 6.1 BV: A cryptoscheme based on RLWE

Brakerski and Vaikuntanathan present a SHE cryptoscheme (BV) based on RLWE [5], that could be turned into a FHE scheme using the standard techniques bootstrapping and squashing. For all schemes so far an extra assumption was required to include squashing in the scheme, namely circular security. In this case they proved the scheme is circular secure w.r.t linear functions of the secret key. Furthermore, not just one bit gets encrypted but a polynomial with binary coefficients $m \in \mathbb{Z}_2[x]/\langle x^n + 1\rangle$. As we will see, performing multiplication on ciphertexts outputs a longer ciphertext. There will be a bound $B \in \mathbb{N}$ to how many times this can be done and $B + 1$ will thus be the maximal length of a ciphertext. When multiple ciphertexts of differents lengths serve as input for a function we can pad with extra zeroes, except when we perform multiplication: in this case the difference in length is no problem. Also the secret key is extended to have length $B+1$ and for decryption we can assume without loss of generality that ciphertext length is $B + 1$. Now follows the proposed scheme:

- **KeyGen**: sample $s$ uniformly from $\chi$. To extend the length of the secret key we set $\mathbf{s} = (1, s, s^2, \ldots, s^B)$. A public key is formed by sampling $a$ from $R$, $e$ from $\chi$ set $b = as + 2e$ and output: $(a, b)$. We can not reuse this public key, but it's easy to generate new pairs given the first one. Sample $v, e'$ from $\chi$ and $e''$ from $\chi'$, which is distribution like $\chi$ but with a larger standard deviation. Now set $a' = av + 2e'$, $b' = bv + 2e''$. Then we have a new pair $(a', b')$ of the same form as the original pair:

$$\begin{aligned} b' = bv + 2e'' &= (as + 2e)v + 2e'' \\ &= avs + 2ev + 2e'' \\ &= avs + 2e's + 2ev + 2e'' - 2e's \\ &= (av + 2e')s + 2ev + 2e'' - 2e's \\ &= a's + 2(ev + e'' - e's) \end{aligned}$$

  In a lemma the authors show that these newly generated pairs are computationally indistinguishable from the desired distribution.
  *Output:* $(sk, pk) = (\boldsymbol{s}, (a, b))$

- **Enc**$(m, pk)$: generate a pair $pk = (a', b')$ as explained above and set $x = b' + m$ and $y = -a'$.
  *Output:* $(x, y)$

- **Dec**$(\mathbf{x}, \mathbf{s})$: we assume length $\mathbf{x}$ is $B + 1$, because we can pad with zeroes. Compute $z = \langle \mathbf{x}, \mathbf{s} \rangle \mod 2$.
  *Output:* $z$

- **Eval:add**$(\mathbf{x}, \mathbf{y})$: add every coordinate separately. If the input vectors don't have the same length we pad with zeroes. Assume $\mathbf{x}, \mathbf{y}$ have length $b \leq B + 1$

after padding.
*Output:* $(x_1 + y_1, \ldots, x_b + y_b)$

- **Eval:mult**(**x**, **y**): the output will be a longer tuple where every entry is one of the coefficients if we would see this as polynomial multiplication. For example, if input is $(x_1, y_1), (x_2, y_2)$, output is $(x_1x_2, x_1y_2 + x_2y_1, y_1y_2)$. In general if we see $\mathbf{x} = (x_1, \ldots, x_{b_1})$, $\mathbf{y} = (y_1, \ldots, y_{b_2})$ and we see them as polynomials in $a$ that we want to multiply; $(\sum_{i=1}^{b_1} x_i a^i)(\sum_{i=1}^{b_2} y_i a^i) = \sum_{i=1}^{b_1+b_2} c_i a^i$ then output is $\mathbf{c} = (c_1, \ldots, c_{b_1+b_2})$. Thus length of output is $b_1 + b_2 - 1$ for inputs of length $b_1$ and $b_2$.
*Output:* $\boldsymbol{c}$

When the correct parameters are chosen the LWE assumption holds, thus the secret key **s**, and specifically $s$, will be difficult to find when given public key $(a, b)$.

Decryption for a fresh ciphertext $(x, y)$ works as follows: $z = x \cdot 1 + y \cdot s \mod 2 \equiv (b' + m) - a' \cdot s \equiv (a' \cdot s + 2e + m) - a' \cdot s \equiv m$ if error $2e$ is small enough. To see that **Eval:add** works correctly we look at the easiest situation after adding 2 fresh ciphertexts $(x_1, y_1)$ and $(x_2, y_2)$ and try to decrypt:

$$\begin{aligned} x_1 + x_2 + (y_1 + y_2) \cdot s &= (a_1 + a_2) \cdot s + 2(e_1 + e_2) + (m_1 + m_2) + (-a_1 - a_2) \cdot s \\ &= 2(e_1 + e_2) + (m_1 + m_2) \\ &\equiv m_1 + m_2 \mod 2 \end{aligned}$$

Which shows this homomorphic operation indeed works if the new error $e_1 + e_2$ is small enough. This can easily be extended to larger ciphertexts, in which case the error will also grow faster. Also for **Eval:mult** we consider an easy example that can be extended to longer ciphertexts. Again we have the two ciphertexts $(x_1, y_1)$ and $(x_2, y_2)$ and if they get multiplied the outcome is $(x_1x_2, x_1y_2 + x_2y_1, y_1y_2)$. Decryption is given by $\langle (x_1x_2, x_1y_2 + x_2y_1, y_1y_2), \mathbf{s} \rangle \mod 2$ which can be split up in the following three terms:

1. $x_1x_2 = (a_1s + 2e_1 + m_1)(a_2s + 2e_2 + m_2)$
   $= a_1a_2s^2 + 2(a_1e_2 + a_2e_1)s + (a_1m_2 + a_2m_2)s + 2(e_1m_2 + e_2m_1) + 2(2e_1e_2) + m_1m_2$
2. $(x_1y_2 + x_2y_1)s = (-(a_1s + 2e_1 + m_1)a_2 - (a_2s + 2e_2 + m_2)a_1)s$
   $= -2a_1a_2s^2 - 2(e_1a_2 + e_2a_1)s - (a_1m_1 + a_2m_2)s$
3. $y_1y_2s^2 = a_1a_2s^2$

Adding these together most of the terms cancel out:

$$\begin{aligned} x_1x_2 + (x_1y_2 + x_2y_1)s + y_1y_2s^2 &= 2(e_1m_2 + e_2m_1) + 2(2e_1e_2) + m_1m_2 \\ &= 2(e_1m_2 + e_2m_1 + 2e_1e_2) + m_1m_2 \\ &\equiv m_1m_1 \mod 2 \end{aligned}$$

This will decrypt correctly as long as the new error term $2(e_1m_2 + e_2m_1 + 2e_1e_2)$ is not too big. For longer ciphertexts decryption goes in a similar way. Again we see that the growing error term when performing **Eval** is the reason why we cannot execute arbitrary functions and only have a somewhat homomorphic scheme so far. As mentioned before, by using the techniques squashing and bootstrapping also this scheme can be turned into a FHE scheme.

## 6.2 FHE without bootstrapping based on LWE or RLWE

Brakerski, Gentry and Vaikuntanathan constructed the first FHE scheme that doesn't need bootstrapping [4]. This is achieved by relying heavily on the modulus-switching technique from the previous discussed scheme based on LWE by Brakerski and Vaikuntanathan, as well as their key switching technique. Ironically it is possible to use bootstrapping as an optimization. The scheme provides the option to base security on LWE or RLWE, the latter providing better performance. This scheme has been implemented in the open source library HElib [10] along with many proposed optimizations.

**Basic scheme** First the basic encryption scheme BGV is based on is given, which is extended to a FHE scheme. Choose integers $q$ and $d$ and set $R = \mathbb{Z}[x]/\langle x^d + 1\rangle$, $R_q = R/qR$. Furthermore $\chi$ is a distribution on $R_q$ and $N = \lceil (2n+1)\log q \rfloor$. For now we will focus on the RLWE setting and thus $n = 1$.

- **KeyGen:** sample $s' \in \chi$ and set $sk = \mathbf{s} = (1, s')$. Generate uniformly at random $B \in R_q^N$ and error vector $e \in \chi^N$. Set $b = Bs' + 2e$ and $A = [b, -B]$.
  *Output:*$(sk, pk) = (\mathbf{s}, A)$
- **Enc**$(m, pk)$: for message $m \in R_2$, set $\mathbf{m}$=$(m, 0)$. Sample $r \in R_2^N$ and set $c = \mathbf{m} + A^T r = (c_0, c_1)$.
  *Output:* $(c_0, c_1)$
- **Dec**$(c,\ sk)$: compute $z = [[\langle (c_0, c_1), (1, s')\rangle]_q]_2$.
  *Output:* $z$

**Turning it into a FHE scheme** Addition of ciphertext can simply be defined by adding the different entries; if we add ciphertexts $(c_0, c_1), (d_0, d_1)$ in this way and decrypt, we get the following:

$$\langle (c_0{+}d_0, c_1{+}d_1), (1, s')\rangle = c_0{+}c_1s'{+}d_0{+}d_1s' = \langle (c_0, c_1), (1, s')\rangle{+}\langle (d_0, d_1), (1, s')\rangle \approx m_1{+}m_2$$

Multiplication is slightly trickier, and will show the need for dimension reduction. Denote the inner product calculated during decryption as a linear equation based on the ciphertext $c$: $L_c(x) = c_0 + c_1 x$. Then homomorphic multiplication of ciphertexts $c_1$ and $c_2$ is defined as $L_{c_1}(x)L_{c_2}(x)$, for ciphertexts encrypted under the same key, which gives a quadratic equation in $x$, or a linear equation in $x \otimes x = (1, x, x^2)$:

$$L_{c_1}(x)L_{c_2}(x) = (c_0{+}c_1x)(d_0{+}d_1x) = c_0d_0{+}(c_0d_1{+}c_1d_0)x{+}c_1d_1x^2 = Q_{c_1,c_2}(x) = L_{c_1,c_2}(x{\otimes}x)$$

If we consider the last representation of the homomorphic product, it becomes clear that this growth in dimension for the ciphertext and secret key has to be managed. The solution here is a key-switching procedure which allows to change a ciphertext $c$ encrypted under key $s_1$ to a ciphertext $c'$, encrypted under a different and possibly shorter key $s_2$. For the concrete implementation we need the **BitDecomp** as defined for the GSW scheme and additionally the following function:

**PowersOf2**$(x \in R_q, q)$: output $(x, 2x, 2^2x, \ldots, 2^{\lfloor \log q \rfloor}x) \in R_q^{\lfloor \log q \rfloor}$.

To allow key-switching from $s_1$ to $s_2$, additional information is added to the public key of $s_1$, which is produced in the routine **SwitchKeyGen**. In a later stadium

this additional information is used in **SwitchKey** to do the actual switching.

**SwitchKeyGen**$(s_1, (s_2, A_2))$: add **PowersOf2**$(s_1)$ to the first column of $A_2$, call this matrix $\tau_{s_1 \to s_2}$ and output it.

**SwitchKey**$(\tau_{s_1 \to s_2}, c)$: output $\tau_{s_1 \to s_2}$**BitDecomp**$(c)^T$.

For proof of correctness see [4]. On the cost of a slightly larger error the key can be switched to a shorter one and thus also the ciphertext is shortened. During the key generation, instead of a single secret key, a sequence of public/private keypair is generated, along with the additional information $\tau$ that allows to switch from key $s_j \otimes s_j$ to the next key $s_{j+1}$.

The other main technique used in the BGV scheme is modulus-switching, in order to reduce the noise. If we have a ciphertext $c \mod q$ and want to transform this into a new ciphertext $c' \mod p$ for modulus $p < q$, we simple scale down: $\frac{p}{q} \cdot c$ and round to the closest integer which we call $c'$. Then with the following lemma we see that for a small key $s$ the noise also scales down.

**Lemma modulus switching [4]**. For $p, q$ odd moduli and $c$ an integer vector, define $c'$ as above, such that $c = c' \mod 2$. Then for any $s$ with $|[\langle c, s\rangle]_q| < \frac{q}{2} - \frac{q}{p} l_1(s)$ we have the following:

$$[\langle c, s\rangle]_q = [\langle c', s\rangle]_p \mod 2 \text{ and } |[\langle c', s\rangle]_p| < \frac{p}{q}|[\langle c, s\rangle]_q| + l_1(s)$$

A proof is given in [4]. If we assume $l_1(s)$ is small enough and $p$ and $q$ are chosen such that $p$ is sufficiently smaller than $q$ we see that the noise scales down by approximately a factor of $\frac{p}{q}$. The ratio noise/modulus doesn't change but the following example will show how we can use this switching to our advantage, because of the decreasing noise magnitude.
Consider a sequence of moduli $(Q_d, Q_{d-1}, \ldots, Q_0)$ defined as $Q_d = x^{d+1}$, $Q_{j-1} = Q_j/x$ for integers $x$ and $d$. Assume we have 2 ciphertexts $c_1, c_2$ both with noise of magnitude $x$ and will compare the following 2 situations:

(a) Calculate $c_3 = c_1 c_2$, $c_4 = c_3 c_3$ and $c_5 = c_4 c_4$

(b) Do the same calculation but after every operation switch from modulus $Q_j$ to $Q_{j-1}$

For procedure (a) we get noise level $x^2$ for $c_3 \mod Q_d$, $x^4$ for $c_4 \mod Q_d$ and $x^8$ for $c_5 \mod Q_d$.
With (b) we start with noise level $x^2$ for $c_3 \mod Q_d$ and then scale down to noise level $x$ for $c_3' \mod (Q_d/x)$. We continue with squaring $c_3'$ and get ciphertext $c_4 \mod (Q_d/x)$ with noise level $x^2$. Scale this down to noise level $x$ for $c_4' \mod (Q_d/x^2)$. Finally the noise level for $c_5 = c_4' c_4' \mod (Q_d/x^2)$ is $x^2$ and after scaling down it becomes $x$ for $c_5' \mod (Q_d/x^3)$.
Now compare the noise/modulus ratio of the final outcome of (a): $x^8/Q_d$ and (b): $x/(Q_d/x^3) = x^4/Q_d$. This shows that by decreasing the magnitude of the noise, even though the ratio after switching stays the same, we can reduce the pace in which the noise ceiling is achieved and thus perform more multiplications. By using this technique a leveled FHE scheme without bootstrapping can be achieved.

**BGV scheme** Putting all the pieces together, we can now define the BGV leveled FHE scheme. According to the number of levels $L$ the ladder of decreasing moduli $\{q_L, \ldots, q_0\}$ will be constructed, with $q_L$ consisting of $(L+1)\mu$ bits and $q_0$ $\mu$ bits.

- **KeyGen**: for $j = L$ to 0 repeat: get keypair $(s_j, A_j)$ from the basic scheme. Set $s'_j = \mathbf{BitDecomp}(s_j \otimes s_j)$ and compute $\tau_{s'_j \to s_{j-1}}$ with **SwitchKeyGen** (omit this final step for $j = 0$).
  *Output*: $(sk, pk) = ((s_L, \ldots, s_0), (A_L, \ldots, A_0, \tau_{s'_L \to s_{L-1}}, \ldots, \tau_{s'_1 \to s_0}))$
- **Enc**$(pk, m)$: as in basic scheme.
- **Dec**$(C,\ sk)$: as in basic scheme.
- **Eval:add**$(c, d)$: for $c = (c_0, c_1), d = (d_0, d_1)$ add every entry.
  *Output*: $(c_0 + d_0, c_1 + d_1)$
- **Eval:mult**$(pk, c, d)$: set $c_3$ to be the coefficient vector of $L_{c,d}(x \otimes x)$. Set $c'_3 = \mathbf{Powersof2}(c_3)$ and for $c'_3$ switch from the current modulus $q_j$ to the smaller modulus $q_{j-1}$ as described before. Finally apply **SwitchKey** on the ciphertext using the correct $\tau$ from the public key $pk$, and output this result.

After addition also key and modulus switching could be done, by interpreting the new ciphertext as encrypted under key $s'_j = s_j \otimes s_j$ instead of under $s_j$ (which is possible because $s'_j$ contains all the powers $s_j$ contains and more).

# 7 Multi-key FHE

So far we have been looking at the single-user setting, where all ciphertexts must be encrypted under the same key. It is also interesting to look at the situation where we don't necessarily need the same key for all messages, in order to outsource computations on data coming from different sources. The notion of multi-key FHE encryption was introduction in [13] and realized for any number of keys based on the NTRU cryptoscheme [11]. Furthermore it was shown that every FHE scheme can be made multi-key for a constant number of keys, a high level overview of this construction will follow. Clear and McGoldrick proposed in [8] a multi-key variant based on Learning With Errors (LWE) which was followed by a simpler version in [16].

In the multi-key FHE setting we have $N$ participants with their own keypair $(sk_i, pk_i)$ and message $m_i$, who want to perform computations on all data without revealing any private information to each other. After the computation decryption should only be possible when all the secret keys that were used to encrypt the messages are involved.

In [13] a construction is given to make any FHE scheme multi-key for a constant number of keys. This is achieved by making use of an 'onion' encryption and decryption, where ciphertext are repeatedly encrypted or decrypted with a sequence of keys. In a standard FHE scheme a message $m \in \{0,1\}$ is encrypted into a ciphertext $c \in \{0,1\}^\lambda$. If a ciphertext has to be encrypted, under a different key, we need a definition for encrypting $x \in \{0,1\}^l$. Let $x_1, \ldots, x_l$ be the bits of $x$ then let encryption be as follows:

$$\overline{\mathbf{Enc}}(pk, x) = (\mathbf{Enc}(pk, x_1), \ldots, \mathbf{Enc}(pk, x_l))$$

Now we define "onion" encryption $\mathbf{Enc}^*$ for $k \in \mathbb{N}$ recursively:

$$\begin{aligned}\mathbf{Enc}^*(pk, m) &= \overline{\mathbf{Enc}}(pk, m)\\ \mathbf{Enc}^*(pk_1, \ldots, pk_k, m) &= \mathbf{Enc}^*(pk_1, \ldots, pk_{k-1}, \overline{\mathbf{Enc}}(pk_k, m))\\ &= \overline{\mathbf{Enc}}(pk_1, \overline{\mathbf{Enc}}(\ldots \overline{\mathbf{Enc}}(pk_k, m)\cdots)\end{aligned}$$

Note that a ciphertext produced by $\mathbf{Enc}^*$ encrypting a message under $N$ keys has size $\lambda^N$.
In a similar way we define decryption:

$$\begin{aligned}\mathbf{Dec}^*(sk, c) &= \mathbf{Dec}(sk, c)\\ \mathbf{Dec}^*(sk_1, \ldots, sk_k, c) &= \mathbf{Dec}^*(sk_2, \ldots, sk_k, \mathbf{Dec}(sk_1, c))\\ &= \mathbf{Dec}(sk_k, \mathbf{Dec}(\ldots \mathbf{Dec}(sk_1, c))\cdots)\end{aligned}$$

A ciphertext $c_i$ encrypting message $m_i$ with public key $p_i$ can be turned into a new ciphertext $z_i$ that is encrypting the same message under keys $p_1, \ldots, p_k$. This is done by homomorphically evaluating the function $\mathbf{Enc}^*(p_{i+1}, \ldots, p_N, c_i)$ which gives $\mathbf{Enc}^*(p_i, \ldots, p_N, m_i)$. Then encrypt this new ciphertext with the remaining keys to obtain $\mathbf{Enc}^*(p_1, \ldots, p_N, m_i)$.

When the ciphertexts involved have been changed into ciphertext encrypting the same message under all keys, it is possible to perform homomorphic operations on them, keeping in mind the order of the keys involved. Since the size of a ciphertext $z_i$ is $\lambda^N$ and $N$ is the number of keys, this can only be efficient if $N = O(1)$; in other words only a constant number of keys can be involved. More details can be found in [13].

## 7.1 Multi-key NTRU

NTRU was introduced as a public key cryptosystem in the nineties [11]. When it was shown that a modified version of the NTRU scheme [19] can actually be used for FHE, it turned out it could support multiple keys [13]. The security is based on RLWE and the Decisional Small Polynomial Ratio (DSRP) assumption that is the following.

**DSPR assumption [13]** Define ring $R = \mathbb{Z}[x]/\langle\phi(x)\rangle$ for $\phi(X) \in \mathbb{Z}[x]$ a polynomial of degree $n$. Let $q \in \mathbb{Z}$ be a prime integer and $\chi$ a distribution over $R$. Furthermore $R_q = R/qR$. Then the $\mathbf{DRPR}_{\phi,q,\chi}$ says it is hard to distinguish between these distributions:

- polynomial $h$, where $h = [2gf^{-1}]_q$ for $f = 2f' + 1$ and $f', g$ sampled from $\chi$ (and $f^{-1}$ is the inverse of $f$ in $R_q$)
- polynomial $u$, sampled uniformly at random from $R_q$

First the modified NTRU scheme is given, followed by the multi-key fully homomorphic version.

**Modified NTRU scheme** Define ring $R = \mathbb{Z}[x]/\langle x^n + 1\rangle$ for $n$ a power of 2. Let $q \in \mathbb{Z}$ be an odd prime integer and $\chi$ a $B$-bounded distribution over $R$ ($B \ll q$), which means the magnitude of the coefficients of a polynomial sampled from $\chi$ is less than $B$.

- **NTRU.KeyGen**: sample $f', g$ from $\chi$. Set $f = 2f' + 1$ (note that $f \mod 2 \equiv 1$). If $f$ is not invertible resample, otherwise compute $f^{-1}$ and set $h = [2gf^{-1}]_q$.
  *Output:* $(sk, pk) = (f, h)$
- **NTRU.Enc**$(m, pk)$: for $m \in \{0, 1\}$, sample $s, e$ from $\chi$ and parse $h = pk$.
  *Output:* $c = [hs + 2e + m]_q$
- **NTRU.Dec**$(c,\ sk)$: parse $f = sk$ and compute $z = [fc]_q$.
  *Output:* $\mu = z \mod 2$

Decryption works as follows:

$$\begin{aligned}\mu &= z \mod 2\\ &= [fc]_q \mod 2\\ &= [fhs + 2fe + fm]_q \mod 2\\ &= [2gs + 2fe + fm]_q \mod 2\end{aligned}$$

Because all elements $f, g, s, e$ come from $\chi$ and $B \ll q$, there is no reduction mod $q$. Furthermore recall that $f \equiv 1 \mod 2$. Then we get:

$$\mu \equiv 2gs + 2fe + fm \equiv fm \equiv m \mod 2$$

**Multi-key FHE based on NTRU** In the multi-key fully homomorphic setting we have two ciphertexts that encrypt different messages $m_1, m_2$ with different public keys $h_1, h_2$. Thus we have the ciphertexts $c_1 = [h_1 s_1 + 2e_1 + m_1]_q$ and $c_2 = [h_2 s_2 + 2e_2 + m_2]_q$. If we simply add them and try to decrypt with the joint secret key $f_1 f_2$, we get the following:

$$\begin{aligned}f_1 f_2(c_1 + c_2) &= f_1 f_2 h_1 s_1 + 2 f_1 f_2 e_1 + f_1 f_2 m_1 + f_1 f_2 h_2 s_2 + 2 f_1 f_2 e_2 + f_1 f_2 m_2\\ &= f_1 f_2 h_1 s_1 + f_1 f_2 h_2 s_2 + 2 f_1 f_2 e_1 + 2 f_1 f_2 e_2 + f_1 f_2 (m_2 + m_1)\\ &= 2(g_1 f_2 s_1 + g_2 f_1 s_2 + f_1 f_2 e_1 + f_1 f_2 e_2) + f_1 f_2 (m_2 + m_1)\\ &= 2\mathbf{e}_{add} + f_1 f_2 (m_2 + m_1)\end{aligned}$$

Which will decrypt correctly if the new error $\mathbf{e}_{add}$ is not too large. This is possible because $s_i, e_i, f_i'$ and $g_i$ were sampled from $\chi$ and $f_i = 2f_i' + 1$, and thus are all relatively small. Now we repeat the same for multiplication:

$$\begin{aligned}f_1 f_2(c_1 c_2) &= f_1 f_2 (h_1 s_1 + 2e_1 + m_1)(h_2 s_2 + 2e_2 + m_2)\\ &= (f_1 f_2 h_1 s_1 + 2 f_1 f_2 e_1 + f_1 f_2 m_1)(h_2 s_2 + 2e_2 + m_2)\\ &= f_1 f_2 h_1 h_2 s_1 s_2 + 2 f_1 f_2 h_1 s_1 e_2 + f_1 f_2 h_1 s_1 m_2 + 2 f_1 f_2 h_2 s_2 e_1 + 4 f_1 f_2 e_1 e_2 + 2 f_1 f_2 e_1 m_2\\ &\quad + f_1 f_2 h_2 s_2 m_1 + 2 f_1 f_2 m_1 e_2 + f_1 f_2 m_1 m_2\\ &= 4 g_1 g_2 s_1 s_2 + 4 g_1 f_2 s_1 e_2 + 2 g_1 f_2 s_1 m_1 + 4 g_2 f_1 s_2 e_1 + 4 f_1 f_2 e_1 e_2 + 2 f_1 f_2 e_1 m_2\\ &\quad + 2 g_2 f_1 s_2 m_1 + 2 f_1 f_2 m_1 e_2 + f_1 f_2 m_1 m_2\\ &= 2\mathbf{e}_{mult} + f_1 f_2 m_1 m_2\end{aligned}$$

Again, it must hold that the new multiplication error is not too large and thus $\chi$ must be chosen appropriately.

For one addition and multiplication this simple approach seems to work. Apart from being able to do the computation this scheme allows the use of multiple keys for multiple ciphertexts, which thus makes it a multi-key scheme. It becomes more tricky when this is extended to circuits where multiple of these operations are performed. First of all it is clear only a limited number of operations can be performed, because of the growing error term. More importantly consider a situation where we have 3 ciphertexts $c_1, c_2, c_3$, similarly to the situation above, encrypted under 3 different keys $f_1, f_2, f_3$. Say we have computed $c_1c_2$ and $c_2c_3$, which then respectively need key $f_1f_2$ and $f_2f_3$ for decryption. With these ciphertexts we compute $c_1c_2 + c_2c_3$ and want to decrypt with the new key $f_1f_2f_3$:

$$\begin{aligned} f_1f_2f_3(c_1c_2 + c_2c_3) &= f_3(f_1f_2c_1c_2) + f_1(f_2f_3c_2c_3) = f_3(2\mathbf{e}_{mult_1} + f_1f_2m_1m_2) + f_1(2\mathbf{e}_{mult_2} + f_2f_3m_2m_3) \\ &= 2\mathbf{e}_{add'} + f_1f_2f_3(m_1m_2 + m_2m_3) \end{aligned}$$

This works, so it seems in general we can make a new keys by appending all keys that were used for encryption of the involved ciphertexts. However, for $c_1c_2 \cdot c_2c_3$ the key $f_1f_2f_3$ will not work! We need the key $f_1f_2^2f_3$:

$$\begin{aligned} \mathbf{c} = f_1f_2^2f_3(c_1c_2 \cdot c_2c_3) &= (f_1f_2c_1c_2)(f_2f_3c_2c_3) = (2\mathbf{e}_{mult_1} + f_1f_2m_1m_2)(2\mathbf{e}_{mult_2} + f_2f_3m_2m_3) \\ &= 2\mathbf{e}_{mult'} + f_1f_2^2f_3(m_1m_2 \cdot m_2m_3) \end{aligned}$$

This shows that it is necessary to be aware of the circuit that was evaluated on the ciphertexts and additionally the size of the key grows faster than just the number of different keys involved.

To solve these problems, the authors used the key-switching technique from Brakerski and Vaikuntanathan [3], also known as relinearization. An evaluation key is added to the public key, that is a "pseudo-encryption" of the powers of 2 of the secret key: $\mathrm{ek}_f = [hs+2e+Pow(f)]_q$, for newly sampled $s, e \in \chi$. This is not a real encryption because we have defined decryption only for a binary message, which the secret key is not. When ciphertexts encrypted under the set of keys $F_1 = \{f_{1_i}, \ldots, f_{1_k}\}$ and $F_2 = \{f_{2_j}, \ldots, f_{2_l}\}$ are multiplied, the keys in $F_1 \cap F_2$ would appear as a square in the new key. For each of those keys the ciphertext is "corrected" with the evaluation key, such that the power of this key goes down in the new key. More precisely, the inner product is taken between the bitexpansion of the ciphertext and each of the evaluation keys. In our example that gives the following: **c'** $= \langle Bit(\mathbf{c}), \mathrm{ek}_{f_2} \rangle \mod q$ with $\mathrm{ek}_{f_2} = [hs + 2e + Pow(f_2)]_q$. Then **c'** can be decrypted with the key $f_1f_2f_3$ as desired.
This gives the following multi-key FHE scheme:

- **KeyGen**: run **NTRU.KeyGen** to obtain $(sk, pk)$ and additionally sample $s', e' \in \chi$ and compute $ek = [hs' + 2e' + Pow(f)]_q$.
  *Output:* $(sk, pk, ek)$
- **Enc**$(m, pk)$: run **NTRU.Enc** to obtain $c$.
  *Output:* $c$
- **Dec**$(c,\ sk_1, \ldots, sk_N)$: compute $z = [f_1 \cdots f_N c]_q$.
  *Output:* $\mu = z \mod 2$
- **Eval:add**$(c_1, c_2)$: $c_{add} = [c_1 + c_2]_q$.
  *Output:* $c_{add}$

- **Eval:mult**$((c_1, F_1), (c_2, F_2))$: where $F_1 = \{pk_1, ek_1\}$ denotes the set of public keys and corresponding evaluation keys associated with ciphertext $c_1$ and $F_2$ similarly for $c_2$. Let $c_0 = [c_1 c_2]_q$ and $F_1 \cap F_2$ contain the evaluation keys $\{ek_{i_1}, \ldots, ek_{i_l}\}$. Set $j = 1$ and repeat until $j = l$: $c_j = [\langle Bit(\mathbf{c}_{j-1}), \mathrm{ek}_{i_j}\rangle]_q$ and set $c_{mult}$ to the resulting $c_l$.
  *Output:* $c_{mult}$

## 7.2 Multi-key GSW

Clear and McGoldrick presented a masking scheme that can be used to make identity based FHE (IBFHE) schemes which use LWE for security, support multiple keys [8]. In an identity based cryptoscheme the public key for a specific person can be deduced from the publicly known user-identity, without any interaction necessary with the individual. GSW was the first scheme that allowed an IBFHE scheme, because no evaluation keys were necessary to perform homomorphic operations on the ciphertexts, which was always the case in previous schemes. Such an evaluation key cannot be computed with an ID by a third party as is the case for a public key, which makes schemes involving evaluation keys unfit to become an IBFHE scheme.

Mukherjee and Wichs gave an implementation of the masking scheme for the GSW scheme [16] and gave the option to perform a 1-round decryption protocol where every party computes and broadcasts a partial decryption which are finally combined to give the resulting plaintext. Their scheme will be discussed.

**Intuition** The GSW scheme was making use of eigenvectors properties for matrices. Because of the involvement of gadget matrices we have the following property for a ciphertext $C$: $C\mathbf{s_i}=mG\mathbf{s_i}+\mathbf{e}$ for secret key $\mathbf{s_i}$, a small error $\mathbf{e}$, gadget matrix $G$ and message $m$. In the multi-key situation we would like to achieve a similar situation, with the new secret key $\mathbf{s}=(\mathbf{s_1}, \ldots, \mathbf{s_k})^T$, larger ciphertext $C'$ and larger gadget matrix $\mathcal{G} = G \cdot I_k$: $C'(\mathbf{s_1}, \ldots, \mathbf{s_k})^T = m\mathcal{G}(\mathbf{s_1}, \ldots, \mathbf{s_k})^T + \mathbf{e'}$.
This will be achieved by transforming an existing ciphertext that encrypts a message $m$ under a certain key $\mathbf{s_i}$ into a new ciphertext encrypting $m$ under the concatenated key $\mathbf{s}$. To achieve this, every ciphertext will carry additional encryptions of the randomness matrix $R$ used to encrypt $m$, which will allow the correct expansion to be performed. Once the expanded ciphertext satisfies said property, homomorphic operations can be done on it as in the single key GSW scheme (with larger parameters).

To achieve the multi-key setting, 2 major changes are applied to the GSW-scheme: the public keys of the participants will depend on each other and, as said, additional information is added to every ciphertext. First we elaborate on the public keys, later it will become clear what the extra information has to be.

**Public and secret keys** The public keys of the participants will be related to each other in the sense that a part of it will be exactly same. Recall that if the secret key is $\mathbf{s}=(\mathbf{-t},1)$, the corresponding public key is $A = \begin{bmatrix} B \\ \mathbf{b} \end{bmatrix}$ for $\mathbf{b}=B\mathbf{t}+\mathbf{e}$.
The LWE assumption states that finding the secret key is hard when given the public key. The crucial observation is that security will not be weakened if the same random matrix $B$ is used for all the public keys of participants of the scheme.
Fix uniformly sampled matrix $B \in \mathbb{Z}^{n-1 \times m}$, and for every secret key $\mathbf{s_i}=(\mathbf{-t}_i,1)$ calculate $\mathbf{b}_i=B\mathbf{t}_i + \mathbf{e}_i$ and set the public key $pk_i$ to be $A_i = \begin{bmatrix} B \\ \mathbf{b}_i \end{bmatrix}$.

**Expanding ciphertext** First we will focus on the situation for 2 participants and then extend this to $k$ participants. We have an encrypted message $m_1$ as a ciphertext $C_1$ and want to expand this ciphertext into $C_1'$ to make it satisfy the property $C_1'(\mathbf{s_1}, \mathbf{s_2})^T = m_1\mathcal{G}(\mathbf{s_1}, \mathbf{s_2})^T + \mathbf{e} \approx m_1\mathcal{G}(\mathbf{s_1}, \mathbf{s_2})^T$. The message is encrypted with public key $A_1$ and randomness matrix $R$. The expanded ciphertext $C_1'$ will have the form $\begin{pmatrix} C_1 & X \\ Y & C_1 \end{pmatrix}$ and we want it to satisfy the following:

$$C_1' \begin{pmatrix} \mathbf{s_1} \\ \mathbf{s_2} \end{pmatrix} = \begin{pmatrix} C_1 & X \\ Y & C_1 \end{pmatrix} \begin{pmatrix} \mathbf{s_1} \\ \mathbf{s_2} \end{pmatrix} \approx m_1\mathcal{G} \begin{pmatrix} \mathbf{s_1} \\ \mathbf{s_2} \end{pmatrix}$$

This means we need the following two properties:

1. $C_1\mathbf{s_1} + X\mathbf{s_2} \approx m_1 G\mathbf{s_1}$
2. $Y\mathbf{s_1} + C_1\mathbf{s_2} \approx m_1 G\mathbf{s_2}$

The first property is satisfied if $X$ is a matrix with every entry 0, for ease this is denoted as $X = 0$. The second case is more difficult, but it is helpful to see what happens when trying to 'decrypt' $C_1$ with the incorrect key $\mathbf{s_2}$:

$$C_1\mathbf{s_2} = (RA_1 + m_1G)\mathbf{s_2} = RA_1\mathbf{s_2} + m_1G\mathbf{s_2}$$

Furthermore we know $A_1 = \begin{bmatrix} B \\ \mathbf{b}_1 \end{bmatrix}, A_2 = \begin{bmatrix} B \\ \mathbf{b}_2 \end{bmatrix}$ and $\mathbf{s_2}$=(**-t**$_2$, 1). We would like the term $RA_1\mathbf{s_2}$ to actually be $RA_2\mathbf{s_2}$ because this would give a small error, while we cannot say anything about the size of $RA_1\mathbf{s_2}$. This means that the actual and the ideal term only differ a factor of $R$:

$RA_2\mathbf{s_2} + m_1G\mathbf{s_2} = R(A_2 - A_1)\mathbf{s_2} + RA_1\mathbf{s_2} + m_1G\mathbf{s_2} = R(\mathbf{b_2} - \mathbf{b_1}) + RA_1\mathbf{s_2} + m_1G\mathbf{s_2}$

Now the challenge is to construct $R(\mathbf{b_2} - \mathbf{b_1})$ *without* knowing $R$.
It turns out there is a trick we can use for this; write $\delta = \mathbf{b_2} - \mathbf{b_1}$, which can be done without any knowledge of the secret key because $\mathbf{b_1}$, $\mathbf{b_2}$ are parts of the public keys $B_1, B_2$. Let $r_{i,j}$ be the entries of matrix $R$ and let participant 1 encrypt every one of them with $B_1$ and a fresh randomness: $U_{i,j} = R'B_1 + r_{i,j}G$. Furthermore let matrices $Z_{i,j}$ have entries all 0 except for the last column, which is equal to the vector $\delta$. Apply $G^{-1}$ to get $Z'_{i,j} = G^{-1}(Z_{i,j})$. Now we have:

$$\begin{aligned} Z'_{i,j}U_{i,j}\mathbf{s_1} &\approx Z'_{i,j}r_{i,j}G\mathbf{s_1} \\ &\approx r_{i,j}G^{-1}(Z_{i,j})G\mathbf{s_1} \\ &\approx r_{i,j}Z_{i,j}\mathbf{s_1} \\ &\approx r_{i,j}\delta \end{aligned}$$

If this is done for all $i, j$ and put together, the complete term $R(\mathbf{b_2} - \mathbf{b_1}) = R\delta$ can be recovered by multiplying with secret key $\mathbf{s_1}$. Fortunately this is exactly what we want, since the intention is to have $Y\mathbf{s_1} + C_1\mathbf{s_2} \approx m_1G\mathbf{s_2}$ and the term $C_1\mathbf{s_2}$ differs about $R(\mathbf{b_2} - \mathbf{b_1})$ from the ideal outcome $m_1G\mathbf{s_2}$.
In conclusion to find matrix $Y$ we needed encryptions of $r_{i,j}$ and the difference between public keys $B_1, B_2$. Thus participant 1 should include the encryptions of $r_{i,j}$ along with the ciphertext $C_1$, producing a tuple of ciphertexts rather than a single ciphertext. Then the expansion of ciphertext $C_1$ to $C_1'$ means calculating matrices $Z'_{i,j}$ and $U_{i,j}$ and put it together in a new matrix:

$$\begin{pmatrix} C_1 & 0 \\ \sum_{i,j} Z'_{i,j}U_{i,j} & C_1 \end{pmatrix}$$

Including the encryptions of $r_{i,j}$ will not weaken the security of the scheme because of the security of GSW encryption. Also, expansion of ciphertexts can be done by any party because no secret keys are involved, which is very convenient.

**$k$ participants** So far only 2 participants were involved in the multi-key GSW scheme, but this can be extended to $k$ participants. The expanded ciphertext grows with the number of participants but it will have the same form. Assume we again have a ciphertext $C_1$ encrypted with $B_1$ which must be expanded, this time to a ciphertext that can only be decrypted by the concatenation of all $k$ keys. This expanded ciphertext will look like this:

$$\begin{pmatrix} C_1 & 0 & \dots & 0 \\ Y_2 & C_1 & \dots & 0 \\ \vdots & & \ddots & \vdots \\ Y_k & 0 & \dots & C_1 \end{pmatrix}$$

The first column contains the ciphertext $C_1$ and the 'correction' matrices $Y_j$ that will give the correction factors $R(\mathbf{b_j} - \mathbf{b_1})$ when multiplied with key $\mathbf{s_1}$. Note that these matrices can be determined just as in the 2-key scheme, only with different $\delta = \mathbf{b_j} - \mathbf{b_1}$ for different rows $j$. In general for a message encrypted by participant $i$, column $i$ will contain correction matrices $Y_j$ for all $j \neq i$ and the diagonal of the matrix will consist of the original ciphertext $C_i$.

**'Threshold' decryption** This multi-key scheme gives the possibility for 'threshold' decryption; every participant should do a piece of the decryption with its own key and these partial decryptions together make it possible to retrieve the message. To allow this process we need the following lemma, to make sure no secret keys can be deduced from the ciphertexts and partial decryptions.

**Smudging noise lemma[16].** For $I_1, I_2$ positive integers and $e_1 \in [-I_1, I_1]$ fixed integer, choose $e_2 \in [-I_2, I_2]$ uniformly at random. Then the distribution of $e_2$ is statistically indistinguishable from that of $e_1 + e_2$ if $B_1/B_2 = \text{negl}(\lambda)$ where negl is a negligible function.

A proof can be found in [2]. Now the adjusted decryption process can be described. Assume we have a ciphertext $C'$ in the expanded form (possibly homomorphic computations have been performed on it). This can be seen as a vector of submatrices: $C' = \left(\mathcal{M}_1 \dots \mathcal{M}_k\right)$ where every $\mathcal{M}_i$ is a 'column' of matrices. Now $\forall i$ participant $i$ :

1. receives $\mathcal{M}_i$
2. computes $p_i' = \langle \mathcal{M}_i s_i, G^{-1}(w') \rangle$ for $w'$ a vector with all zeros and last entry $\lceil \frac{q}{2} \rceil$
3. adds 'smudging noise' $e \in \mathbb{Z}_q$, set $p_i = p_i' + e$, and outputs $p_i$

Finally the message can be retrieved from $p = \sum_{i=1}^{k} p_i$ through dividing by $\frac{q}{2}$ and rounding correctly. This smudging noise will make sure the secret key $s_i$ cannot be deduced from $\mathcal{M}_i$ and $p_i$. The correct parameters with respect to the original noise and the lemma must be used to generate this noise.

### 7.3 Multi-key B(G)V for logarithmic number of participants

A very limited multi-key version for the BV scheme [5] was proposed in [13], which is also applicable to the BGV scheme. The addition of ciphertexts $c_1, c_2$ will be defined as $(c_1, c_2)$, with new secret key $(s_1, s_2)$. This works because decryption gives the following:

$$\langle (c_1, c_2), (s_1, s_2) \rangle = \langle c_1, s_1 \rangle + \langle c_2, s_2 \rangle$$

For the multi-key setting multiplication of 2 ciphertexts $c_1, c_2$ is defined as the coefficient vector $L_{c,d}(x \otimes y)$. For decryption set $x = s_1$ and $y = s_2$, in order words the new key is $s_1 \otimes s_2$. No key-switching will take place, which means the size of the key and ciphertext grows. If we have 2 ciphertexts of length $l_1$ and $l_2$ the multiplied ciphertext will have length $l_1 \cdot l_2$. After $N - 1$ operations the size of the ciphertext and decryption key can be at most $2^N$ to allow $\log \lambda$ number of participants (keys). This means one of the ciphertexts involved in multiplication must always be fresh and thus have size 2. For addition there is no such restriction because the size of the new ciphertext is always the addition of the sizes of the original ciphertexts. Thus these small changes result in a scheme that can support $O(\log \lambda)$ keys.

Note that the new secret key is different after addition and multiplication. It is possible to make this equal, on the cost of also requiring addition to also always have at least one fresh ciphertext as entry. Define addition of $c = (c_0, c_1), d = (d_0, d_1)$ as $(c_0 + d_0, d_1, c_1, 0)$ for new key $s_1 \otimes s_2 = (1, s_2, s_1, s_1 s_2)$. Decryption gives:

$$\langle (c_0 + d_0, d_1, c_1, 0), (1, s_2, s_1, s_1 s_2) \rangle = \langle c_1, s_1 \rangle + \langle c_2, s_2 \rangle$$

Even though this is possible, it is still necessary to know which keys and in which order were involved in which sequence of operations in order to assemble the correct decryption key at the end. It is probably simplest to keep track of the form of the new key and output this additionally to a new ciphertext.

Apart from the fact that the scheme only supports a logarithmic number keys, also the complexity of the functions that are supported by the scheme is limited, thus this is merely a SHE scheme. In the multi-key setting it is not clear how to apply relinearization or squashing. Modulus switching could be a challenge, because participants would have to settle on a fixed sequence of moduli, but seems to be possible.

## 8 Overview

First the notion of FHE was introduced, along with a high level overview of the first FHE scheme, introduced by Gentry. His solution of taking a SHE scheme and turning it into a FHE one by squashing the decryption circuit and then bootstrapping has been the blueprint for many schemes that followed. Several LWE/RLWE based schemes have been discussed which showed different techniques have been introduced along the way to differ from Gentry's blueprint. Basing the scheme on LWE or RLWE, rather than on ideal lattices as was the case with Gentry, has made the systems easier to understand. Among the newly introduced techniques are relinearization and modulus-switching from Brakerski and Vaikuntanathan. The first technique aims to switch keys, from a larger one to a shorter one, while making the ciphertext shorter in the process. By using modulus switching the noise level grows at a slower pace than before. Obviously both techniques influence the amount of

operations can be performed in a positive way. In the GSW scheme a gadget matrix was used to keep the noise level down. No key-switching was done in this leveled FHE scheme, the focus was on developing an easy to understand scheme where homomorphic addition and multiplication corresponded simply with matrix addition and multiplication. Along with the new scheme a IBFHE compiler was given, making it possible to turn LWE-based schemes with certain properties into IBFHE schemes.

Fully homomorphic encryption is a very interesting notion by itself, allowing a variation of applications. With the introduction of multi-key FHE this range of application grows further. In such a scheme multiple secret keys are involved, and every participant encrypts its own message under its own key. Evaluations on ciphertexts encrypted under different keys are possible in this setting, resulting in a ciphertext that can only be decrypted with the use of all secret keys that were used to encrypt the inputs of the evaluation. When the standard cryptoscheme NTRU was made FHE it turned out to support multiple keys, which gave rise to this new notion. It turned out every FHE scheme can be made multi-key in an inefficient way, by using onion encryption and decryption. The IBFHE compiler that was given before was transformed into a M-IBFHE compiler, showing it was possible to let LWE-based schemes support multiple keys. An implementation of this for GSW was given, along with a way to distribute the decryption procedure among the participants. By definition of M-FHE it is always necessary to use all secret keys involved with the computation to decrypt the result, while participants of course do not want to share their key. Making it possible to let each participant do a partial decryption which finally add up to the actual result is a nice addition to make the scheme more practical.

Still many developments are happening in this area of research, especially because performance seems to be a bottleneck. This has not been the focus of this report, but it is worth mentioning there is only one open source library implementing a FHE scheme (BGV) at this moment (HElib). There are examples of the actual use of FHE for practical solutions, for example for secure health care monitoring as done in [1] (it must be mentioned that instead of circuits, branching programs were used for computations). However, it seems performing statistics on health care data with the use of multi-key FHE, as suggested in the introduction, has not been done yet and improvements of the schemes might be necessary to make it realizable.

# References


[1] Scott Ames, Muthuramakrishnan Venkitasubramaniam, Alex Page, Övünç Kocabaş, and Tolga Soyata. Secure health monitoring in the cloud using homomorphic encryption, a branching-program formulation. 2015.

[2] Gilad Asharov, Abhishek Jain, and Daniel Wichs. Multiparty computation with low communication, computation and interaction via threshold fhe. 2011.

[3] Z. Brakerski and V. Vaikuntanathan. Efficient fully homomorphic encryption from (standard) lwe. In *Foundations of Computer Science (FOCS), 2011 IEEE 52nd Annual Symposium on*, pages 97–106, Oct 2011.

[4] Zvika Brakerski, Craig Gentry, and Vinod Vaikuntanathan. (leveled) fully homomorphic encryption without bootstrapping. In *Proceedings of the 3rd Innovations in Theoretical Computer Science Conference*, pages 309–325. ACM, 2012.

[5] Zvika Brakerski and Vinod Vaikuntanathan. Fully homomorphic encryption from ring-lwe and security for key dependent messages. In *Advances in Cryptology–CRYPTO 2011*, pages 505–524. Springer, 2011.

[6] A. Sahai C. Gentry and B. Waters. Homomorphic encryption from learning with errors: Conceptually-simpler, asymptotically-faster, attribute-based. In *Advances in Cryptology–CRYPTO 2013*, pages 75–92. Springer, 2013.

[7] P.K. Cameron. *Introduction to algebra*. Oxford University Press, 2012.

[8] Michael Clear and Ciarán McGoldrick. Multi-identity and multi-key leveled fhe from learning with errors. Technical report, Cryptology ePrint Archive, Report 2014/798, 2014. http://eprint. iacr. org.

[9] Craig Gentry et al. Fully homomorphic encryption using ideal lattices. In *STOC*, volume 9, pages 169–178, 2009.

[10] Shai Halevi and Victor Shoup. Helib - an implementation of homomorphic encryption. `https://github.com/shaih/HElib`, 2013.

[11] Jeffrey Hoffstein, Jill Pipher, and Joseph H Silverman. Ntru: A ring-based public key cryptosystem. In *Algorithmic number theory*, pages 267–288. Springer, 1998.

[12] Jing Liu. Verifiable delegation of computation in the setting of privacy-preserving biometric authentication. Master's thesis, Chalmers University of Technology, 2015.

[13] Adriana López-Alt, Eran Tromer, and Vinod Vaikuntanathan. On-the-fly multiparty computation on the cloud via multikey fully homomorphic encryption. In *Proceedings of the forty-fourth annual ACM symposium on Theory of computing*, pages 1219–1234. ACM, 2012.

[14] Vadim Lyubashevsky, Chris Peikert, and Oded Regev. On ideal lattices and learning with errors over rings. In *In Proc. of EUROCRYPT, volume 6110 of LNCS*, pages 1–23. Springer, 2010.

[15] Daniele Micciancio and Shafi Goldwasser. *Complexity of lattice problems: a cryptographic perspective*, volume 671. Springer Science & Business Media, 2012.

[16] Pratyay Mukherjee and Daniel Wichs. Two round mutliparty computation via multi-key fhe. Cryptology ePrint Archive, Report 2015/345, 2015. `http://eprint.iacr.org/`.

[17] Oded Regev. On lattices, learning with errors, random linear codes, and cryptography. *Journal of the ACM (JACM)*, 56(6):34, 2009.

[18] Ronald L Rivest, Len Adleman, and Michael L Dertouzos. On data banks and privacy homomorphisms. *Foundations of secure computation*, 4(11):169–180, 1978.

[19] Damien Stehlé and Ron Steinfeld. Making ntru as secure as worst-case problems over ideal lattices. In *Advances in Cryptology–EUROCRYPT 2011*, pages 27–47. Springer, 2011.