Introduction to the BFV FHE Scheme

Inferati Inc.

Washington, USA

1 Scope

This is the first article of a blog series about Fully Homomorphic Encryption (FHE) and its applications. In this article, we introduce the levelled Brakerski/Fan-Vercauteren [Bra12, FV12] scheme, a Ring-Learning with Errors (RLWE)-based cryptosystem.

2 Introduction

The Fan-Vercauteren (FV) scheme, (also known as the Brakerski-Fan-Vercauteren (BFV) scheme) is considered as one of the second generation of FHE schemes that is constructed based on the Ring-Learning with Errors (RLWE) problem [LPR13]. BFV is instantiated over two rings: 1) the plaintext ring which includes encodings of unencrypted or intelligible messages and 2) the ciphertext ring which includes encrypted messages. Similar to any other FHE scheme, BFV allows an untrusted party to induce meaningful computation over encrypted data without access to the decryption key. This is possible due to the homomorphism property which offers a map (or function) between the plaintext and ciphertext spaces that preserves the operations in these two spaces.

This can be illustrated concretely as follows. Let f be a function that maps $\mathcal{P} \to \mathcal{C}$, where $\mathcal{P}$ and $\mathcal{C}$ are mathematical structures (sets for e.g.) with a defined binary operation $\diamond$, if for every x and y in $\mathcal{P}$ Equation (1) holds, then f is a homomorphic map that preserves $\diamond$ between $\mathcal{P}$ and $\mathcal{C}$.

$$f(x \diamond y) = f(x) \diamond f(y) \tag{1}$$

As a simple numerical example, consider the homomorphism $f(r) = 2r$ that maps $\mathbb{Z}$ to $\mathbb{Z}$, where $\mathbb{Z}$ is the set of all integers. Let $x = -1$, $y = 4$ and

$\diamond = +$. Let's check if Equation 1 holds for f.

$$\begin{aligned} f(x+y) &\overset{?}{=} f(x) + f(y) \\ f(-1+4) &\overset{?}{=} f(-1) + f(4) \\ f(3) &\overset{?}{=} -2 + 8 \\ 6 &= 6 \end{aligned} \tag{2}$$

To reflect the above on FHE and for the sake of clarification, one can think of f as the encryption function that maps plaintext messages in $\mathcal{P}$ to ciphertext messages in $\mathcal{C}$. The left side of Equation (1) corresponds to the encrypted result one would like to have after manipulating the encrypted messages on the right side (homomorphic evaluation). Note that the results of manipulating ciphertext messages in FHE are also encrypted.

There are a few worth noting points here that need to be emphasized:

- The aforementioned analogy is only illustrative, FHE includes more complex mappings and spaces as we shall see later.
- $f(r)$ preserves addition only. Applying Equation (1) with $(\diamond = \times)$ does not work.
- For the encryption scheme to be really FHE (i.e., able to compute arbitrary functions on data), it must preserve at least two operations: addition and multiplication. If the scheme supports only addition, it is known as additive homomorphic scheme, whereas if it supports only multiplication, it is known as multiplicative homomorphic scheme.

2.1 BFV Primitives

Now we have a basic idea of the notion of homomorphism, we can define the basic primitives of BFV [ACC+18].

BFV includes the following primitives:

- $\mathsf{ParamGen}(\lambda) \rightarrow \mathsf{Params}$:
  Parameter generator (ParamGen) takes as input the security parameter λ, which is a number used to define the security level of BFV, and returns a set of encryption parameters used in BFV. Possible values of λ with increasing security level are $80, 128$ and 256, with 80 being deprecated according to the FHE standardization consortium [ACC+18]. One can view λ as the computational cost of successful attacks on the scheme. In order for these attacks to succeed with probability 1, they would require 2^{λ} basic computational operations.
- $\mathsf{KeyGen}(\mathsf{Params}) \rightarrow \{\mathsf{SK}, \mathsf{PK}, \mathsf{EK}\}$:
  Key generation (KeyGen) takes as input the encryption parameters and

returns a set of keys: 1) the secret key (SK) which is mainly used for decryption[1], 2) the public key (PK) which is used for encryption and 3) the evaluation key (EK) which is used to evaluate homomorphic operations on ciphertexts as we shall see later. While SK should be kept private by the user (sensitive data owner), PK and EK can be shared publicly without affecting the security of the scheme.

- $\mathsf{Encrypt}(\mathsf{PK}, \mathsf{M}) \rightarrow \mathsf{C}$:
  Encrypt takes as input PK and a plaintext message M in the plaintext space $\mathcal{P}$, and returns a valid ciphertext C from the ciphertext space $\mathcal{C}$.

- $\mathsf{Decrypt}(\mathsf{SK}, \mathsf{C}) \rightarrow \mathsf{M}$:
  Decrypt takes as input SK and a valid ciphertext C in $\mathcal{C}$, which encrypts message M in $\mathcal{P}$, and returns M.

- $\mathsf{EvalAdd}(\mathsf{Params}, \mathsf{EK}, \mathsf{C}^{(1)}, \mathsf{C}^{(2)}) \rightarrow \mathsf{C}^{(3)}$:
  EvalAdd takes as input the encryption parameters Params, EK, two valid ciphertexts $\mathsf{C}^{(1)}$ encrypting $\mathsf{M}^{(1)}$ and $\mathsf{C}^{(2)}$ encrypting $\mathsf{M}^{(2)}$, and returns a valid ciphertext $\mathsf{C}^{(3)}$ encrypting $(\mathsf{M}^{(1)}+\mathsf{M}^{(2)})$.

- $\mathsf{EvalAddConst}(\mathsf{Params}, \mathsf{EK}, \mathsf{C}^{(1)}, \mathsf{M}^{(2)}) \rightarrow \mathsf{C}^{(3)}$:
  EvalAddConst is similar to EvalAdd but with either of the operands is a plaintext. For the input $\mathsf{C}^{(1)}$ encrypting $\mathsf{C}^{(1)}$, it outputs a ciphertext $\mathsf{C}^{(3)}$ encrypting $(\mathsf{M}^{(1)}+\mathsf{M}^{(2)})$.

- $\mathsf{EvalMult}(\mathsf{Params}, \mathsf{EK}, \mathsf{C}^{(1)}, \mathsf{C}^{(2)}) \rightarrow \mathsf{C}^{(3)}$:
  EvalMult takes as input the encryption parameters Params, EK, two valid ciphertexts $\mathsf{C}^{(1)}$ encrypting $\mathsf{M}^{(1)}$ and $\mathsf{C}^{(2)}$ encrypting $\mathsf{M}^{(2)}$, and returns a valid ciphertext $\mathsf{C}^{(3)}$ encrypting $(\mathsf{M}^{(1)}\times\mathsf{M}^{(2)})$.

- $\mathsf{EvalMultConst}(\mathsf{Params}, \mathsf{EK}, \mathsf{C}^{(1)}, \mathsf{M}^{(2)}) \rightarrow \mathsf{C}^{(3)}$:
  EvalMultConst is similar to EvalMult but with either of the operands is a plaintext. For the input $\mathsf{C}^{(1)}$ encrypting $\mathsf{M}^{(1)}$, it outputs ciphertext $\mathsf{C}^{(3)}$ encrypting $(\mathsf{M}^{(1)}\times\mathsf{M}^{(2)})$.

- $\mathsf{Relinearize}(\mathsf{Params}, \mathsf{EK}, \mathsf{C}') \rightarrow \mathsf{C}$:
  Relinearize takes as input the encryption parameters Params, EK and an ill-shaped (more on this later) C' encrypting M, and returns a well-shaped compact C encrypting M.

Later on in this note, we provide a more concrete description of these primitives alongside correctness analyses.

[1]The secret key can also be used for encryption in the symmetric mode of operation, i.e., SK is used in encryption and decryption.

3 Plaintext and Ciphertext Spaces

The plaintext and ciphertext spaces in BFV are defined over two distinct polynomial rings denoted by $\mathcal{P} = R_t = \mathbb{Z}_t[x]/(x^n+1)$ and $\mathcal{C} = R_q \times R_q$, where $R_q = \mathbb{Z}_q[x]/(x^n+1)$, $n \in \mathbb{Z}$ is the ring dimension and $t \in \mathbb{Z}$ and $q \in \mathbb{Z}$ are the plaintext and ciphertext coefficients, respectively. The notation $\mathbb{Z}_a[x]/(x^n+1)$ can be viewed as the set of polynomials with integer coefficients modulo both a and (x^n+1), i.e., with coefficients in $\{\left\lceil -\frac{a}{2} \right\rceil, \ldots, \left\lfloor \frac{a-1}{2} \right\rfloor\}$ and of degree less than n. For efficiency purposes, n is usually set as a power of 2 integer. Note that in practice, q is usually much greater than t, hence, the cardinality of $\mathcal{C}$ is much larger than that of $\mathcal{P}$, which also means a plaintext message M in $\mathcal{P}$ can be mapped to multiple valid ciphertexts in $\mathcal{C}$. The order (or number of distinct elements) in R_t and R_q is t^n and q^n, respectively. Table 1 shows Examples of valid plaintext messages for the parameters $n = 4$ and $t = 5$.

Table 1: Examples of valid plaintext messages for $(n, t) = (4, 5)$

$\mathsf{M}^{(0)}$	$1 + 2x + 1x^2 - 1x^3$
$\mathsf{M}^{(1)}$	$-1 - 2x - 1x^2 + 2x^3$
$\mathsf{M}^{(2)}$	$1 + 1x + 1x^2 + 1x^3$

4 Parameters

We describe here the main parameters used in BFV.

Besides the plaintext and ciphertext space parameters (t, q, n) introduced in Section 3, BFV uses a number of random distributions defined as follows:

- R_2: is the key distribution used to sample polynomials with integer coefficients in $\{-1, 0, 1\}$.
- $\mathcal{X}$: is the error distribution defined as a discrete Gaussian distribution with parameters μ and σ over R bounded by some integer β. According to the current version of the homomorphic encryption standard [ACC+18], (μ, σ, β) are set as $(0, \frac{8}{\sqrt{2\pi}} \approx 3.2, \lfloor 6 \cdot \sigma \rceil = 19)$.
- R_q: is a uniform random distribution over R_q.

We note that the choice of the parameters (t, q, n) is application-specific and is also driven by the desired security level. For a set of contemporary accepted parameters, we refer the reader to Tables 1 and 2 in the homomorphic encryption standard [ACC+18]. As a rule of thumb, one should opt for minimizing these parameters so long as the application of interest can still be implemented in FHE.

5 Plaintext Encoding and Decoding

Recall that the plaintext space is the polynomial ring R_t. This means that messages need to be converted to polynomials in R_t. This conversion is referred to by encoding. The literature includes plenty of encoding schemes proposed for FHE. We review here two simple schemes for integer datatypes.

Let m denote an integer message we would like to encrypt in FHE. The first encoding scheme (let's call it the naive encoding scheme) composes the plaintext element (polynomial) as: $\mathsf{M} = m + 0x + 0x^2 + \cdots + 0x^{n-1}$. I.e., we set M as a constant polynomial with m being the constant term. While this encoding scheme is simple in principle, it is extremely inefficient especially with large messages m. The reason is that while performing homomorphic operations on ciphertexts, the magnitude of plaintext coefficients grow larger. To ensure that the results of homomorphic evaluation matches the expected results of the computation of interest, we need to ensure that the plaintext coefficients do not wrap around t[2]. Thus we need to ensure that t is sufficiently larger than the inputs, any intermediate result and outputs of the computation to be implemented in FHE. The naive encoding scheme exhibits a fast growth of coefficients. We remark that this encoding scheme suffers from a more serious limitation, that is, wasting $n-1$ coefficients in the plaintext polynomial. Other more efficient encoding schemes make use of a subset or even all of the coefficients, known as packed or batched encoding schemes.

Decoding for this scheme works by simply reading the constant term of the plaintext polynomial that results from decryption.

The other encoding scheme we would like to introduce, which is known as the integer encoding scheme, works as follows:

1. represent m in binary representation $m = a_{n-1} \cdots a_2 a_1 a_0$
2. compose $\mathsf{M} = a_{n-1}x + \cdots + a_2 x^2 + a_1 x + a_0$. Since n is too large in practice, unused bits are set to 0.

Due to the smaller magnitude of plaintext coefficients, the coefficient growth during homomorphic evaluation is typically slower. Note that homomorphic evaluation with this encoding may cause the degree of the plaintext polynomial to grow. To ensure that the results of homomorphic evaluation matches the expected results of the computation of interest, we need to ensure that the degree of the plaintext coefficient does not wrap around n and the coefficients do not wrap around t.

[2]This applies to homomorphic computing via arithmetic circuits. Binary or base-p circuits might tolerate such wrap around by design.

Note that for the integer encoding scheme, one can choose any integer decomposition base other than 2 such as ternary, quaternary or k-ary base decomposition. Decoding for the integer encoding scheme works by evaluating the plaintext polynomial at k, i.e., computing $\mathsf{M}(k)$.

6 Key Generation

The secret key SK is generated as a random ternary polynomial from R_2, a polynomial of degree n with coefficients in $\{-1, 0, +1\}$.

The public key PK is a pair of polynomials $(\mathsf{PK}_1, \mathsf{PK}_2)$ calculated as follows:

$$\begin{aligned} \mathsf{PK}_1 &= [-1(a \cdot \mathsf{SK} + e)]_q \\ \mathsf{PK}_2 &= a \end{aligned} \tag{3}$$

where a is a random polynomial in R_q, and e is a random error polynomial sampled from $\mathcal{X}$. The notation $[\cdot]_q$ means that polynomial arithmetic should be done modulo q. Note that as PK_2 is in R_q, polynomial arithmetic should also be performed modulo the ring polynomial modulus $(x^n + 1)$.

7 Encryption and Decryption

To encrypt a plaintext message M in $\mathcal{P}$, one first generates three small random polynomials u from R_2 and e_1 and e_2 from $\mathcal{X}$ and returns the ciphertext $\mathsf{C} = (\mathsf{C}_1, \mathsf{C}_2)$ in $\mathcal{C}$ as follows:

$$\begin{aligned} \mathsf{C}_1 &= [\mathsf{PK}_1 \cdot u + e_1 + \Delta\mathsf{M}]_q \\ \mathsf{C}_2 &= [\mathsf{PK}_2 \cdot u + e_2]_q \end{aligned} \tag{4}$$

The parameter Δ is defined as the quotient of dividing q by t, i.e., $\Delta = \lfloor \frac{q}{t} \rfloor$. It is used to scale the message.

Decryption is performed by evaluating the ciphertext on the secret key as follows and inverting the scaling factor applied in encryption:

$$\mathsf{M} = \left[\left\lfloor \frac{t[\mathsf{C}_1 + \mathsf{C}_2 \cdot \mathsf{SK}]_q}{q} \right\rceil \right]_t \tag{5}$$

In order to check why decryption works and under which conditions, let's expand Equation (5) as follows:

$$\begin{aligned} \mathsf{C}_1 + \mathsf{C}_2 \cdot \mathsf{SK} &= \mathsf{PK}_1 \cdot u + e_1 + \Delta\mathsf{M} + (\mathsf{PK}_2 \cdot u + e_2) \cdot \mathsf{SK} \\ &= -(a \cdot \mathsf{SK} + e) \cdot u + e_1 + \Delta\mathsf{M} + a \cdot u \cdot \mathsf{SK} + e_2 \cdot \mathsf{SK} \\ &= \cancel{-a \cdot u \cdot \mathsf{SK}} - e \cdot u + e_1 + \Delta\mathsf{M} + \cancel{a \cdot u \cdot \mathsf{SK}} + e_2 \cdot \mathsf{SK} \\ &= \Delta\mathsf{M} - e \cdot u + e_1 + e_2 \cdot \mathsf{SK} \\ &= \Delta\mathsf{M} + v \end{aligned} \tag{6}$$

It should be noted that infinity norm of v, which is the largest absolute coefficient in the polynomial v denoted by $\|v\|$, is very small since e, e_1, e_2 and SK are all small polynomials. More concretely, given that these small polynomials are bounded by the parameter β, it is straightforward to show that $\|v\| \leq n\beta^2 + \beta + n\beta^2 = 2n\beta^2 + \beta$.

To carry on with the proof, we need to expand Equation (6) modulo q and complete evaluating the decryption function. This can be done as follows:

$$\mathsf{C}_1 + \mathsf{C}_2 \cdot \mathsf{SK} = \Delta\mathsf{M} + v + q \cdot r \tag{7}$$

where r is some polynomial. Scaling Equation (7) by $\frac{t}{q}$ results in $\mathsf{M} + \frac{t}{q} \cdot v + t \cdot r$. The rounding function in decryption removes $\frac{t}{q} \cdot v$ and the final modulo t operation removes $t \cdot r$ and recovers M. In short, for decryption to recover M correctly, we need to ensure that $\frac{t}{q} \cdot \|v\| < \frac{1}{2}$, otherwise the noise will overflow and destroy the message.

8 Homomorphic Evaluation

In this section, we explain how homomorphic evaluation procedures work. Much of this section is based on the correctness analysis that can be found in the proposal of BFV [FV12].

8.1 EvalAdd

Let's take EvalAdd as a reference to understand how homomorphic addition works. This procedure is quite simple, we just add the corresponding polynomials in each ciphertext as shown in Equation (8).

$$\mathsf{EvalAdd}(\mathsf{C}^{(1)}, \mathsf{C}^{(2)}) = ([\mathsf{C}_1^{(1)} + \mathsf{C}_1^{(2)}]_q, [\mathsf{C}_2^{(1)} + \mathsf{C}_2^{(2)}]_q) = (\mathsf{C}_1^{(3)}, \mathsf{C}_2^{(3)}) = \mathsf{C}^{(3)} \tag{8}$$

In order to see why this works, let's break Equation (8) as follows: Assume that C^1 and C^2 are fresh encryptions of M^1 and M^2. Algebraically, they can be expressed as follows (See Equation (4)):

$$\begin{aligned}\mathsf{C}^{(1)} &= ([\mathsf{PK}_1 \cdot u^{(1)} + e_1^{(1)} + \Delta\mathsf{M}^{(1)}]_q, [\mathsf{PK}_2 \cdot u^{(1)} + e_2^{(1)}]_q) \\ \mathsf{C}^{(2)} &= ([\mathsf{PK}_1 \cdot u^{(2)} + e_1^{(2)} + \Delta\mathsf{M}^{(2)}]_q, [\mathsf{PK}_2 \cdot u^{(2)} + e_2^{(2)}]_q)\end{aligned} \tag{9}$$

By substituting the $\mathsf{C}^{(1)}$ and $\mathsf{C}^{(2)}$ in Equation (8) we get the following:

$$\begin{aligned}\mathsf{C}^{(3)} &= (\mathsf{C}_1^{(3)}, \mathsf{C}_2^{(3)}) && (10)\\ &= ([\mathsf{PK}_1 \cdot (u^{(1)} + u^{(2)}) + (e_1^{(1)} + e_1^{(2)}) + \Delta(\mathsf{M}^{(1)} + \mathsf{M}^{(2)})]_q, \\ &\quad [\mathsf{PK}_2 \cdot (u^{(1)} + u^{(2)}) + (e_2^{(1)} + e_2^{(2)})]_q) \\ &= ([\mathsf{PK}_1 \cdot u^{(3)} + e_1^{(3)} + \Delta(\mathsf{M}^{(1)} + \mathsf{M}^{(2)})]_q, [\mathsf{PK}_2 \cdot u^{(3)} + e_2^{(3)}]_q) && (11)\end{aligned}$$

It is straightforward to notice that Equation (11) has the form of a valid ciphertext encrypting $\mathsf{M}^{(3)} = \mathsf{M}^{(1)} + \mathsf{M}^{(2)}$. Note that the error term in $\mathsf{C}^{(3)}$ is approximately, following a worst-case scenario analysis, the sum of noise terms in the input ciphertexts, i.e., the noise grows additively.

One can follow the procedure used above to derive the expression for EvalAddPlain, the plaintext message can be transformed to a ciphertext form by encryption with no error terms, i.e., $e_1 = e_2 = 0$.

8.2 EvalMult

Homomorphic multiplication is more complex compared to homomorphic addition. We present here the basic procedure and refer the reader to external resources for further details.

It is useful to write the ciphertext as an evaluation at SK similar to what we did in the derivation of decryption:

$$\begin{aligned}\mathsf{C}^{(1)}(\mathsf{SK}) &= \Delta\mathsf{M}^{(1)} + v_1 + q \cdot r_1 \\ \mathsf{C}^{(2)}(\mathsf{SK}) &= \Delta\mathsf{M}^{(2)} + v_2 + q \cdot r_2\end{aligned} \tag{12}$$

Multiplying the ciphertexts would give us:

$$\begin{aligned}(\mathsf{C}^{(1)} \cdot \mathsf{C}^{(2)})(\mathsf{SK}) =& \Delta^2\mathsf{M}^{(1)} \cdot \mathsf{M}^{(2)} + \Delta(\mathsf{M}^{(1)} \cdot v_2 + \mathsf{M}^{(2)} \cdot v_1) + \\ & q(v_1 \cdot r_2 + v_2 \cdot r_1) + q \cdot \Delta(\mathsf{M}^{(1)} \cdot r_2 + \mathsf{M}^{(2)} \cdot r_1) + \\ & v_1 \cdot v_2 + q^2 \cdot r_1 \cdot r_2\end{aligned} \tag{13}$$

The product ciphertext looks close to an encryption of what we want $\Delta \cdot [\mathsf{M}^{(1)} \cdot \mathsf{M}^{(2)}]_t$. We notice that scaling by $\frac{1}{\Delta}$ gives us exactly what we want in the first term plus some noise. However, the last term $(q^2 \cdot r_1 \cdot r_2)$ would generate large noise since q^2 does not divide Δ. Instead, we would scale by the factor $\frac{t}{q}$. Now, we can write $\mathsf{M}^{(1)} \cdot \mathsf{M}^{(2)} = [\mathsf{M}^{(1)} \cdot \mathsf{M}^{(2)}]_t + t \cdot r_M$. We can also write $v_1 \cdot v_2 = [v_1 \cdot v_2]_\Delta + \Delta \cdot r_v$. Scaling by $\frac{t}{q}$ and substituting these expressions in Equation (13), we obtain the following:

$$\begin{aligned}\frac{t}{q}(\mathsf{C}^{(1)} \cdot \mathsf{C}^{(2)})(\mathsf{SK}) =& \Delta[\mathsf{M}^{(1)} \cdot \mathsf{M}^{(2)}]_t + (\mathsf{M}^{(1)} \cdot v_2 + \mathsf{M}^{(2)} \cdot v_1) + \\ & t(v_1 \cdot r_2 + v_2 \cdot r_1) + r_v + (q - [q]_t) \cdot (r_M + \mathsf{M}^{(1)} \cdot r_2 + \mathsf{M}^{(2)} \cdot r_1) + \\ & q \cdot t \cdot r_1 \cdot r_2 + \frac{t}{q}[v_1 \cdot v_2]_\Delta - \\ & \frac{[q]_t}{q}(\Delta \mathsf{M}^{(1)} \cdot \mathsf{M}^{(2)} + \mathsf{M}^{(1)} \cdot v_2 + \mathsf{M}^{(2)} \cdot v_1 + r_v)\end{aligned} \tag{14}$$

The final step in the derivation is reducing Equation (14) modulo q, which gives us:

$$\begin{aligned}\frac{t}{q}(\mathsf{C}^{(1)} \cdot \mathsf{C}^{(2)})(\mathsf{SK}) =& \Delta[\mathsf{M}^{(1)} \cdot \mathsf{M}^{(2)}]_t + (\mathsf{M}^{(1)} \cdot v_2 + \mathsf{M}^{(2)} \cdot v_1) + \\ & t(v_1 \cdot r_2 + v_2 \cdot r_1) + r_v - \\ & [q]_t(r_M + \mathsf{M}^{(1)} \cdot r_2 + \mathsf{M}^{(2)} \cdot r_1) + r_e\end{aligned} \tag{15}$$

, where r_e is the rounding error generated from the last two terms in Equation (14).

The noise growth for multiplication grows by a linear factor that is approximately $2 \cdot t \cdot n^2 \cdot \|SK\|$, i.e., $\|v_p\| = \|v_i\| \cdot 2 \cdot t \cdot n^2 \cdot \|SK\|$, where v_p is the noise in the product ciphertext, and v_i is the noise in the input ciphertexts.

In short, to evaluate EvalMult, we compute the tensor product of the input ciphertexts and scale by $\frac{t}{q}$ as follows:

$$\begin{aligned}\mathsf{EvalMult}(\mathsf{C}^{(1)}, \mathsf{C}^{(2)}) = \Bigg(& \left[\left\lfloor \frac{t(\mathsf{C}_1^{(1)} \cdot \mathsf{C}_1^{(2)})}{q} \right\rceil\right]_q, \left[\left\lfloor \frac{t(\mathsf{C}_1^{(1)} \cdot \mathsf{C}_2^{(2)} + \mathsf{C}_2^{(1)} \cdot \mathsf{C}_1^{(2)})}{q} \right\rceil\right]_q, \\ & \left[\left\lfloor \frac{t(\mathsf{C}_2^{(1)} \cdot \mathsf{C}_2^{(2)})}{q} \right\rceil\right]_q \Bigg)\end{aligned} \tag{16}$$

It can be noticed that EvalMult increases the number of terms in the resulting ciphertext from 2 to 3 ring elements. Moreover, in order to decrypt the resulting ciphertext, a slightly different decryption procedure has to be used. Fortunately, these complications can be resolved via Relinearization which will be described in the subsequent section.

9 Relinearization

The main purpose of this procedure is to reduce the size of product ciphertexts, those resulting from EvalMult, back to 2 ring elements. This problem can be formulated concretely as follows: Let $\mathsf{C}^* = \{\mathsf{C}_1^*, \mathsf{C}_2^*, \mathsf{C}_3^*\}$. Our goal is to find $\hat{\mathsf{C}}^* = \{\hat{\mathsf{C}}_1^*, \hat{\mathsf{C}}_2^*\}$ such that:

$$[\mathsf{C}_1^* + \mathsf{C}_2^* \cdot \mathsf{SK} + \mathsf{C}_3^* \cdot \mathsf{SK}^2]_q \approx [\hat{\mathsf{C}}_1^* + \hat{\mathsf{C}}_2^* \cdot \mathsf{SK} + r]_q \tag{17}$$

Access to SK^2 is provided by means of the evaluation key $\mathsf{EK} = (-(a \cdot \mathsf{SK} + e) + \mathsf{SK}^2, a)$. Note that this is a masked version of SK^2 since $\mathsf{EK}_1 + \mathsf{EK}_2 \cdot \mathsf{SK} = \mathsf{SK}^2 - e$. Now we can compute $\hat{\mathsf{C}}^*$ as follows:

$$\begin{aligned} \hat{\mathsf{C}}_1^* &= [\mathsf{C}_1^* + \mathsf{EK}_1 \cdot \mathsf{C}_3^*]_q \\ \hat{\mathsf{C}}_2^* &= [\mathsf{C}_2^* + \mathsf{EK}_2 \cdot \mathsf{C}_3^*]_q \end{aligned} \tag{18}$$

If we write the decryption formula for Equation (18) we obtain:

$$\begin{aligned} \hat{\mathsf{C}}_1^* + \hat{\mathsf{C}}_2^* \cdot \mathsf{SK} &= \mathsf{C}_1^* + \mathsf{EK}_1 \cdot \mathsf{C}_3^* + \mathsf{SK} \cdot (\mathsf{C}_2^* + \mathsf{EK}_2 \cdot \mathsf{C}_3^*) \\ &= \mathsf{C}_1^* + \mathsf{C}_2^* \cdot \mathsf{SK} + \mathsf{C}_3^* \cdot (\mathsf{EK}_1 + \mathsf{EK2} \cdot \mathsf{SK}) \\ &= \mathsf{C}_1^* + \mathsf{C}_2^* \cdot \mathsf{SK} + \mathsf{C}_3^* \cdot \mathsf{SK}^2 + \mathsf{C}_3^* \cdot e \end{aligned} \tag{19}$$

It should be noted that the term $\mathsf{C}_3^* \cdot e$ can be quite large as C_3^* has coefficients in $\mathbb{Z}_q$. Nevertheless, there is a technique to reduce this error using base decomposition. For further details, we refer the reader to [FV12].

10 Security of the Scheme

Analyzing the security of the BFV scheme is quite complex and beyond the scope of this article. Choosing optimal BFV parameters that maximize performance and respect security and functionality constraints is an art that is practiced by expert cryptographers. That said, for a brief security analysis and a set of recommended parameters for BFV (and other FHE schemes), we refer the reader to the homomorphic encryption standard [ACC+18].

References

[ACC+18] Martin Albrecht, Melissa Chase, Hao Chen, Jintai Ding, Shafi Goldwasser, Sergey Gorbunov, Shai Halevi, Jeffrey Hoffstein, Kim Laine, Kristin Lauter, Satya Lokam, Daniele Micciancio, Dustin Moody, Travis Morrison, Amit Sahai, and Vinod Vaikuntanathan. Homomorphic encryption security standard. Technical report, HomomorphicEncryption.org, Toronto, Canada, November 2018.

[Bra12] Zvika Brakerski. Fully homomorphic encryption without modulus switching from classical gapsvp. In *Annual Cryptology Conference*, pages 868–886. Springer, 2012.

[FV12] Junfeng Fan and Frederik Vercauteren. Somewhat practical fully homomorphic encryption. 2012. https://eprint.iacr.org/2012/144.

[LPR13] Vadim Lyubashevsky, Chris Peikert, and Oded Regev. On ideal lattices and learning with errors over rings. *Journal of the ACM (JACM)*, 60(6):1–35, 2013.