Multi-key fully homomorphic encryption report

Elena Fuentes Bongenaar

July 12, 2016

1 Introduction

Since Gentry's first Fully Homomorphic Encryption (FHE) scheme in 2009 [6] multiple new schemes have been proposed. In 2012 the notion of multi-key FHE was introduced by [9] and realized based on the NTRU cryptoscheme. They also showed that every FHE scheme can be made multi-key for only a constant number of keys. Clear and McGoldrick proposed in [5] a multi-key variant based on Learning With Errors (LWE) which was followed by a simpler version in [11]. The BGV FHE scheme [3] has been implemented in HElib [7] and has good performance, thus it would be interesting if this scheme could be made multi-key. The only known variant will support a logarithmic number of keys (in the security parameter). This report will give an introduction to multi-key FHE and discuss the schemes proposed in [9], [5] and [11].

2 Notation and preliminaries

The security parameter is denoted by λ. Reduction mod q will reduce into the set $\{-\lfloor \frac{q}{2} \rfloor, \dots, \lfloor \frac{q}{2} \rfloor\}$. A vector $(v_1, \dots, v_n) \in \mathbb{Z}^n$ is represented as $\mathbf{v}$, with v_i being the ith component. The l_1 norm for vector $\mathbf{v}= (v_1, \dots, v_n)$ is defined as $\sum_{i=1}^{n} |v_i|$.

Learning with errors (LWE) Many recent proposed schemes base their security on the hardness of the learning with errors problem introduced by Regev [12]. Given an integer q, a (Gaussian) distribution χ on $\mathbb{Z}_q$, 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 χ 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.

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 [10]. 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 χ 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 χ. Set $b = a \cdot s + e$.

The RLWE assumption is that this is a hard problem. There is a quantum reduction of this problem to a hard problem in ideal lattices. 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.

3 Fully Homomorphic Encryption

Fully Homomorphic Encryption (FHE) is a special type of encryption that allows arbitrary computations to be performed on the ciphertexts in a meaningful way. This means operations performed in the ciphertextspace will translate back to operations on the plaintexts after decrypting the result. This makes it possible to outsource computations on sensitive data, because computations on the ciphertexts can be performed without knowing the secret key.

Such a FHE scheme **E** will consist of four algorithms; **E** = (**KeyGen**, **Enc**, **Dec**, **Eval**) [6]. **Eval** is the function that makes this encryption scheme special, because for every function f and ciphertexts $c_1 = \mathbf{Enc}(m_1, p_k), \ldots, c_n = \mathbf{Enc}(m_n, p_k)$, which can be decrypted with the secret key s_k, it should provide the following:

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

Note that the plaintexts have to be encrypted under the same public key and the output of **Eval** is again encrypted under that same key.

Ideally the number of operations one can perform on ciphertexts is unlimited, but this is in practice hard to achieve. Also, for many applications it is not necessary to be able to compute an arbitrary number of operations. *Somewhat* homomorphic encryption (SHE) allows computations of limited complexity to be performed, opposed to any complexity for *fully* HE schemes. Another option is a 'leveled FHE scheme', in this case the size of the public key is depending linearly on the number of computations that can be performed.

4 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 could be interesting to look at the situation where we don't necessarily need the same key for all messages, in order to be able to outsource computations on data coming from different sources. For example to do statistics on health care data from different hospitals, while keeping all input secret from each other.

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.

By [9] it is shown any FHE scheme is 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 we encrypt a message $m \in \{0,1\}$ into a ciphertext $c \in \{0,1\}^\lambda$. If we want to encrypt a ciphertext again, 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 λ^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 λ^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 [9].

5 Multi-key NTRU

NTRU was introduced as a public key cryptosystem in the nineties [8]. When it was shown that a modified version of the NTRU scheme [13] can actually be used for FHE, it turned out it could support multiple keys [9]. The security is based on RLWE and the Decisional Small Polynomial Ratio (DSRP) assumption that is the following.

DSPR assumption [9] 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 χ 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 χ (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 χ a B-bounded distribution over R ($B \ll q$), which means the magnitude of the coefficients of a polynomial sampled from χ is less than B.

- **NTRU.KeyGen**: sample f', g from χ. 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 χ 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 χ 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 $c_1 = [h_1s_1 + 2e_1 + m_1]_q$ and $c_2 = [h_2s_2 + 2e_2 + m_2]_q$. If we simply add them and try to decrypt with the joint secret key f_1f_2, we get the following:

$$\begin{aligned}f_1f_2(c_1 + c_2) &= f_1f_2h_1s_1 + 2f_1f_2e_1 + f_1f_2m_1 + f_1f_2h_2s_2 + 2f_1f_2e_2 + f_1f_2m_2\\ &= f_1f_2h_1s_1 + f_1f_2h_2s_2 + 2f_1f_2e_1 + 2f_1f_2e_2 + f_1f_2(m_2 + m_1)\\ &= 2(g_1f_2s_1 + g_2f_1s_2 + f_1f_2e_1 + f_1f_2e_2) + f_1f_2(m_2 + m_1)\\ &= 2\mathbf{e}_{add} + f_1f_2(m_2 + m_1)\end{aligned}$$

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 χ and $f_i = 2f'_i + 1$, and thus are all relatively small. Now we repeat the same for multiplication:

$$\begin{aligned}
f_1f_2(c_1c_2) &= f_1f_2(h_1s_1+2e_1+m_1)(h_2s_2+2e_2+m_2)\\
&= (f_1f_2h_1s_1+2f_1f_2e_1+f_1f_2m_1)(h_2s_2+2e_2+m_2)\\
&= f_1f_2h_1h_2s_1s_2+2f_1f_2h_1s_1e_2+f_1f_2h_1s_1m_2+2f_1f_2h_2s_2e_1+4f_1f_2e_1e_2+2f_1f_2e_1m_2\\
&\quad +f_1f_2h_2s_2m_1+2f_1f_2m_1e_2+f_1f_2m_1m_2\\
&= 4g_1g_2s_1s_2+4g_1f_2s_1e_2+2g_1f_2s_1m_1+4g_2f_1s_2e_1+4f_1f_2e_1e_2+2f_1f_2e_1m_2\\
&\quad +2g_2f_1s_2m_1+2f_1f_2m_1e_2+f_1f_2m_1m_2\\
&= 2\mathbf{e}_{mult}+f_1f_2m_1m_2
\end{aligned}$$

Again, it must hold that the new multiplication error is not too large and thus χ 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 [2], 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: $\text{ek}_f = [hs+2e+Pow(f)]_q$, 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: $\mathbf{c'} = \langle Bit(\mathbf{c}), \text{ek}_{f_2}\rangle \mod q$ with $\text{ek}_{f_2} = [hs+2e+Pow(f_2)]_q$. Then $\mathbf{c'}$ 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}

6 GSW FHE scheme

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 λ such that $A\mathbf{v}=\lambda\mathbf{v}$ and in this case λ 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

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

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 **a**, **a'** be vectors 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:

- **BitDecomp**$(\mathbf{a}) = (a_{1,0}, \ldots, a_{1,l-1}, a_{2,0}, \ldots, a_{m,l-1})$, the binary representation of vector **a** with LSB first. Input has dimension m, output N.
- **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 **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 χ 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 $\mathbf{s}$=(-$\mathbf{t}$,1) $\in \mathbb{Z}_q^n$. Sample uniformly matrix B from $\mathbb{Z}_q^{m\times n-1}$ and **e** from χ^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:* $(C_1 G^{-1}(C_2))$

7 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 [5]. 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 [11] 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 $\mathbf{s}$=$(\mathbf{s_1}, \dots, \mathbf{s_k})^T$, larger ciphertext C' and larger gadget matrix $\mathcal{G} = G \cdot I_k$: $C'(\mathbf{s_1}, \dots, \mathbf{s_k})^T$=$m\mathcal{G}(\mathbf{s_1}, \dots, \mathbf{s_k})^T$+**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 $C_1'(\mathbf{s_1}, \mathbf{s_2})^T$=$m_1\mathcal{G}(\mathbf{s_1}, \mathbf{s_2})^T$+$\mathbf{e}$≈ $m_1\mathcal{G}(\mathbf{s_1}, \mathbf{s_2})^T$. 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. $C_1\mathbf{s_1} + X\mathbf{s_2} \approx m_1G\mathbf{s_1}$
2. $Y\mathbf{s_1} + C_1\mathbf{s_2} \approx m_1G\mathbf{s_2}$

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 $\mathbf{s_2}$=(-$\mathbf{t}_2$, 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 δ. 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[11]. 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 [1]. 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: $C' = \begin{pmatrix}\mathcal{M}_1 \dots \mathcal{M}_k\end{pmatrix}$ 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.

8 The BGV scheme

Brakerski, Gentry and Vaikuntanathan constructed the first FHE scheme that doesn't need bootstrapping [3]. 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 [7] along with many proposed optimizations. In [9] a multi-key variant of the BV-scheme [2], a predecessor of the BGV scheme that has many similarities, for $O(\log \lambda)$ number of keys was given. Unfortunately it is not clear how to turn this scheme into a leveled or FHE variant. Because of the similarities between BV and BGV, and the popularity of the latter scheme it is still interesting to take another look at this multi-key proposal.

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

- **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 $c = m + A^T r = (c_0, c_1)$.
  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: $L_c(x) = c_0 + c_1x$. 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 [3]. 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 τ 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 [3]. 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 [3]. 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 μ bits.

- **KeyGen**: for $j = L$ to 0 repeat: get keypair (s_j, A_j) from the basic scheme. Set $s_j' = \textbf{BitDecomp}(s_j \otimes s_j)$ 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 $c_3' = \textbf{Powersof2}(c_3)$ 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 τ 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).

8.1 Multi-key BGV for $O(\log \lambda)$ number of participants

A very limited multi-key version for the BV scheme [4] was proposed in [9], 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.

References

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

[2] 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.

[3] 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.

[4] 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.

[5] 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.

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

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

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

[9] 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.

[10] 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.

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

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

[13] 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.