BFV Homomorphic Encryption Part 1 - The Original Scheme

Notes by Linus Tang.

These notes have not been thoroughly reviewed. Any errors below are my own responsibility.

Sources:
- The original scheme by (Zvika Brakerski), Junfeng Fan, Frederik Vercauteren:
  - https://eprint.iacr.org/2012/144.pdf
- An improvement by Hao Chen and Kyoohyung Han:
  - https://eprint.iacr.org/2018/067.pdf

Assumed background knowledge:
- Know what (fully) homomorphic encryption is
  - https://en.wikipedia.org/wiki/Homomorphic_encryption
  - Familiarity with at least one FHE scheme may help but is not required
  - Then again, I think BFV is a reasonable first FHE scheme to learn.
- Some familiarity with polynomial rings

What these notes cover:

This is meant to be an intuitive and explanation of the BFV homomorphic encryption scheme as described in the original 2012 paper. We show how to encrypt and decrypt ciphertexts and perform operations homomorphically, and go into detail about bootstrapping, the method of noise reduction. We hint at a faster method of bootstrapping proposed in 2018.

Details about choosing parameters are outside the scope of these notes. We also don't always give specific details about the noise analysis, and merely point out when coefficients are "small" compared to the noise tolerance.

Contents

BFV Homomorphic Encryption Part 1 - The Original Scheme 1
1. Notation 1
2. Key Generation, Encryption, and Decryption 2
3. Evaluation and Decryption for Leveled FHE 3
Adding two ciphertexts 3
Multiplying two ciphertexts 3
Adding and multiplying by constants 4
Recap + correctness guarantees and noise bounds 4
4. Bootstrapping 5
General framework 5
Preprocessing 6
Refined framework 6
Discussion of multiplicative depth 7
Designing the decryption circuit, part 1: analyzing rounding and noise 7
Designing the decryption circuit, part 2: binary addition 9
5. Hint at Better Bootstrapping [CH'18] 10

1. Notation

- The *user* generates keys and ciphertexts of private inputs and the *server* computes on the ciphertexts.
- We will use sans letters to denote keys, ciphertexts, and their components, which are polynomials. We use bold letters to denote polynomials that are not components of keys or ciphertexts.
- Important: We will use $\mathbb{Z}_a$ to denote the set of a integers lying in $(-\frac{a}{2}, \frac{a}{2}]$, not the ring $\mathbb{Z}/a\mathbb{Z}$.

- For any integer r, we let $[r]_a$ denote the residue of r belonging to $\mathbb{Z}_a$. For a polynomial $\boldsymbol{r}$, we let $[\boldsymbol{r}]_a$ denote the same operation applied coefficient-wise.
- We will see later that arithmetic on messages is in $(\mathbb{Z}/t\mathbb{Z})[x]/(x^N+1)$ but analyzing the correctness of arithmetic on ciphertexts cannot be done entirely in $(\mathbb{Z}/q\mathbb{Z})[x]/(x^N+1)$.
- For real r we will let $\lfloor r \rceil$ denote r rounded to the nearest integer (this operation will typically be applied to a quotient in order to denote more general rounding). Similarly, this operation also applies coefficient-wise to polynomials.

- Let N be a power of 2 and R denote the ring $\mathbb{Z}[x]/(x^N+1)$ and R_a denote the set of polynomials in R whose coefficients lie in $\mathbb{Z}_a$ (when written with degree $< N$).
- We will use "$=$" to denote equality in R and "$\equiv$" to denote equality in $(\mathbb{Z}/q\mathbb{Z})[x]/(x^N+1)$.

2. Key Generation, Encryption, and Decryption

- Let t, q be powers of 2 and $\Delta = q/t$ so that $t \ll \Delta$. We will see later than Δ is, in a sense, a level of noise tolerance.[1]
- Messages $\boldsymbol{M}$ lie in R_t but are multiplied by scaling factor Δ to raise them to R_q.
- The secret key $\mathsf{SK} = s_0 x^{N-1} + \cdots + s_{N-2}x + s_{N-1}$ is a random element of R_2, i.e. a polynomial whose coefficients are chosen from $\{0, 1\}$.
- The public key is

$$(\mathsf{PK}_1, \mathsf{PK}_2) = \big([-\boldsymbol{a} \cdot \mathsf{SK} + \boldsymbol{e}]_q, [\boldsymbol{a}]_q\big),$$

  where $\boldsymbol{a}$ is a random polynomial in R_q and $\boldsymbol{e}$ is a polynomial in R whose coefficients are chosen iid from some distribution of χ over small integers (such as a bounded Gaussian over integers).
  - The Ring Learning With Errors (RLWE) assumption states that efficient adversaries (without knowledge of SK) cannot distinguish $(\mathsf{PK}_1, \mathsf{PK}_2)$ as defined above from a two independent uniform random draws from R_q. The security of the BFV scheme relies on the RLWE assumption.
  - Formal security proofs are outside the scope of these notes, but a helpful intuition is that if $(\mathsf{PK}_1, \mathsf{PK}_2)$ looks indistinguishable from uniform randomness to the server, then the public key gives the server no information about the underlying secret key SK.
- A ciphertext of a message $\boldsymbol{M}$ is

$$(\mathsf{C}_1, \mathsf{C}_2) = \Big([\mathsf{PK}_1 \cdot \boldsymbol{u} + \boldsymbol{e}_1 + \Delta \boldsymbol{M}]_q, [\mathsf{PK}_2 \cdot \boldsymbol{u} + \boldsymbol{e_2}]_q\Big),$$

  where $\boldsymbol{u}, \boldsymbol{e}_1, \boldsymbol{e}_2$ are random polynomials in R_2 with coefficients drawn from χ.
- The key property that will be used for decryption is that over $(\mathbb{Z}/q\mathbb{Z})[x]/(x^N+1)$,

$$\begin{aligned}\mathsf{C}_+ &= \mathsf{C}_1 + \mathsf{C}_2 \cdot \mathsf{SK} \\ &\equiv \Delta \boldsymbol{M} + \boldsymbol{e} \cdot \boldsymbol{u} + \boldsymbol{e}_1 + \boldsymbol{e}_2 \cdot \mathsf{SK} \\ &= \Delta \boldsymbol{M} + \boldsymbol{v},\end{aligned}$$

  for some polynomial $\boldsymbol{v}$ with small coefficients. In fact, a more careful analysis shows that

$$\mathsf{C}_+ = \Delta \boldsymbol{M} + \boldsymbol{v} + q\boldsymbol{r}$$

  where the coefficients of $\boldsymbol{r}$ are also small.

  Thus, so long as we can control the noise v to have coefficients $< \Delta/2$ in magnitude, a holder of the secret key can recover the message $\boldsymbol{M} = \left[\left\lfloor \frac{\mathsf{C}_1 + \mathsf{C}_2 \cdot \mathsf{SK}}{\Delta} \right\rceil\right]_t$.

[1] There is a more general version of the scheme in which q and t need not be powers of 2, and t need not even divide q. See the original paper for details. In these notes we cover the specific case where t and q are powers of 2 because it is easiest to understand and has the most efficient implementation.

- There is also a public evaluation key EK and a public bootstrapping key BSK, each of which will be discussed later when it can be better motivated.

3. Evaluation and Decryption for Leveled FHE

Recall that messages $\boldsymbol{M}$ lie in the message space R_t. The operations over ciphertexts described below correspond to operations on messages over $(\mathbb{Z}/t\mathbb{Z})[x]/(x^N+1)$.

Adding two ciphertexts

Suppose ciphertexts $\left(\mathsf{C}_1^{(1)}, \mathsf{C}_2^{(1)}\right)$ and $\left(\mathsf{C}_1^{(2)}, \mathsf{C}_2^{(2)}\right)$ encoding $\boldsymbol{M}^{(1)}$ and $\boldsymbol{M}^{(2)}$. If we want a ciphertext that encodes $\left[\boldsymbol{M}^{(1)} + \boldsymbol{M}^{(2)}\right]_t$, we add the ciphertexts component-wise, i.e. we compute

$$\left(\mathsf{C}_1^{(1)} + \mathsf{C}_1^{(2)}, \mathsf{C}_2^{(1)} + \mathsf{C}_2^{(2)}\right),$$

which encrypts $\left[\boldsymbol{M}^{(1)} + \boldsymbol{M}^{(2)}\right]_t$ with slightly increased noise.

Multiplying two ciphertexts

Say we have two messages $\boldsymbol{M}^{(1)}$ and $\boldsymbol{M}^{(2)}$ with ciphertexts $\left(\mathsf{C}_1^{(1)}, \mathsf{C}_2^{(1)}\right)$ and $\left(\mathsf{C}_1^{(2)}, \mathsf{C}_2^{(2)}\right)$.

Define $\mathsf{C}_+^{(1)} = \mathsf{C}_1^{(1)} + \mathsf{C}_2^{(1)} \cdot \mathsf{SK} = \Delta\boldsymbol{M}^{(1)} + \boldsymbol{v}^{(1)} + q\boldsymbol{r}$ and define $\mathsf{C}_+^{(2)}$, $\boldsymbol{v}^{(2)}$, and $\boldsymbol{r}^{(2)}$ similarly, so that the $\boldsymbol{v}^{(i)}$ and $\boldsymbol{r}^{(i)}$ are small.

Then over $(\mathbb{Z}/q\mathbb{Z})[x]/(x^N+1)$ we have

$$\begin{aligned}
&\left(\mathsf{C}_1^{(1)} + \mathsf{C}_2^{(1)} \cdot \mathsf{SK}\right)\left(\mathsf{C}_1^{(2)} + \mathsf{C}_2^{(2)} \cdot \mathsf{SK}\right) = \mathsf{C}_+^{(1)}\mathsf{C}_+^{(2)} \\
&= \Delta\boldsymbol{M}^{(1)} + \boldsymbol{v}^{(1)} + q\boldsymbol{r} \\
&= \Delta^2\boldsymbol{M}_1\boldsymbol{M}_2 + \Delta\big(\boldsymbol{M}_1\boldsymbol{v}^{(2)} + \boldsymbol{M}_2\boldsymbol{v}^{(1)}\big) + \boldsymbol{v}^{(1)}\boldsymbol{v}^{(2)} + q\big(\boldsymbol{v}^{(1)}\boldsymbol{r}^{(2)} + \boldsymbol{v}^{(2)}\boldsymbol{r}^{(1)} + \Delta(\boldsymbol{r})\big).
\end{aligned}$$

for some polynomial $\boldsymbol{r}$.

We expand the left side of the above, divide by Δ, and round, getting

$$\left\lfloor \frac{\mathsf{C}_1^{(1)}\mathsf{C}_1^{(2)}}{\Delta} \right\rceil + \left\lfloor \frac{\mathsf{C}_1^{(1)}\mathsf{C}_2^{(2)} + \mathsf{C}_2^{(1)}\mathsf{C}_1^{(2)}}{\Delta} \right\rceil \cdot \mathsf{SK} + \left\lfloor \frac{\mathsf{C}_2^{(1)}\mathsf{C}_2^{(2)}}{\Delta} \right\rceil \cdot \mathsf{SK}^2 \equiv \Delta\boldsymbol{M}_1\boldsymbol{M}_2 + \boldsymbol{v}^{(3)}$$

where the error $\boldsymbol{v}^{(3)}$ is larger than $\boldsymbol{v}^{(1)}$ and $\boldsymbol{v}^{(2)}$, but "not by too much" (for an explicit bound, see the recap below or Lemma 2 in the original paper). The division by Δ cannot be done over $(\mathbb{Z}/q\mathbb{Z})[x]/(x^N+1)$, which is why we had to keep track of the terms that are divisible by q but not by $q\Delta$.

We can imagine passing

$$(\mathsf{C}_1', \mathsf{C}_2', \mathsf{C}_3') = \left(\left[\left\lfloor \frac{\mathsf{C}_1^{(1)}\mathsf{C}_1^{(2)}}{\Delta} \right\rceil\right]_q, \left[\left\lfloor \frac{\mathsf{C}_1^{(1)}\mathsf{C}_2^{(2)} + \mathsf{C}_2^{(1)}\mathsf{C}_1^{(2)}}{\Delta} \right\rceil\right]_q, \left[\left\lfloor \frac{\mathsf{C}_2^{(1)}\mathsf{C}_2^{(2)}}{\Delta} \right\rceil\right]_q \right)$$

to back to the user, who can compute $\left[\mathsf{C}_1' + \mathsf{C}_2' \cdot \mathsf{SK} + \mathsf{C}_3' \cdot \mathsf{SK}^2\right]_q$ to get a noisy version of $[\Delta\boldsymbol{M}_1\boldsymbol{M}_2]_q$. Provided the noise is less than $\Delta/2$ the user could divide by Δ and round to get the product $[\boldsymbol{M}_1\boldsymbol{M}_2]_t$.

However, our ciphertext format has changed, and we now have 3 polynomials instead of 2. If we want to compose multiplications, we must get the ciphertext format back to the original (or generalize the multiplication protocol and allow the ciphertext size to grow, but this is impractical for deep circuits). This is called relinearization, and the main idea is for the user to provide an approximate quadratic

equation in SK (encrypted so that the server can't learn SK), which the server can then use to convert the third component into the first two.

In particular, the user provides the server with an evaluation key

$$(\mathsf{EK}_1, \mathsf{EK}_2) = \left(\left[-(\boldsymbol{a} \cdot \mathsf{SK} + \boldsymbol{e}) + p \cdot \mathsf{SK}^2 \right]_{p \cdot q}, \boldsymbol{a} \right)$$

for some parameter p, with $\boldsymbol{a}$ sampled from $R_{p \cdot q}$ and the noise $\boldsymbol{e}$ sampled from χ.

You can think of the evaluation key as an encryption of SK^2 (with scaling factor p instead of Δ), in the sense that $\mathsf{EK}_1 + \mathsf{EK}_2 \cdot \mathsf{SK}$ is close to $p\mathsf{SK}^2$.

Given this encryption of SK^2, the server can compute

$$\left(\left[\mathsf{C}'_1 + \left\lfloor \frac{\mathsf{C}'_3 \mathsf{EK}_1}{p} \right\rceil \right]_q, \left[\mathsf{C}'_1 + \left\lfloor \frac{\mathsf{C}'_3 \mathsf{EK}_2}{p} \right\rceil \right]_q \right),$$

which decrypts to $\boldsymbol{M}_1 \boldsymbol{M}_2$ with slightly more noise than $(\mathsf{C}'_1, \mathsf{C}'_2, \mathsf{C}'_3)$. So, we've successfully reduced the ciphertext back to the original two-polynomial format.

Adding and multiplying by constants

If instead of having ciphertexts for both $\boldsymbol{M}^{(1)}$ and $\boldsymbol{M}^{(2)}$ we have a ciphertext $\mathsf{C}^{(1)}$ for $\boldsymbol{M}^{(1)}$ and direct access to $\boldsymbol{M}^{(2)}$ (e.g. if $\boldsymbol{M}^{(2)}$ is a public input), then we can compute a ciphertext for $\boldsymbol{M}^{(1)} + \boldsymbol{M}^{(2)}$ by first generating a ciphertext $\mathsf{C}^{(2)}$ for $\boldsymbol{M}^{(2)}$, then performing addition of two ciphertexts $\mathsf{C}^{(1)}$ and $\mathsf{C}^{(2)}$.

Our ciphertext $\mathsf{C}^{(2)}$ for $\boldsymbol{M}^{(2)}$ need not have noise because $\boldsymbol{M}^{(2)}$ is public; in particular, we can simply set $\mathsf{C}^{(2)} = (\Delta \boldsymbol{M}^{(2)}, 0)$.

The same applies to multiplying a ciphertext by a public input.

Recap + correctness guarantees and noise bounds

Here we recap our scheme so far and write the correctness guarantees and noise bounds of our homomorphic operations:

- The user generates a secret key SK which is a random polynomial in R_2.
- The user generates a public key $(\mathsf{PK}_1, \mathsf{PK}_2) = \left([-\boldsymbol{a} \cdot \mathsf{SK} + \boldsymbol{e}]_q, [\boldsymbol{a}]_q \right)$ where $\boldsymbol{a}$ is a random polynomial in R_q and $\boldsymbol{e}$ is small noise.
- The user chooses parameter p not too far from q and generates an evaluation key

  $$(\mathsf{EK}_1, \mathsf{EK}_2) = \left(\left[-(\boldsymbol{a}_1 \cdot \mathsf{SK} + \boldsymbol{e}) + p \cdot \mathsf{SK}^2 \right]_{p \cdot q}, \boldsymbol{a}_1 \right)$$

  where $\boldsymbol{a}_1$ is sampled uniformly from R_{pq} and $\boldsymbol{e}$ is small noise.
- The user sends the public key and evaluation key to the server.
- To encrypt a message $\boldsymbol{M} \in R_t$, the user generates a polynomial $\boldsymbol{u} \in R$ with small coefficients and two small noise polynomials $\boldsymbol{e}_1$ and $\boldsymbol{e}_2$. The ciphertext is

  $$(\mathsf{C}_1, \mathsf{C}_2) = \left([\mathsf{PK}_1 \cdot \boldsymbol{u} + \boldsymbol{e}_1 + \Delta \boldsymbol{M}]_q, [\mathsf{PK}_2 \cdot \boldsymbol{u} + \boldsymbol{e_2}]_q \right).$$

  - The random $\boldsymbol{u}, \boldsymbol{e}_1, \boldsymbol{e}_2$ are for security purposes only, so if the server is creating a ciphertext for a public message $\boldsymbol{M}$ (which is done before adding or multiplying a ciphertext to a public input $\boldsymbol{M}$), the server can simply set all of them to 0 to get the noiseless ciphertext $(\Delta \boldsymbol{M}, 0)$.
- To decrypt a ciphertext $(\mathsf{C}_1, \mathsf{C}_2)$, compute

$$\boldsymbol{M} = \left[\left\lfloor \frac{\mathsf{C}_1 + \mathsf{C}_2 \cdot \mathsf{SK}}{\Delta} \right\rceil\right]_t.$$

- We say that $(\mathsf{C}_1, \mathsf{C}_2)$ encodes $\boldsymbol{M}$ with noise at most E if

$$\mathsf{C}_1 + \mathsf{C}_2 \cdot \mathsf{SK} \equiv \Delta \boldsymbol{M} + \boldsymbol{v}$$

  for some $\boldsymbol{v}$ whose coefficients are all at most E. Roughly speaking, decryption yields $[\boldsymbol{M}]_t$ if noise is at most $\Delta/2$.
- Adding two ciphertexts:
  - If $\left(\mathsf{C}_1^{(1)}, \mathsf{C}_2^{(1)}\right)$ and $\left(\mathsf{C}_1^{(2)}, \mathsf{C}_2^{(2)}\right)$ are ciphertexts encoding $\boldsymbol{M}^{(1)}$ and $\boldsymbol{M}^{(2)}$ with noise at most $E^{(1)}$ and $E^{(2)}$, the server computes

$$\left(\mathsf{C}_1^{(1)} + \mathsf{C}_1^{(2)}, \mathsf{C}_2^{(1)} + \mathsf{C}_2^{(2)}\right),$$

    which encrypts $\left[\boldsymbol{M}^{(1)} + \boldsymbol{M}^{(2)}\right]_t$ with noise at most $E^{(1)} + E^{(2)} + t$.
- Multiplying two ciphertexts:
  - If $\left(\mathsf{C}_1^{(1)}, \mathsf{C}_2^{(1)}\right)$ and $\left(\mathsf{C}_1^{(2)}, \mathsf{C}_2^{(2)}\right)$ are ciphertexts encoding $\boldsymbol{M}^{(1)}$ and $\boldsymbol{M}^{(2)}$ with noise at most $E^{(1)}$ and $E^{(2)}$, the server computes

$$(\mathsf{C}_1', \mathsf{C}_2', \mathsf{C}_3') = \left(\left[\left\lfloor \frac{\mathsf{C}_1^{(1)}\mathsf{C}_1^{(2)}}{\Delta} \right\rceil\right]_q, \left[\left\lfloor \frac{\mathsf{C}_1^{(1)}\mathsf{C}_2^{(2)} + \mathsf{C}_2^{(1)}\mathsf{C}_1^{(2)}}{\Delta} \right\rceil\right]_q, \left[\left\lfloor \frac{\mathsf{C}_2^{(1)}\mathsf{C}_2^{(2)}}{\Delta} \right\rceil\right]_q\right)$$

    and relinearize to

$$\left(\left[\mathsf{C}_1' + \left\lfloor \frac{\mathsf{C}_3'\mathsf{EK}_1}{p} \right\rceil\right]_q, \left[\mathsf{C}_1' + \left\lfloor \frac{\mathsf{C}_3'\mathsf{EK}_2}{p} \right\rceil\right]_q\right).$$

    This new ciphertext encrypts $\left[\boldsymbol{M}^1 + \boldsymbol{M}^2\right]_t$ with noise $O\big(N^2 t\big(E^{(1)} + E^{(2)}\big)\big)$.

    (As we can see by this bound, Δ needs to be huge compared to N and t in order to tolerate the noise of stringing together several multiplications.)

4. Bootstrapping

All logarithms below are base 2. **Recall that q, t, N are powers of 2 and $\Delta = q/t$.**

Above, we have achieved levelled homomorphic encryption: we can homomorphically evaluate circuits of bounded depth before the worst-case noise exceeds $\Delta/2$ and risks corrupting the message. In order to achieve fully homomorphic encryption, we need a way to take a ciphertext and reduce its noise to a constant level.

General framework

A general approach for bootstrapping in FHE is to run the decryption circuit homomorphically. To spell it out, we have a ciphertext $(\mathsf{C}_1, \mathsf{C}_2)$ encrypting $\boldsymbol{M}$ with high noise (but still considerably less than $\Delta/2$). We want to refresh the noise, i.e. obtain a ciphertext $\mathsf{C}' = (\mathsf{C}_1', \mathsf{C}_2')$ of $\boldsymbol{M}$ with a constant amount of noise. To do this, we write the decryption function

$$(\mathsf{C}_1, \mathsf{C}_2, \mathsf{SK}) \mapsto \left[\left\lfloor \frac{\mathsf{C}_1 + \mathsf{C}_2 \cdot \mathsf{SK}}{\Delta} \right\rceil\right]_t$$

as a circuit $\mathcal{C}$ whose gates are additions and multiplications. Then if we have ciphertexts of C_1, C_2, and SK (yes, ciphertexts for ciphertexts and a ciphertext of the secret key), we can evaluate the circuit $\mathcal{C}$ homomorphically to get a ciphertext C' for $\boldsymbol{M} = \left[\left\lfloor \frac{\mathsf{C}_1 + \mathsf{C}_2 \cdot \mathsf{SK}}{\Delta} \right\rceil\right]_t$. Furthermore, the noise in this new ciphertext will only come from the computations that are part of the circuit $\mathcal{C}$.

Preprocessing

A ciphertext of SK will be provided to the server, and we call it the bootstrapping key BSK.

Actually, we will want to preprocess the ingredients C_1, C_2, and SK before we obtain ciphertexts for them, for two reasons:

- Since C_1 and C_2 are members of R_q whereas the message space is R_t, we can't encrypt each of $\mathsf{C}_1, \mathsf{C}_2$ in one piece.
- It turns out to be difficult to write a circuit $\mathcal{C}$ whose inputs are C_1, C_2, and SK and whose gates are addition and multiplication that computes the decryption function (we run into trouble with the "divide by Δ and round" step). Instead, we preprocess these ingredients before encrypting them so that the circuit $\mathcal{C}$ can take the processed versions as inputs instead.

In the case of the original BFV scheme, this preprocessing consists of breaking S into its coefficients, rounding the coefficients of C_1 and C_2 to an appropriate amount of precision, and encrypting individual bits of coefficients instead of entire coefficients or entire polynomials.

Specifically:

- Let K_i be the X^i coefficient SK. Recall that $\mathsf{SK} \in R_2$, so the coefficients are binary; $K_i \in \{0, 1\}$.
- We round each coefficient of C_1 and of C_2 to the nearest multiple of $\frac{\Delta}{2N}$. We will see later why this rounding still offers enough precision to carry out the computation, assuming that $\mathsf{C} = (\mathsf{C}_1, \mathsf{C}_2)$ encrypts $\boldsymbol{M}$ with not too much noise.
  - Let $C_{1,i}$ be the X^i coefficient of C_1.
  - Let $\tilde{C}_{1,i} = \left\lfloor \frac{2N\mathsf{C}_{1,i}}{\Delta} \right\rceil$.
  - Define $C_{2,i}$ and $\tilde{C}_{2,i}$ similarly.
- Note that we only care about C_1 and C_2 mod q. Consequently, after dividing and rounding, we only care about $\tilde{C}_{1,i}$ and $\tilde{C}_{2,i}$ mod $\frac{2Nq}{\Delta} = 2Nt$. So, each can be represented by a k-bit integer with $k = \log(2Nt)$. In particular, write

$$\left[\tilde{C}_{1,i}\right]_{2Nt} = \left[\sum_{j=0}^{k-1} c_{1,i,j} 2^j\right]_{2Nt}$$

  where $c_{1,i,j} \in \{0, 1\}$ are bits. Define $c_{2,i,j}$ similarly.

Refined framework

Now we want to design a circuit $\mathcal{C}$ with addition and multiplication gates whose inputs are all of the $c_{1,i,j}$, $c_{2,i,j}$, and K_i and whose output is $\left\lfloor \frac{\mathsf{C}_1 + \mathsf{C}_2 \cdot \mathsf{SK}}{\Delta} \right\rceil$.

Assuming we have designed this circuit, the bootstrapping goes as follows:

- At the beginning of the protocol, the user sends the server a bootstrapping key

$$\mathsf{BSK} = (\mathsf{BSK}_0, \ldots, \mathsf{BSK}_{N-1})$$

  where BSK_i is a ciphertext for K_i.
- Whenever the server has a ciphertext $(\mathsf{C}_1, \mathsf{C}_2)$ encrypting message $\boldsymbol{M}$ with high noise $E < \frac{\Delta}{8}$ and wants to reduce the noise, the server extracts bits $c_{1,i,j}$ and $c_{2,i,j}$ for $0 \le i < N$ and $0 \le j < k$. Then the server computes $\mathsf{CC}_{\cdot,i,j} = \left(\Delta c_{\cdot,i,j}, 0\right)$, which is a ciphertext for $c_{\cdot,i,j}$ with zero noise.

- The reason we require $E < \frac{\Delta}{8}$ is so that we still have some room for error beneath the actual noise threshold of $\frac{\Delta}{2}$, which allows for the rounding of $C_{\cdot,i}$ to $\tilde{C}_{\cdot,i}$ mentioned earlier. The reason why this works will be explained in more detail soon.

- Recall that the circuit $\mathcal{C}$ of additions and multiplications on $c_{1,i,j}$, $c_{2,i,j}$, and K_i produces output $\boldsymbol{M} = \left\lfloor \frac{\mathsf{C}_1 + \mathsf{C}_2 \cdot \mathsf{SK}}{\Delta} \right\rceil$. So, the server can run the same circuit on the respective ciphertexts $\mathsf{CC}_{1,i,j}$, $\mathsf{CC}_{2,i,j}$, and BSK_i, and the output is a ciphertext $\mathsf{C}' = (\mathsf{C}'_1, \mathsf{C}'_2)$ of $\boldsymbol{M}$. Furthermore, the noise with which $(\mathsf{C}'_1, \mathsf{C}'_2)$ encrypts $\boldsymbol{M}$ will come only from the operations in $\mathcal{C}$; the noise from the original ciphertext $(\mathsf{C}_1, \mathsf{C}_2)$ does not carry over.

Discussion of multiplicative depth

We want the circuit $\mathcal{C}$ to be small because it means we can perform fewer operations each time we want to bootstrap a ciphertext. But there is another property we care about, called multiplicative depth.

As a simple example, consider two ways of multiplying eight polynomials $\boldsymbol{p}_1, \ldots, \boldsymbol{p}_8$ belonging to the plaintext space.

- Method 1: We can compute $((((((\boldsymbol{p}_1\boldsymbol{p}_2)\boldsymbol{p}_3)\boldsymbol{p}_4)\boldsymbol{p}_5)\boldsymbol{p}_6)\boldsymbol{p}_7)\boldsymbol{p}_8$.
- Method 2: We can compute $((\boldsymbol{p}_1\boldsymbol{p}_2)(\boldsymbol{p}_3\boldsymbol{p}_4))((\boldsymbol{p}_5\boldsymbol{p}_6)(\boldsymbol{p}_7\boldsymbol{p}_8))$.

These both involve seven multiplications and will lead to the same answer. However, if we want to multiply these polynomials homomorphically (i.e. multiply their corresponding ciphertexts), assuming the level of noise in the eight ciphertexts is roughly the same, method 1 will result in a product ciphertext with greater noise levels than method 2.

To see why, recall the noise bound for multiplication (which we did not prove):

- If $\left(\mathsf{C}_1^{(1)}, \mathsf{C}_2^{(1)}\right)$ and $\left(\mathsf{C}_1^{(2)}, \mathsf{C}_2^{(2)}\right)$ encode $\boldsymbol{M}^{(1)}$ and $\boldsymbol{M}^{(2)}$ with noise upper bounded by $E^{(1)}$ and $E^{(2)}$, respectively, then the result of ciphertext multiplication encodes $\boldsymbol{M}^{(1)}\boldsymbol{M}^{(2)}$ with noise upper bounded by $O\big(N^2 t\big(E^{(1)} + E^{(2)}\big)\big)$. Abbreviate $N^2 t$ by y, our "noise multiplier".

Then we can see that if the ciphertexts for $\boldsymbol{p}_i$ all have noise E, then multiplying them as in method 1 results in noise $O(y^7 E)$, whereas multiplying them as in method 2 results in noise $O(y^3 E)$ (with a larger constant term, but in practice not nearly enough to offset the extra factor of y^4).

We call this phenomenon the *multiplicative depth* of a circuit, formally defined as the greatest number of multiplications along a directed path through the circuit. We can check that methods 1 and 2 represent circuits of multiplicative depths 7 and 3, respectively. (Although our circuit $\mathcal{C}$ may have both additions and multiplications, we only consider multiplications since they amplify noise a lot more.)

The key takeaway is that circuits with lower multiplicative depth are better for keeping noise relatively low. In particular, we want to design $\mathcal{C}$ to have low multiplicative depth.

Designing the decryption circuit, part 1: analyzing rounding and noise

We now design the decryption circuit $\mathcal{C}$.

We want to extract the coefficients $M_0, \ldots, M_{N-1}$ of

$$\boldsymbol{M} = \left[\left\lfloor \frac{\mathsf{C}_1 + \mathsf{C}_2 \cdot \mathsf{SK}}{\Delta} \right\rceil\right]_t = M_{N-1}x^{N-1} + \cdots + M_1 x + M_0$$

individually. As an example, we will focus on the extraction of M_{N-1}, which is given by

$$M_{N-1} = \left[\left\lfloor \frac{C_{1,N-1} + \left(C_{2,0}K_{N-1} + \cdots + C_{2,N-1}K_0\right)}{\Delta} \right\rceil\right]_t.$$

There are similar equations for the other coefficients M_i. These equations are given by expanding $\mathsf{C}_1 + \mathsf{C}_2 \cdot \mathsf{SK} \pmod{x^N + 1}$ in terms of the coefficients of C_1, C_2, and SK. We also note that if we assume that $\boldsymbol{M}$ is a constant polynomial (an assumption which holds if the overall computation being run homomorphically has inputs which are constant polynomials), then the only coefficient we need to compute is M_0.

Returning to the formula for coefficient M_{N-1}, we are about to replace the terms $\frac{C_{\cdot,\cdot}}{\Delta}$ with $\frac{\tilde{C}_{\cdot,\cdot}}{2N}$, recalling that $\tilde{C}_{\cdot,\cdot} = \left\lfloor \frac{2N\mathsf{C}_{\cdot,\cdot}}{\Delta} \right\rceil$.

Before we do this, we demonstrate why the result will still round to the same integer:

Since the noise E of the original ciphertext is $< \frac{\Delta}{8}$, it follows that $\frac{C_{1,N-1}+\left(C_{2,0}K_{N-1}+\cdots+C_{2,N-1}K_0\right)}{\Delta}$ is less than $\frac{1}{8}$ away from an integer.

Now the discrepancy we are about to incur by replacing Cs with $\tilde{C}$s is

$$\begin{aligned}
&\left|\frac{C_{1,N-1} + \left(C_{2,0}K_{N-1} + \cdots + C_{2,N-1}K_0\right)}{\Delta} - \frac{\tilde{C}_{1,N-1} + \left(\tilde{C}_{2,0}K_{N-1} + \cdots + \tilde{C}_{2,N-1}K_0\right)}{2N}\right| \\
&\leq \left|\frac{C_{1,N-1}}{\Delta} - \frac{\tilde{C}_{1,N-1}}{2N}\right| + \sum_{i=0}^{N-1} K_{N-1-i}\left|\frac{C_{2,i}}{\Delta} - \frac{\tilde{C}_{2,i}}{2N}\right| \\
&\leq \frac{1}{4N} + \sum_{i=0}^{N-1}\frac{1}{4N} \\
&< \frac{3}{8}
\end{aligned}$$

assuming $N > 2$ (which it had better be, since the secret key has N bits of information).

Recalling that

$$\frac{C_{1,N-1} + \left(C_{2,0}K_{N-1} + \cdots + C_{2,N-1}K_0\right)}{\Delta}$$

is less than $\frac{1}{8}$ away from an integer, it now follows that

$$\frac{\tilde{C}_{1,N-1} + \left(\tilde{C}_{2,0}K_{N-1} + \cdots + \tilde{C}_{2,N-1}K_0\right)}{2N}$$

is less than $\frac{1}{2}$ away from the same integer, and in particular

$$\begin{aligned}
M_{N-1} &= \left[\left\lfloor \frac{C_{1,N-1} + \left(C_{2,0}K_{N-1} + \cdots + C_{2,N-1}K_0\right)}{\Delta} \right\rceil\right]_t \\
&= \left[\left\lfloor \frac{\tilde{C}_{1,N-1} + \left(\tilde{C}_{2,0}K_{N-1} + \cdots + \tilde{C}_{2,N-1}K_0\right)}{2N} \right\rceil\right]_t.
\end{aligned}$$

We can now write each $\tilde{C}_{\cdot,\cdot}$ in binary with $k = \log(2Nt)$ digits. We have

$$\tilde{C}_{1,N-1} = \langle c_{1,N-1,k-1}, \ldots, c_{1,N-1,0}\rangle_2$$

where $\langle \cdot \rangle_2$ means "read in binary". We also have

$$\tilde{C}_{2,i} K_{N-1-i} = \langle d_{2,i,k-1}, ..., d_{2,i,0} \rangle_2$$

where $d_{2,i,k-1} = c_{2,i,k-1} K_{N-1-i} \in \{0, 1\}$.

Designing the decryption circuit, part 2: binary addition
Now we have

$$M_{N-1} = \left[\left\lfloor \frac{\langle c_{1,N-1,k-1}, ..., c_{1,N-1,0} \rangle_2 + \sum_{i=0}^{N-1} \langle d_{2,i,k-1}, ..., d_{2,i,0} \rangle_2}{2N} \right\rceil \right]_t.$$

Conceptually, we've reduced the problem of computing M_{N-1} to that of adding $N+1$ binary numbers given their k digits each, dividing the sum by $2N$, and rounding it. The division and rounding will be easy because our sum will be written in binary.

The binary addition circuit used in the original BFV scheme can be described as follows:

Repeatedly perform an operation which takes three binary numbers and returns two binary numbers with the same sum. (This is called the carry-save adder.) In particular, replace the three binary numbers

$$\langle a_{k-1}, ..., a_2, a_1, a_0 \rangle_2 + \langle b_{k-1}, ..., b_2, b_1, b_0 \rangle_2 + \langle c_{k-1}, ..., c_2, c_1, c_0 \rangle_2$$

with $\langle d_{k-1}, ..., d_2, d_1, d_0 \rangle_2 + \langle e_{k-1}, ..., e_2, e_1, e_0 \rangle_2$, where the d_j are "mod 2 sum bits" and the e_j are the "carry bits". They are the bits such that $\langle e_{j+1}, d_j \rangle_2 = a_j + b_j + c_j$ where we define $a_{-1} = b_{-1} = c_{-1} = 0$.

Since our circuit should only have additions and multiplications, we check that d_j and e_{j+1} can equivalently be computed by

$$e_{j+1} = a_j b_j + a_j c_j + b_j c_j - 2 a_j b_j c_j$$

and

$$d_j = a_j + b_j + c_j - 2\big(a_j b_j + a_j c_j + b_j c_j\big) + 4 a_j b_j c_j.$$

(Note that we don't have to worry about carrying beyond index $k-1$ because we end up taking mod t when computing M_{N-1}.)

We apply this carry save adder, replacing three binary numbers with two, until there are only two numbers left. We need to apply the carry save adder in a balanced way so that the computation tree is wide instead of deep. Doing so, we get a depth of $O(\log N)$ so far.

We now need to add the last two $(\log k)$-digit binary numbers. In the original scheme, the authors suggest using schoolbook addition (also called a ripple-carry adder), which incurs $O(\log k)$ multiplicative depth.

Actually, a carry-lookahead adder (https://en.wikipedia.org/wiki/Carry-lookahead_adder) can be used to add the last two binary numbers instead, which would incur multiplicative depth of $O(\log \log k)$ instead of $O(\log k)$, appreciably decreasing the multiplicative depth of the overall circuit $\mathcal{C}$. Thanks to Brian Lawrence for this observation!

(For the rest of these notes, we analyze the original scheme which uses schoolbook addition to add the last two numbers.)

After this final binary addition, we get the digits s_j of the sum

$$[\langle s_{k-1}, ..., s_0 \rangle_2]_{2^k} = \left[\langle c_{1,N-1,k-1}, ..., c_{1,N-1,0} \rangle_2 + \sum_{i=0}^{N-1} \langle d_{2,i,k-1}, ..., d_{2,i,0} \rangle_2 \right]_{2^k}.$$

Now the coefficient we seek is simply

$$\begin{aligned} M_{N-1} &= \left[\left\lfloor \frac{\langle s_{k-1}, ..., s_0 \rangle_2}{2N} \right\rceil \right]_t \\ &= \langle s_{k-1}, ..., s_{\log(2N)} \rangle_2 + s_{\log(2N)-1} \\ &= \left(\sum_{j=0}^{k-1-\log(2N)} 2^j s_{j+\log(2N)} \right) + s_{\log(2N)-1}. \end{aligned}$$

Finally, after extracting all N coefficients in a similar manner, our circuit $\mathcal{C}$ can arrive at the output

$$\boldsymbol{M} = M_{N-1} x^{N-1} + \cdots + M_1 x + M_0.$$

To recap, we have just designed a circuit that inputs bits $c_{1,i,j}$, $c_{2,i,j}$, and k_i for $0 \leq i < N$ and $0 \leq j < k$ and outputs $\boldsymbol{M}$. If we run this circuit homomorphically (i.e. on the ciphertexts $\mathsf{CC}_{1,i,j}$, $\mathsf{CC}_{2,i,j}$, and BSK_i), we can get a ciphertext C' for $\boldsymbol{M}$, as desired!

Recalling that $k = \log(2Nt)$ and retracing all of the steps of the circuit, we can see that this decryption circuit has multiplicative depth $O(\log t + \log N)$. Thus, roughly speaking, our bound on the noise of C' is equal to that on a ciphertext that has undergone $O(\log t + \log N)$ multiplications.

This means that our "noise capacity" Δ must be large enough to support that many multiplications (plus some more, so that we can perform multiplications between bootstraps). The $\log t$ term in the depth can be rather large if we seek to perform arithmetic in large rings like $\mathbb{Z}_{2^{64}}$, which makes this version of the scheme at best feasible for small t.

5. Hint at Better Bootstrapping [CH'18]

Here we very briefly describe a major subsequent improvement to the bootstrapping.

Recall that when constructing the decryption circuit $\mathcal{C}$, we split each $\tilde{C}_{\cdot,i}$ into bits $c_{\cdot,i,j}$ and performed bitwise addition. The purpose of this was to have a way of rounding off binary digits of the sum, thus rounding off the noise bits attached to the coefficient of the message.

In 2018, Hao Chen and Kyoohyung Han found an alternate method of rounding off the noise bits for the decryption circuit. Their bootstrapping does not require splitting coefficients into bits. They write a decryption circuit with a multiplicative depth of $O(\log \log t + \log N)$, by constructing a polynomial that computes the "divide and round" step. Their method also uses far fewer total multiplications. This is a significant improvement over the $O(\log t + \log N)$ achieved by the original BFV scheme and makes BFV practical with large message spaces.

I'll write notes soon that cover this improvement in more detail; there's some really nice math in the construction of this circuit. If you're interested, keep an eye out for part 2 coming soon!