New Strategies for Bootstrapping Large-Error Ciphertext in Large-Precision FHEW/TFHE Cryptosystem [⋆]

Hongbo Li[1,2], Dengfa Liu[1,2], and Guangsheng Ma[1,3]

[1] Academy of Mathematics and Systems Science, Chinese Academy of Sciences,
[2] University of Chinese Academy of Sciences,
[3] North China Electric Power University, Beijing, China.
E-mail: hli@mmrc.iss.ac.cn, dengfa@amss.ac.cn, 627362183@qq.com.

Abstract. Bootstrapping is the core task in fully homomorphic encryption. It is designed to self-clean encrypted data to support unlimited level of homomorphic computing. FHEW/TFHE cryptosystem provides the fastest bootstrapping machinery in addition to the unique homomorphic evaluation functionality. In 2021, the problem of large-precision bootstrapping was investigated in the literature, with fast algorithms proposed and implemented. A common strategy to all the algorithms is to decompose the plaintext homomorphically into blocks from the tail up, at the same bootstrap the blocks sequentially.
This paper proposes two new strategies to improve the efficiency of large-precision plaintext bootstrapping. Both strategies are based on a new design of continuous nega-cyclic function with varying resolution, for making accurate computation with blockwise approximate computing. To minimize the approximation error in each block, optimizations are proposed based on rigorous error estimation, and are illustrated by error bounds in power-of-two binomial representation.
The first strategy is to make homomorphic approximate decomposition of the input plaintext from the head on. Compared with the tail-up approach, the head-on approach reduces the number of blocks at most by half asympotitically, at the same time reducing the final refreshed error by at most $1 - 1/\sqrt{2} \approx 29.3\%$.
The second strategy extends the head-on approach from large-precision plaintext bootstrapping to large error reduction. It can be used directly to the input ciphertext for the purpose of plaintext bootstrapping; it can also be used after plaintext bootstrapping to further reduce the refreshed error.
Two algorithms based on the above two strategies are proposed, together with some variants combining the tail-up approach. The tail-up approach is completely re-developed for optimal blocksize control based on careful error analysis, and a corresponding algorithm is proposed. All the algorithms are implemented on PALISADE, and experiments based on real data show that the by the new strategies, the speed of large-precision plaintext bootstrapping can be improved to as many as 7 times.

Keywords: Fully homomorphic encryption, bootstrapping, FHEW/TFHE, large-precision plaintext, power-of-two binomial representation of integers.

1 Introduction

Fully homomorphic encryption (FHE) is designed to compute directly on data in encrypted form. In the popular LWE/RLWE based FHE cryptosystem, an encrypted data m when scrutinized with the aid of a secret key, is always accompanied with and protected by a head error qI and a tail error e. During homomorphic computing, the errors grow towards the plaintext data. To make correct decryption, the data needs to keep safe distance from the errors. At some point the error will be too big to allow any further homomorphic computing. A scheme allowing a limited level of homomorphic computing is called a somewhat homomorphic encryption system (SWHE).

FHE differs from SWHE by it capability of supporting unlimited level of homomorphic computing. In 2009, a revolutionary idea was proposed to refresh an encrypted data by reducing the error homomorphically via the decryption circuit [16]. This idea is called *bootstrapping*. The first efficient bootstrapping method is based on the BGV scheme [3], where the plaintext m is behind both the head error qI and the tail error e in the phase $qI + e + m$ of the encrypted data. By taking qI as part of the tail error in a bigger-modulus BGV ciphertext, polynomial

[⋆] This paper is supported partially by China National Key Research and Development Projects 2020YFA0712300, 2018YFA0704705.

functions defined on a finite field [18], [21] can be designed to run on the encrypted data, and output a new ciphertext encrypting the same plaintext but with much smaller tail error. This original bootstrapping can be called *backup bootstrapping*, indicating that the plaintext is backed up after the bootstrapping.

Another efficient bootstrapping method follows a similar strategy. In the phase of a CKKS-format ciphertext [11], the plaintext m is sandwiched by the head error qI and the tail error e in the phase $qI + m + e$ of the encrypted data. Usually the tail error overlaps with the plaintext, and is taken as part of the plaintext, so bootstrapping the ciphertext aims at pushing the head error farther away from the plaintext. This is done by taking qI as part of the plaintext in a bigger-modulus CKKS ciphertext, running some appropriate polynomial functions [6], [5] that approximate the modulo-q function on plaintext, and outputting a new ciphertext encrypting approximately the same plaintext but with much farther away head error.

The most efficient bootstrapping method for unpacked accurate data is FHEW/TFHE. It is based on both the BFV scheme [2], [15] and the ring variant RGSW of the GSW scheme [19]. In the phase of a BFV-format ciphertext, the plaintext m is also sandwiched by the head error qI and the tail error e in the form $qI + m + e$, but unlike the CKKS-format, here m is separated from e and is immediately next to qI. In other words, m is a modular integer with modulus q.

To separate m from the errors, the FHEW/TFHE method chooses an RGSW working environment where the plaintext is stored in the exponent of a monomial. The underlying ring of RGSW has the property that $x^N = -1$ for an integer N of power of two, so that after changing the modulus from q to $2N$ and lifting the phase $qI + m + e$ to the exponent, the original head error qI disappears, and by $x^{m+e} = x^m x^e$, m and e can be easily separated in the exponent. RGSW scheme is utilized to control the error growth efficiently during the homomorphic lifting [1], [20]. Lifting the data from the coefficient to the exponent is a salient feature of FHEW/TFHE.

Another salient feature of FHEW/TFHE is programmable functional bootstrapping. To bring down plaintext m from the exponent to the coefficient, meanwhile inserting it to the argument of a prescribed function f, a test polynomial is used, which is essentially the look-up table of function f, so that when m is brought down to the coefficient, the error factor x^e is removed, and the encrypted plaintext becomes $f(m)$. This functionality of FHEW/TFHE has important applications in privacy-preserving machine learning [10].

The FHEW/TFHE bootstrapping is a blooming topic in FHE. Since its discovery in 2015, there have been a lot of papers in the literature devoted to further explore and develop the method [4], [8], [9], [13], [22], [23], [26], [27], [28]. On the other hand, this method has two obvious shortcomings. The first is its inadequacy for packed data bootstrapping, due to the harsh prerequisite of RGSW scheme in the error control of the homomorphic lifting. The second is the plaintext size limitation due to the exponent space restriction. The coefficient space in BFV/RGSW scheme can have hundreds of bits, and even thousands of bits if RNS representation is used. On the contrary, the exponent space cannot exceed 20 bits, due to the inability of current computer systems in manipulating large-degree polynomials.

In 2021, several methods [12], [25], [29] were proposed to overcome the second shortcoming, namely to handle the bootstrapping of large-precision data in FHEW/TFHE cryptosystem. These methods used the strategy of decomposing the long plaintext into shorter blocks from the tail up, bootstrapping each block sequentially, and finally concatenating all the blocks. In FHEW/TFHE, a cycle of bootstrapping by lifting a block of plaintext to the exponent and then down to the argument of a programmable function, with both input and output of the same ciphertext format, can be rightly called a *uni-bootstrap*. It is denoted by `GenPBS` in [12], by `Boot` in [25], and by `FBS` in [29].

- [12] proposed a fast method to bootstrap each block with a single uni-bootstrap, with the price of consuming one more bit in the exponent space, and enlarging the refreshed error by a BFV-format multiplication.
- [25] proposed a fast method to obtain the sign bit of the long plaintext homomorphically, by bootstrapping each block with two uni-bootstraps, one for the MSB (Most Significant Bit) of the block, the other for the lower bits. The efficiency is further improved by a sequence of modulus switches. Without reducing the moduli, the method can also be used to backup the original long plaintext homomorphically, and output a ciphertext with small refreshed error.
- [29] proposed a fast method to backup long plaintexts that can be either accurate or approximate. It uses the same strategy as [25] in bootstrapping each block: two uni-bootstraps, one for the MSB and the other for the lower bits. One difference is the implementation. While [25] is based on the public PALISADE library, [29] made its own C++ implementation from scratch.

We observe that in [25], [29], the phenomenon of requiring two uni-bootstraps for one plaintext block m' is caused by the disappearance of the MSB of m' after it is lifted to the exponent. Before the lift, let $m' = m'_{\text{MSB}}\, N + m'' \in \mathbb{Z}_{2N}$ be the plaintext block after modulus switch from q to $2N$, where $m'_{\text{MSB}} \in \{0, 1\}$ is the MSB of m', and m'' is the lower bits. Then $x^{m'} = x^{m'_{\text{MSB}}\, N} x^{m''} = (-1)^{m'_{\text{MSB}}} x^{m''}$ because $x^N = -1$. So after the lift, the MSB drops from the exponent to the coefficient, only the lower bits remains in the exponent. Because of this, two uni-bootstraps are required to recover m', one for m'_{MSB}, the other for m''.

Consider the following special case: in two's complement representation, plaintext block m' has it two leading bits, one being m'_{MSB}, the other being the MSB of m'' that is denoted by m''_{MSB}, having identical bit-value, *i.e.*, both 0 in the positive case, or both 1 in the negative case. Let us call such a block a "compliant block". When such a block is lifted, although m'_{MSB} is not in the exponent, its value is disclosed by m''_{MSB}, which is still in the exponent. Bootstrapping such a block needs only one uni-bootstrap.

How to obtain compliant blocks? For a long plaintext, suppose that it is divided into blocks from the head on, and the first block is already a compliant one containing at least three bits. Then one uni-bootstrap suffices to finish bootstrapping the first block. The second block is no longer a compliant one in general. However, when we subtract the refreshed first block from the input plaintext, then in the leftover plaintext, the bits that were previously occupied by the first block all have bitvalue 0 if the leftover plaintext is non-negative, or all have bitvalue 1 otherwise. So if we enlarge the second block by including two extra bits as the leading bits, we get a compliant block. This trick can be used for all later blocks, so that each successive block is compliant after the previous blocks are bootstrapped and then subtracted from the input.

The above blockwise bootstrapping is from the head of the plaintext on. As to the leftmost block, it can be bootstrapped by two uni-bootstraps, one for the MSB, the other for the lower bits. If everything works well, then on one hand, including two redundant bits in every succeeding block reduces the blocksize by 2. On the other hand, the number of uni-bootstraps is reduced by half for each succeeding block. Suppose that in the tail-up approach to blockwise bootstrapping, each block has bitsize $d + 2$ where $d > 0$. Then in the head-on approach, from the second block on, each block has bitsize d, where the two leading extra bits are not counted. For sufficiently long plaintext, the bootstrapping efficiency of the head-on approach is about d bits per uni-bootstrap, while for the tail-up approach, the bootstrapping efficiency is $d/2 + 1$ bits per uni-bootstrap. Another benefit of reducing the block number is the reduction of the refreshed error in the output ciphertext.

Unfortunately, not everything works well. In a uni-bootstrap, Before lifting to the exponent, the modulus of an encrypted block needs to be down switched from q to $2N$. This incurs a rounding error e_{MDS} that generally occupies more than half the bits of $\mathbb{Z}_{2N}$. In the tail-up approach, error e_{MDS} does not overlap with the plaintext, and can be completely eliminated after the plaintext block is brought down from the exponent to the coefficient. Once a block is bootstrapped and then subtracted from the input plaintext, the bits previously occupied by the block are vacant. In the head-on approach, the situation is different. After the modulus down switch and before the lifting, rounding error e_{MDS} generally covers the succeeding plaintext blocks, if there are any. The consequence is that before the lift, in $\mathbb{Z}_{2N}$ the plaintext bits that are not covered by error e_{MDS} are inaccurate, and there is no way to obtain an accurate result from the plaintext bits.

For the leftmost block of the input plaintext, the situation is even worse. In FHEW/TFHE, the programmable function must be *nega-cyclic* [14], [8]. Usually the function can be divided into two branches, only one branch is correct and returns the desired bootstrapping reuslt, while the other branch is designed solely to meet the nega-cyclicity requirement, and generally returns wrong result if the rounding error e_{MDS} forces the leftmost plaintext block to fall within the domain of the wrong branch. If the programmable function is discontinuous, controlling the error caused by misuse of the wrong branch is practically impossible.

To cope with the above situations, the programmable function for bootstrapping the lower bits of a plaintext block must be re-designed, which should be at least continuous, so that a small change in the domain of definition caused by e_{MDS} does not change the result too much.[4] The programmable function should also have a parameter called resolution that controls the output plaintext precision. If there are various resolutions available, then optimization is needed to minimize the approximation error. To handle approximate bootstrapping of the plaintext blocks, at the same time to obtain accurate bootstrapping of the whole long plaintext, the classical idea of accurate computation by approximate computing in computer algebra can be borrowed.

This paper realizes the above ideas. A continuous programmable function with nega-cyclicity is designed to bootstrap the lower bits of a block. A parameter called resolution is introduced to control the precision of approx-

[4] In contrast, the MSB function is not continuous, nor is any programmable function to bootstrap the MSB.

imate bootstrapping. Rigorous error estimations are made for various resolution values. For a plaintext block, if there is more than one resolution value available for the programmable function, then a series of lemmas are set up to optimize the resolution value for the purpose of maximizing the block size. To make accurate bootstrapping for the whole plaintext, the approximate plaintext is strictly separated from the refreshed tail error after each uni-bootstrap. To facilitate error control and illustrate optimization result, the error bound is usually measured by the so-called power-of-2 binomial representation, which is an integer of the form $2^l(1 \pm 2^{-k})$ where $0 < k \leq l$, and provides more detail than the usual power-of-2 bound. Usually the minus sign is used in the representation, so that there are more values approximating a power-of-2 bound from below.

Based on the above ideas, two new strategies are proposed for large-precision plaintext bootstrapping. The first strategy is to decompose homomorphically the input plaintext into blocks from the head on. Each block is bootstrapped approximately, whose size is maximized by optimizing the approximation precision. Every two adjacent blocks overlap. This overlap is not redundant in that the bootstrapping result of one block also repairs the inaccurate result of the preceding block. After all the blocks are bootstrapped, by summing up all the bootstrapping results, a ciphertext whose plaintext equal the input one is obtained.

For the purpose of backup bootstrapping, blockwise bootstrapping the tail error instead of the plaintext, can reach the same goal of error reduction. To bootstrap the tail error blockwise, starting from the head, namely the head-on approach, is the best choice. On one hand, bootstrapping the tail error is usually simpler than bootstrapping the plaintext, because for the tail error, the first block is usually compliant. On the other hand, the tail error can never be completely eliminated, and accurate bootstrapping of the tail error is impossible.

The second strategy is to decompose dynamically and homomorphically the tail error into blocks from the head on, back up each block approximately, and then subtract the result of bootstrapping once the block is bootstrapped. The block size changes dynamically as a result of error optimization. In this way, the tail error is gradually reduced, its higher bits are gradually vacant, while the plaintext keeps unchanged. This strategy can be called (tail) error bootstrapping. It is not only applicable to the input ciphertext, but also to the bootstrapping result of any plaintext bootstrapping algorithm, such as the algorithms in [12], [25], [29], or any algorithm based on the head-on approach to plaintext bootstrapping.

In order to make fair comparison, we re-develop and further improve the tail-up approach to plaintext bootstrapping. In [25], [29], the block size in the tail-up approach is set to be constant. This simplifies the algorithm design with the cost of more blocks and relatively small initial tail error. In our improvement, the block size no longer keeps constant, and each block size is the result of optimization. This improvement reduces the block number, at the same time allows for bigger initial tail error in the input ciphertext. When the initial tail error is very big, in the extreme case the the first plaintext block in the tail-up approach, which is at the tail of the whole plaintext, contains only one bit, namely the LSB (least significant bit). The smaller the first block, the bigger the allowed initial tail error. Bootstrapping the LSB of the plaintext first is called LSB precursor. It turns out that for big initial tail error, running the LSB precursor is necessary not only for the tail-up approach, but also for the head-on approach to plaintext bootstrapping, and in some scenario, also indispensable for the error bootstrapping.

In this paper, for each of the tail-up approach, the head-on approach, and the error bootstrapping, an algorithm is designed. Furthermore, some variants are designed to include special tricks to handle the first block, for example the LSB precursor. All the algorithms are implemented in the open-source FHEW/TFHE system on PALISADE platform. For real test data such as those used in [25], the head-on approach can save run-time by 17% to 27%, and the error bootstrapping can improve run-time speed to 7 times in the most favorable case.

The content of this paper is arranged as follows. Section 2 introduces the basics of FHEW/TFHE programmable bootstrapping, and presents a method of bootstrapping simulation called phase bootstrapping. Section 3 presents a continuous programmable function and the concept “resolution”, and investigates the high-resolution variants of some programmable functions. Section 4 re-develops the tail-up approach together with the LSB precursor. Section 5 develops the head-on approach. Section 6 develops the error bootstrapping, together with combinations of various bootstrapping strategies for FHE ciphertexts of different formats. Section 7 presents our experimental results on the three approaches. The Appendix contains a series of lemmas on the error estimation, block size optimization, and bootstrapping feasibility of the three approaches based on the power-of-2 binomial representation of error bounds.

2 Basics

2.1 FHEW/TFHE programmable bootstrapping

In a programmable functional uni-bootstrap, the plaintext is first lifted from the coefficient to the exponent of a monomial, then brought down to the argument of a programmable function, so that the output ciphertext encrypts the value of the programmable function at the input plaintext. Because the coefficient space is much larger than the exponent space, for large-precision plaintext only a block of it can be lifted to the exponent at a time. Bootstrapping an encrypted long plaintext block by block is called *block bootstrapping.*

The input of bootstrapping is an LWE ciphertext $\texttt{ct} = (\mathbf{a}, b)$, where $\mathbf{a} = (a_1, \ldots, a_n) \in \mathbb{Z}_q^n$ and $b \in \mathbb{Z}_q$. Suppose each entry is represented by an integer in $[-q/2, q/2)$. The entries s_i of the secret key $\mathbf{s} \in \mathbb{Z}_q^n$ are assumed to be sampled uniformly in $\{0, 1, -1\}$. The *modular phase* of the ciphertext is $m_0 \in \mathbb{Z}_q$ such that

$$m_0 = \text{phase}(\texttt{ct}) := b - \mathbf{a} \cdot \mathbf{s} = b - \sum_{s_i \neq 0} a_i s_i \mod q. \tag{2.1}$$

Later on when we talk about phase, we usually mean the modular one.

The first step of uni-bootstrap is to change the modulus from q to $2N$, so that the MSB of the block to be bootstrapped becomes the MSB of $\mathbb{Z}_{2N}$. This can be done by first making modulus down without scaling the input ciphertext, then making modulus switch by scaling the ciphertext and then rounding.

Assume

$$m_0 = \Delta m_{\text{init}} + e_{\text{init}} \mod q, \tag{2.2}$$

where $m_{\text{init}} \in \mathbb{Z}_t$ is the original plaintext,

$$\Delta = \lfloor q/t \rfloor, \tag{2.3}$$

and the original error $|e_{\text{init}}| \leq E_{\text{init}} < \Delta/2$, so that decoding m_0 gives $\lfloor m_0/\Delta \rceil = m_{\text{init}} \mod t$.

Let $\Delta < q' \leq q$. Since any integer in $[-q/2, q/2)$ can be taken as a representative of a modular integer in $\mathbb{Z}_{q'}$, $\texttt{ct} \in \mathbb{Z}_q^{n+1}$ can be taken as a ciphertext with modulus q', so that if $q \mid tq'$, then the encrypted plaintext is the following tail block of m_{init}:

$$m' = (m_{\text{init}} \mod tq'/q) \cap [-tq'/(2q), tq'/(2q)). \tag{2.4}$$

Taking $\texttt{ct}$ as an LWE ciphertext with modulus $q' < q$ is called *modulus down.* It changes the modulus and the plaintext, but does not change the ciphertext components $\mathbf{a}, b$. The phase of $\texttt{ct}$ after modulus down is $\Delta m' + e_{\text{init}} = m_0 \mod q'$.

Let there be an RGSW homomorphic computing environment with modulus $Q \gg q$ and ring dimension $N > n$, that has the same security level with the input LWE ciphertext. When $tq'/q \in \mathbb{Z}$ and is small, the FHEW bootstrapping of $\texttt{ct} \in \mathbb{Z}_q^{n+1}$ by taking it as encrypting $m' \in \mathbb{Z}_{tq'/q}$, is composed of three steps:

(1) **Modulus down switch from** $\text{LWE}_{n,q'}$ **to** $\text{LWE}_{n,2N}$:

Take $\text{LWE}_{n,q}$ ciphertext $\texttt{ct} \in$ as one with modulus $q' < q$, change the modulus to $2N \ll q$ by scaling and rounding:

$$\texttt{ct'} = (\mathbf{a}', b') := (\lfloor \mathbf{a} \times 2N/q' \rceil, (\lfloor b \times 2N/q' \rceil) \in \mathbb{Z}_{2N}^{n+1}. \tag{2.5}$$

The encrypted plaintext is m', the rounding error is

$$e_{\text{MDS}} := (b' - b \times 2N/q') - (\mathbf{a}' - \mathbf{a} \times 2N/q') \cdot \mathbf{s} \in \mathbb{Q}_{2N}. \tag{2.6}$$

For large t, it is possible that $\Delta \times 2N/q' = (q/q') \times (2N/t) < 1$, then the phase of $\texttt{ct'}$ is

$$\begin{aligned} m_0' &:= \lfloor b \times 2N/q' \rceil - \lfloor \mathbf{a} \times 2N/q' \rceil \cdot \mathbf{s} \\ &= e_{\text{MDS}} + (b - \mathbf{a} \cdot \mathbf{s}) \times 2N/q' \\ &= e_{\text{MDS}} + (\Delta m' + e_{\text{init}}) \times 2N/q' \\ &= \Delta m' \times 2N/q' + e' \mod 2N, \end{aligned} \tag{2.7}$$

where

$$e' = e_{\text{MDS}} + e_{\text{init}} \times 2N/q' \in \mathbb{Q}, \tag{2.8}$$

For `ct'` to be decryptable, $|e'| < \Delta N/q'$ is required.

(2) **Phase lift-up from** $\mathrm{LWE}_{n,2N}$ **to** $\mathrm{RGSW}_{N,Q}$:

Use identity

$$x^{\mathrm{phase}(\mathtt{ct'})} = x^{b'} \prod_{s_i \neq 0} x^{-a_i s_i} \tag{2.9}$$

to lift phase(`ct'`) up to the exponent of a monomial homomorphically, by encrypting each $x^{-a_i s_i}$ in RGSW format, and making successive multiplications between RGSW ciphertexts. This procedure is called homomorphic *accumulation* in the exponent. The result is an RGSW ciphertext `ct"` with modulus Q and ring dimension N, that encrypts monomial $x^{m'_0} = x^{(2N/q')\Delta m'} x^{e'}$.

(3) **Phase bring-down from** $\mathrm{RGSW}_{N,Q}$ **to** $\mathrm{LWE}_{n,q}$:

A so-called *programmable function* $f(x)$ is needed to bring down the plaintext m' from the exponent to the constant term of a polynomial. For the most general-purpose functional bootstrapping, f is from $\mathbb{Z}_{2N}$ to $\mathbb{Z}_Q$. A trivial $\mathrm{RGSW}_{N,Q}$ encryption of test polynomial

$$c(x) := \sum_{d \in \mathbb{Z}_{2N}} f(d) x^{-d} \tag{2.10}$$

is multiplied with `ct"`, and the result is an $\mathrm{RLWE}_{N,Q}$ ciphertext encrypting a plaintext having constant term $f(m'_0)$. This RLWE ciphertext is then changed into $\mathrm{LWE}_{n,q}$ form by key switch and modulus switch.

To get rid of the error e' in the exponent of $x^{m'_0}$,

$$f(2N\Delta m'/q' + e') = f(2N\Delta m'/q') \tag{2.11}$$

is sufficient. Let

$$\begin{aligned} g(x) &= \sum_{i=-\lfloor tq'/(2q) \rfloor}^{tq'/(2q) - \lfloor tq'/(2q) \rfloor - 1} f(2N\Delta i/q')\, x^{-2N\Delta i/q'}, \\ g_0(x) &= \sum_{e \in (-\Delta N/q', \Delta N/q') \cap \mathbb{Z}} x^e, \end{aligned} \tag{2.12}$$

then the test polynomial $c(x) = g(x)g_0(x)$, and $x^{m'_0}c(x)$ has constant term $f(2N\Delta m'/q')$ as desired. Notice that when expanding the product of polynomials $g(x), g_0(x)$, no like terms occur, due to $|e| < \Delta N/q'$.

We see that to get rid of error e' from the exponent, if $c(x)$ takes the form (2.10), then the programmable function f should satisfy (2.11). In the general case where f does not satisfy this requirement, the test polynomial should take the form $c(x) = g(x)g_0(x)$, where factor $g(x)$ serves as a look-up table of function f, and factor $g_0(x)$ is used to eliminate the error e'.

If $f(2N\Delta m'/q') = m'$, the we would have obtained a refreshed ciphertext encrypting m'. Unfortunately this is possible only for the special case where $tq'/q = 2$ and N is a power of two, so that $m' \in \mathbb{Z}_2$ and $x^{(2N/q')\Delta m'} = x^{Nm'} = (-1)^{m'}$. When $tq'/q > 2$, the programmable function is required to be $2N$-periodic nega-cyclic. A $2N$-*periodic nega-cyclic function* on $\mathbb{Z}$ is a function that satisfies

$$f(x + N) = -f(x), \quad \forall x \in \mathbb{Z}. \tag{2.13}$$

The nega-cyclicity requirement is a direct consequence of the fact that after lifting the phase of an LWE ciphertext to the exponent of a monomial, the MSB of m'_0 drops to the coefficient and becomes a sign factor. If $m'_0 = N + c' \bmod 2N$ for some $0 \leq c' < N$, then $x^{m'_0} = -x^{c'}$, and the constant term of $x^{m'_0}c(x)$ is $f(N + c') = f(m'_0) = -f(c')$. Hence the nega-cyclicity guarantees no contradiction in the result.

TFHE bootstrapping is an improvement of FHEW bootstrapping in that every (matrix) multiplication between two RGSW ciphertexts is replaced by a (vector-matrix) multiplication between an RLWE ciphertext and a GSW ciphertext. This is achieved by putting a trivial $\mathrm{RLWE}_{N,Q}$ encryption of plaintext polynomial $x^{b'}c(x)$ at the beginning of the second step, so that the product of $x^{b'}c(x)$ and $x^{a_i s_i}$ is homomorphically realized by the multiplication of the RLWE ciphertext with an RGSW ciphertext encrypting $x^{a_i s_i}$.

In TFHE bootstrapping for the case where $tq'/q \in \mathbb{Z}$ is small, there are also the above three steps, the difference lies in the latter two steps. In the second step, the starting ciphertext is a trivial RLWE encryption of polynomial $x^{b'}c(x)$; the middle and final ciphertexts, being multiplications of an RLWE ciphertext by an RGSW ciphertext, are RLWE ones encrypting $x^{b'}c(x)x^d$ for varying integer d. So this step essentially rotates the list of coefficients of polynomial $c(x)$ by a blind/secret amount d in each iteration, called *blind rotation*. The third step of TFHE bootstrapping is simply changing the RLWE ciphertext into $\mathrm{LWE}_{n,q}$ form by key switch and modulus switch.

When viewed from the content of the encrypted plaintext in the second step, FHEW gains accumulation on the exponent of its plaintext monomial, while TFHE gains synchronous accumulation on the exponent of each term of its test polynomial, or equivalently, gains rotation on the list of coefficients of its test polynomial. As the latter two steps are closely related to each other, it is better to put them together and name them the step of "*accumulative blind rotation*".

The procedure consisting of the above two steps: modulus down switch and accumulative blind rotation, is called a *uni-bootstrap*. The result is a ciphertext with refreshed error bound $E_B \ll E_{\text{init}}$ that is independent of the input m_{init} and e_{init}.

2.2 Backup block bootstrapping

If the output encrypts the input long plaintext, the bootstrapping is called *backup bootstrapping*.

From now on for simplicity of statement, it is assumed that q, q', t, n, N are all powers of two. For example, q can be changed into a power of two by by modulus switch. Denote

$$\begin{array}{l} l_q = \log(q),\ \ l_t = \log(t), \quad l_\Delta = \log(\Delta), \\ l_n = \log(n),\ l_N = \log(N),\ l_{q'} = \log(q'). \end{array} \tag{2.14}$$

Due to the nega-cyclicity constraint of the programmable function, the MSB of $m' \in \mathbb{Z}_{tq'/q}$ and all the lower bits cannot be recovered all at once by a single uni-bootstrap using only one programmable function. The MSB of m' is assumed to be the i-th bit of m' from the right and have bit-value $b_i \in \{0, 1\}$, where

$$i = \log(tq'/q) = l_{q'} - l_\Delta > 0. \tag{2.15}$$

When $tq'/q = 2^i$ is so small that in $\mathbb{Z}_{2N}$, the block $m' \times 2N/2^i$ occupied by plaintext m' does not overlap with the rounding error of modulus down switch, or more strictly, $|e'| < N/2^i$, then a refreshed ciphertext encrypting m' can be obtained by one or two uni-bootstraps, in one of the following ways:

Double uni-bootstrap approach [25], [29]:

1. Starting from a ciphertext `ct` encrypting m', the first uni-bootstrap returns a ciphertext $\texttt{ct}_1$ encrypting the integer $b_i\, 2^i$ corresponding to the MSB (2.15) of m', based on the $2N$-periodic nega-cyclic programmable function f^{init} depicted in Fig. 1.

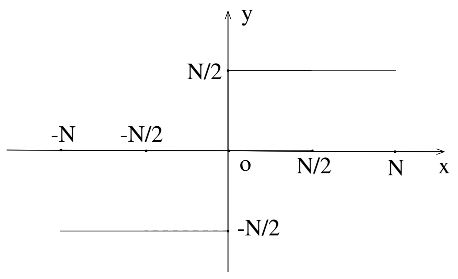

Fig. 1. Programmable function f^{init} for MSB bootstrapping.

2. Starting from ciphertext $\texttt{ct} - \texttt{ct}_1$, the second uni-bootstrap returns a ciphertext $\texttt{ct}_2$ encrypting all the lower bits m'' of m', based on the $2N$-periodic nega-cyclic programmable function f' depicted in Fig. 2.

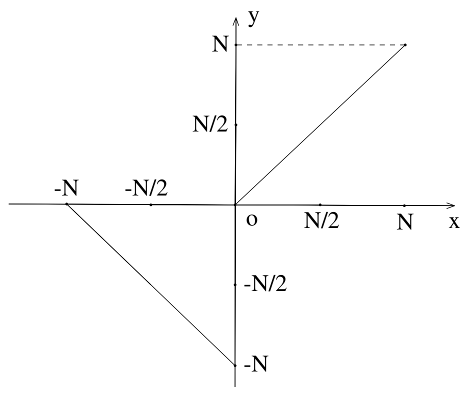

Fig. 2. Discontinuous programmable function f' for lower bits bootstrapping.

3. Return $\texttt{ct}_1 + \texttt{ct}_2$.

Single uni-bootstrap approach [12]:

1. Choose a new modulus q' that is half the one used in the previous approach. Then the bit-size i of m' in (2.15) is reduced by 1. The error $e' \in \mathbb{Z}_{2N}$ is the same as before, but now satisfies $|e'| < N/2^{i+1}$ for the reduced i. Use both programmable functions $f^{\text{init}}(x), f'(x)$ to construct test polynomial

$$c(x) = \{g_1(x) + x^{-N/2^i} g_2(x)\} g_0(x), \tag{2.16}$$

where g_0 is defined in (2.12), and for $j = 1, 2$, $g_j(x)$ is obtained from $g(x)$ in (2.12) by replacing f with f^{init}, f' respectively. Because $2N\Delta/q' > N/2^i$ and $|e'| < N/2^{i+1}$, no like terms occur in the polynomial addition and multiplication of (2.16).
Do one uni-bootstrap to get two ciphertexts $\texttt{ct}_1$ and $\texttt{ct}_2$ encrypting respectively the coefficients of x^0 and $x^{N/2^i}$. $\texttt{ct}_1$ encrypts the integer $2b_i$ where b_i is the MSB of m', while $\texttt{ct}_2$ encrypts the absolute value $|m'|$.
2. Make RLWE ciphertext multiplication to get $\texttt{ct}_3 = \texttt{ct}_1 \times \texttt{ct}_2$.
3. Return $\texttt{ct}_2 - \texttt{ct}_3$, because

$$m' = (-1)^{b_i}|m'| = |m'| - (2b_i)|m'|. \tag{2.17}$$

The single uni-bootstrap approach has the benefit of fewer uni-bootstraps, with the cost of much larger refreshed error. For better-quality bootstrapping, the double uni-bootstrap approach is preferred.

When tq'/q is not small, so that the block of plaintext m' overlaps with error e' in $\mathbb{Z}_{2N}$, then q' must be reduced in order not to corrupt the plaintext. This is the case of large precision l_t. The original plaintext m_{init} must be divided into blocks, so that each block can backed up by one of the two alternate procedures. This is called *backup block bootstrapping*. In all the three references [25], [29], [12], block bootstrapping is done from the tail block to the head block sequentially, where each block has the same size.

2.3 Phase simulation of bootstrapping

In error analysis and correctness verification, once the three basic error variables: the initial error e_{init}, the rounding error e_{MDS} of modulus down switch, and the refreshed error e_B of uni-bootstrap, are given either variances as subgaussian random variables, or absolute bounds as regular variables, then the two steps of a uni-bootstrap can be both simulated by acting on the input plaintext directly. This *phase simulation* can dramatically simply error analysis.

Furthermore, due to the narrow exponent space constraint of the RLWE/RGSW cryptosystem, FHEW/TFHE method cannot apply to ring dimension bigger than 20. To investigate the asymptotic behavior of a bootstrapping algorithm by experimentation involving large FHE parameters, phase simulation must be resorted to.

There are two approaches to error analysis. The first is the absolute error approach, in that the sum of errors e_{Mi} is bounded by the sum $\sum_i e_{Mi}$ of their absolute bounds $e_{Mi} \geq |e_{Mi}|$. The second is the heuristic error approach [7], in that the following errors are taken to be independent random variables (this is called the *error independence heuristic*):

- the initial error e_{init} of the input ciphertext $\texttt{ct}$;
- the refreshed error e_{Bi} in the resulting ciphertext $\texttt{ct}_i$ of the i-th uni-bootstrap;
- the rounding error e_{Mi} of ciphertext $\texttt{ct} - \sum_{j=1}^{i} \texttt{ct}_j$ in modulus down switch, where $i \geq 0$. When $i = 0$, the ciphertext is just the input $\texttt{ct}$.

Notice that the independence heuristic is on both the errors within each class and the errors of different class.

In the heuristic approach, an error e is assumed to be subgaussian with mean zero and variance σ_e^2, such that the bound E_e of $|e|$ is determined with some failure rate by

$$E_e = \mu \sigma_e, \tag{2.18}$$

where $\mu > 0$ is a constant independent of e, and is determined first by subgaussian tail estimation and then trimmed by experiments. The sum of errors with scaling: $\lambda_1 e_1 + \ldots + \lambda_n e_n$, is then subgaussian with mean zero and variance $\sum_i \lambda^2 \sigma_{e_{Mi}}^2$. Since μ is a universal constant, the bound of the sum of errors is $\sqrt{\sum_i \lambda_i^2 E_i^2}$ for $|e_{Mi}| \leq e_{Mi}$.

For example for the error in (2.8), its absolute bound is $1/2 + E_{\text{MDS}} + E_{\text{init}} \times 2N/q'$, while its heuristic bound is

$$\sqrt{E_{\text{MDS}}^2 + E_{\text{init}}^2 \times (2N/q')^2}, \tag{2.19}$$

where e_r is too small to be considered. In this paper, we use the absolute approach and the heuristic approach in parallel, so that the former is failure-free, while the latter is more practical.

For e_{MDS}, let

$$\begin{aligned} \epsilon_0 &= b \times 2N/q' - \lfloor b \times 2N/q' \rceil, \\ \epsilon_i &= (a_i \times 2N/q' - \lfloor a_i \times 2N/q' \rceil) s_i. \end{aligned} \tag{2.20}$$

When the secret key is ternary uniformly random in $\{0, \pm 1\}$, one can generate e_{M0} and the e_{Mi} for all $s_i \in \{\pm 1\}$ as i.i.d. uniform distribution on interval $[-1/2, 1/2]$. There are $1 + 2n/3$ of them all together, so random variable $e_{\text{MDS}} = \epsilon_0 - \sum_{i=1}^{n} s_i \epsilon_i$ for large n is approximately Gaussian by the central limit theorem, with standard deviation

$$\sigma_{\text{MDS}} = \sqrt{(1 + 2n/3) \times (1/12)} \approx \sqrt{2n}/6. \tag{2.21}$$

So we can set

$$E_{\text{MDS}} = \mu \sqrt{2n}/6. \tag{2.22}$$

Theoretically μ determines the error failure probability: for a zero-centered Gaussian distribution $e(x)$ with standard deviation σ_e,

$$P(|e(x)| > E) = 1 - \text{erf}(E/(\sqrt{2}\sigma_e)) = \text{erfc}(\mu/\sqrt{2}). \tag{2.23}$$

To facilitate error analysis and algorithm implementation, we usually simulate the bounds of two important errors by the following *power-of-2-binomial integers on the upper side*:

$$\begin{aligned} E_{\text{MDS}} &= 2^{l}(1 - 2^{-k}) &&= 2^{-d}(1 - 2^{-k})N/4, \\ E_B &= 2^{l_B}(1 - 2^{-k_B}) &&= 2^{-d_B}(1 - 2^{-k_B})\Delta/2, \\ E_{\text{init}} &= 2^{l_I}(1 - 2^{-k_B}) &&= 2^{-d_I}(1 - 2^{-k_I})\Delta/2, \end{aligned} \tag{2.24}$$

where

$$\begin{aligned} d &= l_N - l - 2, \\ d_B &= l_\Delta - l_B - 1, \\ d_I &= l_\Delta - l_I - 1, \end{aligned} \tag{2.25}$$

for non-negative integers $k, l, k_B, d_B, k_I, d_I, d, d_B, D_I$ satisfying the following:

- $1 < k \leq l,\ 1 < k_B \leq l_B,\ \ 1 < k_I \leq l_I$;
- $0 < l \leq l_N - 3,\ \ 4 \leq l_B < l_I \leq l_\Delta - 1$;
- $0 < d \leq l_N - 3,\ \ 0 \leq d_I < d_B \leq l_\Delta - 5$.

Below we explain the ranges. The right side of (2.24) are all integers. Usually $l \geq 5$, if $k = 1$, then $2^{l+1}(1-2^{-k}) = 2^l \approx 2^l - 1 = 2^l - 2^{l-l}$, so we choose $2 \leq k \leq l$ to avoid redundancy in resolution. For the same reason we have $k_B, k_I > 1$.

By FHE standards, the error in a new ciphertext has standard deviation 3.19, the fresh error bound can be set to be about 2^4. So $l_B \geq 4$ for refreshed error E_B. As a quality requirement of bootstrapping, $\lfloor \log E_B \rceil < \lfloor \log E_{\text{init}} \rceil$ is necessary, which is just $l_B < l_I$. That $E_{\text{init}} < \Delta/2$ requires $l_I \leq l_\Delta - 1$.

That $d > 0$ is for bootstrapping of long plaintexts, where $d{+}2$ is often the regular block length. So $l = l_N - d - 2 \leq l_N - 3$ and $d \leq l_N - 3$. In practice, at least $d \geq 3$ and $l \geq 4$. In our experiments, it also holds that $l_N \geq l + k/2$.

Alternatively, there are bounds of the form $2^l(1+2^{-k})$ where $2 \leq k \leq l$, which are power-of-2 binomial integers on the lower side. They are not as useful as those on the upper side in exploring the limits of approximation to power-of-2 upper bounds, and will not be used in this work.

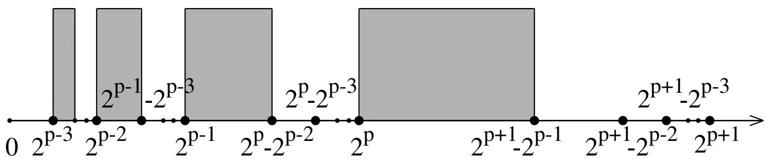

Fig. 3. power-of-2 binomial integers on the upper side. The shadow zones on the real axis are on the lower side and are empty.

For example, for $\mu = 6, 12$, the corresponding failure probabilities are both very small:

$$\text{erfc}(6/\sqrt{2}) \approx 1.97 \times 10^{-9}, \quad \text{erfc}(12/\sqrt{2}) \approx 3.55 \times 10^{-33}; \tag{2.26}$$

the parameters of E_{MDS} and E_B are the following:

- If $n = 2^9$, then $\sqrt{2n} = 2^5$. If $\mu = 6$, then $E_{\text{MDS}} = 32 = 2^5 \approx 2^5(1-2^{-5})$, so $l = k = 5$. If $\mu = 12$, then $E_{\text{MDS}} = 64 = 2^6 \approx 2^6(1-2^{-6})$, so $l = k = 6$.
- If $N = 2^{11}$, then $\sqrt{2N} = 2^6$. If $\mu = 6$, then $E_B = 2^6$, so $l_B = k_B = 6$. If $\mu = 12$, then $E_B = 2^7$, so $l_B = k_B = 7$.
- If $n = 2^{10}$, then $\sqrt{2n} = 2^5\sqrt{2}$. If $\mu = 6$, then $E_{\text{MDS}} = 2^5\sqrt{2} < 2^6(1-2^{-2})$, so $l = 6, k = 2$. If $\mu = 12$, then $E_{\text{MDS}} = 2^6\sqrt{2} < 2^7(1-2^{-2})$, and $l = 7, k = 2$.
- If $N = 2^{12}$, then $\sqrt{2N} = 2^6\sqrt{2}$. If $\mu = 6$, then $E_B = 2^6\sqrt{2}$, so $l_B = 7, k_B = 2$. If $\mu = 12$, then $E_B = 2^7\sqrt{2}$, so $l_B = 8, k_B = 2$.

In the phase space $\mathbb{Z}_{2N}$, let the plaintext occupies the region from the left first bit to the left (d_0+2)-bit, where $d_0 \geq -1$, then there are $(l_N+1) - (d_0+2)$ bits left in the space. For correct decryption, there must be a gap of at least 1 bit between the plaintext and the error. So the error occupies at most $l_N - d_0 - 2 = l_0$ bits. For example in the notations of (2.25), if $d_0 = d$ then $l_0 = l$. Let E be the error bound in $\mathbb{Z}_{2N}$. The following is the *error correctness constraint* in $\mathbb{Z}_{2N}$:

$$E < 2^{l_0} = N/2^{d_0+2}. \tag{2.27}$$

In the phase space $\mathbb{Z}_q$ represented by integers bounded by $q/2$, the two intervals $(0, q/4)$ and $(q/4, q/2)$ have the same number of positive integers. For a modular number $x \in \mathbb{Q}_q$, if $|[x]_{\pm q/2}| \leq q/4$, then x is said to be *small*; if $|[x]_{\pm q/2}| \in (q/4, q/2]$, then x is said to be *large*. For example, for the initial error bound, if $E_{\text{init}} \leq \Delta/4$, the initial error is small; if $\Delta/4 < E_{\text{init}} < \Delta/2$, the initial error is large.

The modulus down switch step can be simulated by defining a rounding function r from $\mathbb{Z}_q \times \mathbb{Z}$ to $\mathbb{Z}_{2N}$:

$$r_{2N/q'}(m, e) = m \times (2N/q') + e \mod 2N, \tag{2.28}$$

where error $e \in \mathbb{Q}$ simulates e_{MDS}. The benefit of this simulation over the usual one $r_{2N/q'}(m, e) = \lfloor m \times (2N/q') \rceil + e$ where $e \in \mathbb{Z}$, is that the 1/2-bounded rounding error e_r between $m \times (2N/q')$ and $\lfloor m \times (2N/q') \rceil$ is integrated with e, which simplifies error analysis; the drawback is that the randomness of e is harmed by the constraint $m \times (2N/q') + e \in \mathbb{Z}$. Alternatively, $r_{2N/q'}$ can be defined as $\lfloor m \times (2N/q') + e \rceil$. In error analysis, the above 1/2-bounded rounding error is too small to make any effect.

For the refreshed error e_B after uni-bootstrap, a Gaussian distribution with heuristic bound E_B is sufficient to simulate it. The estimation of E_B from both theoretical deduction and statistical experiments, can be found in [17]. Later in this paper we also report our own estimation of E_B by statistical experiments.

The accumulative blind rotation step can be simulated by the sum of a programmable function $f : \mathbb{Z} \longrightarrow \mathbb{Z}_q$ and an error $e_B \in \mathbb{Z}_q$, where $|e_B| \leq E_B$ and f satisfies (2.11). Then the whole uni-bootstrap procedure can be simulated by the following function mapping $m \in \mathbb{Z}_q$ to $m_{\text{UniBoot}} \in \mathbb{Z}_q$:

$$m_{\text{UniBoot}} = f(r_{2N/q'}(m, e)) + e_B \mod q. \tag{2.29}$$

For backup bootstrapping, defining a new programmable function $f : \mathbb{Z}_{2N} \longrightarrow \mathbb{Z}_{2N}$ simplifies error estimation tremendously. To further simplify error analysis, we redefine the rounding function $r_{2N/q'}$ so that it is from $\mathbb{Z} \times \mathbb{Z}$ to $\mathbb{Z}$. The purpose of changing the domain of definition to $\mathbb{Z}$ is to make the modular addition in (2.28) an integer addition. Changing the range of value to $\mathbb{Z}$ has no influence upon error analysis, because $r_{2N/q'}$ is always coupled with $2N$-periodic function f in uni-bootstrap simulation.

3 New techniques for block bootstrapping error analysis

In this section, we first investigate the difficulties in realizing the idea of head-on block bootstrapping, then develop some techniques to overcome the difficulties. The techniques include a continuous programmable function, resolute numbers, and some inequalities related to them.

Suppose the two leading bits of a block m' have identical value. When making uni-bootstrap, the first step is modulus down switch that generates a rounding error e_{MDS} that may occupy most of the bits of $\mathbb{Z}_{2N}$. This newly introduced error may change the values of the two leading bits in $\mathbb{Z}_{2N}$, so that they are no longer identical.

For example, if m' has two leading bits of value 00, the rounding error may change them into 01 or 11: the first may occur when the lower bits of m' all have value 1, and $e_{\text{MDS}} > 0$; the second may occur when the lower bits of m' all have value 0, and $e_{\text{MDS}} < 0$. In duality, if m' has two leading bits of value 11, the rounding error may change them into 10 or 00: the first may occur when the lower bits of m' all have value 0, and $e_{\text{MDS}} < 0$; the second may occur when the lower bits of m' all have value 1, and $e_{\text{MDS}} > 0$.

The instability of leading bits also exists in previous bootstrapping methods. When bootstrapping is from the tail up, there is a "firewall" separating the plaintext m' from the error e_{MDS}, which is composed of at least one vacant bit. For example, if $|e'| < 2^p$ for the error in (2.8), and m' is embedded into $\mathbb{Z}_{2N}$ as $m' \, 2^{p+1}$, then the bit occupied by 2^p via its unique nonzero bit-value serves as a firewall between the plaintext and the error. The MSB of the firewall behaves as the rounding bit of the plaintext, whose role is to stabilize not only the leading bits, but also the lower bits of m'. Indeed, the rounding bit is already in use in factor $g_0(x)$ of the test polynomial (2.12) to get rid of the error part.

If bootstrapping starts from the head on, there is no longer a firewall separating the plaintext from the error, because in modulus down switch, the rounding error overlaps with the input long plaintext. Although m' is not covered by e_{MDS} in $\mathbb{Z}_{2N}$, those bits behind m' in m_{init} are flooded by the error, leading to the consequence that the last bit of m' cannot be recovered even after the block of m' has finished bootstrapping, because there is no longer a "clean" rounding bit following m'.

In m_{init} after subtracting the result of bootstrapping block m', the next block should include some number of bits that were previously occupied by m', so that the block starts with exactly two leading bits of identical value. The bootstrapping result of the block will repair the last few bits of the previous block m'. Such repairing finishes only when the LSB of the input plaintext m_{init} is bootstrapped. Before that, the sum of bootstrapping results is not equal to the sum of the bootstrapped blocks.

Another error in bootstrapping is caused by falling into the wrong branch of the programmable function that is designed only to meet the nega-cyclicity constraint. For example, in Fig. 2 of function f' that is designed to recover the lower bits of m', the branch defined on interval $[-N, 0)$ does not return the correct backup result $f'(x) = x$, but rather returns $f'(x) = -x - N$. The function in Fig. 2 is discontinuous at points $-N, 0$, making error control very difficult.

3.1 Continuous programmable function for block bootstrapping

We observe that in Fig. 1 of function f^{init}, $|x - f^{\text{init}}(x)| < N/2$ for any $x \in (-N, 0) \cup (0, N)$, and the function is continuous on the two intervals. Formally,

$$\begin{aligned} f^{\text{init}} : \mathbb{Z} &\longrightarrow [-N/2, N/2] \\ x &\mapsto \begin{cases} N/2, & \text{if } x \in [0, N) \\ -N/2, & \text{if } x \in [-N, 0) \end{cases} \end{aligned} \tag{3.1}$$

If we choose a continuous programmable function f for lower bits recovery, so that it agrees with the identity function on interval $(-N/2, N/2)$, then by

$$f^{\text{init}}(x) + f(x - f^{\text{init}}(x)) = x, \quad \forall x \in (-N, 0) \cup (0, N), \tag{3.2}$$

the two functions f^{init}, f correctly backup all $x \in (-N, 0) \cup (0, N)$ by first returning $f^{\text{init}}(x)$, then returning the value of f at the remainder of x after subtracting $f^{\text{init}}(x)$, and then adding the two returns up. By continuity and $2N$-periodic nega-cyclicity, function f is of the following form:

$$\begin{aligned} f : \mathbb{Z} &\longrightarrow [-N/2, N/2] \\ x &\mapsto \begin{cases} x, & \text{if } x \in [-N/2, N/2] \\ N - x, & \text{if } x \in [N/2, N] \\ -N - x, & \text{if } x \in [-N, -N/2] \end{cases} \end{aligned} \tag{3.3}$$

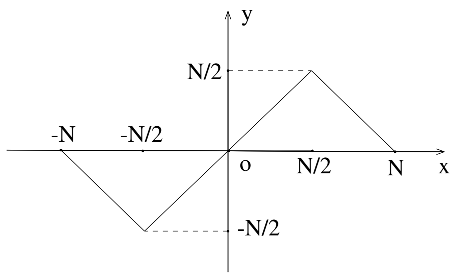

Fig. 4. Continuous programmable function f for bootstrapping lower bits.

When x is nearby $-N$ or 0, error $|x - f^{\text{init}}(x)|$ may exceed $N/2$ a little bit, then f enlarges the error furthermore, but not too much due to continuity. Only the block of plaintext that is not flooded by the error can be approximately backed up. The error is a mixture of the input ciphertext error, the modulus down switch error, the plaintext rounding error, and the programmable function branch misuse error.

From the viewpoint of two's complement representation, a block in $\mathbb{Z}_{2N}$ having two leading bits of identical value corresponds to a number of $(-N/2, N/2)$ in the interval $[-N, N)$ representation of $\mathbb{Z}_{2N}$. If the two leading bits are changed into different values by a rather small error, this only occurs when the number is of the binary form 110...0 or 001...1. In $[-N, N)$, the number is either $-N/2$ or almost $N/2$, which are exactly the two points from which the programmable function starts to deviate from the identity function.

3.2 Resolute numbers

In Subsection 2.1, we see that in a uni-bootstrap, to get rid of error e_{MDS}, it is sufficient for the programmable function f to satisfy (2.11). However, none of the three programmable functions: f' of Fig. 2, f^{init} of (3.1), and f of (3.3), meets the requirement. To revise these functions so that (2.11) is satisfied, we need a convenient concept to name a class of numbers that has been used popularly in LWE plaintext decoding.

Denote

$$\begin{aligned} \mathbb{Q}_{[p+1]} &:= \{x \in \mathbb{Q} \,\big|\, |x - y2^{p+1}| < 2^p, \ \exists\, y \in \mathbb{Z}\}, \\ \mathbb{Z}_{[p+1]} &:= \mathbb{Q}_{[p+1]} \cap \mathbb{Z}, \end{aligned} \tag{3.4}$$

and call them respectively *resolute rational numbers* and *resolute integers* with $(p+1)$-*bit resolution*. They can also be called $(p+1)$-*resolute numbers* unanimously. When $x \in \mathbb{Q}_q$ (or $\mathbb{Z}_q$) for some $q > 2^{p+1}$, then if a rational number (or integer) representative of x is in $\mathbb{Q}_{[p+1]}$ (or $\mathbb{Z}_{[p+1]}$), so are all other representatives of x. Hence for a modular number x, $x \in \mathbb{Q}_{[p+1]}$ (or $\mathbb{Z}_{[p+1]}$) makes sense. Usually we require $p \geq 0$, but $p = -1$ is still useful: $\mathbb{Q}_{[0]} = \mathbb{Q}$.

Let $x = y2^{p+1} + z \in \mathbb{Q}_{[p+1]}$ for some integer y and rational number z with $|z| < 2^p$. Then $y2^{p+1}$ is called the *center* of x, denoted by $(x)^{[p+1]}$, and z is called the *error* or *residue* of x. For a fixed $E_{\text{MDS}} > 0$, if $|z| < 2^p - E_{\text{MDS}}$, then x is said to be E_{MDS}-*tolerant* in $\mathbb{Q}_{[p+1]}$. If z is a random variable, E_{MDS} is the bound of another independent random variable, and $|z| < \sqrt{2^{2p} - E^2_{\text{MDS}}}$, then x is said to be *Pythagorean* E_{MDS}-*tolerant*.

Every $x \in \mathbb{Q}_q$ or $\mathbb{Z}_q$ has infinitely many rational or integral representatives of the form $x + kq$, where $k \in \mathbb{Z}$. Since comparison and ordering make sense only in $\mathbb{Z}$, modular integers should be given common interval representatives and then be compared as regular integers. Usually a zero-centered interval is used to offer representatives for modular numbers, because this representation gives smaller absolute value. For example, $x \in \mathbb{Z}_q$ can be represented by an integer in interval $[-q/2, q/2)$, and $|x|$ makes sense. Another example is the two's complement representation when q is a power of two, which is just the $[0, q)$-interval representation in binary form. Still these are not enough.

We introduce the following notations. The unique representative of $x \in \mathbb{Q}_q$ in interval $[a, a+q)$ can be denoted by $[x]_{[a,a+q)}$. The following are special shorthand notations:

$$\begin{aligned} [x]_{\pm q/2} &:= [x]_{[-q/2,q/2)}; \\ [x]_{\pm q/2-\epsilon} &:= [x]_{[-q/2-\epsilon,q/2-\epsilon)}. \end{aligned} \tag{3.5}$$

The default representative of $x \in \mathbb{Q}_q$ is $[x]_{\pm q/2}$.

When $x \in \mathbb{Q}$ is a regular number, the notation "$[\ \ ]_{[a,a+q)}$" serves as a map from $\mathbb{Q}$ to interval $[a, a+q)$, by taking x as a representative of some modular number in $\mathbb{Q}_q$. In this case, the simplified notations (3.5) also make sense.

We further introduce the following simplified notation: for any $x \in \mathbb{Q}_{[p+1]}$,

$$[x]^{[p+1]}_{\pm q/2} := [(x)^{[p+1]}]_{\pm q/2} \in \mathbb{Z} \cap [-q/2, q/2). \tag{3.6}$$

Notice that

$$[(x)^{[p+1]}]_{\pm q/2} = \big([x]_{\pm q/2-2^p}\big)^{[p+1]}. \tag{3.7}$$

The following are some simple facts:

– For any $q' < q$, both of which are powers of two, for any $x \in \mathbb{Q}_q$, $x - (x \mod q') \in \mathbb{Z}_{[l_{q'}]}$.
– For any two integers $y_1 \neq y_2$, $(y_1 2^{p+1} - 2^p, y_1 2^{p+1} + 2^p) \cap (y_2 2^{p+1} - 2^p, y_2 2^{p+1} + 2^p) = \emptyset$.
– $\mathbb{Q} \backslash \mathbb{Q}_{[p+1]} = \mathbb{Z} \backslash \mathbb{Z}_{[p+1]} = \{2^p(1+2k) \,\big|\, k \in \mathbb{Z}\}$.

For any $x \in \mathbb{Z}_q$, it has the following unique *0-centered tail-up blockwise representative*:

$$x = \sum_{i=0}^{u} y_i 2^{d'_{i-1}} \in [-q, q), \tag{3.8}$$

where $d'_j = d'_{j-1} + d_j$, $d'_{-1} = 0$, and $d_0, \ldots, d_u$ are a sequence of positive integers such that $d'_u = \sum_{i=0}^{u} d_i = l_q$; where for all $0 \leq i \leq u$, integer $y_i \in [-2^{d_i-1}, 2^{d_i-1})$ is unique.

If $x \in \mathbb{Z}_q$ is represented by (3.8), then for $u \geq 1$, $x \in \big[-(\sum_{i=0}^{u} 2^{d'_i})/2, (\sum_{i=0}^{u} 2^{d'_i})/2\big)$. In the extreme case where $d_i = 1$ for all i, then $u = l_q - 1$, and $\sum_{i=0}^{u} 2^{d_i-1} \times 2^{d'_{i-1}} = \sum_{i=0}^{u} 2^i = 2^{u+1} - 1 = q - 1$, so $x \in (-q, q)$. This representation is suitable for plaintexts, but not for errors, because errors are not modular integers.

The representative can be obtained as follows. In regular binary form, let $x = \sum_{i=0}^{u} x_i 2^{d'_{i-1}}$, where $|x_i| \leq 2^{d_i} - 1$, and if $x \geq 0$, then all the $x_i \geq 0$, while if $x < 0$, then all the $x_i \leq 0$. For $i = 0$ to u, do the following:

1. Compute the regular binary form of $x \mod q$, which is $x = \sum_{j=i}^{u} x_j 2^{d'_{j-1}}$.
2. If $x_i \geq 2^{d_i-1}$, set $y_i = -2^{d_i} + x_i$, $x = x + (2^{d_i} - x_i)2^{d'_{i-1}}$; else if $x_i < -2^{d_i-1}$, set $y_i = 2^{d_i} + x_i$, $x = x - (2^{d_i} + x_i)2^{d'_{i-1}}$; else, set $y_i = x_i$, $x = x - x_i 2^{d'_{i-1}}$.

For example, when $q = 2^4 = 16$, let $d_0 = d_1 = 2$, then $d'_0 = 2, d'_1 = 4$. In binary form, $7 = -9 \mod 16$ has the following 0-centered tail-up blockwise representative:

$$0111 = 11 + 01 \times 2^2 = -01 + (01 + 01) \times 2^2 = -01 + 10 \times 2^2 = -01 - 10 \times 2^2 =: y_0 + y_1 2^{d'_0}. \tag{3.9}$$

In blockwise bootstrapping, to clear a signed integer instead of a modular integer, it is the 0-centered blockwise representative that should be used. The two's complement representation changes every signed integer into non-negative integer, clearing such a representative in general does not clear the signed integer. This is another reason to use the two programmable functions f^{init}, f in blockwise bootstrapping. The two functions are themselves almost anti-symmetric with respect to the origin, making them well suited for generating 0-centered blockwise representatives for sign integers.

3.3 High-resolution variants of programmable functions

The following is a high-resolution variant of function f^{init} in (3.1), with resolution $p+1$ bits for some $0 \le p \le l_N - 1$. It is first defined on the subset $\mathbb{Z}_{[p+1]}$ of $\mathbb{Z}$, then extended to $\mathbb{Z}$ by $2N$-periodicity and by supplementing the points outside $\mathbb{Z}_{[p+1]}$ with value 0:

$$\begin{aligned} f_p^{\text{init}} : \mathbb{Z} &\longrightarrow \mathbb{Z}_{2N} \\ x &\mapsto \begin{cases} N/2, & \text{if } x \in \mathbb{Z}_{[p+1]},\ (x)^{[p+1]} \in [0, N); \\ -N/2, & \text{if } x \in \mathbb{Z}_{[p+1]},\ (x)^{[p+1]} \in [-N, 0); \\ f_p^{\text{init}}(x - 2^p), & \text{if } x \in \mathbb{Z} \backslash \mathbb{Z}_{[p+1]}. \end{cases} \end{aligned} \tag{3.10}$$

It is easy to see that f_p^{init} is $2N$-periodic nega-cyclic.

The following is a high-resolution variant of function f in (3.3), with resolution $p+1$ bits for some $0 \le p \le l_N - 2$, by first defining on the subset $\mathbb{Z}_{[p+1]}$ of $\mathbb{Z}$, then supplementing the points outside $\mathbb{Z}_{[p+1]}$ with suitable values and extending to $\mathbb{Z}$ by $2N$-periodicity:

$$\begin{aligned} f_p : \mathbb{Z} &\longrightarrow \mathbb{Z}_{2N} \\ x &\mapsto \begin{cases} (x)^{[p+1]}, & \text{if } x \in \mathbb{Z}_{[p+1]}, \text{ and} \\ & \quad (x)^{[p+1]} \in [-N/2, N/2]; \\ N - (x)^{[p+1]}, & \text{if } x \in \mathbb{Z}_{[p+1]}, \text{ and} \\ & \quad (x)^{[p+1]} \in [N/2, N]; \\ -N - (x)^{[p+1]}, & \text{if } x \in \mathbb{Z}_{[p+1]}, \text{ and} \\ & \quad (x)^{[p+1]} \in [-N, -N/2]; \\ y2^{p+1}, & \text{if } x = y2^{p+1} - 2^p \\ & \quad \in [-N/2, N/2],\ y \in \mathbb{Z}; \\ N - y2^{p+1}, & \text{if } x = y2^{p+1} - 2^p \\ & \quad \in [N/2, N],\ y \in \mathbb{Z}; \\ -N - y2^{p+1}, & \text{if } x = y2^{p+1} - 2^p \\ & \quad \in [-N, -N/2],\ y \in \mathbb{Z}. \end{cases} \end{aligned} \tag{3.11}$$

It is easy to see that f_p is $2N$-periodic nega-cyclic, and

$$f_p(x) \in 2^{p+1} \times \left([-N/2^{p+2}, N/2^{p+2}] \cap \mathbb{Z}\right),\ \forall x \in \mathbb{Z}. \tag{3.12}$$

In fact, for all $x \in \mathbb{Z}_{[p+1]} \cap \mathbb{Z}_{2N}$,

$$\begin{aligned} f_p^{\text{init}}(x) &= f^{\text{init}}([x]_{\pm N}^{[p+1]}) = \text{sign}(\lfloor x/2^{p+1} \rceil) \times N/2, \\ f_p(x) &= f([x]_{\pm N}^{[p+1]}) \quad = \lfloor f(x)/2^{p+1} \rceil \times 2^{p+1}, \end{aligned} \tag{3.13}$$

and for all $x \in (-N/2 - 2^p, 3N/2 + 2^p) \cap \mathbb{Z}$,

$$f_p(x) = \begin{cases} \lfloor x/2^{p+1} \rceil 2^{p+1}, & \text{if } x \in (-N/2 - 2^p, \\ & \qquad N/2 + 2^p); \\ N - \lfloor x/2^{p+1} \rceil 2^{p+1}, & \text{if } x \in (N/2 - 2^p, \\ & \qquad 3N/2 + 2^p). \end{cases} \tag{3.14}$$

Furthermore, it can be easily verified that for $x \in [-N, N)$,

$$f(x) = x - 2\text{sign}(x)\text{ReLU}(|x| - N/2), \tag{3.15}$$

and

$$f^{\text{init}} = f_{l_N - 1}. \tag{3.16}$$

Let $x = \sum_{j=0}^{l_N - 2} x_j 2^j \in [0, N/2)$. Then

$$f_p(x) = \sum_{j=p+1}^{l_N} x_j 2^j + x_p 2^{p+1}. \tag{3.17}$$

For $x = N/2 + \sum_{j=0}^{l_N-2} x_j 2^j \in [N/2, N)$,

$$f_p(x) = N/2 - \sum_{j=p+1}^{l_N} x_j 2^j - x_p 2^{p+1}. \tag{3.18}$$

expand by $2N$-periodic nega-cyclicity.

In general, we assume $p \geq 0$. Still sometimes negative resolution is needed. Notice that $f_{-1}^{\text{init}} = f^{\text{init}}$ and $f_{-1} = f$. For any $p < 0$, we can define

$$f_p^{\text{init}} = f^{\text{init}}, \quad f_p = f. \tag{3.19}$$

Lemma 1 Let $x_1, x_2 \in \mathbb{Z}$ such that $x_2 = x_1 + e_1$, where $|e_1| \leq E_1 < N/2$.

(1) If $|x_1| \leq N/2$, $E_1 < 2^p$ for some integer $p > 0$, and $x_1 \in \mathbb{Z}_{[p+1]}$ with error bound $\epsilon < 2^p - E_1$, *i.e.*, $|x_1 - y \times 2^{p+1}| \leq \epsilon$ for some integer y, then

$$|f_p(x_2) - x_1| \leq \epsilon. \tag{3.20}$$

(2) If $|x_1| \leq N/2 + \epsilon$ for some $\epsilon \in [0, N/2)$, and $|x_2| < N/2 + 2^{p+1} + 2^p \leq N$ for some integer $p \geq 0$, then

$$|f_p(x_2) - x_1| \leq \max(2^p + E_1, 2^{p+1} + \epsilon). \tag{3.21}$$

Proof. When $x_2 \in [-N/2, N/2]$, then $f(x_2) = x_2$ and $|f_p(x_2) - x_2| \leq 2^p$, so

$$|x_1 - f_p(x_2)| \leq |x_1 - x_2| + |x_2 - f_p(x_2)| \leq 2^p + E_1. \tag{3.22}$$

In particular, if $|x_1 - y \times 2^{p+1}| \leq \epsilon$ where $\epsilon < 2^p - E_1$, then $x_2 \in \mathbb{Z}_{[p+1]}$ with $(x_2)^{[p+1]} = y \times 2^{p+1}$, and $|x_1 - f_p(x_2)| \leq \epsilon$ follows. So when $x_2 \in [-N/2, N/2]$, both inequalities hold.

When $x_2 \in (N/2, N] \cup [-N, -N/2)$, by symmetry we only need to consider the situation where both x_1, x_2 are close to the border point $N/2$. Let $x_1 = N/2 + a$ and $x_2 = N/2 + b$, where $|a|, |b| < N/2$, $b > 0$ and $|a - b| \leq E_1$. Then $f(x_2) = N/2 - b$.

If $b \in (0, 2^p)$, then $f_p(x_2) = N/2$, and $f_p(x_2) - x_1 = -a$; if $b \in (0, 2^{p+1} + 2^p)$, then $f_p(x_2) \in \{N/2, N/2 - 2^{p+1}\}$, and $f_p(x_2) - x_1 \in \{-a, -a - 2^{p+1}\}$. There are two cases: (i) $a \leq 0$; (ii) $a > 0$.

In case (i), $x_1 \leq N/2 < x_2$. When $x_1 \in \mathbb{Z}_{[p+1]}$ and $E_1 < 2^p$, then $x_2 \in \mathbb{Z}_{[p+1]}$, $N/2$ is the common center of x_1, x_2, $-a = |x_1 - N/2| \leq \epsilon < 2^p - E_1$, so $|f_p(x_2) - x_1| = -a \leq \epsilon$. This proves (3.20).

When $x_2 < N/2 + 2^{p+1} + 2^p$, by $0 \leq -a < E_1$, we have $|f_p(x_2) - x_1| \leq |a + 2^{p+1}| \leq \max(E_1, 2^{p+1})$.

In case (ii), only (3.21) needs proof. So $0 < a \leq \epsilon$ and $0 < b < 2^{p+1} + 2^p$, and $|f_p(x_2) - x_1| \leq |a + 2^{p+1}| \leq 2^{p+1} + \epsilon$. Q.E.D.

The following lemma is the high-resolution version of (3.2).

Lemma 2 For all $0 \leq p \leq l_N - 1$, all $x \in \mathbb{Z}_{2N/2^{p+1}} \cap [-N/2^{p+1}, N/2^{p+1})$,

$$f_p^{\text{init}}(x2^{p+1}) + f_p(x2^{p+1} - f_p^{\text{init}}(x2^{p+1})) = x2^{p+1}. \tag{3.23}$$

Proof. Let $y = f_p^{\text{init}}(x2^{p+1})$ and $z = x2^{p+1} - y$. Now that $x2^{p+1} \in [-N, N) \cap \mathbb{Z}_{[p+1]}$, if $x2^{p+1} \in [-N, 0)$, then $y = -N/2$, $z \in [-N/2, N/2) \cap \mathbb{Z}_{[p+1]}$, so $f_p(z) = z$, and (3.23) is just the trivial relation $x2^{p+1} = y + z$. If $x2^{p+1} \in [0, N)$, then $y = N/2$, $z \in [-N/2, N/2) \cap \mathbb{Z}_{[p+1]}$, and $f_p(z) = z$. Again (3.23) is trivial. Q.E.D.

Lemma 3 Let $x_1 \in (-N, N)$, and let $x_2 - x_1 = e_0$ such that $2^p < |e_0| \leq N/2$ for some integer $p \geq 0$.

– If $x_2 \in [-N, N)$, then
$$|x_1 - f^{\text{init}}(x_2)| \leq N/2 + |e_0|. \tag{3.24}$$
If $x_2 \in [-N, N - 2^p) \cap \mathbb{Z}_{[p+1]}$, then
$$|x_1 - f_p^{\text{init}}(x_2)| \leq N/2 + |e_0|. \tag{3.25}$$
– If $x_2 \geq N$, then
$$|x_1 - 2N - f^{\text{init}}(x_2)| \leq N/2 + |e_0|. \tag{3.26}$$
If $x_2 \in [N, 3N/2) \cap \mathbb{Z}_{[p+1]}$, then
$$|x_1 - 2N - f_p^{\text{init}}(x_2)| \leq N/2 + |e_0|. \tag{3.27}$$
– If $x_2 < -N$, then
$$|x_1 + 2N - f^{\text{init}}(x_2)| \leq N/2 + |e_0|. \tag{3.28}$$
If $x_2 \in (-3N/2, -N - 2^p) \cap \mathbb{Z}_{[p+1]}$, then
$$|x_1 + 2N - f_p^{\text{init}}(x_2)| \leq N/2 + |e_0|. \tag{3.29}$$

Proof. Let
$$x_0 = \begin{cases} x_2 - f^{\text{init}}(x_2), & \text{if } x_2 \in [-N, N); \\ x_2 - 2N - f^{\text{init}}(x_2), & \text{if } x_2 \geq N; \\ x_2 + 2N - f^{\text{init}}(x_2), & \text{if } x_2 < N. \end{cases} \tag{3.30}$$
Then
$$x_1 - f^{\text{init}}(x_2) = (x_1 - x_2) + (x_2 - f^{\text{init}}(x_2)) = e_0 + x_0. \tag{3.31}$$
When $x_2 \in [-N, N)$, then $|x_0)| \leq N/2$. By
$$x_1 - f^{\text{init}}(x_2) = (x_1 - x_2) + (x_2 - f^{\text{init}}(x_2)) = e_0 + x_0, \tag{3.32}$$
we get $|x_1 - f^{\text{init}}(x_2)| \leq N/2 + |e_0|$.

When $x_2 \geq N$, then $N \leq x_2 < N + E \leq 3N/2$. So $-N \leq x_2 - 2N < -N/2$, $f^{\text{init}}(x_2) = f^{\text{init}}(x_2 - 2N) = -N/2$, and $-N/2 \leq x_0 < 0$. By
$$x_1 - 2N - f^{\text{init}}(x_2) = (x_1 - x_2) + (x_2 - 2N - f^{\text{init}}(x_2)) = e_0 + x_0, \tag{3.33}$$
we get $|x_1 - 2N - f^{\text{init}}(x_2)| \leq N/2 + |e_0|$.

When $x_2 < -N$, the proof is similar. When $x_2 \in \mathbb{Z}_{[p+1]}$, the proof for f_p^{init} is much the same. Q.E.D.

The following lemma is the counterpart of Lemma 1 for independent subgaussian errors.

Lemma 4 let e_0, e_1, e_2 be independent 0-centered subgaussian random variables with heuristic bound E_0, E_1, E_2 respectively, where $E_0 < N/2$, and $E_1 \leq E_0$. Let $x_2 = x_1 + e_1$ satisfy $|x_2| < N$.

1. If $|x_1| \leq N/2$, then for any $p' \geq 0$,
$$\begin{aligned} |x_1 - f(x_2) - e_2| &\leq \sqrt{E_1^2 + E_2^2}, \\ |x_1 - f_{p'}(x_2) - e_2| &\leq 2^{p'} + \sqrt{E_1^2 + E_2^2}. \end{aligned} \tag{3.34}$$

2. If $|x_1| \leq N/2+\epsilon < N$, then
$$|x_1 - f(x_2) - e_2| \leq 2\epsilon + \sqrt{E_1^2 + E_2^2}. \tag{3.35}$$
3. If $x_1 = x_0 + e_0$, where $|x_0| \leq N/2$, then
$$|x_1 - f(x_2) - e_2| \leq \sqrt{4E_0^2 + E_1^2 + E_2^2}. \tag{3.36}$$

If $|x_2| < N/2 + 2^p + 2^{p-1} \leq N$ for some integer $p > 1$, then for any $p' \leq p$,
$$\begin{aligned} |x_1 - f_{p-1}(x_2) - e_2| &\leq 2^p + \sqrt{E_0^2 + E_2^2}, \\ |x_1 - f_{p'}(x_2) - e_2| &\leq 2^{p+1} + \sqrt{E_0^2 + E_2^2} \end{aligned} \tag{3.37}$$

If instead $|x_2| < N/2 + 2^p \leq N$, then
$$|x_1 - f_p(x_2) - e_2| \leq 2^p + \sqrt{E_0^2 + E_2^2}. \tag{3.38}$$

Proof. If $x_2 \in [-N/2, N/2]$, then $f(x_2) = x_2$, and $f(x_2) - x_1 = e_1$. All conclusions are valid in this case. If $x_2 \notin [-N/2, N/2]$, by symmetry, assume $x_2 \in (N/2, N)$. There are three cases.

Case 1. $x_1 \in [-N/2, N/2]$. Since $x_2 > N/2$, $e_1 > 0$. It can be decomposed into $e_1 = e_1' + e_1''$, where $e_1' \geq 0$ and $e_1'' > 0$, such that $x_1 = N/2 - e_1'$ and $x_2 = N/2 + e_1''$. Then $f(x_2) = N/2 - e_1''$, and $x_1 - f(x_2) = e_1'' - e_1'$. Since $-e_1'' - e_1' \leq e_1'' - e_1' \leq e_1'' + e_1'$, the variance of $e_1'' - e_1'$ is bounded by that of e_1. So
$$|x_1 - f(x_2) - e_2| = |e_1'' - e_1' - e_2| \leq \sqrt{E_1^2 + E_2^2}. \tag{3.39}$$
For any $p' \geq 0$, $|x_1 - f_{p'}(x_2) - e_2| \leq |x_1 - f(x_2) - e_2| + |f(x_2) - f_{p'}(e_2)| \leq 2^{p'} + \sqrt{E_1^2 + E_2^2}$.

Case 2. $N/2 < x_2 \leq x_1 < N$. Let $x_1 = N/2 + a$ where $a > 0$. From $x_2 = N/2 + a + e_1$, we get $0 \geq e_1 > -a$. So $f(x_2) = N/2 - a - e_1$, and $x_1 - f(x_2) = e_1 + 2a$.

If $x_1 \leq N/2 + \epsilon$, then $a \leq \epsilon$, and
$$|x_1 - f(x_2) - e_2| = |e_1 + 2a - e_2| \leq 2a + |e_2| \leq 2\epsilon + E_2. \tag{3.40}$$

If $x_1 = x_0 + e_0$, where $|x_0| \leq N/2$ and $|e_0| \leq E_0$, then e_0 has decomposition $e_0 = e_0' + a$ where $e_0' \geq 0$, such that $x_0 = N/2 - e_0'$. Since $2a \geq e_1 + 2a > a$ and $0 < a < e_0$, the variance of $e_1 + 2a$ is bounded by that of $2e_0$ in this case, so
$$|x_1 - f(x_2) - e_2| = |e_1 + 2a - e_2| \leq \sqrt{4E_0^2 + E_2^2}. \tag{3.41}$$
If $x_2 < N/2 + 2^p$, then $f_p(x_2) = N/2$, and $|x_1 - f_p(x_2) - e_2| = |a - e_2| \leq \sqrt{E_0^2 + E_2^2}$.

If $x_2 < N/2 + 2^p + 2^{p-1}$, then $f_p(x_2) \in \{N/2, N/2 + 2^p\}$, and $x_1 - f_{p-1}(x_2) = a$ or $a - 2^p$, so $|x_1 - f_{p-1}(x_2) - e_2| \leq 2^p + \sqrt{E_0^2 + E_2^2}$. For any $p' < p-1$, $N/2 \leq f_{p'}(x_2) \leq N/2 + 2^p + 2^{p-1}$, $a \geq x_1 - f_{p-1}(x_2) \geq a - 2^p - 2^{p-1}$, so $|x_1 - f_{p'}(x_2) - e_2| \leq 2^{p+1} + \sqrt{E_0^2 + E_2^2}$. For $p' = p$, $f_p(x_2) \in \{N/2, N/2 + 2^{p+1}\}$, so $|x_1 - f_p(x_2) - e_2| \leq 2^{p+1} + \sqrt{E_0^2 + E_2^2}$.

Case 3. $N/2 < x_1 < x_2$. Again let $x_1 = N/2 + a$ where $a > 0$. Then $e_1 > 0$. From $x_2 = N/2 + a + e_1$, we get $f(x_2) = N/2 - a - e_1$, and $f(x_2) - x_1 = -e_1 - 2a$.

If $x_1 \leq N/2 + \epsilon$, then $a \leq \epsilon$, and $|x_1 - f(x_2) - e_2| = |e_1 + 2a - e_2| \leq 2\epsilon + \sqrt{E_1^2 + E_2^2}$.

If $x_1 = x_0 + e_0$ with $|x_0| \leq N/2$, again e_0 has decomposition $e_0 = e_0' + a$ where $e_0' \geq 0$, such that $x_0 = N/2 - e_0'$. The variance of a is bounded by that of e_0, so
$$|x_1 - f(x_2) - e_2| = |e_1 + 2a - e_2| \leq \sqrt{4E_0^2 + E_1^2 + E_2^2}. \tag{3.42}$$

Just as in Case 2, if $x_2 < N/2 + 2^p$, then $f_p(x_2) = N/2$. If $x_2 < N/2 + 2^p + 2^{p-1}$, then $f_{p-1}(x_2) \in \{N/2, N/2 + 2^p\}$, $f_p(x_2) \in \{N/2, N/2 + 2^{p+1}\}$, and for any $p' < p-1$, $N/2 \leq f_{p'}(x_2) \leq N/2 + 2^p + 2^{p-1}$. Both (3.37) and (3.38) remain valid in this case. Q.E.D.

4 Tail-up backup bootstrapping

For the purpose of both improving the bootstrapping efficiency and maximizing the initial error that allows tail-up bootstrapping, we completely re-develop the tail-up bootstrapping approach [25], [29]. There are the following advantages of the re-development:

– the bootstrapping efficiency is improved in that the size of every block is maximized;
– the initial error is maximized under the minimal requirement that the first block to be bootstrapped has at least two bits;
– the initial error is further maximized by the LSB precursor.

In contrast, in the literature all blocks have the same size.

The block bootstrapping from the tail up is composed of three groups of uni-bootstraps. The first group contains a couple of uni-bootstraps, and is designed to recover the tail block, whose size ranges from 2 up to $d+2$ bits, depending mainly on the input error size. The second group is composed of a sequence of uni-bootstrap couples, each recovering one block. All blocks of the group except for the first one, have size $d+2$ bits, while the first one may have size $d+1$ bits. The third group contains either one or two uni-bootstraps, depending on how many bits are left.

In the following, we first introduce each group of uni-bootstraps using phase simulation, with rigorous error analysis whose proofs can be found in the Appendix, then introduce the tail-up bootstrapping algorithm with illustration, and finally introduce the LSB precursor. In the following subsections,

– $m_0 \in \mathbb{Z}_q$ is the phase of the input ciphertext, or after modulo q, the plaintext with error; for $i > 0$, $m_i \in \mathbb{Z}_q$ is the leftover of m_0 after subtracting the results of the previous i uni-bootstraps.
– $m'_i \in \mathbb{Z}_q$ is the result of the i-th uni-bootstrap. When $i = 2j$, m'_i contains the MSB of the tail j-th block with error; when $i = 2j+1$, m'_i contains the lower bits of the tail j-th block with error.
– Each couple of phases m'_{2j} and m'_{2j+1} are generated by the same rounding function but two different programmable functions f^{init} and f respectively with appropriate resolution.
– $\check{m}_i$ is the last block of m_i. When $i = 2j$, $\hat{m}_i$ is the leftover of $\check{m}_{i-1}$ after subtracting m'_{i-1}.
– e_{Mi} is the modulus down switch error of the i-th uni-bootstrap, e_{Bi} is the refreshed error after the i-th uni-bootstrap, and e_{-i} is the error of block $\check{m}_i$.

4.1 First group of tail uni-bootstraps

Suppose the input BFV ciphertext encrypts plaintext $m_{\text{init}} \in \mathbb{Z}_t$, where $t \geq 2^2$. The phase of the ciphertext is

$$m_0 = m_{\text{init}}\Delta + e_{\text{init}} \in \mathbb{Z}_{[l_\Delta]}. \tag{4.1}$$

Its center is just the embedded plaintext: $[m_0]^{[l_\Delta]}_{\pm q/2} = \Delta[m_{\text{init}}]_{\pm t/2}$. We can image that at this beginning stage, there is a pointer that points at the end of the plaintext in $m_0 \in \mathbb{Z}_q$, which is the l_t-th bit counted from the left.

The first plaintext block to be bootstrapped has d_0+2 bits, where $0 \leq d_0 \leq d$ is the maximal integer satisfying

$$\sqrt{E^2_{\text{MDS}} + 2^{-2(d_0+2)}(E^2_{\text{init}} + E^2_B)(2N/\Delta)^2} < 2^{-(d_0+2)}N, \tag{4.2}$$

or in terms of $l_0 = l_N - d_0 - 2 \geq l$,

$$\sqrt{E^2_{\text{MDS}} + (E^2_{\text{init}} + E^2_B)(2^{l_0+1}/\Delta)^2} < 2^{l_0}. \tag{4.3}$$

The reason why this constraint is imposed will be clear very soon. The existence of $d_0 \geq 0$ is equivalent to the inequality

$$\sqrt{E^2_{\text{MDS}} + (E^2_{\text{init}} + E^2_B)N^2/(2\Delta)^2} < N/4. \tag{4.4}$$

It is the prerequisite for regular tail-up bootstrapping.

Let $d_0 \geq 0$ be given. If $d_0 + 2 \geq l_t$, then the first two groups are skipped. The first block to be backed up has d_0+2 bits, so we move the pointer leftward by d_0+2 bits, so that it points at the beginning of the block to be backed up. The truncated phase from the pointer down to the end of m_0, is a new phase denoted by an integer $\check{m}_0$. In the 0-centered tail-up blockwise representation of modular integer m_0, phase $\check{m}_0$ is just the last block of phase m_0.

Let $y_0 \in \mathbb{Z}_{2^{d_0+2}}$ be the plaintext in phase $\check{m}_0$. Then

$$y_0 = [m_{\text{init}}]_{\pm 2^{d_0+1}} \in [-2^{d_0+1}, 2^{d_0+1}), \tag{4.5}$$

and $\check{m}_0 = y_0 \Delta + e_{\text{init}} \in \mathbb{Z}_q$. In integer form,

$$\check{m}_0 = [m_0]_{\pm 2^{d_0+1}\Delta - \Delta/2} \in \mathbb{Z}_{2^{d_0+2}\Delta} \cap \mathbb{Z}_{[l_\Delta]}. \tag{4.6}$$

By $2^{d_0+2} = 2N/2^{l_0+1}$, in $\mathbb{Z}_{2N}$, the error must be $< N/2^{d_0+2} = 2^{l_0}$, so that there is a one-bit gap between the error and the plaintext. Accordingly, the first uni-bootstrap handles the MSB of the block using programmable function $f_{l_0}^{\text{init}}$, and the second uni-bootstrap handles the other bits of the block using programmable function f_{l_0}.

Each uni-bootstrap precedes by a modulus down switch from $\mathbb{Z}_{2^{d_0+2}\Delta}$ to $\mathbb{Z}_{2N}$. It can be simulated by function

$$r_{2^{l_0+1}/\Delta}(m, e) = m2^{l_0+1}/\Delta + e \in \mathbb{Z}, \tag{4.7}$$

where $m \in \mathbb{Z}$ and $|e| \leq E_{\text{MDS}}$. The function maps a phase m of $\mathbb{Z}_q$ to a phase of $\mathbb{Z}_{2N}$.

For simplicity, denote

$$x_0 = r_{2^{l_0+1}/\Delta}(\check{m}_0, e_{M0}), \tag{4.8}$$

where $|e_{M0}| \leq E_{\text{MDS}}$. The error part of phase x_0 is $e_{\text{init}}2^{l_0+1}/\Delta + e_{M0}$, and is bounded strictly by 2^{l_0}, according to (4.3). So $x_0 \in \mathbb{Z}_{[l_0+1]}$.

After the modulus down switch, the first uni-bootstrap returns a refreshed ciphertext in $\mathbb{Z}_q$ with phase

$$m_0' = f_{l_0}^{\text{init}}(x_0) \times \Delta/2^{l_0+1} + e_{B0} \mod q, \tag{4.9}$$

where $|e_{B0}| \leq E_B$. By $\check{m}_0 2^{l_0+1}/\Delta = y_0 2^{l_0+1} + e_{\text{init}}2^{l_0+1}/\Delta$, and $|e_{\text{init}}2^{l_0+1}/\Delta| < 2^{l_0}$, we get

$$f_{l_0}^{\text{init}}(x_0) = f_{l_0}^{\text{init}}(y_0 2^{l_0+1}). \tag{4.10}$$

Now subtract phase m_0' from the input phase m_0. The leftover is a phase denoted by

$$m_1 = m_0 - m_0' \mod q. \tag{4.11}$$

In the second uni-bootstrap, m_1 is to be converted to a phase in $\mathbb{Z}_{2N}$ by modulus down switch, where only the last (d_0+2)-bit plaintext block is preserved.

Let the (d_0+2)-bit plaintext block in phase m_1 be $y_1\Delta$. Then

$$y_1 = y_0 - f_{l_0}^{\text{init}}(x_0)/2^{l_0+1} = y_0 - (2/N) \times f_{l_0}^{\text{init}}(x_0) \times 2^{d_0}. \tag{4.12}$$

If $y_0 \in [-2^{d_0+1}, 0)$, then $f_{l_0}^{\text{init}}(x_0) = -N/2$, so $y_1 \in [-2^{d_0}, 2^{d_0})$. If $y_0 \in [0, 2^{d_0+1})$, then $f_{l_0}^{\text{init}}(x_0) = N/2$, again $y_1 \in [-2^{d_0}, 2^{d_0})$. So $|y_1| \leq 2^{d_0}$. When viewed as a modular integer in $\mathbb{Z}_{2^{d_0+2}}$, then y_1 has its first two bits having identical value.

The error part of phase m_1 is

$$e_{-1} := e_{\text{init}} - e_{B0}. \tag{4.13}$$

By (4.3), $|e_{-1}| < \Delta/2$. In fact, inequality prerequisite (4.3) provides exactly what is needed:

$$|e_{-1}2^{l_0+1}/\Delta| < \sqrt{2^{2l_0} - E_{\text{MDS}}^2}. \tag{4.14}$$

Combining error e_{-1} with the (d_0+2)-bit plaintext block $y_1\Delta$, we get a new phase represented by integer

$$\check{m}_1 = y_1\Delta + e_{-1} \in \ \mathbb{Z}_{[l_\Delta]} \cap [-2^{d_0+1}\Delta - \Delta/2, 2^{d_0+1}\Delta - \Delta/2). \tag{4.15}$$

Obviously, $\check{m}_1$ is the truncation of m_1 from the pointer to the right end:

$$\check{m}_1 = [m_1]_{\pm 2^{d_0+1}\Delta - \Delta/2} = \check{m}_0 - m_0' \mod 2^{d_0+2}\Delta. \tag{4.16}$$

Furthermore, by (4.14),

$$\check{m}_1 2^{l_0+1}/\Delta = y_1 2^{l_0+1} + e_{-1} 2^{l_0+1}/\Delta \in \mathbb{Q}_{[l_0+1]} \tag{4.17}$$

is Pythagorean E_{MDS}-tolerant.

The second uni-bootstrap precedes by the modulus down switch of phase m_1 that is simulated by $r_{2^{l_0+1}/\Delta}$. Denote the result by

$$x_1 = r_{2^{l_0+1}/\Delta}(\check{m}_1, e_{M1}), \tag{4.18}$$

where $|e_{M1}| \le E_{\mathrm{MDS}}$. By (4.17), $x_1 \in \mathbb{Z}_{[l_0+1]}$. After the modulus down switch, based on programmable function f_{l_0}, the second uni-bootstrap returns a refreshed phase in $\mathbb{Z}_q$ as follows:

$$m_1' = f_{l_0}(x_1) \times \Delta/2^{l_0+1} + e_{B1} \mod q. \tag{4.19}$$

Since $x_1 \in \mathbb{Z}_{[l_0+1]}$, $f_{l_0}(x_1) = y_1 2^{l_0+1}$. Now subtract m_1' from phase m_1:

$$m_2 = m_1 - m_1' \mod q. \tag{4.20}$$

Denote

$$e_{-2} = e_{-1} - e_{B1} = e_{\mathrm{init}} - e_{B0} - e_{B1}. \tag{4.21}$$

We check two things: Whether or not the last (d_0+2)-bit block of the plaintext in m_2 are cleared? If so, whether or not the plaintext in $m_0' + m_1'$ is exactly the cleared plaintext block of m_0?

Denote by $\hat{m}_2$ the last d_0+2 bits of plaintext in m_2 represented by an integer in the following interval:

$$\hat{m}_2 = [m_2]_{\pm 2^{d_0+1}\Delta - \Delta/2} \in \mathbb{Z}_{2^{d_0+2}\Delta}. \tag{4.22}$$

By $(N/2) \times \Delta/2^{l_0+1} = 2^{d_0}\Delta < 2^{d_0+2}\Delta$, we get $\hat{m}_2 = \check{m}_1 - m_1' \mod 2^{d_0+2}\Delta$. So

$$\hat{m}_2 = \check{m}_1 - y_1 2^{l_0+1} \times \Delta/2^{l_0+1} - e_{B1} = e_{-1} - e_{B1} \in (-3\Delta/4, 3\Delta/4). \tag{4.23}$$

As a corollary, $m_2 \in \mathbb{Z}_{[d_0+2+l_\Delta]}$ with error e_{-2} bounded by $3\Delta/4$.

By (3.23) and $y_0 2^{l_0+1} \in [-N, N)$,

$$f_{l_0}^{\mathrm{init}}(y_0 2^{l_0+1}) + f_{l_0}(y_0 2^{l_0+1} - f_{l_0}^{\mathrm{init}}(y_0 2^{l_0+1})) = y_0 2^{l_0+1}, \tag{4.24}$$

which when written as

$$(f_{l_0}^{\mathrm{init}}(y_0 2^{l_0+1}) + f_{l_0}(y_1 2^{l_0+1})) \times \Delta/2^{l_0+1} = [m_{\mathrm{init}}]_{\pm 2^{d_0+1}} \times \Delta, \tag{4.25}$$

gives

$$[m_0' + m_1']_{\pm q/2}^{[l_\Delta]} = [m_{\mathrm{init}}]_{\pm 2^{d_0+1}} \times \Delta. \tag{4.26}$$

In 0-centered tail-up blockwise representation, the last (d_0+2)-bit block of m_{init} is accurately backed up into $m_0' + m_1'$ as the plaintext.

For example, if $m_{\mathrm{init}} = 1011$, $d_0 = 0$, then the first block to be bootstrapped is composed of the last two bits 11 of m_{init}, so $y_0 = [1011]_{\pm 2} = -1$. In the first uni-bootstrap, $f_{l_0}^{\mathrm{init}}(y_0 2^{l_0+1}) = -N/2$. In the second uni-bootstrap, $y_1 = y_0 - (2/N) \times f_{l_0}^{\mathrm{init}}(x_0) \times 2^{d_0} = 0$. So $m_0' + m_1'$ backs up the last block of m_{init} in 0-centered tail-up blockwise representation, which is $y_0 + y_1 = -1$.

In the first group, the key parameter is the size $d_0 + 2$ of the last block. Since d_0 is the biggest integer satisfying (4.2), we have

$$d_0 = l_N - 3 + \left\lceil (1/2)\log\left(1 - 4(E_{\mathrm{init}}^2 + E_B^2)/\Delta^2\right) - \log E_{\mathrm{MDS}} \right\rceil. \tag{4.27}$$

The computation can be greatly simplified if using in power-of-2 binomial bounds.

In the Appendix, Lemma 14 and Lemma 17 are on the existence of $d_0 \ge 0$ when the error bounds are in power-of-2 binomial form. For small initial error, d_0 not only exists, but is at least $d-1$. For large initial error, $k_I \le 2\min(d, d_B)$ is usually required. Lemma 15 and Lemma 16 provides complete lists of all possible results of d_0. For small initial error, d_0 is always between $d-1$ and d; when $d_0 = d$, the tail-up bootstrapping is said to be in *greedy mode*. For large initial error, d_0 ranges from 2 to $d-1$.

4.2 Second group of tail uni-bootstraps

The input to the second group is the leftover phase $m_2 \in \mathbb{Z}_{[d_0+2+l_\Delta]}$ with error e_{-2}. The second group consists of $w \geq 0$ pairs of uni-bootstraps, where the i-th pair backs up a d_i-bit block of plaintext. Integer w satisfies

$$d'_w < l_t, \quad d'_{w+1} \geq l_t. \tag{4.28}$$

Let $d'_{-1} = 0$, and for $1 \leq i \leq w+1$, let

$$d'_{i-1} = d'_{i-2} + d_{i-1} + 2 = \sum_{j=0}^{i-1}(d_j + 2). \tag{4.29}$$

For $2 \leq i \leq w$, d'_{i-1} is the number of bits backed up by the first $i-1$ pairs of uni-bootstraps in the second group; when $i = 1$, $d'_0 = d_0 + 2$ is the number of bits backed up in the first group.

By Lemma 15, if $d_I > 0$, then the first block has $d'_0 \geq d+1$ bits. That $d+1 \geq (k+2)/2-1$, namely $l_N \geq l+1+k/2$ is always satisfied in our experiments. So for small initial error, in practice every block in the second group has $d+2$ bits.

If $d_I = 0$, by Lemma 16, the first block has $2 \leq d'_0 \leq d+1$ bits. By Lemma 18, the second block has at least $d+1$ bits, so $d'_1 \geq d+3$. That $d+3 \geq (k+2)/2$ is just $l_N \geq l + k/2$. So for large initial error, in practice from the second block to the last block in the second group, each block has $d+2$ bits, while the first block in the second group has $d+1$ to $d+2$ bits.

In general, after finishing $h+1$ uni-bootstraps in total, where $h \geq 1$, the leftover of the input phase m_0 after subtracting the sum of phases $\sum_{j=0}^{h} m'_j$ obtained by the previous $h+1$ uni-bootstraps, is

$$m_h := m_{h-1} - m'_{h-1} = m_0 - \sum_{j=0}^{h-1} m'_j \mod q. \tag{4.30}$$

The error part of m_h is

$$e_{-h} := e_{-(h-1)} - e_{B(h-1)} = e_{\text{init}} - \sum_{j=0}^{h-1} e_{Bj} \mod q, \tag{4.31}$$

where e_{Bh} is the refreshed error in m'_h. Error e_{-h} is bounded by $\sqrt{E^2_{\text{init}} + (h+1)E^2_B}$.

For $h \in \{2i-1, 2i\}$ where $i > 0$, to bootstrap the last plaintext block in m_h, whose block size is $d_i + 2$, it is mandatory that after modulus down switch from $\mathbb{Z}_{2^{d'_i}\Delta}$ to $\mathbb{Z}_{2N}$, the error part does not overlap with the plaintext part. The plaintext part in $\mathbb{Z}_{2N}$ occupies the first $d_i + 2$ bits, so the error part occupies only the last

$$l_N - (d_i + 2) =: l_i \tag{4.32}$$

bits. In $\mathbb{Z}_{2N}$, the error part must be bounded by

$$\sqrt{E^2_{\text{MDS}} + 2^{-2d'_i}(E^2_{\text{init}} + (h+1)E^2_B) \times (2N/\Delta)^2} < 2^{l_i}, \tag{4.33}$$

or in terms of d_i,

$$2^{2d_i}(4/N)^2 E^2_{\text{MDS}} + 2^{-2d'_{i-1}}(E^2_{\text{init}} + (h+1)E^2_B)(2/\Delta)^2 < 1. \tag{4.34}$$

For fixed $d'_{i-1} > 0$, let d_i be the biggest integer satisfying (4.34) for $h = 2i$, then

$$d_i = l_N - 3 + \left\lceil (1/2)\log\left(1 - 4^{1-d'_{i-1}}(E^2_{\text{init}} + (2i+1)E^2_B)/\Delta^2\right) - \log E_{\text{MDS}} \right\rceil. \tag{4.35}$$

This formula can be used to compute d_i. Theoretically, the second group consists of the following loop:

1. Set $i = 1$. Set $d'_0 = d_0 + 2$. Compute d_1.
2. While $d'_i < l_t$, do the following:
   (a) Do uni-bootstrap to phase m_{2i} by programmable function $f^{\text{init}}_{l_i}$. Denote the resulting phase by m'_{2i}.
   (b) Do phase subtraction: $m_{2i+1} = m'_{2i} - m_{2i} \mod q$.

(c) Do uni-bootstrap to phase m_{2i+1} by programmable function f_{l_i}. Denote the resulting phase by m'_{2i+1}.
(d) Do phase subtraction: $m_{2i+2} = m'_{2i+1} - m_{2i+1} \mod q$.
(e) Set $i = i + 1$. Compute d_i.

Let E_{fin} be the error bound of the output ciphertext after the whole bootstrapping. The *quality constraint of bootstrapping* refers to $E_{\text{fin}} \leq E_{\text{init}}/2$. The inequality ensures that the output ciphertext has an error bound that is at most half that of the input ciphertext, so that the refreshed ciphertext can undergo either one addition or one multiplication followed by modulus switch. This is a very humble requirement on the quality of the bootstrapping.

If there are all together v uni-bootstraps in the whole bootstrapping, then

$$E_{\text{fin}} = \sqrt{v}E_B \leq E_{\text{init}}/2. \tag{4.36}$$

On the other hand, $\sqrt{v} \leq E_{\text{init}}/2E_B$ imposes a strong constraint on the number of uni-bootstraps. As a corollary,

$$\sqrt{E_{\text{init}}^2 + vE_B^2} \leq \sqrt{5}\, E_{\text{init}}/2 < (\sqrt{5}/4)\varDelta < 3\varDelta/4. \tag{4.37}$$

To simplify the computation of d_i, and also to gain better insight of the second group, we can use the fact that $i+1 < v$, the latter being the total number of uni-bootstraps, to enlarge $(i+1)E_B^2$ to

$$(i+1)E_B^2 \leq (v-1)E_B^2 = E_{\text{fin}}^2 - E_B^2 \leq E_{\text{init}}^2/4 - E_B^2. \tag{4.38}$$

Then (4.34) can be strenghtened to

$$2^{2d_i}E_{\text{MDS}}^2(4/N)^2 + 2^{-2d'_{i-1}}((1+2^{-2})E_{\text{init}}^2 - E_B^2)(2/\varDelta)^2 < 1. \tag{4.39}$$

The values of d_i for $1 \leq i \leq w$ under the strenghtened constraint (4.39) in power-of-2 binomial bounds, are given by Lemma 18 in the Appendix. For small initial error, in practice each block in the second group has $d+2$ bits; for large initial error, in practice from the second block to the last block in the second group, each block has $d+2$ bits, while the first block in the second group has $d+1$ to $d+2$ bits.

In the rest of this subsection, we make step-by-step analysis of the error growth and plaintext backup accuracy during the second group. Assume $w > 0$ and the d_i are given. The first block in the second group has $d_1 + 2$ bits, so we move the pointer in the phase space $\mathbb{Z}_q$ leftward by $d_1 + 2$ bits, so that it points at the beginning of the block to be backed up. The truncated phase from the pointer to the right end, is a new phase denoted by an integer $\check{m}_2$. In the 0-centered tail-up blockwise representation of modular integer m_2, phase $\check{m}_2$ is just the last block of phase m_2:

$$\check{m}_2 = [m_2]_{\pm 2^{d'_1-1}\varDelta - 3\varDelta/4} \ \in \ \mathbb{Z}_{2^{d'_1}\varDelta}. \tag{4.40}$$

Inherited from m_2, phase $\check{m}_2 \in \mathbb{Z}_{[d'_0+l_\varDelta]}$ with error e_{-2}:

$$\check{m}_2 = y_2 \times 2^{d'_0}\varDelta + e_{-2} \tag{4.41}$$

for some $y_2 \in [-2^{d_1+1}, 2^{d_1+1})$. In fact, by $(d_1+1) + (d'_0) = d'_1 - 1$ and (4.26),

$$\begin{aligned}
[y_2]_{\pm 2^{d_1+1}} \times 2^{d'_0}\varDelta &= [m_0 - m'_0 - m'_1]^{[l_\varDelta]}_{\pm 2^{d'_1-1}\varDelta} \\
&= [m_0]^{[l_\varDelta]}_{\pm 2^{d'_1-1}\varDelta} - [m'_0 + m'_1]^{[l_\varDelta]}_{\pm 2^{d'_1-1}\varDelta} \\
&= [m_0]^{[l_\varDelta]}_{\pm 2^{d'_1-1}\varDelta} - [m_0]^{[l_\varDelta]}_{\pm 2^{d_0+1}\varDelta} \\
&= \left([m_{\text{init}}]_{\pm 2^{d'_1-1}} - [m_{\text{init}}]_{\pm 2^{d'_0-1}}\right) \times \varDelta.
\end{aligned} \tag{4.42}$$

In 0-centered tail-up blockwise representation, $y_2/\varDelta$ is the next to the last block of m_{init}.

For example, if $m_{\text{init}} = 1011$, $d_0 = d_1 = 0$, then $d'_0 = 2$, and the first block backs up the last block -1 in 0-centered tail-up blockwise representation, and the leftover phase m_2 contains plaintext $1011 - (-1) = 1100$. In the second group, the last two vacant bits of the leftover plaintext will not be backed up. The first two bits 11 will be backed up just as in the first group. The result will be $-1 \times 2^{d'_0} = -100$ in binary form. Summing it up with the backed up result -1 of the first group, the 0-centered tail-up blockwise representation $-100-1=-101$ of m_{init} mod 2^4 is recovered.

To bootstrap the first block in the second group, by $2N/(\Delta \times 2^{d'_1}) = 2^{1-d'_1}N/\Delta = 2^{l_1+1-d'_0}/\Delta$, the rounding function for modulus down switch is

$$r_{2^{l_1+1-d'_0}/\Delta}(m,e) = m2^{l_1+1-d'_0}/\Delta + e \in \mathbb{Z}. \tag{4.43}$$

Denote

$$x_2 = r_{2^{l_1+1-d'_0}/\Delta}(\check{m}_2, e_{M2}), \tag{4.44}$$

where $|e_{M2}| \le E_{\text{MDS}}$.

By (4.41),

$$\check{m}_2 2^{l_1+1-d'_0}/\Delta = y_2 \times 2^{l_1+1} + e_{-2} \times 2^{l_1+1-d'_0}/\Delta. \tag{4.45}$$

By (4.33),

$$\sqrt{(e_{-2}^2(2^{l_1+1-d'_0}/\Delta)^2 + E_{\text{MDS}}^2} < 2^{l_1}. \tag{4.46}$$

so $x_2 \in \mathbb{Z}_{[l_1+1]}$.

The first uni-bootstrap of the block gives

$$\begin{aligned} m'_2 &= f_{l_1}^{\text{init}}(x_2) \times \Delta/2^{l_1+1-d'_0} + e_{B2} \mod q, \\ m_3 &= m_2 - m'_2 \mod q, \\ \check{m}_3 &= [m_3]_{\pm 2^{d'_1-1}\Delta - 3\Delta/4}, \end{aligned} \tag{4.47}$$

where $|e_{B2}| \le E_B$. Integer $\check{m}_3$ is chosen to represent the leftover phase m_3.

We have

$$f_{l_1}^{\text{init}}(x_2) = f_{l_1}^{\text{init}}(y_2\, 2^{l_1+1}). \tag{4.48}$$

Furthermore, since $(N/2) \times \Delta/2^{l_1+1-d'_0} = 2^{d'_1-2}\Delta$, we have

$$\check{m}_3 = \check{m}_2 - m'_2 \mod 2^{d'_1}\Delta. \tag{4.49}$$

Below we check the resolute structure and bound of integer $\check{m}_3$.

Let

$$y_3 = y_2 - (2/N) \times f_{l_1}^{\text{init}}(x_2) \times 2^{d_1} \in \mathbb{Z}. \tag{4.50}$$

From $y_2 \in [-2^{d_1+1}, 2^{d_1+1})$, we get $y_3 \in [-2^{d_1}, 2^{d_1})$. So

$$\begin{aligned} \check{m}_3 &= y_2\, 2^{d'_0}\Delta + e_{-2} - f_{l_1}^{\text{init}}(y_2\, 2^{l_1+1}) \times \Delta/2^{l_1+1-d'_0} - e_{B2} \\ &= y_3\, 2^{d'_0}\Delta + e_{-3} \in \mathbb{Z}_{[d'_0+l_\Delta]} \cap [-2^{d'_1-1}\Delta - 3\Delta/4, 2^{d'_1-1}\Delta - 3\Delta/4). \end{aligned} \tag{4.51}$$

As to the bound of $\check{m}_3$, by (4.51) and (4.33),

$$|\check{m}_3 2^{l_1+1-d'_0}/\Delta| \le N/2 + |e_{-3}| \times 2^{l_1+1-d'_0}/\Delta < N/2 + \sqrt{2^{2l_1} - E_{\text{MDS}}^2}. \tag{4.52}$$

For the same reason,

$$|\check{m}_3 2^{l_1+1-d'_0}/\Delta - y_3\, 2^{l_1+1}| \le |e_{-3}| \times 2^{l_1+1-d'_0}/\Delta < \sqrt{2^{2l_1} - E_{\text{MDS}}^2}. \tag{4.53}$$

So $\check{m}_3 2^{l_1+1-d'_0}/\Delta \in \mathbb{Q}_{[l_1+1]}$ is Pythagorean E_{MDS}-tolerant.

The second uni-bootstrap of the block is realized by acting programmable function f_{l_1} on the modulus down switch result

$$x_3 := r_{2^{l_1+1-d'_0}/\Delta}(\check{m}_3, e_{M3}), \tag{4.54}$$

where $|e_{M3}| \le E_{\text{MDS}}$. By (4.52) and (4.53), using (3.20), we get $f_{l_1}(x_3) = y_3 2^{l_1+1}$.

The second uni-bootstrap gives

$$\begin{aligned} m_3' &= f_{l_1}(x_3) \times \Delta/2^{l_1+1-d_0'} + e_{B3} \mod q, \\ m_4 &= m_3 - m_3' \mod q, \\ \hat{m}_4 &= [m_4]_{\pm 2^{d_1'-1}\Delta - 3\Delta/4}, \end{aligned} \tag{4.55}$$

where $|e_{B3}| \le E_B$.

Again by $(N/2) \times \Delta/2^{l_1+1-d_0'} = 2^{d_1'-2}\Delta$, we have $\hat{m}_4 = \check{m}_3 - m_3' \mod 2^{d_1'}\Delta$. So

$$\hat{m}_4 = \check{m}_3 - y_3 2^{l_1+1} \times \Delta/2^{l_1+1-d_0'} - e_{B3} = e_{-3} - e_{B3}, \tag{4.56}$$

both $\hat{m}_4, m_4 \in \mathbb{Z}_{[d_1'+l_\Delta]}$ have error bound $3\Delta/4$.

By (3.23),

$$f_{l_1}^{\text{init}}(y_2 2^{l_1+1}) + f_{l_1}(y_2 2^{l_1+1} - f_{l_1}^{\text{init}}(y_2 2^{l_1+1})) = y_2 2^{l_1+1}, \tag{4.57}$$

which when written as

$$(f_{l_1}^{\text{init}}(y_2 2^{l_1+1}) + f_{l_1}(y_3 2^{l_1+1})) \times \Delta/2^{l_1+1-d_0'} = ([m_{\text{init}}]_{\pm 2^{d_1'-1}} - [m_{\text{init}}]_{\pm 2^{d_0'-1}}) \times \Delta, \tag{4.58}$$

gives

$$[m_2' + m_3']_{\pm q/2}^{[d_0'+l_\Delta]} = \left([m_{\text{init}}]_{\pm 2^{d_1'-1}} - [m_{\text{init}}]_{\pm 2^{d_0'-1}}\right) \times \Delta. \tag{4.59}$$

In 0-centered tail-up blockwise representation, $m_2' + m_3'$ backs up the next to the last block of m_{init}.

Extending the above results by mathematical induction to other blocks of the second group is straightforward. We present the following conclusion but omit its proof.

Lemma 5 For $i \ge 1$, let $r_{2^{l_i+1-d_{i-1}'}/\Delta}(m, e) = m2^{l_i+1-d_{i-1}'}/\Delta + e \in \mathbb{Z}$. For $j \in \{2i, 2i+1\}$, denote

$$x_j = r_{2^{l_i+1-d_{i-1}'}/\Delta}(\check{m}_j, e_{Mj}), \tag{4.60}$$

where $|e_{Mj}| \le E_{\text{MDS}}$. Define

$$\begin{aligned} \check{m}_{2i} &= [m_{2i}]_{\pm 2^{d_i'-1}\Delta - 3\Delta/4}, \\ m_{2i}' &= f_{l_i}^{\text{init}}(x_{2i}) \times \Delta/2^{l_i+1-d_{i-1}'} + e_{B(2i)} \mod q, \\ m_{2i+1} &= m_{2i} - m_{2i}' \mod q, \\ \check{m}_{2i+1} &= [m_{2i+1}]_{\pm 2^{d_i'-1}\Delta - 3\Delta/4}, \\ m_{2i+1}' &= f_{l_i}(x_{2i+1}) \times \Delta/2^{l_i+1-d_{i-1}'} + e_{B(2i+1)} \mod q, \\ m_{2i+2} &= m_{2i+1} - m_{2i+1}' \mod q, \end{aligned} \tag{4.61}$$

where $|e_{Bj}| \le E_B$. Then

1. $x_{2i}, x_{2i+1} \in \mathbb{Z}_{[l_i+1]}$.
2. $\check{m}_{2i}, \check{m}_{2i+1} \in \mathbb{Z}_{[d_{i-1}'+l_\Delta]}$, with errors e_{-2i}, $e_{-(2i+1)}$ bounded by $3\Delta/4$. In more details,

$$\begin{aligned} \check{m}_{2i} &= y_{2i}\, 2^{d_{i-1}'}\Delta + e_{-2i}, \\ \check{m}_{2i+1} &= y_{2i+1}\, 2^{d_{i-1}'}\Delta + e_{-(2i+1)}, \end{aligned} \tag{4.62}$$

where integers $y_{2i} \in [-2^{d_i+1}, 2^{d_i+1})$, $y_{2i+1} \in [-2^{d_i}, 2^{d_i})$ satisfy

$$\begin{aligned} y_{2i}\, 2^{d_{i-1}'} &= [m_{\text{init}}]_{\pm 2^{d_i'-1}} - [m_{\text{init}}]_{\pm 2^{d_{i-1}'-1}}, \\ y_{2i+1} &= y_{2i} - f_{l_i}^{\text{init}}(y_{2i}\, 2^{l_i+1})/2^{l_i+1}. \end{aligned} \tag{4.63}$$

3. $m'_{2i}, m'_{2i+1} \in \mathbb{Z}_{2^{d'_i}\Delta}$, and

$$[m'_{2i} + m'_{2i+1}]^{[d'_{i-1}+l_\Delta]}_{\pm q/2} = \left([m_{\text{init}}]_{\pm 2^{d'_i-1}} - [m_{\text{init}}]_{\pm 2^{d'_{i-1}-1}}\right) \times \Delta. \tag{4.64}$$

4. $m_{2i+2} \in \mathbb{Z}_{[d'_i+l_\Delta]}$ with error $e_{-(2i+2)}$ bounded by $3\Delta/4$.

After the second group, the number of remaining bits in plaintext m_{init} is

$$d_{\text{last}} := l_t - d'_w \in [1, d_w + 2]. \tag{4.65}$$

The leftover of m_{init} is $m_{2w+2} \in \mathbb{Z}_{[d'_w+l_\Delta]}$, with error $e_{-(2w+2)}$ bounded by $3\Delta/4$.

4.3 Third group of tail uni-bootstraps

In the third group, if $d_{\text{last}} = 1$, then only one uni-bootstrap based on function $f^{\text{init}}_{l_N-1} - N/2$ is needed, otherwise two uni-bootstraps are needed as before. Denote

$$l_e := l_N - d_{\text{last}}. \tag{4.66}$$

Then $l_e \in [l_w, l_N - 1]$.

First consider the simple case $d_{\text{last}} = 1$. Then $m_{2w+2} \in \mathbb{Z}_{[l_q-1]} \cap \mathbb{Z}_q$. Let

$$\check{m}_{2w+2} = [m_{2w+2}]_{\pm q/2-3\Delta/4}. \tag{4.67}$$

Then

$$\check{m}_{2w+2} = y_{2w+2} \times q/2 + e_{-(2w+2)} \tag{4.68}$$

for some integer $y_{2w+2} \in \{0, -1\}$, and $\check{m}_{2w+2} \times (2N/q) \in \mathbb{Q}_{[l_N]}$ is Pythagorean E_{MDS}-tolerant, by (4.34).

The modulus down switch in the uni-bootstrap uses the rounding function $r_{2N/q}(m, e) = m(2N/q) + e \in \mathbb{Z}$. Denote

$$x_{2w+2} = r_{2N/q}(\check{m}_{2w+2}, e_{M(2w+2)}), \tag{4.69}$$

where $|e_{M(2w+2)}| \le E_{\text{MDS}}$. Then $x_{2w+2} \in \mathbb{Z}_{[l_N]}$.

The uni-bootstrap is based on function $f^{\text{init}}_{l_N-1} - N/2$, so that

$$f^{\text{init}}_{l_N-1}(x_{2w+2}) - N/2 = f^{\text{init}}_{l}(y_{2w+2}N) - N/2 = y_{2w+2}N. \tag{4.70}$$

The uni-bootstrap gives

$$m'_{2w+2} = f^{\text{init}}_{l_N-1}(x_{2w+2}) \times q/(2N) + e_{B(2w+2)} - q/4 \mod q, \tag{4.71}$$

where $|e_{B(2w+2)}| \le E_B$. By (4.70),

$$m'_{2w+2} - e_{B(2w+2)} = y_{2w+2} \times q/2. \tag{4.72}$$

Next consider the general case $d_{\text{last}} > 1$. Then (4.67) is still valid, but now that $m_{2w+2} \in \mathbb{Z}_{[l_q-d_{\text{last}}]}$, (4.68) is replaced by

$$\check{m}_{2w+2} = y_{2w+2} \times q/2^{d_{\text{last}}} + e_{-(2w+2)} \tag{4.73}$$

for some integer $y_{2w+2} \in [-2^{d_{\text{last}}-1}, 2^{d_{\text{last}}-1})$. In fact, by (4.64),

$$\begin{aligned}
&[y_{2w+2}]_{\pm 2^{d_{\text{last}}-1}} \times q/2^{d_{\text{last}}} \\
&= [m_0 - \textstyle\sum_{j=0}^{2w+1} m'_i]^{[l_\Delta]}_{\pm q/2} \\
&= [m_0]^{[l_\Delta]}_{\pm q/2} - \textstyle\sum_{j=0}^{w} [m'_{2j} + m'_{2j+1}]^{[l_\Delta]}_{\pm q/2} \\
&= \left([m_{\text{init}}]_{\pm t/2} - [m_{\text{init}}]_{\pm 2^{d'_w-1}}\right) \times \Delta.
\end{aligned} \tag{4.74}$$

Furthermore, by (4.34) and $d_{w+1} \ge d_{\text{last}}$, $\check{m}_{2w+2} \times (2N/q) \in \mathbb{Q}_{[l_e+1]}$ is Pythagorean E_{MDS}-tolerant.

The first uni-bootstrap in the third group makes modulus down switch to $\check{m}_{2w+2}$ by $r_{2N/q}$ to get $x_{2w+2} \in \mathbb{Z}_{[l_e+1]}$. The uni-bootstrap is based on $f_{l_e}^{\text{init}}$, and gives let

$$\begin{aligned} m'_{2w+2} &= f_{l_e}^{\text{init}}(x_{2w+2}) \times q/(2N) + e_{B(2w+2)} \mod q, \\ m_{2w+3} &= m_{2w+2} - m'_{2w+2} \mod q, \\ \check{m}_{2w+3} &= [m_{2w+3}]_{\pm q/2-3\Delta/4}, \end{aligned} \tag{4.75}$$

where $|e_{B(2w+2)}| \leq E_B$.

Similar to the case of the second group, we have $f_{l_e}^{\text{init}}(x_{2w+2}) = f_{l_e}^{\text{init}}(y_{2w+2} \times 2^{l_e+1})$. Let

$$y_{2w+3} = y_{2w+2} - (2/N) \times f_{l_e}^{\text{init}}(x_{2w+2}) \times 2^{d_{\text{last}}-2}. \tag{4.76}$$

Then $y_{2w+3} \in [-2^{d_{\text{last}}-2}, 2^{d_{\text{last}}-2})$. Furthermore,

$$\begin{aligned} \check{m}_{2w+3} &= y_{2w+2} \times q/2^{d_{\text{last}}} - f_{l_e}^{\text{init}}(x_{2w+2}) \times q/(2N) - e_{B(2w+2)} \\ &= y_{2w+3} \times q/2^{d_{\text{last}}} + e_{-(2w+3)}. \end{aligned} \tag{4.77}$$

By (4.34), $\check{m}_{2w+3} \times (2N/q) \in \mathbb{Z}_{[l_e+1]}$ is Pythagorean E_{MDS}-tolerant.

Denote

$$x_{2w+3} = r_{2N/q}(\check{m}_{2w+3}, e_{2w+3}), \tag{4.78}$$

where $|e_{2w+3}| \leq E_{\text{MDS}}$. Then $f_{l_e}(x_{2w+3}) = y_{2w+3} \times 2^{l_e+1}$. The second uni-bootstrap of the third group is based on f_{l_e}, and gives

$$m'_{2w+3} = f_{l_e}(x_{2w+3}) \times q/(2N) + e_{B(2w+3)} \mod q, \tag{4.79}$$

where $|e_{B(2w+3)}| \leq E_B$. By (3.23),

$$\begin{aligned} m'_{2w+2} + m'_{2w+3} - e_{B(2w+2)} - e_{B(2w+3)} &= \{f_{l_e}^{\text{init}}(y_{2w+2}\, 2^{l_e+1}) + f_{l_e}(y_{2w+3}\, 2^{l_e+1})\} \times q/(2N) \\ &= y_{2w+2} \times q/2^{d_{\text{last}}}. \end{aligned} \tag{4.80}$$

Theorem 6 The correctness of the whole tail-up bootstrapping is guaranteed by

$$\lfloor \sum_{i=0}^{v_T-1} m'_i/\Delta \rceil = \lfloor m_0/\Delta \rceil \mod t. \tag{4.81}$$

Proof. When $d_{\text{last}} = 1$, (4.74) is still valid, *i.e.*,

$$[y_{2w+2}]_{\pm 1} \times q/2 = [m_0]_{\pm q/2}^{[l_\Delta]} - \sum_{j=0}^{w} [m'_{2j} + m'_{2j+1}]_{\pm q/2}^{[l_\Delta]}. \tag{4.82}$$

By (4.72), $[m'_{2w+2}]_{\pm q/2}^{[l_\Delta]} = [y_{2w+2}]_{\pm 1} \times q/2$. By (4.36),

$$[m_0]_{\pm q/2}^{[l_\Delta]} = [m'_{2w+2}]_{\pm q/2}^{[l_\Delta]} + \sum_{j=0}^{w} [m'_{2j} + m'_{2j+1}]_{\pm q/2}^{[l_\Delta]} = \left[\sum_{j=0}^{2w+2} m'_j \right]_{\pm q/2}^{[l_\Delta]}. \tag{4.83}$$

When $d_{\text{last}} > 1$, by (4.80),

$$[y_{2w+2}]_{\pm 2^{d_{\text{last}}-1}} \times q/2^{d_{\text{last}}} = [m'_{2w+2} + m'_{2w+3}]_{\pm q/2}^{[l_\Delta]}. \tag{4.84}$$

Then by (4.74) and (4.36),

$$[m_0]_{\pm q/2}^{[l_\Delta]} = [m'_{2w+2} + m'_{2w+3}]_{\pm q/2}^{[l_\Delta]} + \sum_{j=0}^{w} [m'_{2j} + m'_{2j+1}]_{\pm q/2}^{[l_\Delta]} = \left[\sum_{j=0}^{2w+3} m'_j \right]_{\pm q/2}^{[l_\Delta]}. \tag{4.85}$$

Q.E.D.

4.4 Putting everything together into ciphertext form

In FHEW/TFHE ciphertext bootstrapping, assume that the following items are all global and public after the safety parameter is fixed:

– parameters $n, q, t, \delta = q/t$ of the input LWE ciphertext `ct`;
– parameters N, Q, B_g, B_{ks} of the $\text{GSW}_{N,Q}$ working environment, where B_g is the base integer of the gadget matrix in GSW encryption, and B_{ks} is the base integer for key switch from $\text{RLWE}_{N,Q}$ to $\text{LWE}_{n,q}$;
– public keys for bootstrapping in $\text{GSW}_{N,Q}$ working environment; public keys for key switch from $\text{RLWE}_{N,Q}$ to $\text{LWE}_{n,q}$;
– error bound E_{init} of the input LWE ciphertext; error bound E_{MDS} for modulus down switch; error bound E_B for refreshed error after a uni-bootstrap; In practice, these parameters may be given in power-of-2 binomial form.

The uni-bootstrap procedure $\texttt{UniBoot}(f, \texttt{ct}, q')$, on input $\text{LWE}_{n,q}$ ciphertext `ct`, programmable function f and modulus $q' \in (2N, q]$, outputs another $\text{LWE}_{n,q}$ ciphertext, by the following two steps: first modulus down switch from q' to $2N$, then accumulative blind rotation:

1. $\texttt{ModDownSwitch}(\texttt{ct}, q')$:
   Taking $\texttt{ct} \in \mathbb{Z}_q^{n+1}$ as a ciphertext with modulus q', make modulus down switch to obtain a ciphertext $\texttt{ct'} \in \mathbb{Z}_{2N}^{n+1}$.
2. $\texttt{AccumuBlindRot}(f, \texttt{ct'})$:
   For $\texttt{ct'} = (a'_1, \ldots, a'_n, b') \in \mathbb{Z}_{2N}^{n+1}$, start with a trivial $\text{RLWE}_{N,Q}$ encryption of a plaintext constructed by b' and f, multiply it successively with GSW ciphertexts encrypting $x^{a'_i s_i}$ for i from 1 to n.
   From the above multiplication result, which is an $\text{RLWE}_{N,Q}$ ciphertext, extract the constant term of the encrypted plaintext polynomial homomorphically to get an $\text{LWE}_{N,Q}$ ciphertext, then make key switch and modulus switch to obtain an $\text{LWE}_{n,q}$ ciphertext.

Details of $\texttt{UniBoot}(f, \texttt{ct}, q')$ and its two sub-procedures can be found in standard FHEW reference [14] and TFHE reference [7].

Algorithm 1 "`TailBoot`": Backup bootstrapping from tail up

```
Input: LWE ciphertext ct to bootstrap;
    public FHEW/TFHE parameters.
    Assume the bit-size of the first block $\geq 2$.
Output: LWE ciphertext ct'.
    {First group}
1: Compute $d_0, l_0 = l_N - d_0 - 2, d' = d_0 + 2$. ($d_0 + 2$ is the bit-size of the first block)
2: If $d' + d_0 + 2 \geq l_t$ then go to 7: Third group.
     ct' $\longleftarrow$ UniBoot($f_{l_0}^{\text{init}}$, ct, $2^{d'}\Delta$);
     ct $\longleftarrow$ ct $-$ ct' mod $q$;
     ct' $\longleftarrow$ ct' $+$ UniBoot($f_{l_0}$, ct, $2^{d'}\Delta$) mod $q$;
     ct $\longleftarrow$ ct $-$ ct' mod $q$.
    {Second group}
3: Compute the bit-size of the first block of the second group, still denoted by $d_0 + 2$.
4: while $d' + d_0 + 2 < l_t$ do
5:     ct' $\longleftarrow$ ct' $+$ UniBoot($f_{l_0}^{\text{init}}$, ct, $2^{d'}\Delta$) mod $q$;
       ct $\longleftarrow$ ct $-$ ct' mod $q$;
       ct' $\longleftarrow$ ct' $+$ UniBoot($f_{l_0}$, ct, $2^{d'}\Delta$) mod $q$;
       ct $\longleftarrow$ ct $-$ ct' mod $q$;
       set $d' = d' + d_0 + 2$;
       compute the bit-size of the next block, still denoted by $d_0 + 2$; compute $l_0 = l_N - d_0 - 2$.
6: end while
    {Third group}
```

```
7:  Compute $d_{\text{last}} = l_t - d'$, $l_e = l_N - d_{\text{last}}$. ($d_{\text{last}}$ is the bit-size of the last block)
8:  ct' ⟵ ct' + UniBoot($f^{\text{init}}_{l_e}$, ct, $q$)  mod $q$.
9:  if $d_{\text{last}} = 1$ then
10:     ct' ⟵ ct' − $q/4$.
11: else
12:     ct ⟵ ct − ct'  mod $q$;
        ct' ⟵ ct' + UniBoot($f_{l_e}$, ct, $q$)  mod $q$.
13: end if
14: return ct'.
```

Obviously we can merge the first group into the second group as a unified **while**-loop. Separating the two groups facilitates replacing the first group with the LSB precursor to be introduced in the next subsection.

The complexity of the algorithm is dominated by the number of uni-bootstraps. For small initial error, the first block has $d+1$ to $d+2$ bits, and the second group are all $(d+2)$-bit blocks, if not empty. We use "$\texttt{is}$(statement)" to denote the Boolean function that returns 1 if the argument "statement" is true, and 0 if it is false. For example,

$$\texttt{is}((d+2)|l_t) = \begin{cases} 1, \text{ if } (d+2)|l_t, \\ 0, \text{ else}, \end{cases} \tag{4.86}$$

i.e., $\texttt{is}((d+2)|l_t) = 1$ if and only if $d_{\text{last}} = 1$.

If the first block has $d+2$ bits, then the tail-up bootstrapping is in greedy mode, and the total number of uni-bootstraps is

$$v'_T := 2\lceil l_t/(d+2)\rceil - \texttt{is}((d+2) \mid (l_t - 1)). \tag{4.87}$$

In the general case, the total number of uni-bootstraps for small initial error is

$$v_T := 2\lceil (l_t+1)/(d+2)\rceil - \texttt{is}((d+2) \mid l_t). \tag{4.88}$$

For example, when $d_I = 1$ and $k = 2$, by Lemma 15, the tail-up bootstrapping is in greedy mode. When $d = 4$, for input plaintext of $d+2 = 6$ bits, a single block suffices to finish the bootstrapping.

For large initial error, in the worst case the first block contains only 2 bits. For example, if $l = k = 5$, $d_I = 0$, $d \geq 3$, $d_B = d+1$ and $k_I = 2d_B - 4 = 2d - 2 > 3$, then by Lemma 16, the first block has 2 bits; by Lemma 18, the second block has $d-1$ bits. When $d = 4$, for input plaintext of 6 bits, three blocks and 5 uni-bootstraps are needed to finish the bootstrapping, the bit-sizes of which are $2, 3, 1$ respectively.

4.5 LSB precursor for large initial error

The previous subsections have introduced the tail-up bootstrapping based on the requirement that the first block to be bootstrapped has at least two bits. This imposes a constraint on the initial error, such that in general $k_I \leq 2\min(d, d_B)$ by Lemma 14. When the initial error goes beyond this bound, can the tail-up bootstrapping still be executed? In this subsection, we present a simple method to loosen the requirement by allowing the first block to be bootstrapped to have only one bit. By Lemma 19, in power-of-2 binomial bounds, the new method requires $k_I \leq 2d+3$.

The new method is essentially an LSB-first approach, in that the LSB of the plaintext is first backed up, then the other blocks are backed up sequentially from the tail up as usual. The purpose of plaintext LSB backup is to reduce the large initial error to a small one with respect to the new error bound Δ, instead of the old bound $\Delta/2$.

The LSB-first approach to tail-up bootstrapping also consists of three groups. The second group and the third group are the same as before. The first group contains only one uni-bootstrap, which is based on programmable function $f^{\text{init}}_{l_N-1} - N/2$, just as the case of $d_{\text{last}} = 1$ in the third group of regular tail-up bootstrapping.

The first uni-bootstrap, or equivalently, the first group of uni-bootstraps, is called the *LSB precursor*. It makes modulus down switch from $\mathbb{Z}_{2\Delta}$ to $\mathbb{Z}_{2N}$, so that the plaintext occupies only the first bit of $\mathbb{Z}_{2N}$, and the error bound must be $N/2$. For the initial error E_{MDS}, this gives the constraint

$$\sqrt{E^2_{\text{init}}(N/\Delta)^2 + E^2_{\text{MDS}}} < N/2, \tag{4.89}$$

which is the inequality prerequisite for LSB precursor. Lemma 19 characterizes the constraint in power-of-2 binomial bounds, which allows all small initial errors, and allows all large initial errors where $k_I \le 2d+3$.

As before, let $m_0 = \Delta m_{\text{init}} + e_{\text{init}} \mod q$. Let

$$\check{m}_0 = [m_0]_{\pm\Delta - \Delta/2}. \tag{4.90}$$

Then $\check{m}_0 = y_0 \times \Delta + e_{\text{init}}$ for some $y_0 \in \{0, -1\}$. The modulus down switch uses rounding function $r_{N/\Delta}$.

The uni-bootstrap on the input phase represented by integer $\check{m}_0$, outputs a new phase $m'_0 \in \mathbb{Z}_q$. Let

$$m_2 = m_0 - m'_0 \mod q. \tag{4.91}$$

Then just like (4.72), $[m'_0]^{[l_\Delta]}_{\pm q/2} = [y_0]_{\pm 1} \times q/2$, so that $m_2 = (m_{\text{init}} - y_0)\Delta \in \mathbb{Z}_{[l_\Delta+1]}$, with error bound $\sqrt{E^2_{\text{init}} + E^2_B}$.

As to the second group and the third group, for large initial error, after the LSB precursor finishes, by Lemma 18 for $d'_0 = 1$, the second block has at least $d+1$ bits, and has $d+2$ bits if and only if $k = 3$, and either $k_I = 2$, or $k_I = 3$ and $d_B = 2$. From the third block to the next to the last block, each block has $d+2$ bits.

In algorithm 1, the first group is changed into the following:

- Set $l_0 = l_N - 1, d' = 1$.
- `ct'` $\longleftarrow$ `UniBoot`$(f^{\text{init}}_{l_0}$, `ct`, $2^{d'}\Delta) - \Delta/2$;
- `ct` $\longleftarrow$ `ct` $-$ `ct'` $\mod q$.

In Lemma 19, the inequality $k \le 2d+4$ can be written as $l_N \ge l + k/2$, which is always satisfied in our experiments. For large initial error, (4.89) requires $k_I \le 2d+3$. On the other hand, $k_I \le l_I \le l_\Delta - 1$. When $2d+3 \ge l_\Delta - 1$, namely $l_\Delta \le 2(d+2)$, then (4.89) is true for the extreme initial error value $E_{\text{init}} = \Delta/2 - 1$. For parameters $l_q = 29, d = 4$, $l_\Delta \le 2(d+2)$ is equivalent to $l_t \ge 17$.

Compared with the LSB-first approach, the regular tail-up bootstrapping is generally more efficient. By Lemma 16 and Lemma 18, when the second block after the LSB precursor has d+2 bits, then in regular tail-up bootstrapping, the first group has $d+1$ bits if $k_I = 2$, and d bits if $k_I = 3$; the second block always has $d+2$ bits. When $d = 4$, then in the LSB-first approach, the first two blocks contain $d+3 = 7$ bits, so each uni-bootstrap in the first two blocks handles $7/3$ bits on average. In contrast, in the regular approach the first two blocks contain at least $2d+2 = 10$ bits, so each uni-bootstrap in the first two blocks handles at least $10/4 > 7/3$ bits on average.

When the second block after the LSB precursor has $d+1$ bits, then in regular tail-up bootstrapping, the first group has 2 to $d+1$ bits, the second block has $d+1$ to $d+2$ bits. When $d = 4$, the LSB-first approach is more efficient if the regular approach has only 2 bits in its first group. However, if the first group in the regular approach has more than 2 bits, then by Lemma 18, for $k \le 5$ in our experiments, the second block has $d+2$ bits, so the regular approach is more efficient.

Usually a large-precision plaintext means that l_t is large. But how large is large? There should be some ruler to measure with. When compared with bit-lengths in $\mathbb{Z}_{2N}$, An input plaintext of bit-size $l_t \le d+2$ is said to be *short*. It is *medium* if $d+2 < l_t \le l_N$, and *long* if $l_t > l_N$.

From the viewpoint of tail-up bootstrapping, a ciphertext is said to have *single-block plaintext*, if under the error bound of the ciphertext, the plaintext it encrypts can be bootstrapped as a single block. Single-bit plaintexts are trivially single-block ones. Besides this special class, other single-block plaintexts are provided in Lemma 15 and Lemma 16, where the block size d_0+2 is denoted by d_{single}, so that a plaintext is single-block in a ciphertext if and only if its bit-length $\le d_{\text{single}}$.

By Lemma 15, for small initial error, $d_{\text{single}} \in \{d+1, d+2\}$; for large initial error, $d_{\text{single}} \in \{2, d+1\}$. So all single-blocked plaintexts are short.

When compared with bit-lengths in $\mathbb{Z}_q$, a plaintext is said to be *error-squeezing* if $l_t \ge l_q - l_B - 3$. By $d_B = l_q - l_t - l_B - 1$, this implies $d_B \le 2$. The limit of l_t is $l_q - l_B - 1$, for which $l_I = l_B$, so no backup bootstrapping is necessary. That $l_t = l_q - l_B - 3$ is the limit for plaintext bootstrapping; $l_t = l_q - l_B - 2$ is the limit for effective error bootstrapping; $l_t = l_q - l_B - 1$ is the limit for sign bit extraction.

Usually $l_q - l_B > 2l_N$, so $l_q - l_B - 3 > l_N$, and error-squeezing plaintexts are the longest plaintexts. From the viewpoint of plaintext bootstrapping, a large-precision plaintext should be defined to be either medium or long, namely $l_t > d+2$. Indeed, in greedy mode a short plaintext can be bootstrapped as a single block from the tail up.

For example, when $l_N = 11$, $d = 4$, $l_q = 29$, $l_B = 6$, small plaintexts have size at most 6 bits; medium plaintexts have size from 7 to 11 bits, large plaintexts have size over 11 bits, very large plaintexts have size 20 to 22 bits.

4.6 Plaintext bootstrapping in two's complement representation

In the previous subsections we have been concentrating on obtaining the 0-centered blockwise representative of the plaintext in a ciphertext. In practice it is more useful to obtain the popular two's complement representative of the plaintext. This subsection deals with this problem.

First, the programmable function f need to be replaced by the discontinuous function f' depicted in Fig. 2, namely

$$\begin{aligned} &f' : \mathbb{Z} \longrightarrow \mathbb{Z}_{2N} \\ &x \mapsto \begin{cases} y, & \text{if } x = 2kN + y, \quad k, y \in \mathbb{Z}, \quad y \in [0, N); \\ -y, & \text{if } x = (2k+1)N + y, \quad \text{with } k, y \text{ as above.} \end{cases} \end{aligned} \tag{4.92}$$

The resolution-p version of f', denoted by f'_p, where $0 \le p \le l_N - 1$, is

$$\begin{aligned} &f'_p : \mathbb{Z} \longrightarrow \mathbb{Z}_{2N} \\ &x \mapsto \begin{cases} y2^{p+1}, & \text{if } x = 2kN + y2^{p+1} + z, \quad k, y, z \in \mathbb{Z}, \quad y \in [0, N/2^{p+1}), \quad z \in [-2^p, 2^p); \\ -y2^{p+1}, & \text{if } x = (2k+1)N + y2^{p+1} + z, \quad \text{with } k, y, z \text{ as above.} \end{cases} \end{aligned} \tag{4.93}$$

Notice that $f'_{l_N-1} \neq f^{\text{init}}$, instead, $f'_{l_N-1}(x) = 0$ for all $x \in \mathbb{Z}$.

The following lemma is the counterpart of (3.23).

Lemma 7 For all $0 \le p \le l_N - 1$, all $x \in \mathbb{Z}_{2N/2^{p+1}} \cap [-N/2^{p+1}, N/2^{p+1})$,

$$f_p^{\text{init}}(x2^{p+1}) - N/2 + f'_p(x2^{p+1} - f_p^{\text{init}}(x2^{p+1}) + N/2) = x2^{p+1}. \tag{4.94}$$

Proof. Let $y = f_p^{\text{init}}(x2^{p+1}) - N/2$ and $z = x2^{p+1} - y$. Now that $x2^{p+1} \in [-N, N) \cap \mathbb{Z}_{[p+1]}$, if $x2^{p+1} \in [-N, 0)$, then $y = -N$, $z = x2^{p+1} + N \in [0, N) \cap \mathbb{Z}_{[p+1]}$, so $f'_p(z) = z$, and (4.94) is just the trivial relation $x2^{p+1} = y + z$. If $x2^{p+1} \in [0, N)$, then $y = 0$, $z = x2^{p+1} \in [0, N) \cap \mathbb{Z}_{[p+1]}$, again $f'_p(z) = z$, and (4.94) is still the trivial relation $x2^{p+1} = y + z$. Q.E.D.

By the above lemma, in plaintext bootstrapping from the tail up, each block is backed up by two uni-bootstraps, the first is based on $f_p^{\text{init}} - N/2$ where p is the same as before, the second is based on f'_p. The leftover error bound after each uni-bootstrap is also the same as before. In Algorithm 1, every f_p is replaced by f'_p, and every f_p^{init} is replaced by $f_p^{\text{init}} - N/2$. Furthermore in the third group, the command "if $d_{\text{last}} = 1$, then `ct'` $\longleftarrow$ `ct'` $- q/4$, else ......" is replaced by "if $d_{\text{last}} \neq 1$, then ......".

5 Head-on backup bootstrapping

From the viewpoint of plaintext blockwise representation, the tail-up bootstrapping based on f^{init}, f recovers the 0-centered tail-up blockwise representative of the input plaintext. An alternative approach to backup bootstrapping is from the head on. In this approach, the plaintext blockwise representation is dramatically different in that practically every block of plaintext except for the last one is at random to some scale. With the ongoing of blockwise bootstrapping, the plaintext error (not the ciphertext error!) of the backup result from the input decreases, and in the end the plaintext error disappears.

For example, for plaintext $m_0 = 01011100 \in \mathbb{Z}_{2^8}$ in binary form, suppose there are three blocks of size 3,3,2 bits respectively.

- The first block gives the following approximate to m_0: $m'_0 = 01010011$. Then $m_0 - m'_0 = 00001011$, so m_1 approximates m_0 with 3 bits of precision: $|m_0 - m'_0| \times 2^{-8} < 2^{-3}$. Denote $m_1 = m_0 - m'_0 = 1011$.
- The second block gives the following approximate to m_1: $m'_1 = 1001$. Then $m_1 - m'_1 = 0010$, so m_1 approximates m_0 with 6 bits of precision: $|m_1 - m'_1| \times 2^{-8} = 2^{-6}$. Denote $m_2 = m_1 - m'_1 = 11$.
- The third block is just $m'_2 = m_2$. So $m_0 = m'_0 + m'_1 + m'_2$, with the property that each block improves the precision of approximation.

Let $m_0 \in \mathbb{Z}_t$. let there be a sequence of non-negative integers $d_0, \ldots, d_u$ whose sum equals l_t, and where only d_u can be zero. Let $d'_{-1} = 0$, and for $i > 0$, let $d'_i = d'_{i-1} + d_i$. If there is a sequence of integers $m'_0, \ldots, m'_u \in [-t, t)$, such that (1) $\check{m}_0 = \sum_{i=0}^{u} m'_i$ for some $\check{m}_0 \in (-t, t)$, (2) $\check{m}_0 = m_0 \mod t$, (3) for all $0 \le i < u$, $|\check{m}_0 - \sum_{j=0}^{i} m'_j| \le t/2^{d'_i}$, then $\sum_{i=0}^{u} m'_i$ is called a *0-centered head-on blockwise representative* of modular number m_0.

For any two integers $x, y \in [-q/2, q/2)$, if $|x - y| \le q/2^{p+1}$ for some $p \ge 0$, they are said to approximate each other with p-bit *precision*. The bits of precision are counted from the left, while the bits of resolution are counted from the right.

For example, if $q = 2^3$, then in binary form, $011, 010, 001, 000$ approximate 000 with 0-bit, 1-bit, 2-bit, 3-bit precision, respectively. Another example is the 0-centered head-on blockwise representative $\sum_{i=0}^{u} m'_i$ of integer $\check{m}_0$: for all $0 \le j < u$, $\sum_{i=0}^{j} m'_i$ approximates $\check{m}_0$ with $(d'_j - 1)$-bit precision.

In addition to the different recovered representatives, the head-on bootstrapping has the following two major differences:

– For small initial error, each block is bootstrapped by one uni-bootstrap; for large initial error, only the first block may need two uni-bootstrap. In contrast, the tail-up approach generally requires two uni-bootstraps for each block.
– Most blocks have $d + 1$ bits. In contrast, most blocks in the tail-up approach have $d + 2$ blocks.

Because of the above differences, the head-on approach is more efficient: on average one uni-bootstrap handles $d+1$ bits of plaintext, while in the tail-up approach, averagely one uni-bootstrap handles $d/2 + 1$ bits of plaintext.

In bootstrapping from the tail up, the rounding error generated by modulus down switch is always separated from the plaintext in $\mathbb{Z}_{2N}$, it does not influence the resulting plaintext of each uni-bootstrap. In sharp contrast, during a head-on bootstrapping, every rounding error generated by modulus down switch floods the plaintext blocks that are not yet bootstrapped. Because of this, the 0-centered head-on blockwise representative of the input plaintext is random and denies accurate prediction.

For single-block plaintexts, the head-on approach agrees with the tail-up approach. So in the head-on approach, it is always assumed that $l_t > d_{\text{single}}$. The head-on block bootstrapping also consists of three groups of uni-bootstraps. The first group is composed of one couple of uni-bootstraps, and aims at approximately backing up the head block of the plaintext. The second group is composed of a sequence of single uni-bootstraps, each backing up one block approximately. The third group contains at most one uni-bootstrap.

As before, we shall first introduce each group by phase simulation, then introduce the general algorithm for head-on ciphertext bootstrapping, and finally introduce the LSB precursor in the end.

In the following subsections,

– $d_0 + 2$ is the size of the first block; for $i \ge 1$, d_i is the size of the $(i+1)$-st plaintext block; d'_i is the sum of d_j for $0 \le j \le i$; d_e is the maximal size allowed for the last block for correct decryption.
– $l_{s0} = l_s = l_N - l_t$, and for $i \ge 1$, l_{si} is the resolution of programmable function f in bootstrapping the $(i+1)$-st block.
– $m_0 \in \mathbb{Z}_q$ is the original plaintext with error; for $i > 0$, $m_i \in \mathbb{Z}_q$ is the leftover of m_0 after subtracting the results of the previous i uni-bootstraps.
– $m'_i \in \mathbb{Z}_q$ is the result of the i-th uni-bootstrap. m'_0 involves the MSB of the whole block; m'_1 approximates the lower bits of the leading block; for $i > 1$, m'_i approximates the i-th block. $\check{m}_0 \in (-q, q) \cap \mathbb{Z}_{[l_\Delta]}$ has its center equal to the sum of plaintexts in all the m'_i, with each plaintext taking value in $[-q/2, q/2)$.
– e_{Mi} is the rounding error in modulus down switch of the i-th uni-bootstrap, and e_{Bi} is the refreshed error by the i-th uni-bootstrap.

In this section, we always use $[-q/2, q/2)$ interval representatives for modular numbers in $\mathbb{Z}_q$, and use $[-N, N)$ interval representatives for modular numbers in $\mathbb{Q}_{2N}$ by default. We assume that the plaintext is not single-block, so that the total number of uni-bootstraps $v \ge 3$.

5.1 First group of head uni-bootstraps

In the head-on approach, if the initial error is small, then the first group is skipped. For large initial error, two uni-bootstraps are used to backup the leading block of m_{init} approximately. The first uni-bootstrap uses f^{init} to obtain the MSB of the block approximately, and the second one uses f with appropriate resolution to obtain the lower bits approximately.

First consider long plaintexts. For such plaintexts, any resolution of f is correct. Preceding the first uni-bootstrap, the modulus down switch is from $\mathbb{Z}_q$ to $\mathbb{Z}_{2N}$, where the rounding error occupies the last l bits of $\mathbb{Z}_{2N}$. For long plaintexts, $t \geq 2N$, the end of the input plaintext is outside the range of $\mathbb{Z}_{2N}$. Any resolution of f can keep the value of the function within the plaintext part in the refreshed ciphertext of $\mathbb{Z}_q$. Similarly, since $f^{\text{init}}(x) \in \{\pm N/2\}$, the value of f^{init} occupies only the first two bits of $\mathbb{Z}_{2N}$, as long as $l_t \geq 2$, any resolution of f^{init} keeps the value of the function within the plaintext part in the refreshed ciphertext of $\mathbb{Z}_q$.

Now that any resolution of the programmable functions is acceptable, the only concern is on the precision of approximation of the resulting phase to the input phase. As to f^{init}, for any $x \in [-N, N)$, any $p \geq 0$,

$$|f^{\text{init}}(x) - x| \leq N/4, \quad |f_p^{\text{init}}(x) - x| \leq N/4 + 2^p, \tag{5.1}$$

so $f^{\text{init}}(x)$ approximates x with 1-bit precision, and introducing any non-negative resolution will reduce the precision. The resolution-free version of f^{init} is optimal.

For function f, the story is different. Since $f(x) = x$ for $x \in [-N, N)$, when the last l bits of x is flooded by the rounding error, accurate information of the leading plaintext block can only be obtained from the first $l_N - l + 1$ bits of $\mathbb{Z}_{2N}$. If $p > l$, then $f_p(x)$ truncates x by extracting its information only from the first $l_N - p + 1$ bits, and the precision cannot be bigger than using f_l. On the other hand, if $p < l$, then $f_p(x)$ extracts information of x from not only its exposed part, but also from the covered part by the rounding error; the smaller the value of p, the more the covered part involved. To gain the best precision of approximation, p should be bounded from the top by l, and somehow close to l.

The modulus down switch from $\mathbb{Z}_q$ to $\mathbb{Z}_{2N}$ is simulated by function $r_{2N/q}$. For the input phase $m_0 = \Delta m_{\text{init}} + e_{\text{init}}$, denote

$$x_0 = r_{2N/q}(m_0, e_{M0}) = m_0 \times (2N/q) + e_{M0} \in \mathbb{Z}, \tag{5.2}$$

where $|e_{M0}| \leq E_{\text{MDS}}$.

The first uni-bootstrap gives a new phase

$$m_0' = f^{\text{init}}(x_0) \times q/(2N) + e_{B0} \mod q, \tag{5.3}$$

where $|e_{B0}| \leq E_B$. As $m_0' \in [-q/2, q/2)$ by default integer representation, and $e_{B0} \ll q$, integer $m_0' = f^{\text{init}}(x_0) \times q/(2N) + e_{B0}$.

We first investigate the plaintext in phase m_0'. Although $[m_0]_{\pm q/2} \times (2N/q) \in [-N, N)$, $x_0 \in (-2N, 2N)$ may go beyond this interval, so that $|x_0 - f^{\text{init}}(x_0)| \leq N/4$ is no longer true. We need to find some rational number $\check{m}_0 \times (2N/q) \in (-N, N)$ satisfying $\check{m}_0 = m_0 \mod q$, such that integer

$$\check{x}_0 := \check{m}_0 \times (2N/q) + e_{M0} \tag{5.4}$$

satisfies $\check{x}_0 = x_0 \mod 2N$ and $|\check{x}_0 - f^{\text{init}}(\check{x}_0)| \leq N/4$. Notice that the plaintext part is always mixed with the error part in phase x_0, so both can be represented by rational numbers.

By Lemma 3,

$$\check{m}_0 = \begin{cases} [m_0]_{\pm q/2}, & \text{if } x_0 \in [-N, N); \\ [m_0]_{\pm q/2} - q, & \text{if } x_0 \geq N; \\ [m_0]_{\pm q/2} + q, & \text{if } x_0 < N. \end{cases} \tag{5.5}$$

So

$$|\check{x}_0 - f^{\text{init}}(x_0)| \leq N/4. \tag{5.6}$$

For phase m_0', the target to approximate is integer $\check{m}_0 \in (-q, q)$. The plaintext in phase $\check{m}_0$ is $(\check{m}_0)^{[l_\Delta]} = \Delta m_{\text{init}} \mod q$.

We next investigate the approximation error of integer $[m_0']_{\pm q}$ to integer $\check{m}_0$. Let

$$m_1 = m_0 - m_0' \mod q. \tag{5.7}$$

Then

$$m_1 = \check{m}_0 - f^{\text{init}}(x_0) \times q/(2N) - e_{B0} = (\check{x}_0 - f^{\text{init}}(x_0)) \times q/(2N) - e_{B0} - e_{M0} \times q/(2N) \mod q. \tag{5.8}$$

As $m_1 \in [-q/2, q/2)$ by default representation,

$$|m_1| \leq |\check{x}_0 - f^{\text{init}}(x_0)| \times q/(2N) + |e_{B0} + e_{M0} \times q/(2N)| \leq q/4 + \sqrt{E_{\text{MDS}}^2 q^2/(2N)^2 + E_B^2}, \tag{5.9}$$

or equivalently,

$$|m_1 \times (2N)/q| \leq N/2 + \sqrt{E_{\text{MDS}}^2 + E_B^2 (2N)^2/q^2}. \tag{5.10}$$

In the second uni-bootstrap, the modulus down switch of m_1 gives

$$\begin{aligned} x_1 &= r_{2N/q}(m_1, e_{M1}) \\ &= m_1 \times (2N)/q + e_{M1} \\ &= (\check{x}_0 - f^{\text{init}}(x_0)) \times q/(2N) - e_{B0} \times q/(2N) - e_{M0} + e_{M1}, \end{aligned} \tag{5.11}$$

where $|e_{M1}| \leq E_{\text{MDS}}$. So

$$\begin{aligned} |x_1| &\leq |\check{x}_0 - f^{\text{init}}(x_0)| \times q/(2N) + | - e_{B0} \times q/(2N) - e_{M0} + e_{M1}| \\ &\leq N/2 + \sqrt{2E_{\text{MDS}}^2 + E_B^2 (2N)^2/q^2}. \end{aligned} \tag{5.12}$$

To continue the uni-bootstrap, we need to fix the resolution of f.

On the right side of (5.10), $E_{\text{MDS}} < 2^l$ and $E_B \ll q$, so if $E_B(2N)/q$ is small, then the square-root term $\leq 2^l$. Similarly, on the right side of (5.12), when $E_B(2N)/q$ is small, the square-root term $\approx \sqrt{2^{l+1}} < 2^l + 2^{l-1}$. For long plaintexts, by Lemma 23, the following two inequalities are always satisfied:

$$\begin{aligned} &\sqrt{E_{\text{MDS}}^2 + 2E_B^2 (2N/q)^2} \leq 2^l, \\ &\sqrt{2E_{\text{MDS}}^2 + E_B^2 (2N/q)^2} < 2^l + 2^{l-1}. \end{aligned} \tag{5.13}$$

For long plaintexts, the second uni-bootstrap is based on f_{l-1}. It gives

$$\begin{aligned} m_1' &= f_{l-1}(x_1) \times q/(2N) + e_{B1} \mod q, \\ m_2 &= m_1 - m_1' \mod q, \end{aligned} \tag{5.14}$$

where $|e_{B1}| \leq E_B$. We check the bound of integer $m_2 \in [-q/2, q/2)$, which measures the approximation error of integer $\check{m}_0$ by the sum of integers $m_0' + m_1'$.

By (5.10) and (5.12), using (3.37), we get

$$\begin{aligned} |m_2| &= |m_1 - f_{l-1}(x_1) \times q/(2N) - e_{B1}| \\ &\leq 2^l \times q/(2N) + |e_{M0} \times q/(2N) + e_{B0} + e_{B1}| \\ &\leq 2^l \times q/(2N) + \sqrt{E_{\text{MDS}}^2 q^2/(2N)^2 + 2E_B^2} \\ &\leq 2^{l+1} \times q/(2N) \;\; = \;\; q/2^{d+2}. \end{aligned} \tag{5.15}$$

So the first block that is just bootstrapped has $d + 2$ bits.

Notice that the error bound in (5.15) does not involve E_{init}. The reason is that at the current stage e_{init} is taken as part of the input plaintext, now that $l_t > l_N$, the plaintexts at lower bits together with e_{init} are all truncated when the phase is in $\mathbb{Z}_{2N}$, so after the uni-bootstrap they become part of the rounding error bounded by the first term $2^l \times q/(2N)$ in the second line of (5.15).

By (3.12),

$$\begin{aligned} (m_0' + m_1') - (e_{B0} + e_{B1}) &= (f^{\text{init}}(x_0) + f_{l-1}(x_1)) \times q/(2N) \\ &\in \pm q/4 + 2^l q/(2N) \times \big([-N/2^{l+1}, N/2^{l+1}] \cap \mathbb{Z}\big) \\ &= q/2^{d+3} \times \big([-2^{d+2}, 2^{d+2}] \cap \mathbb{Z}\big). \end{aligned} \tag{5.16}$$

So $m_0' + m_1' \in \mathbb{Z}_{[l_q-d-3]}$. All nonzero plaintext bits of $m_0' + m_1' \in \mathbb{Z}_q$ are in the first $d+3$ bits. Furthermore, by $m_2 = m_1 - m_1' = m_0 - (m_0' + m_1')$, since $|m_2| \leq q/2^{d+2}$, the plaintext of $m_0' + m_1'$ approximates that of m_0 with $(d+1)$-bit precision. Let

$$m_0' + m_1' = y_1 q/2^{d+3} + e_1, \text{ where } y_1 \in [-2^{d+2}, 2^{d+2}] \cap \mathbb{Z}, \tag{5.17}$$

then $e_1 = e_{B0} + e_{B1}$ is the tail error of the phase, and

$$m_2 = (\check{m}_0)^{[l_\Delta]} - y_1 q/2^{d+3} + (e_{\text{init}} - e_1). \tag{5.18}$$

In m_2, the tail error is $e_{\text{init}} - e_{B0} - e_{B1}$.

Denote

$$l_s = l_N - l_t. \tag{5.19}$$

For long plaintexts, $l_s < 0$; for medium ones, $0 \leq l_s < l$; for short plaintexts, $l_s \geq l$; for single-block plaintexts, $l_s \geq l_N - d_{\text{single}}$.

Then consider medium plaintexts. In practice, the first group of bootstrapping a medium plaintext is the same as bootstrapping a long plaintext. By Lemma 23 in the Appendix, $k \leq 2(d_B + l - l_s)$ is required. By $d_B = l_\Delta - 1 - l_B = l_q - l_N + l_s - 1 - l_B$, we get $d_B + l - l_s = l_q - l_B - d - 3$. In our experiments, $k \leq l = 5$ or 6, $l_B = 6$ or 7, $l_q = 29$, $d = 4$ or 3. So $l_q - l_B - d - 3 = 16$, and $k \leq 2(d_B + l - l_s)$ is well satisfied.

Theorem 8 When $l_t > d+2$, under the hypothesis $k \leq 2(d_B + l - l_s)$, in the second uni-bootstrap of the head-on approach, only using f_{l-1} can the first block take the largest size of $d+2$ bits.

Proof. Denote by T the first term in the second line of (5.15). By (5.12) and (5.13), $|x_1| < N/2 + 2^l + 2^{l-1}$.

If using f_l, then when $x_1 \in (N/2, N/2 + 2^l + 2^{l-1})$, $f_l(x_1)$ can take value $N/2 - 2^{l+1}$, enlarging term T to $2^{l+1} \times q/(2N)$, and resulting in the enlarged power-of-2 bound $2^{l+2} \times q/(2N)$ of $|m_2|$.

If using any f_p where $p \geq l+1$, then when $x_1 \in (-N/2, N/2)$, $|x_1 - f_p(x_1)| \leq 2^{l+1}$, and term T is enlarged to at least $2^{l+1} \times q/(2N)$.

If using any f_p where $p \leq l-2$, then when $x_1 \in (N/2, N/2 + 2^l + 2^{l-1})$, both terms of $2^l + 2^{l-1}$ are discerned by the low-resolute f_p. As a result, term T is enlarged to $(2^l + 2^{l-1}) \times q/(2N)$, and the new bound of $|m_2|$ is $(2^{l+1} + 2^{l-1}) \times q/(2N)$. Q.E.D.

Finally, consider short plaintexts, namely $l_s \geq l$. For short plaintexts, bootstrapping the first block is dramatically different from that of long/medium plaintexts, because after the modulus switch from $\mathbb{Z}_q$ to $\mathbb{Z}_{2N}$, the rounding error floods only the last $l \leq l_s$ bits, the plaintext in $\mathbb{Z}_{2N}$ is not flooded, the resolution of f must be at least $l_s > l-1$. If still using f_{l-1}, then in $\mathbb{Z}_q$, the plaintext in the refreshed ciphertext goes beyond the Δ-border between the plaintext and the tail error, and floods the latter.

In the first group of bootstrapping short plaintexts, the first uni-bootstrap based on f^{init} is the same as before. The leftover phase m_1 has the same bound as (5.10). Before the second uni-bootstrap, the modulus down switch of m_1 gives the same x_1 with bound (5.12). Now the second uni-bootstrap is based on f_{l_s}.

If $l_s > l$, the following counterpart of (5.13):

$$\begin{aligned} \sqrt{E_{\text{MDS}}^2 + 2E_B^2(2N/q)^2} &\leq 2^{l_s}, \\ \sqrt{2E_{\text{MDS}}^2 + E_B^2(2N/q)^2} &< 2^{l_s} \end{aligned} \tag{5.20}$$

is well satisfied, by Lemma 24. Just as in (5.15), but now by (3.38), the leftover phase m_2 is bounded by

$$|m_2| \leq 2^{l_s+1} \times q/(2N) = q/t, \tag{5.21}$$

so the first block has l_t bits. In other words, the first block contains the full length of the plaintext. Is this the end of the whole bootstrapping? Since $l_t > d_{\text{single}}$, the bootstrapping cannot finish here. The plaintext in the bootstrapped result approximates the input plaintext with precision $l_t - 1$ bits. Another uni-bootstrap must be launched to correct the last bit. It belongs to the third group of uni-bootstraps.

If $l_s = l$, then $k \leq 2(d_B + l - l_s)$ is well satisfied in practice, so by Lemma 23, (5.13) is true. As in the proof of Theorem 8, in (5.15) the bound of $|m_2|$ is enlarged to $2^{l+2} \times q/(2N) = 2q/t = q/2^{l_t - 1}$, the first block has $l_t - 1$ bits.

Similar to Theorem 8, it is easy to show that only using f_{l_s} in the second uni-bootstrap can give the largest size for the first block. Similar to (5.16), we have

$$\begin{aligned}(m_0' + m_1') - (e_{B0} + e_{B1}) &= (f^{\text{init}}(x_0) + f_{l_s}(x_1)) \times q/(2N) \\ &\in \pm q/4 + 2^{l_s+1} q/(2N) \times \big([-N/2^{l_s+2}, N/2^{l_s+2}] \cap \mathbb{Z}\big) \\ &= q/t \times \big([-t/2, t/2] \cap \mathbb{Z}\big).\end{aligned} \tag{5.22}$$

So $m_0' + m_1' \in \mathbb{Z}_{[l_\Delta]}$. All nonzero plaintext bits of $m_0' + m_1' \in \mathbb{Z}_q$ are in the first l_t bits. Since $|m_2| \leq q/t$, the plaintext of $m_0' + m_1'$ approximates that of m_0 with $(l_t - 1)$-bit precision. Let

$$m_0' + m_1' = y_1 \Delta + e_{B0} + e_{B1}, \text{ where } y_1 \in [-t/2, t/2] \cap \mathbb{Z}, \tag{5.23}$$

then

$$m_2 = (\check{m}_0)^{[l_\Delta]} - y_1 \Delta + (e_{\text{init}} - e_{B0} - e_{B1}). \tag{5.24}$$

5.2 Second group of head uni-bootstraps

In the first group, two different resolutions of f are used: $l - 1$ for long and medium plaintexts, and l_s for short ones. In the second group, the resolution of f keeps on changing, due to the decrease of the plaintext size in the leftover phase. Short plaintext bootstrapping skips the second group. So in this subsection, $l_t - d - 2 = l - l_s > 0$ is assumed.

For long plaintexts, denote by $d_0 + 2 := d + 2$ the size of the first block. For $i > 0$, let d_i be the size of the $(i+1)$-st block. Denote

$$d_i' = \sum_{j=0}^{i} d_j. \tag{5.25}$$

Then $d_i' + 2$ is the bit-size of the concatenation of the first $i + 1$ blocks. Set $l_{s0} = l_s$, and for $i > 0$, set

$$l_{si} = l_s + d_{i-1}' = l_{s(i-1)} + d_{i-1}. \tag{5.26}$$

For the $(i+1)$-st block, l_{si} plays the role of l_s, and is the resolution of f.

In the second group and the third group, for any block to be bootstrapped, during modulus down switch, two vacant bits are reserved ahead of the block, or equivalently, in two's complement representation, the block starts with two bits of identical value. Because of this, during the uni-bootstrap of the block, we often refer to $d_i + 2$ as the *full size* of the $(i+1)$-st block, including two leading extra bits that were occupied by the previous block. By default, d_i is the *size* of the $(i+1)$-st block where $i > 0$, sometimes also called *non-redundant size*.

The input of the second group is $m_2 \in \mathbb{Z}_q$ satisfying $|m_2| \leq q/2^{d_0+2}$, according to (5.15). The inequality can be written as

$$(m_2\, 2^{d_0}) \times 2N/q \in [-N/2, N/2]. \tag{5.27}$$

From now on till the end of the whole bootstrapping, f^{init} is no longer needed, and the $(i+1)$-st block will be bootstrapped by one uni-bootstrap based on $f_{l_{si}}$.

For the first block of the second group, preceding the uni-bootstrap, the modulus down switch is realized by function

$$r_{2^{d_0'+1}N/q}(m, e) = (m\, 2^{d_0})(2N/q) + e \in \mathbb{Z}. \tag{5.28}$$

Denote

$$x_2 = r_{2^{d_0'+1}N/q}(m_2, e_{M_2}), \text{ where } |e_{M_2}| \leq E_{\text{MDS}}. \tag{5.29}$$

The uni-bootstrap based on f_p, where p is to be determined, generates

$$\begin{aligned}m_2' &= f_p(x_2) \times q/(2^{d_0'+1}N) + e_{B2} \mod q, \\ m_3 &= m_2 - m_2' \mod q,\end{aligned} \tag{5.30}$$

where $|e_{B2}| \leq E_B$. For $m_3 \in [-q/2, q/2)$, by (5.27), using (3.34), we get

$$|m_2\, 2^{d'_0+1} N/q - f_p(x_2) - e_{B2}\, 2^{d'_0+1} N/q| \leq 2^p + \sqrt{E_{\text{MDS}}^2 + (E_B\, 2^{d'_0+1} N/q)^2}, \tag{5.31}$$

for any $p \geq 0$; when $p < 0$, the term 2^p is removed from the right side. So

$$|m_3| \leq |m_2 - f_p(x_2) \times q/(2^{d'_0+1}N) - e_{B2}| \leq 2^p q/(2^{d'_0+1}N) + \sqrt{E_{\text{MDS}}^2 q^2/(2^{d'_0+1}N)^2 + E_B^2}; \tag{5.32}$$

again when $p < 0$, the term involving 2^p is removed from the right side. The smaller the value of p, the better for $|m_3|$.

The resolution p of f is determined by $l_{s1} = l_{s0} + d_0 = l_s + d$. If $l_{s1} < 0$, then obviously the resolution-free version f is optimal, for which (5.32) becomes

$$|m_3| \leq \sqrt{E_{\text{MDS}}^2 q^2/(2^{d'_0+1}N)^2 + E_B^2}. \tag{5.33}$$

If $l_{s1} \geq 0$, then $p \geq l_{s1}$ is mandatory, and $f_{l_{s1}}$ is optimal, for which (5.32) becomes

$$|m_3| \leq 2^{l_{s1}} q/(2^{d'_0+1}N) + \sqrt{E_{\text{MDS}}^2 q^2/(2^{d'_0+1}N)^2 + E_B^2}. \tag{5.34}$$

Let d_1 be the biggest integer such that $|m_3| \leq q/2^{d'_0+d_1+2}$, then d_1 is the size of the block that is just bootstrapped.

The computation of d_1 by the above definition is straightforward. In power-of-2 binomial bounds, the computation can be greatly simplified. The result is presented in Lemma 25. For large-precision plaintexts where $l_t \gg l_N$, $l_{si} < 0$ for the beginning blocks of the second group. When $l_{si} < 0$, then $d_i = d + 1$; when $l > l_{si} \geq 0$, then $d_i = d$; when $l_{si} \geq l$, the block-size ranges from 1 to d.

By (3.12), when $l_{s1} \geq 0$,

$$\begin{aligned} m'_2 - e_{B2} &\in 2^{l_{s1}+1} \times q/(2^{d'_0+1}N) \times \big([-N/2^{l_{s1}+2}, N/2^{l_{s1}+2}] \cap \mathbb{Z}\big) \\ &= \varDelta \times \big([-t/2^{d'_0+2}, t/2^{d'_0+2}] \cap \mathbb{Z}\big). \end{aligned} \tag{5.35}$$

When $l_{s1} < 0$,

$$m'_2 - e_{B2} \in q/(2^{d'_0+1}N) \times [-N/2, N/2] = 2^{-l_{s1}-1} \varDelta \times [-N/2, N/2]. \tag{5.36}$$

So if $l_{s1} < 0$, then $m'_2 - e_{B2} \in \mathbb{Z}_{[l_q-d-l_N-1]}$, else $m'_2 - e_{B2} \in \mathbb{Z}_{[l_q-l_t]}$. Together with $|m'_2 - e_{B2}| \leq q/2^{d+2}$, we get that if $l_{s1} < 0$, then all nonzero plaintext bits of $|m'_2| \in \mathbb{Z}_q$ are in the l_N-bit-string from the $(d+2)$-nd bit to the $(d+l_N+1)$-st bit counted from the left; else, all nonzero plaintext bits of $|m'_2|$ are in the $(l_t - d - 2)$-bit-string from the $(d+2)$-nd bit to the l_t-th bit counted from the left. Recall that all nonzero plaintext bits of $|m'_0 + m'_1| \in \mathbb{Z}_q$ are in the first $d+3$ bits. So the l_N-bit-string or $(l_t - d - 2)$-bit-string of $|m'_2|$ overlaps with the $(d+3)$-bit-string of $|m'_0 + m'_1|$ by 2 bits.

Consider $m_3 = m_2 - m'_2 = m_0 - \sum_{i=0}^{2} m'_i$. Since $|m_3| \leq q/2^{d+d_1+2}$, the plaintext of $\sum_{i=0}^{2} m'_i$ approximates that of m_0 with $(d + d_1 + 1)$-bit precision. Compared with the $(d+1)$-bit precision provided by the plaintext of $m'_0 + m'_1$, we see that the precision is improved by d_1 bits, which is exactly the size of the block just bootstrapped. This interprets the meaning of the term "block-size" in head-on bootstrapping.

If $l_{s1} \geq 0$, let

$$m'_2 = y_2 \varDelta + e_{B2}, \ \text{ where } y_2 \in [-t/2^{d'_0+2}, t/2^{d'_0+2}] \cap \mathbb{Z}; \tag{5.37}$$

else, let

$$m'_2 = y_2 q/(2^{d'_0+1}N) + e_{B2}, \ \text{ where } y_2 \in [-N/2, N/2] \cap \mathbb{Z}. \tag{5.38}$$

Then

$$m_3 = (\check{m}_0)^{[l_\varDelta]} - y_1 q/2^{d+3} - y_2 z_2 + (e_{\text{init}} - \sum_{i=0}^{2} e_{Bi}), \tag{5.39}$$

where $z_2 = \varDelta$ if $l_{s1} \geq 0$, and $z_2 = q/(2^{d'_0+1}N)$ otherwise.

For the other blocks in the second group, the bootstrapping is much the same with the first block of the group. For the i-block of the group where $i \geq 1$, the input phase is $m_{i+1} \in \mathbb{Z}_q$ satisfying $|m_{i+1}| \leq q/2^{d'_{i-1}}$, the modulus down switch is from $\mathbb{Z}_{q/2^{d'_{i-1}}}$ to $\mathbb{Z}_{2N}$, so that in $\mathbb{Z}_{2N}$, $m_{i+1} \times 2^{d'_{i-1}}(2N/q) \in [-N/2, N/2]$.

The uni-bootstrap of the i-block in the second group generates a new phase $m'_{i+1} \in \mathbb{Z}_q$:

$$m'_{i+1} = f_{l_{si}}(r_{2^{d'_{i-1}}(2N/q)}(m_{i+1}, e_{M(i+1)})) + e_{B(i+1)} \mod q, \tag{5.40}$$

where $|e_{B(i+1)}| \leq E_B$. Let

$$m_{i+2} = m_{i+1} - m'_{i+1} \mod q. \tag{5.41}$$

By (3.34), if $l_{si} \geq 0$, then

$$|m_{i+2}| \leq 2^{l_{si}} q/(2N \times 2^{d'_{i-1}}) + \sqrt{E^2_{\mathrm{MDS}} q^2/(2N \times 2^{d'_{i-1}})^2 + E^2_B}; \tag{5.42}$$

if $l_{si} < 0$, then

$$|m_{i+2}| \leq \sqrt{E^2_{\mathrm{MDS}} q^2/(2N \times 2^{d'_{i-1}})^2 + E^2_B}. \tag{5.43}$$

In each case, the right side of each inequality above is bounded by $q/2^{d'_{i-1}+d_i+2}$ for some biggest integer $d_i > 0$, and d_i is called the size of the block. Again the computation of d_1 by the above definition is straightforward, and in power-of-2 binomial bounds, the computation can be greatly simplified, with results presented in Lemma 25.

For the i-block in the second group, denote $d' = d'_{i-1}$ for short. Then $d' < l_t - d - 2 = l_t - l_N + l = l - l_s$. If $l_{si} = l_s + d' < 0$, namely $l_t - d' > l_N$, by (5.43), let d_i be the biggest integer satisfying

$$\sqrt{E^2_{\mathrm{MDS}} q^2/(2^{d'+1}N)^2 + E^2_B} \leq q/2^{d'+d_i+2}. \tag{5.44}$$

If $l_{si} = l_s + d' \geq 0$, namely $d + 3 \leq l_t - d' \leq l_N$, by (5.42), let d_i be the biggest integer satisfying

$$2^{l_{si}} q/(2^{d'+1}N) + \sqrt{E^2_{\mathrm{MDS}} q^2/(2^{d'+1}N)^2 + E^2_B} \leq q/2^{d'+d_i+2}. \tag{5.45}$$

For the i-th uni-bootstrap of the second group, where $i > 1$, just as the $i = 1$ case, it can be proved by induction that the uni-bootstrap generates a phase $m'_{i+1} \in \mathbb{Z}_q$ with the property that if $l_N - 2 > l_{si} \geq 0$, then $d'_{i-1} < l_t - 2$, and

$$\begin{aligned} m'_{i+1} - e_{B(i+1)} &\in 2^{l_{si}+1} \times q/(2^{d'_{i-1}+1}N) \times \big([-N/2^{l_{si}+2}, N/2^{l_{si}+2}] \cap \mathbb{Z}\big) \\ &= \varDelta \times \big([-t/2^{d'_{i-1}+2}, t/2^{d'_{i-1}+2}] \cap \mathbb{Z}\big); \end{aligned} \tag{5.46}$$

if $l_{si} < 0$, then

$$m'_{i+1} - e_{B(i+1)} \in q/(2^{d'_{i-1}+1}N) \times [-N/2, N/2] = 2^{-l_{si}-1}\varDelta \times [-N/2, N/2]. \tag{5.47}$$

So if $l_{si} < 0$, then $m'_{i+1} - e_{B(i+1)} \in \mathbb{Z}_{[l_q - d'_{i-1} - l_N - 1]}$, else $m'_{i+1} - e_{B(i+1)} \in \mathbb{Z}_{[l_\varDelta]}$. Together with $|m'_{i+1} - e_{B(i+1)}| \leq q/2^{d'_{i-1}+2}$, we get that if $l_{si} < 0$, then all nonzero plaintext bits of $|m'_{i+1}| \in \mathbb{Z}_q$ are in the l_N-bit-string from the $(d'_{i-1}+2)$-nd bit to the $(d'_{i-1}+l_N+1)$-st bit counted from the left; else, all nonzero plaintext bits of $|m'_{i+1}|$ are in the $(l_t - d'_{i-1} - 2)$-bit-string from the $(d'_{i-1}+2)$-nd bit to the l_t-th bit counted from the left.

For the i-th block of the second group, its l_N-bit-string or $(l_t - d'_{i-1} - 2)$-bit-string starts at the $(d'_{i-2} + d_{i-1} + 2)$-nd bit from the left, while for its preceding $(i-1)$-st block, the l_N-bit-string ends at the $(d'_{i-2} + l_N + 1)$-st bit from the left. The two bit-strings overlap by $l_N - d_{i-1}$ bits. By Lemma 25, $d_{i-1} \leq d + 1$, so $l_N - d_{i-1} \geq l + 1$.

Furthermore, by $m_{i+2} = m_{i+1} - m'_{i+1} = m_0 - \sum_{j=0}^{i+1} m'_j$, since $|m_{i+2}| \leq q/2^{d'_i+2}$, the plaintext of $\sum_{j=0}^{i+1} m'_j$ approximates that of m_0 with (d'_i+1)-bit precision. Compared with the $(d'_{i-1}+1)$-bit precision provided by the plaintext of $\sum_{j=0}^{i} m'_j$, we see that the precision is improved by d_i bits, which is exactly the size of the i-th block in the second group.

If $l_{si} \geq 0$, let

$$m'_{i+1} = y_{i+1}\varDelta + e_{B(i+1)}, \text{ where } y_{i+1} \in [-t/2^{d'_{i-1}+2}, t/2^{d'_{i-1}+2}] \cap \mathbb{Z}; \tag{5.48}$$

else, let

$$m'_{i+1} = y_{i+1} q/(2^{d'_{i-1}+1}N) + e_{B(i+1)}, \text{ where } y_{i+1} \in [-N/2, N/2] \cap \mathbb{Z}. \tag{5.49}$$

Then

$$m_{i+2} = (\check{m}_0)^{[l_\varDelta]} - y_1 q/2^{d+3} - \sum_{j=2}^{i+1} y_j z_j + (e_{\mathrm{init}} - \sum_{j=0}^{i+1} e_{Bj}), \tag{5.50}$$

where $z_j = \Delta$ if $l_{s(j-1)} \geq 0$, and $z_j = q/(2^{d'_{j-2}+1}N)$ otherwise.

The second group contains $w \geq 0$ blocks, where w satisfies

$$d'_w = \sum_{i=0}^{w} d_i = l_t - 2, \text{ or equivalently, } l_{s(w+1)} = l_s + d'_w = l_N - 2, \tag{5.51}$$

where each $d_i > 0$. When the second group finishes, the leftover phase m_{w+2} has the property that $|m_{w+2}| \leq \Delta$, and the plaintext of $\sum_{j=0}^{w+1} m'_j$ approximates that of m_0 with $(l_t - 1)$-bit precision, just like the result of short plaintext bootstrapping after the first group.

The number of uni-bootstraps w can be directly computed by Lemma 25, together with the size of each block in the second group. By the lemma, as long as $l_{si} < 0$, the block size remains to be the biggest value $d+1$. Once $l_{si} \geq l - k$, then the block size reduces to be at most d. The more the smaller sized blocks, the less efficient the bootstrapping. How many blocks in the second group have size $\leq d$?

by Lemma 25, when $l > l_{si} = l_s + d'_{i-1} \geq l - k$, the i-block of the second group has block size $d_i = d$; when $l_{si} = l$, then $d_i \in \{l_t - 2 - d'_{i-1}, l_t - 3 - d'_{i-1}\}$; when $l_N - 2 > l_{si} > l$, then $d_i = l_t - 2 - d'_{i-1}$. Only when $i = w$ can the block size take value $l_t - 2 - d'_{i-1}$. So if $k < 2d - 1$, there cannot be more than two blocks having size d, and if there are two such blocks, then the number of small sized blocks in the second group is at most 3. The reason is that if $l_{si} > l$, then $i = w$ is the last block; from $l - k$ to l, the interval length $< 2d$, so if l_{si} takes value twice in the interval, then after the two blocks, the third block has its $l_{si} > l$, and is the last one.

For example, for practical parameters $l_N = 11, k = l = 5, d = 4, k_B = l_B = 6$, for any block of small size, l_{si} takes values in interval $[l - k, l_N - 2] = [0, 9]$, with $l = 5$ in between. by Lemma 25, when $l_{si} = l$, it is always true that $d_i = l_t - 2 - d'_{i-1}$. The third group has at least two and at most three small sized blocks. Let the j-th block be the first block of size $\leq d$. Since the $(j-1)$-st block has size $d_{j-1} = d + 1$, $l_{sj} = l_s + d'_{j-2} + d_{j-1} \in [0, d]$. The following are all possibilities of the small sized blocks in the second group:

- $l_{sj} = 0$: then $d_j = d = 4, d_{j+1} = 4, d_{j+2} = 1$.
- $l_{sj} = 1$: then $d_j = d = 4, d_{j+1} = 4$.
- $l_{sj} = 2$: then $d_j = d = 4, d_{j+1} = 3$.
- $l_{sj} = 3$: then $d_j = d = 4, d_{j+1} = 2$.
- $l_{sj} = 4$: then $d_j = d = 4, d_{j+1} = 1$.

For another set of practical parameters $l_q = 29, l_N = 11, k = l = 6, d = 3, k_B = l_B = 7$, when $l_{si} = l$, then $d_i = l_t - 2 - d'_{i-1}$ if $d_B > 2$, and $d_i = l_t - 3 - d'_{i-1}$ if $d_B = 2$. Assume $d_B = 2$. Then $l_\Delta = 10$, $l_t = 19$, $l_s = -8$. There are 6 blocks in the first two groups, with size $2+3, 4, 4, 3, 2, 1$ respectively. The corresponding l_{si} sequence for $0 \leq i \leq w + 1 = 6$ is $-8, -5, -1, 3, 6, 8, 9$.

Theorem 9 below is a direct corollary of Lemma 25. It describes the change of block size during the whole bootstrapping. In particular, at the beginning of the second group, all blocks have size $d+1$; in the middle, when $l_{si} \geq l - k$, the blocks have size d only. In greedy mode, the blocks keep size d till the end, where the last block is viewed as being padded with some vacant bits at its end. In regular mode, when l_{si} grows to l, the blocks have size $d - 1$, and keep it till the end.

Theorem 9 In the second group of head-on bootstrapping, take the last plaintext block as being padded with some vacant bits at the end. Then along with the blockwise bootstrapping starting from the first group, there are all together 5 possible sizes during the whole bootstrapping, and they occur in descending order:

$d+2$**:** the first block;

$d+1$**:** all blocks i with $d'_{i-1} \in [d, l_t - 2 - d - k)$ (start of the second group);

d**:** all blocks i with $d'_{i-1} \in [l_t - 2 - d - k, l_t - 2)$ in greedy mode, or all block i with $d'_{i-1} \in [l_t - 2 - d - k, l_t - 2 - d)$ in regular mode (middle of the second group);

$d-1$**:** all blocks i with $d'_{i-1} + 2 - l_t \in [l_t - 2 - d, l_t - 2)$ in regular mode (end of the second group);

0: the third group.

5.3 Third group of head uni-bootstraps

After the first two groups of uni-bootstraps, not only the input error e_{init} persists, but new errors $\sum_{i=0}^{w} e_{Bi}$ grow at the end of the leftover phase m_{w+2}. The accumulated bootstrapping result $\sum_{j=0}^{w+1} m'_j$ approximates the input m_0 with at least $(l_t - 1)$-bit precision.

There are two possibilities. If the precision of approximation has reached l_t bit, then the whole bootstrapping finishes after the second group, else to compensate for the loss of the 1-bit precision, a new uni-bootstrap must be launched to bootstrap the last block, which has non-redundant size of 0 bit, or equivalently, full size of 2 bits.

Whether or not the third group is needed is determined by the bound of the tail error after the modulus down switch preceding the last uni-bootstrap in the second group. Before the uni-bootstrap, $w+1$ uni-bootstraps have been done starting from the first group, so the tail error in phase $m_{w+1} \in \mathbb{Z}_q$ is bounded by $\sqrt{E_{\text{init}}^2 + (w+1)E_B^2}$. By $d'_{w-1} = l_t - 2 - d_{w-1}$, the modulus down switch from $\mathbb{Z}_{q/2^{d'_{w-1}}} = \mathbb{Z}_{2^{2+d_{w-1}}\Delta}$ to $\mathbb{Z}_{2N}$ introduces a rounding error, and in $\mathbb{Z}_{2N}$, the tail error is bounded by

$$\sqrt{E_{\text{MDS}}^2 + (E_{\text{init}}^2 + (w+1)E_B^2)(2N)^2/(2^{2+d_{w-1}}\Delta)^2} = \sqrt{E_{\text{MDS}}^2 + 2^{-2d_{w-1}}(E_{\text{init}}^2 + (w+1)E_B^2)(N/4)^2(2/\Delta)^2}. \tag{5.52}$$

By $l_{sw} = l_s + d'_{w-1} = l_N - 2 - d_{w-1}$, if in (5.52), the bound $< 2^{l_{sw}} = N/2^{2+d_{w-1}}$, then when viewed in $\mathbb{Z}_q$, this tail error $< \Delta/2$, the uni-bootstrap can cut off the old tail error completely, and replace it with a new one bounded by E_B.

If bound (5.52) $\geq N/2^{2+d_{w-1}}$, then a uni-bootstrap is needed to repair the precision loss. The uni-bootstrap starts with the modulus down switch on m_{w+2} from $\mathbb{Z}_{2^2\Delta}$ to $\mathbb{Z}_{2N}$. By $l_{s(w+1)} = l_s + d'_w = l_N - 2$, the uni-bootstrap is based on f_{l_N-2}, so that only the first two bits of $\mathbb{Z}_{2N}$ are recorded in the result.

After the modulus down switch on m_{w+2}, in $\mathbb{Z}_{2N}$, the tail error is bounded by

$$\sqrt{E_{\text{MDS}}^2 + (E_{\text{init}}^2 + (w+2)E_B^2)N^2/(2\Delta)^2}. \tag{5.53}$$

If in (5.53), the bound $< N/4$, then when viewed in $\mathbb{Z}_q$, this tail error $< \Delta/2$, the uni-bootstrap cuts off the old tail error completely, and the whole bootstrapping succeeds; else, the bootstrapping fails to generate an accurate backup of the input plaintext, instead it outputs a phase whose plaintext loses 1-bit precision when compared with the input.

It is important to know beforehand whether an input phase can be accurately bootstrapped in the head-on approach. In the worst case, the tail error bound $\sqrt{v}E_B$ of the final output of the whole bootstrapping equals $E_{\text{init}}/2$, where v is the total number of uni-bootstraps, then the requirement on the initial error is the most stringent. In the best case, v takes the smallest value 3, the last block has zero non-redundant size, then the requirement on the initial error is the most generous.

Let d_e be the maximal size allowed for the last block in the whole bootstrapping. To clear the tail error of the last phase in $\mathbb{Z}_{2N}$, when viewed in $\mathbb{Z}_q$, the tail error must be strictly bounded by $\Delta/2$, *i.e.*,

$$\sqrt{E_{\text{init}}^2 + (v-1)E_B^2 + E_{\text{MDS}}^2(\Delta/2N)^2 2^{2(d_e+2)}} < \Delta/2. \tag{5.54}$$

Equivalently, when viewed in $\mathbb{Q}_{2N}$,

$$\sqrt{E_{\text{MDS}}^2 + 2^{-2(d_e+2)}(E_{\text{init}}^2 + (v-1)E_B^2)(2N/\Delta)^2} < 2^{-(d_e+2)}N. \tag{5.55}$$

The existence of integer $d_e \geq 0$ is the inequality prerequisite for head-on bootstrapping.

Given l_t, computing v is pretty easy. In the best case $v = 3$, when comparing (5.55) with (4.2) of the tail-up approach, on sees that by replacing E_B with $\sqrt{2}E_B$, or approximately, replacing integer d_B by rational $d_B - 1/2$, the conclusions given by Lemma 15 and Lemma 17 on inequality (4.2) can be transported to conclusions on inequality (5.55). For example, for large initial error, $d_e = d$ or $d-1$; for small initial error, if $k_I \leq \min(2d_B - 1, 2d - 1)$, then $d_e \geq 0$ exists. Indeed, Lemma 22 of the Appendix guarantees that the above transported conclusions are correct for $v = 3$.

When v continues to grow, by the power-of-two upper bound of integer $v-1$, Lemma 15 and Lemma 17 can still be approximately extended to (5.55), but the aberration becomes larger. In the worst case $\sqrt{v}E_B = E_{\text{init}}/2$, we can use the replacement $E_{\text{init}}^2 + (v-1)E_B^2 = (5/4)E_{\text{init}}^2 - E_B^2$, so that (5.54) becomes

$$\sqrt{((5/4)E_{\text{init}}^2 - E_B^2)(2/\Delta)^2 + 2^{2d_e}E_{\text{MDS}}^2(4/N)^2} < 1. \tag{5.56}$$

The existence of integer $d_e \geq 0$ in (5.56) proposes the lowest upper bound on the initial error for head-on bootstrapping.

For (5.56), Lemma 20 gives a full classification of all possible values of d_e. In particular, for small initial error, the head-on bootstrapping always works, and either $d_e = d$, or $d_e = d-1$, so the third group is not needed. Combining this with Lemma 25 on the block size d_i when $d'_{i-1} + l_s = l$, we get that there are two modes of head-on bootstrapping for small initial errors:

Greedy mode: If $k \leq 2d_I + 1$, and either $k \leq 2d_B$, or $k = 2d_B + 1$ and $k_B \leq k+1$, then ever since $l_{si} \geq l-k$, the blocks have size d and keep the size till the end of the whole bootstrapping.

Regular mode: Else, while still the blocks have size d when $l_{si} \geq l-k$ is reached, ever since $l_{si} \geq l$, the blocks have size $d-1$, and keep it till the end of the whole bootstrapping.

In each mode, the last block is viewed as being padded with some vacant bits at its end.

Lemma 21 presents a simplified conclusion on d_e in the practical situation $d \geq 3$ for large initial error, which states that $d_e \geq 0$ exists if and only if the initial error is not larger than $(1-2^{-3})\Delta/2$. When v is given, then depending on v and E_B, the upper bound of k_I for large initial errors ranges from 3 to $\min(2d-1, 2d_B-1)$.

Once the biggest integer d_e satisfying (5.54) is computed, determining whether or not the third group is necessary is very easy: the group is necessary if and only if the last block of the second group has bit-size $\leq d_e$.

Theorem 10 Let there be v_H uni-bootstraps in the head-on bootstrapping. Let $d_e \geq 0$ satisfy (5.55). The correctness of the head-on approach is guaranteed by

$$\lfloor (\sum_{i=0}^{v_H-1} m'_i)/\Delta \rceil = \lfloor \check{m}_0/\Delta \rceil. \tag{5.57}$$

Proof. Preceding the last uni-bootstrap of the second group, the input phase is m_{w+1}, which is of the form (5.50) for $i = w-1$, whose tail error is $e = e_{\text{init}} - \sum_{j=0}^{w} e_{Bj}$, and whose plaintext is of the form Δm, where $m \in [-t/2, t/2]$, such that

$$\Delta m = (\check{m}_0)^{[l_\Delta]} - y_1 q/2^{d+3} - \sum_{j=2}^{w} y_j z_j, \tag{5.58}$$

and $|\Delta m| \leq |m_{w+1}| + |e| \leq q/2^{d'_{w-1}+2} + |e|$. By (5.54), $|e| < \Delta/2$, so $|\Delta m| < (2^{d_{w-1}} + 2^{-1})\Delta$, indicating $|m| \leq 2^{d_{w-1}}$.

After the modulus down switch from $\mathbb{Z}_{q/2^{d'_{w-1}}} = \mathbb{Z}_{2^{2+d_{w-1}}\Delta}$ to $\mathbb{Z}_{2N}$, in $\mathbb{Z}_{2N}$ the phase becomes

$$e_{M(w+1)} + m_{w+1} \times 2N/(2^{2+d_{w-1}}\Delta) = m \times 2^{l_N - 1 - d_{w-1}} + \{e_{M(w+1)} + e \times N/(2^{1+d_{w-1}}\Delta)\}. \tag{5.59}$$

The plaintext $m \times 2^{l_N-1-d_{w-1}}$ is bounded by $2^{l_N-1} = N/2$, the tail error is bounded by (5.52).

If the bound (5.52) $< 2^{l_{sw}} = 2^{l_N-2-d_{w-1}}$, then by definition, $f_{l_{sw}} = f_{l_N-1-d_{w-1}}$ acting on phase (5.59) gives $m \times 2^{l_N-1-d_{w-1}}$. So the last uni-bootstrap of the second group generates the phase

$$m'_{w+2} = m \times 2^{l_N-1-d_{w-1}} \times 2^{1+d_{w-1}}\Delta/N + e_{B(w+2)} = \Delta m + e_{B(w+2)}; \tag{5.60}$$

for $v_H = w+2$, (5.57) is true.

If the bound (5.52) $\geq 2^{l_{sw}}$, then the third group is necessary. Preceding the uni-bootstrap of the third group, the input phase is m_{w+2}, which is of the form (5.50) for $i = w$, whose tail error is $e' = e_{\text{init}} - \sum_{j=0}^{w+1} e_{Bj}$, and whose plaintext is of the form $\Delta m'$, where $m' \in [-t/2, t/2]$, such that

$$\Delta m' = (\check{m}_0)^{[l_\Delta]} - y_1 q/2^{d+3} - \sum_{j=2}^{w+1} y_j z_j, \tag{5.61}$$

and $|\Delta m'| \leq |m_{w+2}| + |e'| \leq \Delta + |e|$. By (5.54), $|e'| < \Delta/2$, so $|\Delta m'| < (1 + 2^{-1})\Delta$, indicating $m' \in \{0, \pm 1\}$.

After the modulus down switch from $\mathbb{Z}_{\Delta/2^2}$ to $\mathbb{Z}_{2N}$, in $\mathbb{Z}_{2N}$ the phase becomes

$$e_{M(w+2)} + m_{w+2} \times 2N/(2^2\Delta) = m' \times N/2 + \{e_{M(w+2)} + e' \times N/(2\Delta)\}. \tag{5.62}$$

The plaintext $m' \times N/2$ is bounded by $N/2$, the tail error is bounded by (5.53).

Now that $d_e \geq 0$ satisfies (5.55), the tail error of (5.62) is strictly bounded by $N/4$. By definition, f_{l_N-2} acting on phase (5.62) gives $m' \times N/2$. So the last uni-bootstrap generates the phase

$$m'_{w+3} = m' \times N/2 \times 2\Delta/N + e_{B(w+3)} = \Delta m' + e_{B(w+3)}; \tag{5.63}$$

for $v_H = w + 3$, (5.57) is true. Q.E.D.

5.4 Putting everything together into ciphertext form

For head-on bootstrapping, the global parameters are the same as in tail-up bootstrapping, so are the uni-bootstrap procedure `UniBoot`$(f, \texttt{ct}, q')$. The following is the general algorithm for head-on bootstrapping.

Algorithm 2 "`HeadBoot`": Backup bootstrapping from head on

Input: LWE ciphertext `ct` to bootstrap;
public FHEW/TFHE parameters.
Assume $l_t \geq 2$. Assume $d_e \geq 0$ exists.
Output: LWE ciphertext `ct'`.

```
    {Control parameters setup}
 1: Set $d = l_N - l - 2$, $d_0 = d, d'_0 = d_0$. ($d_0 + 2$ is the size of the first block)
 2: Compute the block sizes $d_i > 0$ in the second group, such that $\sum_{j=0}^{w} d_j = l_t - 2$. The number of blocks $w \geq 0$ is thus obtained.
 3: Set $l_s = l_N - l_t$. For $1 \leq i \leq w$, set $d'_i = \sum_{j=0}^{i} d_j$, $l_{si} = l_s + d'_{i-1}$.
 4: Compute the allowed maximal size of the last block $d_e$.
    {First group}
 5:   ct' ⟵ UniBoot($f^{\text{init}}$, ct, $q$);
      ct ⟵ ct − ct'  mod $q$.
 6: if $l_t > d + 2$ then
 7:   ct' ⟵ ct' + UniBoot($f_{l-1}$, ct, $q$)  mod $q$.
 8: else
 9:   ct' ⟵ ct' + UniBoot($f_{l_s}$, ct, $q$)  mod $q$.
10: end if
11:   ct ⟵ ct − ct'  mod $q$.
    {Second group}
12: for $i = 1$ to $w$ do
13:   ct' ⟵ ct' + UniBoot($f_{l_{si}}$, ct, $q/2^{d'_{i-1}}$)  mod $q$;
      ct ⟵ ct − ct'  mod $q$.
14: end for
    {Third group}
15: if $d_w > d_e$ then
16:   ct' ⟵ ct' + UniBoot($f_{l_N-2}$, ct, $4q/t$)  mod $q$.
17: end if
18: return ct'.
```

The complexity of the algorithm is also dominated by the number of uni-bootstraps. In the following, we consider head-on bootstrapping for small initial error. The first block has $\min(d + 2, l_t)$ bits. In the second group, when i changes from 1 to $w + 1$, d'_{i-1} increases from d to $l_t - 2$. Denote $d' = d'_{i-1}$ and take it as a variable. By Lemma 9,

– when $d' \in [d, l_t - 2 - d - k)$, the blocks have size $d + 1$. The number of blocks in this interval of d' is

$$v_1 := \lceil \max(0, l_t - 2 - 2d - k)/(d+1) \rceil = \max(0, \lceil (l_t - k)/(d+1) \rceil - 2). \tag{5.64}$$

After these blocks are bootstrapped, the interval of d' is reduced to $[d + v_1(d+1), l_t - 2)$.

– In greedy mode, when $d' \in [d + v_1(d+1), l_t - 2)$, the blocks have size d. The number of blocks in this interval of d' is

$$v_2' := \lceil \max(0, l_t - 2 - d - v_1(d+1))/d \rceil = \max(0, \lceil l_t - 2 - v_1 \rceil - 1 - v_1). \tag{5.65}$$

– In regular mode, when $d' \in [d + v_1(d+1), l_t - 2 - d)$, the blocks have size d. The number of blocks in this interval of d' is

$$v_2 := \lceil \max(0, l_t - 2 - 2d - v_1(d+1))/d \rceil = \max(0, \lceil l_t - 2 - v_1 \rceil - 2 - v_1). \tag{5.66}$$

– In regular mode, when $d' \in [d + v_1(d+1) + v_2 d, l_t - 2)$, the blocks have size $d - 1$. The number of blocks in this region of d' is

$$v_3 := \lceil \max(0, l_t - 2 - d - v_1(d+1) - v_2 d)/(d-1) \rceil = \max(0, \lceil l_t - 3 - 2v_1 - v_2 \rceil - 1 - v_1 - v_2). \tag{5.67}$$

So for small initial error, in greedy mode, the total number of uni-bootstraps is

$$v_H' := 2 + v_1 + v_2'. \tag{5.68}$$

In regular mode, the total number of uni-bootstraps is

$$v_H := 2 + v_1 + v_2 + v_3. \tag{5.69}$$

For example, when $d_I = 1$ and $k = 2$, then the head-on bootstrapping is also in greedy mode. When $d = 4$, for input plaintext of 15 bits, 4 uni-bootstraps suffice to finish the bootstrapping in three blocks, with size 6, 5, 4, respectively. In contrast, the tail-up approach in greedy mode needs 6 uni-bootstraps to bootstrap three blocks from the tail up, with size 6, 6, 3, respectively. The bootstrapping efficiency is improved by $(6-4)/4 = 50\%$.

For large initial error, in the worst case, $d_e = 0$, so the third group is always needed. For the example used in illustrating the tail-up approach, where $l = k = 5$, $d_I = 0$, $d \geq 3$, $d_B = d + 1$ and $k_I = 2d_B - 4 = 2d - 2 > 3$, if $v \leq 4$, then by Lemma 17, with d_B replaced by $d_B - 1$, $k_I \leq \min(2d-1, 2d_B - 2)$ is satisfied, so $d_e \geq 0$ exists. When $d = 4$, for input plaintext of 6 bits, two blocks and 3 uni-bootstraps are needed to finish the head-on bootstrapping, the bit-sizes of which are $6, 0$, respectively. The bootstrapping efficiency is improved by $(5-3)/3 \approx 67\%$.

5.5 Head-on bootstrapping with LSB precursor

When the initial error is too large, the head-on approach fails. To make bootstrapping, we resort to the LSB precursor to reduce the initial error relative to the new border between the plaintext and the tail error, which is now Δ instead of the old $\Delta/2$.

By Lemma 19, the LSB precursor only requires $k_I \leq 2d + 3$ for large initial error. After the LSB precursor is executed, l_t, l_s are replaced by $l_t - 1, l_s + 1$ respectively. Then the block sizes d_i, the maximal allowed size d_e of the last block, and the total number of uni-bootstraps v, all need update. In algorithm 2, the control parameters need update, and the LSB precursor needs to be ahead of the first group.

Enlarging parameter d_e is the sole purpose of the LSB precursor. We investigate the influence of the LSB precursor on d_e. In (5.55), replacing Δ by 2Δ, we get

$$\sqrt{E_{\text{MDS}}^2 + 2^{-2(d_e+2)}(E_{\text{init}}^2 + (v-1)E_B^2)(N/\Delta)^2} < 2^{-(d_e+2)}N, \tag{5.70}$$

or when viewed in $\mathbb{Z}_q$,

$$\sqrt{(E_{\text{init}}/2)^2 + (v-1)(E_B/2)^2 + E_{\text{MDS}}^2 \Delta^2/(2N)^2 2^{2(d_e+2)}} < \Delta/2. \tag{5.71}$$

In the best case $v = 3$, when comparing (5.71) with (A.47), we see that d_I, d_B both increase by 1. By Lemma 22, the new $d_e \in \{d, d-1\}$. In the worst case $\sqrt{v}E_B = E_{\text{init}}/2$, (5.70) becomes

$$\sqrt{((5/4)E_{\text{init}}^2 - E_B^2)/\Delta^2 + 2^{2d_e}E_{\text{MDS}}^2(4/N)^2} < 1. \tag{5.72}$$

Comparing (5.72) with (A.23), we see that the former is a special case of the latter with $d'_{i-1} = 1$ and $d_i = d_e$. By Lemma 18, we get the following corollary:

Corollary 11 Let d_e be the biggest integer satisfying (5.72), Then $d_e = d-1$, except for the following cases where $d_e = d$:

- $k \le 2d_I + 1$;
- $k = 2d_I + 3$, and one of the following is true:
  - $k_I = 2$;
  - $k_I = 3, d_I > 0$;
  - $k_I = 3, d_I = 0, d_B = 2$.

After executing the LSB precursor, the third group of the head-on approach is no longer needed. Moreover, the last block has size at least $d - 1$ (the bootstrapped LSB is not part of the last block). If the third group is not empty, then the LSB precursor not only allows bigger initial error, but also speeds up bootstrapping.

If the third group is empty, and the last block of the second group has only 1 bit, then the LSB precursor only allows bigger initial error, but does not speed up bootstrapping. Indeed in head-on bootstrapping, the $(d-1)$-bit size occurs at most once. By Lemma 25, the next to the last block of the second group has size $d-1$ if and only if for $i = w-1$, $d'_{i-1} + l_s = l$, and either $k > 2d_B + 1$, or $k = 2d_B + 1$ and $k_B > k+1$. When this happens, the last block has 1 bit. However in this case, since $d_B \ge d_I + 2$, $k \ge 2d_I + 5$, by Corollary 11, the new $d_e = d - 1$, and cannot increase to d.

6 Bootstrapping by blockwise error reduction

To put in Section 1:

The idea of CKKS ciphertext bootstrapping by removing the head error with FHEW/TFHE bootstrapping in a larger-modulus phase space, was first proposed by [22]. To meet the Li-Micciancio security, the extra bits in the result of fixed-point CKKS multiplication of two approximate plaintexts need to be removed. A method based on tail-up bootstrapping was proposed in [29]. After removing these bits, although the extra bits that may disclose information of the plaintext are removed, the tail error bound is enlarged by 1 bit, namely the plaintext precision is lost by 1 bit.

After bootstrapping a BFV ciphertext with large-precision plaintext, the tail error is decreased. Due to the number of blocks $v > 1$, the output tail error bound $E_{\text{fin}} = \sqrt{v}E_B > E_B$. Now that E_B is the refreshed error bound of single-block plaintext, this seems to suggest that blockwise bootstrapping of large-precision plaintext cannot obtain a tail error as small as E_B. Is this the truth?

The head-on approach can improve the efficiency from bootstrapping $d/2+1$ bits per uni-bootstrap on average to $d+1$ bits. For long plaintexts this is still not efficient. Can the efficiency be further improved? In this section, we propose a new strategy for large-precision plaintext bootstrapping. Instead of backing up the long plaintexts, the new strategy is to get rid of the tail error as much as possible, by blockwise approximate bootstrapping of the tail error in the input phase. It is called *error bootstrapping*, in contrast to the previous *plaintext bootstrapping* strategy.

Error bootstrapping is obviously more effective when the input tail error is short. In fact, when the input tail error is shorter than the concatenation of the plaintext and the output tail error after plaintext bootstrapping, error bootstrapping is more effective. This is because the error bootstrapping approach is similar to the head-on approach to plaintext bootstrapping. Starting from the first block of the error phase, the old errors are backed up blockwise approximately, and then removed from the input.

What is different is that while the head of the tail error keeps on being cleared, new refreshed errors keep on accumulating at the tail. So error bootstrapping is not simply blockwise backing up tail error and then deleting them from the original phase. Instead, it is reducing the ever-changing tail error, including the newly added refreshed errors that were introduced by bootstrapping previous error blocks. For every uni-bootstrap, the whole tail error in $\mathbb{Z}_\Delta$ is taken as the plaintext, so the plaintext is constantly changing. This is the unique feature of error bootstrapping. In contrast, in head-on plaintext bootstrapping, although the tail error changes every time after each uni-bootstrap, it never floods the plaintext, so the plaintext is constant.

Lemma 27 gives the feasibility condition on error bootstrapping. In practice, $2d+5 \geq k-1$ is well satisfied, so in Lemma 27, if $d_I = 0$, the condition can be replaced by $k_I \leq 2d+5$. This condition is the weakest among all the inequality prerequisites for blockwise bootstrapping.

For practical parameters $l_q = 29$, $d = 4$, when the plaintext has $l_t \geq l_q - k_I - 1 = 29 - 13 - 1 = 15$ bits, all decryptable ciphertexts can be bootstrapped, because $2d+5 = 13$ on one hand, and $k_I \leq l_I \leq l_q - l_t - 1 \leq 13$ on the other hand, so that $k_I \leq 2d+5$ is always true.

When $l_t < 15$,the initial error bound in Lemma 27 that allows bootstrapping is $(1-2^{-13})\Delta/2$ instead of $\Delta/2-1$. In order to keep the property that any decryptable ciphertext can be bootstrapped, the information rate l_t/l_q of the ciphertext cannot be too small, or equivalently, the error-modulus ratio B/q cannot be too big.

As the input of error bootstrapping may be the result of plaintext bootstrapping, the goal is to reduce the tail error as much as possible, or more accurately, as close to E_B as possible. Because of this, the error quality constraint $E_{\text{fin}} \leq E_{\text{init}}/2$ is weakened to the extreme:

$$E_B \leq E_{\text{init}}/2. \tag{6.1}$$

Consequently, in this section only $d_I \geq d_B + 1$ is assumed.

Obviously E_B is the lower bound of tail error in all error bootstrapping results. Can this bound be approached? If not, what is the limit of the tail error bound by error bootstrapping? We will calculate the limit, and show that for many practical parameters, the limit is just E_B, and can be approached in practice.

We will also investigate an extreme case where both the initial error and E_B are too big to make error bootstrapping, but still $E_B \leq E_{\text{init}}/2$, so there is still possibility to reduce the initial error. In such extreme case, we have to resort to the LSB precursor, and use delicate error control to ultimately reduce the initial error to the limit.

The size of the error space can be measured by l_Δ. In consideration of the fact that after bootstrapping, the tail error is always bounded from below by $E_B \approx 2^{-1-d_B}\Delta$, it is more appropriate to measure the size of the error space by d_B. When d_B takes the minimal value 1, then $d_I = 0$, the error space is the smallest in that practically there are only two bits available: the MSB of the initial error, and the MSB of the refreshed error. The lower bits of the error space are constantly occupied, and are redundant when measuring the error space.

Similar to the measurement of the input plaintext, to measure the error space we need some ruler. When the length of $\mathbb{Z}_{2N}$ is used to measure the error space, then when $d_B \leq d+2$, the error space is said to be *short*; when $d+2 < d_B \leq l_N$, the error space is said to be *medium*; when $d_B > l_N$, then error space is said to be *long*. When $d_B = d+2$, then

$$E_B \approx 2^{-3-d}\Delta = 2^l\Delta/(2N) \approx E_{\text{MDS}} \times \Delta/(2N); \tag{6.2}$$

the two errors E_B and $E_{\text{MDS}} \times (2N/\Delta)$ approximately make a tie. When $d_B = l_N$, then $E_B \approx \Delta/(2N)$, all the "reducible" bits of the error space are included in $\mathbb{Z}_{2N}$ after modulus down switch. In fact, when $\Delta < 2N$, the modulus switch from $\mathbb{Z}_\Delta$ to $\mathbb{Z}_{2N}$ is modulus up switch. Only when $\Delta > 2N$ is the modulus switch truly "down". Still we unanimously call it modulus down switch.

As before, we shall first introduce each group by phase simulation, then introduce the general algorithm for head-on ciphertext bootstrapping, and finally introduce the LSB precursor in the end. For simplification, we introduce the rationals:

$$\delta := 2N/\Delta \in \mathbb{Q}, \quad \delta^{-1} = \Delta/(2N). \tag{6.3}$$

In the following subsections,

– $d_0 + 2$ is the size of the first block; for $i \geq 1$, d_i is the size of the $(i+1)$-st plaintext block; d_i' is the sum of d_j for $0 \leq j \leq i$; d_e is the maximal size allowed for the last block for correct decryption.
– $e_0 \in \mathbb{Z}_q$ is the initial error; for $i > 0$, $e_i \in \mathbb{Z}_q$ is the leftover error of e_0 after subtracting the results of the previous i uni-bootstraps.
– $e_i' \in \mathbb{Z}_q$ is the result of the i-th uni-bootstrap. e_0' involves the MSB of the whole error block; e_1' approximates the lower bits of the leading error block; for $i > 1$, e_i' approximates the i-th error block.
– e_{Mi} is the rounding error in modulus down switch of the i-th uni-bootstrap, and e_{Bi} is the refreshed error by the i-th uni-bootstrap.

In this section, we always use $[-\Delta, \Delta/2)$ interval representatives for modular numbers in $\mathbb{Z}_\Delta$, and use $[-N, N)$ interval representatives for modular numbers in $\mathbb{Q}_{2N}$.

6.1 First group of error bootstrapping

If the initial error is small: $E_{\text{init}} \leq \Delta/4$, then the first group is skipped. The purpose of the first group is to reduce the bound of the tail error to $\leq \Delta/4$, so only large initial error needs it.

The first group consists of two uni-bootstraps based on f^{init} and f respectively, similar to all previous blockwise bootstrappings. The modulus down switch is from $\mathbb{Z}_\Delta$ to $\mathbb{Z}_{2N}$, as the tail error must be within $(-\Delta/2, \Delta/2)$. In error bootstrapping, the phase is usually called *error phase*, as only tail error is left in the phase after modulo Δ.

However, this does not mean that the the plaintext of the input phase is no longer taken care of. As shown in (5.5), on head-on plaintext bootstrapping, the sum of plaintexts in the output phase $\sum_{i=0}^{v-1} m'_i$ may not be equal to the input plaintext Δm_{init}, instead it equals $(\check{m}_0)^{[l_\Delta]} \in \Delta m_{\text{init}} + \{0, \pm q\}$. In error bootstrapping, similar phenomenon may occur, namely the sum of tail errors in the output phase may not be equal to the input tail error approximately, instead it may differ from the input by approximately Δ. This will destroy the LSB of the input plaintext. For head-on plaintext bootstrapping according to (5.9), the mistake can be avoided in the first uni-bootstrap of the first group; for error bootstrapping, the mistake can be avoided only by taking care of both uni-bootstraps in the first group.

For the input phase $m_0 = \Delta m_{\text{init}} + e_{\text{init}}$, the input error phase is denoted by integer

$$e_0 := e_{\text{init}} \in \mathbb{Z}_\Delta \cap (-\Delta/2, \Delta/2). \tag{6.4}$$

Obviously,

$$e_0 = [\Delta m_{\text{init}} + e_{\text{init}}]_{\pm\Delta/2} = m_0 \mod \Delta. \tag{6.5}$$

Now that e_{init} becomes the plaintext, it will be removed from the error independence heuristic very soon, but not for now.

Preceding the first uni-bootstrap, the modulus down switch is from $\mathbb{Z}_\Delta$ to $\mathbb{Z}_{2N}$. In $\mathbb{Z}_{2N}$, the tail error of the whole phase is

$$e := e_0 \times (2N/\Delta) + e_{M0} = e_0\delta + e_{M0} \in \mathbb{Z}, \text{ where } |e_{M0}| \leq E_{\text{MDS}}. \tag{6.6}$$

In the sum, the two errors are still independent heuristically, so the tail error is bounded by $\sqrt{E_{\text{MDS}}^2 + E_{\text{init}}^2\delta^2}$.

By Lemma 3, for f^{init} to correctly handle the MSB of integer e_0, it is both sufficient and necessary that $e \in [-N, N)$. In terms of absolute value bound, the constraint is improved to

$$\sqrt{E_{\text{MDS}}^2 + E_{\text{init}}^2\delta^2} < N. \tag{6.7}$$

When viewed in $\mathbb{Z}_q$,

$$\sqrt{E_{\text{init}}^2 + E_{\text{MDS}}^2\Delta^2/(2N)^2} < \Delta/2. \tag{6.8}$$

In other words, $e_{\text{init}} \in \mathbb{Z}_{[l_\delta]}$ is Pythagorean E_{MDS}-tolerant, by taking $e_{\text{init}} = 0 \times 2^{p+1} + z \in \mathbb{Z}_{[p+1]}$ with $p = l_\delta - 1$ and $z = e_{\text{init}}$. Lemma 27 discloses the requirement (6.8) in power-of-2 binomial bounds, which is the most generous requirement so far on large initial error: $k_I \leq 2d + 5$.

Denote

$$x_0 = r_{2N/\Delta}(e_0, e_{M0}) = e_0 \times 2N/\Delta + e_{M0}, \text{ where } |e_{M0}| \leq E_{\text{MDS}}. \tag{6.9}$$

By (6.7), $|x_0| < N$. The first uni-bootstrap generates

$$\begin{aligned} e'_0 &= f^{\text{init}}(x_0) \times \Delta/(2N) + e_{B0} \mod q, \\ e_1 &= e_0 - e'_0 \mod q, \end{aligned} \tag{6.10}$$

where $|e_{B0}| \leq E_B$. We check the bound of e_1 below.

For $e'_0 \in [-q/2, q/2)$, in order for integer $e_0 - e'_0$ to correctly represent $e_1 \in \mathbb{Z}_\Delta$ in interval $[-\Delta/2, \Delta/2)$, the bound $|e_1| < \Delta/2$ is needed. More accurate bound is given as follows. By (6.7), using (3.24), we get

$$e_1 = (x_0 - e_{M0})\delta^{-1} - (f^{\text{init}}(x_0)\delta^{-1} + e_{B0}), \tag{6.11}$$

so

$$|e_1|\delta \leq |x_0 - f^{\text{init}}(x_0)| + |e_{M0} + e_{B0}\delta| \leq N/2 + \sqrt{E_{\text{MDS}}^2 + E_B^2\delta^2}. \tag{6.12}$$

That $|e_1| < \Delta/2$ can be guaranteed if

$$\sqrt{E_{\mathrm{MDS}}^2 + E_B^2\delta^2} < N/2. \tag{6.13}$$

Notice that on the left side of (6.13), nay of the two error terms may be dominant. The situation is dramatically different from the head-on approach, where the counterpart of (6.12) is (5.10), and where Δ is replaced by $q \gg 2N$, so that the error term of (5.10) is completely dominated by E_{MDS}.

Consider the effect of reducing error phase by the first uni-bootstrap. For large initial error, when $E_B\delta \le E_{\mathrm{MDS}}$, the first uni-bootstrap reduces the error bound to about $(1 + 2^{-1-d})N/2 \gtrapprox N/2$, the reduction is less than but almost 1 bit. When $E_B\delta > E_{\mathrm{MDS}}$, the first uni-bootstrap reduces the tail error bound to about $(1 + 2^{1-k_B})N/2$.

When E_B is very close to E_{init}, for example, $d_B = 1 = d_I + 1$, by $E_B \le E_{\mathrm{init}}/2$, $k_I \ge k_B$, then compared with the initial error bound $(1-2^{-k_I})\Delta/2 \times 2N/\Delta = (2-2^{1-k_I})N/2$, the tail error is only slightly reduced; in the worst case $k_I = k_B = 2$, the tail error is not reduced at all. Later in this subsection it will be shown that for the first group using two uni-bootstraps based on f^{init}, f_p respectively for some suitable resolution p, it is required that $d_B \ge 2$. Under this constraint, the first uni-bootstrap based on f^{init} always reduces the tail error.

Preceding the second uni-bootstrap, the modulus down switch of e_1 to $\mathbb{Z}_{2N}$ gives

$$x_1 = r_{2N/\Delta}(e_1, e_{M1}) = e_1\delta + e_{M1}, \text{ where } |e_{M1}| \le E_{\mathrm{MDS}}. \tag{6.14}$$

That $|x_1| < N$ is required so that integer x_1 provides the correct $[-N, N)$-interval representation of the phase. More accurate bound is given as follows. By (6.11),

$$x_1 = (x_0 - f^{\mathrm{init}}(x_0)) - e_{B0}\delta - e_{M0} + e_{M1}, \tag{6.15}$$

so

$$|x_1| \le |x_0 - f^{\mathrm{init}}(x_0)| + |-e_{B0}\delta - e_{M0} + e_{M1}| \le N/2 + \sqrt{2E_{\mathrm{MDS}}^2 + E_B^2\delta^2}. \tag{6.16}$$

Then

$$\sqrt{2E_{\mathrm{MDS}}^2 + E_B^2\delta^2} < N/2 \tag{6.17}$$

is used to guarantee $|x_1| < N$. Once (6.17) is true, so is (6.13). Hence constraint (6.13) is redundant.

The second uni-bootstrap is based on f_p, where resolution p is to be determined. It outputs

$$\begin{aligned} e_1' &= f_p(x_1) \times \Delta/(2N) + e_{B1} \mod q, \text{ where } |e_{B0}| \le E_B; \\ e_2 &= e_1 - e_1' \mod q. \end{aligned} \tag{6.18}$$

e_2 is the resulting leftover error phase of the first group. That $|e_2| \le \Delta/4$ is the bottom line of making the first group of uni-bootstraps.

To gain more accurate bound of e_2, the resolution p must be involved. When $p < 0$, then $f_p = f$. For short, denote

$$\begin{aligned} x_0' &= x_0 - f^{\mathrm{init}}(x_0), \\ x_1' &= e_1\delta \ = \ x_0' - e_{M0} - e_{B0}\delta. \end{aligned} \tag{6.19}$$

Then $x_0' \in (-N/2, N/2)$, $x_1 = x_0' - e_{M0} - e_{B0}\delta + e_{M1} \in (-N, N)$, and $e_2\delta = x_1' - f(x_1) - e_{B1}\delta$. Using (3.36), we get

$$\begin{aligned} |e_2| &= |(x_1' - f(x_1))\delta^{-1} - e_{M0}\delta^{-1} - e_{B0} - e_{B1}| \\ &\le \sqrt{4|-e_{M0}\delta^{-1} - e_{B0}|^2 + |e_{M1}\delta^{-1}|^2 + e_{B1}|^2} \\ &\le \sqrt{5(E_{\mathrm{MDS}}^2 + E_B^2\delta^2)}\Delta/(2N). \end{aligned} \tag{6.20}$$

Let d_0 be the biggest integer satisfying

$$\sqrt{5(E_{\mathrm{MDS}}^2 + E_B^2\delta^2)}\ \Delta/(2N) \le \Delta/2^{d_0+2}. \tag{6.21}$$

Then $d_0 \ge 0$ is required, which implies condition (6.17). Lemma 28 shows that the existence of $d_0 \ge 0$ requires $d_B \ge 2$. The lemma also presents a complete classification of all possible values of d_0. There are four possible values of d_0: $d, d-1; d_B - 2, d_B - 3$. The first two values are taken when $d_B \ge d + 1$; the last two values are taken when

$d_B \leq d+1$. When $d_0 = d_B - 2$, then $d_0 \leq d-1$, the leftover error e_2 is bounded by $\Delta/2^{d_0+2} = 2^{1-d_B}\Delta/2 \approx 2E_B$; when $d_0 = d_B - 3$, the bound is about $4E_B$.

When $p \geq 0$, then $f_p(x)$ has the advantage of suppressing an error in x that is bounded strictly by 2^p, and the disadvantage of adding a term 2^p to the resulting error bound. The goal of optimization is d_0, the bigger the better, as $|e_2| \leq \Delta/2^{d_0+2}$.

Denote

$$l_0 := l_N - d_B. \tag{6.22}$$

Then $E_B\delta = 2^{l_0}(1-2^{-k_B})$ in power-of-2 binomial bound. With this notation, when $E_B\delta \leq E_{\text{MDS}}$, then $\sqrt{E_{\text{MDS}}^2 + E_B^2\delta^2} \leq 2^l + 2^{l-1}$; when $E_B\delta \geq E_{\text{MDS}}$, $\sqrt{E_{\text{MDS}}^2 + E_B^2\delta^2} \leq 2^{l_0} + 2^{l_0-1}$.

In (6.16), if $|x_1| < N/2 + 2^{p+1} + 2^p \leq N$ for some integer $p \geq 0$, then $p \leq l_N - 3$, and using (3.37), we get

$$\begin{aligned} |e_2| &= |(x_1' - f_p(x_1))\delta^{-1} - e_{M0}\delta^{-1} - e_{B0} - e_{B1}| \\ &\leq 2^{p+1}\delta^{-1} + \sqrt{|-e_{M0}\delta^{-1} - e_{B0}|^2 + |e_{B1}|^2} \\ &\leq \left(2^{p+1} + \sqrt{E_{\text{MDS}}^2 + 2E_B^2\delta^2)}\right)\Delta/(2N). \end{aligned} \tag{6.23}$$

So $|e_2| \leq \Delta/2^{d_0+2}$ is satisfied if

$$2^{p+1} + \sqrt{E_{\text{MDS}}^2 + 2E_B^2\delta^2)} \leq N/2^{d_0+1}. \tag{6.24}$$

By (6.16), $|x_1| < N/2 + 2^{p+1} + 2^p$ can be guaranteed if

$$\sqrt{2E_{\text{MDS}}^2 + E_B^2\delta^2} < 2^{p+1} + 2^p \leq (3/8)N. \tag{6.25}$$

This inequality implies condition (6.17).

To maximize d_0, resolution p should be as small as possible. Lemma 29 gives a complete classification of all possible minimal values of p and the corresponding maximal values of d_0 satisfying (6.24) and (6.25). By the lemma, That $d_0 \geq 0$ requires $d_B \geq 3$, now that f_p is used instead of f; $p \in \{l-1, l, l_0-1\}$, and $d_0 \in \{d, d-1, d_B-3\}$. The first two values of d_0 are taken when $d_B \geq d+2$, namely $l_0 \leq l$; the last value is taken when $d_B \leq d+1$, namely $l_0 \geq l+1$. When $d_0 = d_B - 3$, then $d_0 \leq d-2$, the leftover error e_2 is approximately bounded by $4E_B$.

We compare Lemma 29 based on f_p with Lemma 28 based on f for the second uni-bootstrap:

- when $d+5 \leq d_B$, using f_p can generate $d_0 = d$ for all $k < 2(d_B - d - 2)$, while using f to generate $d_0 = d$ requires $k \leq 3$;
- when $d+4 \leq d_B$, both f_p, f generate $d_0 = d$ for $k \leq 3$;
- when $d+3 = d_B$, using f still generates $d_0 = d$ for $k \leq 3$, while using f_p generates $d_0 = d-1$ only;
- when $d+2 = d_B$, although both f, f_p generate $d_0 = d-1$, using f_p requires two different resolutions under different conditions;
- When $d+2 > d_B$, using f can generate $d_0 = d_B - 2$, while using f_p only generates the smallest $d_0 = d_B - 3$.

So when $d+5 \leq d_B$, f_p is optimal; when $d+3 \geq d_B$, f is optimal; when $d+4 = d_B$, f_p, f make a tie.

There are two more observations. The first is that only f allows $d_B = 2$. The second is that only f can lead to a leftover error bound of about $2E_B$ after the first group.

Theorem 12 Let $d_0 + 2 \geq 2$ be the size of the first block in error bootstrapping. For the second uni-bootstrap in the first group, if $d_B \geq d+4$, then using f_{l-1} is optimal, if $d_B \leq d+4$, then using f is optimal.

When the initial error is large but not too large, or more accurately, $\Delta/4 < E_{\text{init}} < 3\Delta/8$, it is possible to reduce the error bound to within $\Delta/4$, *i.e.*, to small error, by using only one uni-bootstrap. This problem will be investigated in Subsection **??**. Another problem is that the first group introduced in this subsection requires $d_B > 1$. The case $d_B = 1$ will be handled in Subsection **??**.

6.2 Second group of error uni-bootstraps

The leftover error phase of the first group is integer e_2 satisfying $|e_2| \leq \Delta/2^{d_0+2}$, or equivalently, $e_2\, 2^{d_0}\delta \in [-N/2, N/2]$. If the first group is skipped, then $d_0 = 0$. In the second group, a single programmable function f will be used in each uni-bootstrap to reduce the error bound. When this group finishes, the leftover error bound will be reduced to within $2^{l_B} \approx E_B$.

The first uni-bootstrap of the second group is as follows. Preceding the uni-bootstrap, the modulus down switch is from $\mathbb{Z}_{\Delta/2^{d_0}}$ to $\mathbb{Z}_{2N}$:

$$x_2 := r_{2^{d_0}\delta}(e_2, e_{M2}) = e_2 2^{d_0}\delta + e_{M2} \in \mathbb{Z}, \text{ where } |e_{M2}| \leq E_{\text{MDS}}. \tag{6.26}$$

The uni-bootstrap generates

$$\begin{aligned} e_2' &= f(x_2)(2^{d_0}\delta)^{-1} + e_{B2} \mod q, \\ e_3 &= e_2 - e_2' \mod q, \end{aligned} \tag{6.27}$$

where $|e_{B2}| \leq E_B$.

Since $|x_2| \leq N/2$, by (3.34),

$$|e_3| \leq \sqrt{E_{\text{MDS}}^2 (2^{d_0}\delta)^{-2} + E_B^2}. \tag{6.28}$$

Let d_1 be the biggest integer satisfying

$$\sqrt{E_{\text{MDS}}^2 (2^{d_0}\delta)^{-2} + E_B^2} \leq \Delta/2^{d_0+d_1+2}. \tag{6.29}$$

Then $|e_3| \leq \Delta/2^{d_0+d_1+2}$. The first block of the second group is said to have block-size d_1 bits. In other words, the first $d_0 + d_1 + 2$ bits of the tail error are cleared.

Set $d'_{-1} = 0$. For all $i \geq 0$, set

$$d_i' = \sum_{j=0}^{i} d_j. \tag{6.30}$$

For example, $d_0' = d_0, d_1' = d_0 + d_1$. By induction, the following can be easily proved: for any $1 \leq i \leq w$, the i-th block of the second group, on input e_{i+1} bounded by $\Delta/2^{d'_{i-1}+2}$, generates

$$\begin{aligned} e'_{i+1} &= f(r_{2^{d'_{i-1}}\delta}(e_{i+1}, e_{M(i+1)})) \times (2^{d'_{i-1}}\delta)^{-1} + e_{B(i+1)} \mod q, \\ e_{i+2} &= e_{i+1} - e'_{i+1} \mod q, \end{aligned} \tag{6.31}$$

where $|e_{Bi}| \leq E_B$ and $|e_{Mi}| \leq E_{\text{MDS}}$. The leftover error e_{i+2} is bounded by $\Delta/2^{d'_{i-1}+d_i+2}$, where d_i is the biggest integer satisfying

$$|e_{i+2}| \leq \sqrt{E_{\text{MDS}}^2 (2^{d'_{i-1}}\delta)^{-2} + E_B^2} \leq \Delta/2^{d'_{i-1}+d_i+2}. \tag{6.32}$$

Let $d' \geq 0$. Let l_i be the smallest integer satisfying

$$\sqrt{E_B^2 (2N/\Delta)^2 \times 2^{2d'} + E_{\text{MDS}}^2} \leq 2^{l_i+1}. \tag{6.33}$$

From the above inequality and $E_{\text{MDS}} < 2^l$, we get $l_i \geq l - 1$. From the fact that in $\mathbb{Z}_{2N}$ after modulus down switch, the first two bits are from the previous block, so that the error part occupies at most the remaining $l_N - 1$ bits, we get $l_i \leq l_N - 2$.

The block-size d_i for $1 \leq i \leq w$ is given by Lemma 30, by setting $d' = d'_{i-1}$ and $d_0 = d_i$ in the lemma. The second group starts with blocks of size $d + 1$, and for every $1 \leq i \leq w$, as long as $d' = d'_{i-1} \leq d_B - d - 2 - k/2$, then $d_i = d + 1$. When $d'_{i-1} \geq d_B - d - (k+3)/2$, then d_i drop to d. For the next block, $d' \geq d_B - (k+3)/2$. If $(k+3)/2 \leq d$, then $d' \geq d_B - d$, the next block will have size $d_B - d' - 1 < d$ or $d_B - d' - 2 < d$.

In our experiments, when $d = 3$, then $k = 6$, and when $d = 4$, then $k = 5$. So $(k+3)/2 \leq d$ is satisfied for $d = 4$ but not for $d = 3$. As a consequence, if $d = 4$, there is at most one d-bit block in the second group; if $d = 3$, there are at most two d-bit blocks in the second group, and the second d-bit block is the first block satisfying $d_i = d_B - d' - j$ for some $j \in \{1, 2\}$.

After the last block of size $\geq d$, we have $d' \geq d_B - d$, so the next block has size $d_i = d_B - d' - j \leq d - j$ for some $j \in \{1, 2\}$. After the d_i-bit block is bootstrapped, the leftover error is bounded by $2^{-(d'+d_i+2)}\Delta = 2^{j-1}(2^{-d_B}\Delta/2) \approx 2^{j-1}E_B$. When $j = 1$, then bound E_B is approximately reached, there is no bit left before E_B. When $j = 2$, the bound is about $2E_B$, there is still 1 bit left before E_B.

By Lemma 30, $d_0 = d_B - 1 - d'$ requires either $d' \geq d_B - d - 2 + k_B/2$, or $d' = d_B - d - 2 + (k_B - 1)/2$ and $k \leq k_B + 1$, otherwise only $d_0 = d_B - 2 - d'$ is possible. Suppose we start with $d' = d_B - 2$. If either $d \leq k_B/2 - 1$, or $d = (k_B - 1)/2$ but $k \geq k_B + 2$, then the new $d_0 = d_B - 2 - d' = 0$, and d' can no longer increase, which means the 1-bit error before E_B cannot be reduced in this case; else, the 1-bit error can still be cleared by another uni-bootstrap. So there are at most two blocks left in the second group after the last block of size $\geq d$. If there are two such blocks, the size of the last block is $d_w = 1 = d_B - d' - 1$; if there is only one such block, its size is $d_w = d_B - d' - 1$.

In our experiments, $k_B \leq l_B \leq 7$ and $d \geq 3$, so $k_B \leq 2d+1$ is satisfied, namely $d \geq (k_B - 1)/2$. If $d = (k_B - 1)/2$, then $k_B = 7$, and since $k \leq l \leq 6$, $k \leq k_B + 1 = 8$ is also satisfied. So in our experiments, $d_w = d_B - 1 - d'$ is always reached, and the second group finishes with no bit left before E_B.

The number of blocks of the second group is the smallest $w \geq 0$ satisfying

$$d_{w+1} = 0, \text{ or in practice, } d'_w = d_B - 1, \tag{6.34}$$

as explained above. After the second group, the leftover error phase e_{w+2} is bounded by $\Delta/2^{d'_w+2} = 2^{-d_B}\Delta/2 \approx E_B$, or more accurately,

$$|e_{w+2}| \leq \sqrt{E_B^2 + E_{\text{MDS}}^2(\delta 2^{d'_{w-1}})^{-2}}, \tag{6.35}$$

where $d'_{w-1} = d_B - 1 - d_w$.

In our experiments, $k_B = k + 1$, and $(k - 1)/2 < d$ is always true. Then $(k_B - 1)/2 = k/2 \leq d$. Under these assumptions, first consider the block size sequence when bootstrapping an error space where $d_B > d + 2 + k/2$. By Lemma 29, the first block has size $d + 2$.

Take $d' = d'_{i-1}$ as a variable. Then $d' \in [0, d_B + 1]$. The first block has size $d + 2$, so $d' = d$ at the beginning of the second group. In the second group, by Lemma 30, as long as $d_B \geq d + d' + 2 + (k - 1)/2$, the blocks have size $d + 1$. When $d_B - d - 2 - (k - 1)/2 < d' \leq d_B - d - 2$, the block size becomes d, and there is at most one such block. When $d_B - d - 2 < d' < d_B - d - 2 + \lfloor (k_B - 1)/2 \rfloor$, the block has size $d_B - 2 - d' \in [1, d + 1 - \lfloor (k_B - 1)/2 \rfloor]$, and is next to the last block; the last block has 1-bit. When $d_B - 1 > d' \geq d_B - d - 2 + (k_B - 1)/2$, the block has size $d_B - 1 - d'$, and is the last block.

Next consider the error space satisfying $d + 2 \leq d_B \leq d + 2 + k/2$. Then the first block has size $d + 1$ by Lemma 29, so $d' = d - 1$ at the beginning of the second group. Since $d_B - d - 2 - (k - 1)/2 \leq 1/2 < d'$, and $d_B - d - 2 + (k_B - 1)/2 = d_B - d - 2 + k/2 \in [0, k]$, by Lemma 30, the first block of the second group has three possible sizes:

- if $d' = d - 1 \leq d_B - d - 2$, namely $d_B \geq 2d + 1$, then the size is d; there are at most three blocks in the second group;
- if $d_B - d - 2 < d' < d_B - 2 - d + \lfloor k/2 \rfloor$, then the size is $d_B - 2 - d'$; there are only two blocks in the second group;
- if $d' \geq d_B - d - 2 + \lfloor k/2 \rfloor$, then the size is $d_B - 1 - d'$; there is only one block in the second group.

Finally consider the error space satisfying $d_B < d + 2$, namely small error space. By Lemma 28, if $d_B < d$ and $k = 2$, then the first block has size d_B, else the size is $d_B - 1$. At the beginning of the second group, either $d' = d_B - 2$ or $d' = d_B - 3 > d_B - d - 1$ for $d \geq 3$. In the former case, by $k = 2$ and $d' \geq d_B - d - 2 + k/2$, the second group has only one block, whose size is 1. In the latter case, if $d - 1 \geq k/2$, then $d' \geq d_B - d - 2 + k/2$, the second group has only one block, whose size is 2; else, the second group has two blocks, each having size 1.

For small initial error with $d_I > 0$, the first group is skipped, so at the beginning of the second group, $d' = d_I - 1 \geq 0$, and there are d' vacant leading bits from the initial If d_1 is the computed size of the first block of the second group based on $d' = d_I - 1$, then the actual size of the first block is $d_1 + (d_I - 1)$.

When $k \leq 3$, then in the second group, the block size becomes d if and only if $d' = d_B - d - 2$, because it is the only integer satisfying $d_B - d - 2 - (k - 1)/2 < d' \leq d_B - d - 2$; if $d_B > d + d' + 2$, the block size is always $d + 1$. Similarly, when $k_B \leq 4$, then $\lfloor (k_B - 1)/2 \rfloor \leq 1$, and no integer d' satisfies $d_B - d - 2 < d' < d_B - d - 2 + \lfloor (k_B - 1)/2 \rfloor$, so no block has size $d_B - 2 - d'$; after the blocks of size $\geq d$, there is only one block left, whose size is $d_B - 1 - d'$.

When both $k \leq 3$ and $k_B \leq 4$, we say the error bootstrapping is in *greedy mode*, no matter if the initial error is large or small.

For practical parameters $l_q = 29, l_N = 11, k = l = 5, d = 4, k_B = l_B = 6$, there is no greedy mode, and (1) $d + 2 + k/2 = 8.5$, (2) $d + 2 + (k-1)/2 = 8$, (3) $d + 2 - (k_B - 1)/2 = d + 2 - k/2 = 3.5$.

– If $l_t = 11$, then $l_\Delta = 29 - 11 = 18, d_B = l_\Delta - 1 - l_B = 11 > d + 2 + k/2$. The first block has size $d + 2 = 6$, so $d' = d = 4$ at the beginning of the second group. As $d_B - d - 2 - (k-1)/2 = 11 - 8 < d' \leq d_B - d - 2 = 11 - 6$, the first block of the second group has size $d = 4$. After this block, $d' = 4 + 4 = 8 \geq d_B - d - 2 + k/2 = 7.5$, so the last block has size $d_B - 1 - d' = 2$. The whole sequence of block sizes is $6, 4, 2$. Notice that the sum of block sizes is $d_B + 1 = 12$.
  In the above analysis, the initial error bound is assumed to be very large: $k_I \leq 2d + 5 = 13$, namely $E_{\text{init}} = 2^{17} - 2^4$. If the initial error is small, say $d_I = 1$, then the first group is skipped, and $d' = d_I - 1 = 0$ at the beginning of the second group. By $d_B \geq d + d' + 2 + (k-1)/2 = 8$, the first block has size $(d+1) + 2 = 7$, where the additional two bits are the two vacant leading bits of the error space. After this block, $d' = 0 + 5 = 5$ satisfies $d_B - d - 2 - (k-1)/2 = 3 < d' \leq d_B - d - 2 = 5$, so the second block has size $d = 4$. The last block has size 1. The whole sequence of block sizes is $7, 4, 1$.
– If $l_t = 15$, then $l_\Delta = 14, d_B = 7 \in [d + 2, d + 2 + k/2]$. The first block has size $d + 1 = 5$. At the beginning of the second group, $d' = d - 1 = 3 \leq d_B - d - 2 = 5$, so the first block of the group has size $d = 4$. After this block, $d' = 3 + d = 7 = d_B - d - 2 + \lfloor k/2 \rfloor$, so the last block has size $d_B - 1 - d' = 3$. The whole sequence of block sizes is $5, 4, 3$.
  For small initial error, if $d_I = 2$, then at the beginning of the second group, $d' = d_I - 1 = 1 \leq d_B - d - 2 = 5$, so the first block has size $(d) + 3 = 7$, where the additional three bits are the vacant leading bits of the error space. The whole sequence of block sizes becomes $7, 4, 1$, as in the case of $l_t = 11$.
– If $l_t = 19$, then $l_\Delta = 10, d_B = 3 < d + 2$. The first block has size $d_B - 1 = 2$. At the beginning of the second group, $d' = 0$. Since $d' \geq d_B - d - 2 + k/2 = -0.5$, the second group has only one block, whose size is 2. The sequence of block sizes is $2, 2$.
  For small initial error case $d_I = 1$, again by $d' = 0 \geq d_B - d - 2 + k/2 = -0.5$, there is only one block, whose size is $(d_B - 1 - d') + 2 = 4$.

6.3 Third group of error uni-bootstraps: limiting error refresher

After the second group, the leftover error phase e_{w+2} is bounded by $2^{-d_B}\Delta/2 \approx E_B$. This is already very good, and the whole bootstrapping can terminate. There is one more question: if one wants to further reduce the error, what is the limit of the error reduction? The third group addresses this question. In the third group, with one uni-bootstrap, the limit of the error reduction is almost reached. Because of this, the last uni-bootstrap is called the *limiting error refresher*. It reduces an error bounded by $2^{-d_B}\Delta/2$ to an error close to the limit of the error reduction. Of course, the third group is optional.

By (6.35), with $d'_{w-1} = d_B - 1 - d_w$, $l_B = l_\Delta - 1 - d_B$ and $(\delta 2^{d'_{w-1}})^{-1} = 2^{l_B - l_N + d_w + 1}$, a more accurate bound of the input e_{w+2} of the third group is the following:

$$\begin{aligned} A_{w+2} &:= \sqrt{E_B^2 + E_{\text{MDS}}^2 (\delta 2^{d'_{w-1}})^{-2}} \\ &= E_B \sqrt{1 + 2^{-2(d+1-d_w)}(1 - 2^{-k})^2/(1 - 2^{-k_B})^2} \\ &\approx E_B\{1 + 2^{-3-2d+2d_w}(1 - 2^{-k})^2/(1 - 2^{-k_B})^2\} \ \leq \Delta/2^{d_B+1}. \end{aligned} \tag{6.36}$$

Preceding the uni-bootstrap, the modulus down switch is from $\mathbb{Z}_{\Delta/2^{d_B+1}}$ to $\mathbb{Z}_{2N}$. The uni-bootstrap generates a leftover error e_{w+3} bounded by

$$\begin{aligned} A_{w+3} &:= \sqrt{E_B^2 + E_{\text{MDS}}^2 (\delta 2^{d_B - 1})^{-2}} \\ &\approx E_B\{1 + 2^{-3-2d}(1 - 2^{-k})^2/(1 - 2^{-k_B})^2\} \ < \ E_B(1 + 2^{-2-2d}(1 - 2^{-k})^2). \end{aligned} \tag{6.37}$$

Comparison between A_{w+2} and A_{w+3} shows that $(A_{w+2} - E_B)/(A_{w+3} - E_B) \approx 2^{2d_w}$.

For practical parameters $d = 4, E_B = 2^6 - 1$, since $2^{-2d-2} < 2^{-7}$, $A_{w+3} = E_B$. For another set of parameters $d = 3, E_B = 2^7 - 1$, since $2^{-2-2d}(1 - 2^{-k})^2 < 2^{-8}$, again $A_{w+3} = E_B$. So one uni-bootstrap suffices for the error to reach E_B in practice. In contrast, for $d_w = 1$ and $d = 3$, $A_{w+2} \neq E_B$; for $d_w > 1$ and $d = 4$, $A_{w+2} \neq E_B$.

There is another viewpoint on the ultimate error refresher. In $\mathbb{Z}_q$, if all the bits preceding E_B by at least two bits are taken as the plaintext bits, namely $l_\Delta = l_B + 2$, then $d_B = 1$, the input error e_{w+2} is bounded by $E_{\text{init}} = A_{w+3} \leq \Delta/4$. Now that $E_B = (1 - 2^{-k_B})\Delta/4 < E_{\text{init}}$, there is still room to reduce the error bound closer to E_B. Below we explore the limit of the error reduction.

By Lemma 30, for $d' = 0$, $d_0 = 0$ is allowed in practice, because $k_B \leq l_B \leq 2d + 1$ is satisfied. So on input e_0 bounded by E_{init}, the first uni-bootstrap generates an error phase e'_0, with leftover error phase $e_1 = e_0 - e'_0$ bounded by

$$A_1 := \sqrt{E_B^2 + E_{\text{MDS}}^2 \Delta^2/(2N)^2} \leq \Delta/4. \tag{6.38}$$

It is easy to see that the above A_1 is just the previous (6.37).

To investigate the limit of error reduction, we now leave the power-of-2 bounds, which have been used in every modulus down switch preceding each uni-bootstrap so far. It must be pointed out that even if the second group can only decrease the left error bound to about $2E_B$, by the following non-power-of-2-bound modulus down switch, with one uni-bootstrap based on f, the error bound can be reduced to about E_B; with another uni-bootstrap, the error bound can be reduced to about the limit.

Since $|e_1| \leq A_1$, we have $|e_1| \times N/(2\lceil A_1 \rceil) \leq N/2$. The modulus down switch preceding the second uni-bootstrap is from $\mathbb{Z}_{4\lceil A_1 \rceil}$ to $\mathbb{Z}_{2N}$. The second uni-bootstrap generates

$$\begin{aligned} e'_1 &= f(r_{N/(2\lceil A_1 \rceil)}(e_1, e_{M1})) \times 2\lceil A_1 \rceil/N + e_{B1} \mod q, \\ e_2 &= e_1 - e'_1 \mod q, \end{aligned} \tag{6.39}$$

where $|e_{B1}| \leq E_B$ and $|e_{M1}| \leq E_{\text{MDS}}$. By (6.38),

$$|e_2| \leq \left\lceil \sqrt{E_B^2 + E_{\text{MDS}}^2 (2\lceil A_1 \rceil/N)^2} \right\rceil \leq \left\lceil \sqrt{E_B^2 + E_{\text{MDS}}^2 \Delta^2/(2N)^2} \right\rceil = \lceil A_1 \rceil. \tag{6.40}$$

For all $j \geq 2$, define

$$A_j = \sqrt{E_B^2 + E_{\text{MDS}}^2 (2\lceil A_{j-1} \rceil/N)^2}. \tag{6.41}$$

Then $\lceil A_j \rceil \leq \lceil A_{j-1} \rceil$. When j tends to infinity, the limit of A_j is

$$E_{\text{lim}} := E_B/\sqrt{1 - (2E_{\text{MDS}}/N)^2} = E_B/\sqrt{1 - 2^{-2-2d}(1 - 2^{-k})^2} \approx E_B(1 + 2^{-3-2d}(1 - 2^{-k})^2). \tag{6.42}$$

Below we make detailed comparison between the two values E_{lim} and $A_1 > E_{\text{lim}}$, under the practical conditions $k_B \leq 2d + 1$ and $k \leq 2d$.

By (6.37), A_1 decreases with the increase of k_B. The minimum of A_1 for variable k_B is obtained when $k_B = 2d+1$, and is

$$A_{1-} := E_B \sqrt{1 + 2^{-2-2d}(1 - 2^{-k})^2/(1 - 2^{-1-2d})^2}. \tag{6.43}$$

Denote $u = 2^{-2-2d}$ and $x = (1 - 2^{-k})^2 \in (0, 1)$. Then

$$(A_{1-}/E_{\text{lim}})^2 = (1 + 2^{-2-2d}(1 - 2^{-k})^2/(1 - 2^{-1-2d})^2)\,(1 - 2^{-2-2d}(1 - 2^{-k})^2) = (1 + xu(1 - 2u)^{-2})(1 - xu) =: g_-(x). \tag{6.44}$$

Fix d and let k vary from 2 to $2d$, or equivalently, x increases from $9/16$ to $(1 - 2^{-2d})^2$. Parabola $g_-(x)$ takes its maximum at $g'_-(x_0) = 0$, where $x_0 = 2(1 - u)$. As $x = (1 - 2^{-k})^2 < 2(1 - u) = 2(1 - 2^{-2-2d})$, when k increases from 2 to $2d$, $g_-(x)$ increases accordingly. The minimal value of $g_-(x)$ is taken at $k = 2$, and is $1 + (495/256)u^2/(1 - 2u)^2 - (9/4)u^3/(1 - 2u)^2 \lessapprox 1 + 2^{-3-2d}$. So

$$(A_{1-} - E_{\text{lim}})/E_{\text{lim}} \lessapprox 2^{-3-2d}. \tag{6.45}$$

The maximum of A_1 for variable k_B is obtained when $k_B = 2$, and is

$$A_{1+} := E_B \sqrt{1 + 2^{-2-2d}(1 - 2^{-k})^2 \times 16/9}. \tag{6.46}$$

Again let $u = 2^{-2-2d}$ and $x = (1-2^{-k})^2 \in (0,1)$. Then

$$(A_{1+}/E_{\text{lim}})^2 = (1+2^{-2-2d}(1-2^{-k})^2 \times 16/9)\ (1-2^{-2-2d}(1-2^{-k})^2) = (1+16xu/9)(1-xu) =: g_+(x). \quad (6.47)$$

Parabola $g_+(x)$ takes its maximum at $g'_+(x_1)=0$, where $x_1 = 7/(32u) = (7/8)2^{2d}$. As $x = (1-2^{-k})^2 < 1 < x_1$, when k increases from 2 to $2d$, $g_+(x)$ increases accordingly. The maximal value of $g_+(x)$ is taken at $x = (1-2^{-2d})^2$, and is $1+(7/9)(1-2^{-2d})^2 2^{-2-2d} - (16/9)(1-2^{-2d})^4 2^{-4-4d} \lessapprox 1+(7/9)2^{-2-2d}$. So

$$(A_{1+} - E_{\text{lim}})/E_{\text{lim}} \lessapprox (1+5/9)2^{-3-2d}. \quad (6.48)$$

Remark: In the first group of uni-bootstraps of Subsection 6.1, after the first uni-bootstrap based on $f^{\text{init}} = f_{l_N-1}$, the leftover error e_1 is bounded by

$$|e_1| \le \varDelta/4 + \sqrt{E_{\text{MDS}}^2 \varDelta^2/(2N)^2 + E_B^2} < \varDelta/2. \quad (6.49)$$

Compared with the initial error, the bound of e_1 is reduced by less than 1 bit, yet starting from this error bound, the second uni-bootstrap reduces the error bound to within $\varDelta/4$. If the initial error is bounded by (6.49), obviously with only one uni-bootstrap based on f, the error bound can be reduced to $\varDelta/4$. This observation suggests using only one uni-bootstrap based on f_p for some appropriate resolution p, to reduce a large but not too large initial error to a small one.

6.4 Putting everything together into ciphertext form

For error bootstrapping, the global parameters are the same as in plaintext bootstrapping. There is also a switch to control the third group: when the switch is on, third group is executed, otherwise the whole bootstrapping finishes. In the following error bootstrapping algorithm, $d_B > 1$ is assumed. If $d_B = 2$, it is further assumed that either $k_B \le 3$, or $d_I > 0$, or $\kappa_I \ge 2$, so that the first group always works.

Algorithm 3 "ErrorBoot": Tail error block bootstrapping

Input: LWE ciphertext ct to bootstrap;
initial error bound $E_{\text{init}} = 2^{-d_I}(1-2^{-\kappa_I})\delta/2$ where $d_I \ge 0$, or $E_{\text{init}} = (1+2^{-\kappa_I})\delta/4$ where $\kappa_I \ge 2$;
third group lock (on/off).
Output: LWE ciphertext ct'.

```
    {First group}
 1: if $d_I = 0$ then
 2:     Compute $p, d_0$ in the first group. ($d_0+2$ is the size of the first block, $p \ge -1$ is the optimal resolution of $f$
        for the second uni-bootstrap)
 3:     ct' ⟵ UniBoot($f^{\text{init}}$, ct, $\varDelta$)  mod $q$;
        ct ⟵ ct − ct'  mod $q$;
        ct' ⟵ ct' + UniBoot($f_p$, ct, $\varDelta$)  mod $q$.
 4: else
 5:     Set $d_0 = d_I - 1$.
 6: end if
    {Second group}
 7: Compute $w$ and $d_i$ for $1 \le i \le w$. ($w \ge 0$ is the number of blocks in the second group, $d_i$ is the size of the $i$-th
    block in the group)
 8: Set $d'_0 = d_0$. Set $d'_i = d'_{i-1} + d_i$ for $1 \le i < w$.
 9: for $i = 1$ to $w$ do
10:     ct' ⟵ ct' + UniBoot($f$, ct, $\varDelta/2^{d'_{i-1}}$)  mod $q$;
        ct ⟵ ct − ct'  mod $q$.
11: end for
    {Third group}
12: if third group = on then
```

```
13:     ct' ⟵ ct' + UniBoot($f$, ct, $\Delta/2^{d_B+1}$)  mod $q$.
14: end if
15: return ct'.
```

The complexity of the algorithm is also dominated by the number of uni-bootstraps. As an example, consider the small initial error case $d_I = 1$ and $d_B > 2$. The bootstrapping starts directly from the second group. Suppose the third group is switched off. In the second group, when i changes from 1 to $w+1$, variable $d' = d'_{i-1}$ increases from $d_I - 1 = 0$ to $d_B - 1$. If we denote

$$l' = d_B - d', \tag{6.50}$$

then variable l' decreases from d_B to 1.

In our experiments, $k_B = k+1$ and $d \geq k/2$. Under these assumptions, by Lemma 30 and $\lceil (k-1)/2 \rceil = \lceil k/2 \rceil - 1$,

– when $l' \in [d+2+\lceil (k-1)/2 \rceil, d_B] = [d+1+\lceil k/2 \rceil, d_B]$, the blocks have size $d+1$. The number of blocks in this interval of l' is

$$v_1 := \lceil \max(0, d_B - d - \lceil k/2 \rceil)/(d+1) \rceil = \max(0, \lceil (d_B + 1 - \lceil k/2 \rceil)/(d+1) \rceil - 1). \tag{6.51}$$

After these blocks are bootstrapped, the interval of l' is reduced to $[1, d_B - v_1(d+1)]$.

– When $l' \in [d+2, d+\lceil k/2 \rceil] \cap [1, d_B - v_1(d+1)] = [d+2, d_B - v_1(d+1)]$, the blocks have size d; the number of blocks in this region of l' is

$$v_2 := \lceil \max(0, d_B - (v_1+1)(d+1))/d \rceil = \max(0, \lceil (d_B - v_1 - 1)/(d+1) \rceil - v_1 - 1). \tag{6.52}$$

– When $l' \in [\max(3, d+3-\lfloor (k_B-1)/2 \rfloor), d+1] \cap [1, d_B - v_1(d+1) - v_2 d] = [d+3-\lfloor k/2 \rfloor, d_B - v_1(d+1) - v_2 d]$, the block has size $l'-2$. This interval contains $\leq (d_B - v_1(d+1) - v_2 d) - 2$ integers, and at the beginning of this stage, $l' = d_B - v_1(d+1) - v_2 d$. So there is at most one block in this stage. The number of blocks in this stage is

$$v_3 := \mathtt{is}(d_B - v_1(d+1) - v_2 d > d + 2 - \lfloor k/2 \rfloor). \tag{6.53}$$

After this stage, $l' = (d_B - v_1(d+1) - v_2 d)(1 - v_3) + 2v_3$.

– When $l' \in [2, d+2-\lfloor (k_B-1)/2 \rfloor] \cap [1, (d_B - v_1(d+1) - v_2 d)(1-v_3) + 2v_3] = [2, (d_B - v_1(d+1) - v_2 d)(1-v_3) + 2v_3]$, the block has size $l'-1$. Since $l' = (d_B - v_1(d+1) - v_2 d)(1-v_3) + 2v_3$ at the beginning of this stage, there is at most one block in this stage, and it is the last block. The number of blocks in this stage is

$$v_4 := \mathtt{is}((d_B - v_1(d+1) - v_2 d)(1 - v_3) + 2v_3 > 1). \tag{6.54}$$

So when $d_I = 1$ and $d_B > 2$, the total number of uni-bootstraps is

$$v_E := v_1 + v_2 + v_3 + v_4. \tag{6.55}$$

In greedy mode, $k = 2$, so

$$v_1 = \max(0, \lceil d_B/(d+1) \rceil - 1), \quad v_2 = v_3 = 0, \quad v_4 = \mathtt{is}(d_B - v_1(d+1) > 1), \tag{6.56}$$

the total number of uni-bootstraps in greedy mode is

$$v'_E := v_1 + v_4. \tag{6.57}$$

For example, when $l_q = 29, d_I = 1, k = 2, k_B = 3$, the error bootstrapping is in greedy mode. When $l = 5, l_B = 6, d = 4$, for long plaintext of $l_t = 15$ bits, since $d_B = l_q - l_t - 1 - l_B = 7$, we have $v_1 = 1 = v_4$, so 2 uni-bootstraps suffice to finish the bootstrapping in two blocks, with size 7, 1 respectively, with final error bound $2^6 \approx E_B = 2^6 - 1$. If the final error bound is controlled to within 2^7, then one uni-bootstrap is sufficient. In contrast, the head-on approach in greedy mode needs 4 uni-bootstraps, with final error bound $\sqrt{4}E_B \approx 2^7$; the tail-up approach in greedy mode needs 6 uni-bootstraps, with final error bound $\sqrt{6}E_B > 2^7$. The bootstrapping efficiency is improved by $4 - 1 = 300\%$ and $6 - 1 = 500\%$ respectively.

If the plaintext is short, say $l_t = d + 2 = 5$ for $l = 6, l_B = 7, d = 3$, suppose that the initial error is small, and both the plaintext bootstrapping and the error bootstrapping are in greedy mode. In plaintext bootstrapping one block and two uni-bootstraps suffices to reduce the error bound to about $2E_B$, and the head-on approach agrees with the tail-up approach. When the goal is to reduce the tail error bound to about E_B, the result of plaintext bootstrapping needs to go through an additional error uni-bootstrap, so three uni-bootstraps are needed for plaintext bootstrapping to reach the goal. In direct error bootstrapping, now that $d_B + 1 = l_q - l_t - l_B = 17$, the sequence of block sizes is 6,4,4,3, so four blocks and four uni-bootstraps are needed to reduce the error bound to about E_B. For short plaintext and large error space, direct error bootstrapping is not as efficient as plaintext bootstrapping followed by error bootstrapping.

The last example is the one used in illustrating the tail-up approach with large initial error, where $l = k = 5$, $d_I = 0$, $d = 4$, $d_B = d + 1 = 5$, $k_I = 2d_B - 4 = 2d - 2 = 6$ and $l_t = 6$. Recall that in the tail-up approach, three blocks and 5 uni-bootstraps are needed; in the head-on approach, two blocks and 3 uni-bootstraps are needed. For error-bootstrapping, by Lemma 29, the first block has size $d + 1 = d_B$, so with one block and two uni-bootstraps, the tail error is reduced to about $2E_B$, just like the other two approaches. The bootstrapping efficiency is improved by $(5 - 2)/2 = 150\%$ and $(3 - 2)/2 = 50\%$ respectively.

6.5 Combination of bootstrapping strategies

By now we have introduced blockwise plaintext bootstrapping and blockwise tail error bootstrapping. For BGV-format FHE ciphertext, if the plaintext is short while the error space is long, then plaintext bootstrapping is more efficient, and after the plaintext bootstrapping finishes, the second group of error bootstrapping can be used to further reduce the error to about E_B. If the plaintext is not short, and the tail is error is not too long, then direct error bootstrapping is efficient for backup bootstrapping.

For large initial error and short plaintext, the first group of error bootstrapping can be used to reduce the tail error before plaintext bootstrapping, which allows the initial error to be the largest among all existing strategies for blockwise bootstrapping. So the first group of error bootstrapping serves as the first-block error cleaner for plaintext bootstrapping. When plaintext bootstrapping finishes, the second group of error bootstrapping serves as the final error cleaner. Of course, the third group of error bootstrapping can continue to reduce the error bound nearly to the limit.

For BGV-format FHE ciphertext, the above blockwise bootstrapping works by first converting the ciphertext into BGV format, and after the whole bootstrapping finishes, convert the format back to BGV. For CKKS-format FHE ciphertext, the strategies need some changes.

First, in the phase of a CKKS ciphertext, the plaintext and the tail error has no gap, while the plaintext and the head error usually has gap of at least two bits. So for plaintext bootstrapping of CKKS ciphertext, the second group of error bootstrapping can be resorted to, where the "error" is just the plaintext together with the tail error. At this time, the leftover error bound can be allowed to be much bigger than E_B, as long as the Li-Michiancio security [24] is guaranteed.

Second, since the main objective of CKKS ciphertext bootstrapping is to push the tail error qI farther away from the plaintext, in a larger ciphertext space with modulus $Q > 2^2 qI$, the second group of head-on plaintext bootstrapping can be used to delete qI from the phase, where the "plaintext" is just the tail error qI, and the usual gap between the plaintext and tail error in head-on plaintext bootstrapping, is now the gap between the head error and the CKKS plaintext.

Third, if the main objective is to smudge the tail error in order to achieve Li-Michiancio security, then starting from a preset bit position i in CKKS plaintext, the error bootstrapping can be used to clear all the bits of the plaintext from bit i to the right end, in the sense that (1) the leftover error bound is not reduced, but enlarged by a term 2^{i+1}, (2) the leftover error has all the bits from bit i to the LSB of E_B vacant.

For CKKS ciphertext, usually the head-error is short while the plaintext is long. From this aspect, bootstrapping the head error is optimal. On the other hand, bootstrapping the head error requires a bigger modulus $Q > q$ to encrypt the phase of the CKKS ciphertext, which slows down each uni-bootstrap.

The reason why $|I|$ is small in head error qI can be shown as follows. Suppose $\mathtt{ct} = (a_1, \ldots, a_n, b) \in \mathbb{Z}_q^{n+1}$ is a CKKS ciphertext with uniform ternary secret $\mathbf{s} = (s_1, \ldots, s_n) \in \mathbb{Z}_3^n$. Then $\text{phase}(\mathtt{ct}) = b - \sum_{s_i \neq 0} a_i s_i$. Since $|b|, |a_i| \leq q/2$, $|\text{phase}(\mathtt{ct})| \leq (1 + 2n/3) \times q/2$. From $\text{phase}(\mathtt{ct}) = qI + m + e$ where $|m + e| \leq q/2$, we get

$$|I| \leq n/3 + 1/2. \tag{6.58}$$

7 Experiments

Estimation and practical tests of E_B:

Bootstrapping (UniBoot) results in a ciphertext with an error from Gaussian distribution of standard deviation $\sigma_B = \sqrt{\frac{q}{Q}(\sigma_{ACC}^2 + \sigma_{KS}^2) + \sigma_{MS}^2}$, where σ_{ACC}, σ_{KS} and σ_{MS} are the variance contributions by the operations: accumulator, key switching, and modulus switching, respectively. Unlike the previous cases [25,26], our long-precision bootstrapping chooses a large cipher module (e.g., $Q = 2^{60}$) at work space, so that the main variance contribution comes from the modulus switching rather than the accumulator or key switching, the contribution of which is negligible by scaling q/Q.

For example, suppose that the bit decomposition for Q is $Q = {B_g}^{d_g}$, and then we have $\sigma_{ACC} = 4d_g nNB_g\sigma_0^2$. With the practical parameters $n = 2^{10}$, $N = 2^{11}$, $Q = 2^{60}$, $q = 2^{29}$, $B_g = 2^{15}$, $d_g = 4$ and $\sigma_0 = 3.7$, it holds that $\sigma_B \approx e_{MS} = \sqrt{2^{11}}/6$, and thus the growth of error in ciphertexts cased by each UniBoot will not exceed 2^6 with failure probability $< 10^{-9}$.

Overall behavior:

When compared to normal head-on bootstrapping approach, the error bootstrapping ("ErrorBoot") is more efficient if the plaintext size l_t is much longer than the error size l_I. Table 1 and Figure 5 show the efficiency comparison between normal and error bootstrapping. Figure gives the under different combination l_t and l_I at $l_q = 200$. Table 1 make the comparison under practical parameters ($l_q = 29$, $l_B = 6$) used in [25,26]. Let the number of uniBoots required for ErrorBoot be V_H, and let V'_H be that number in the greedy mode.

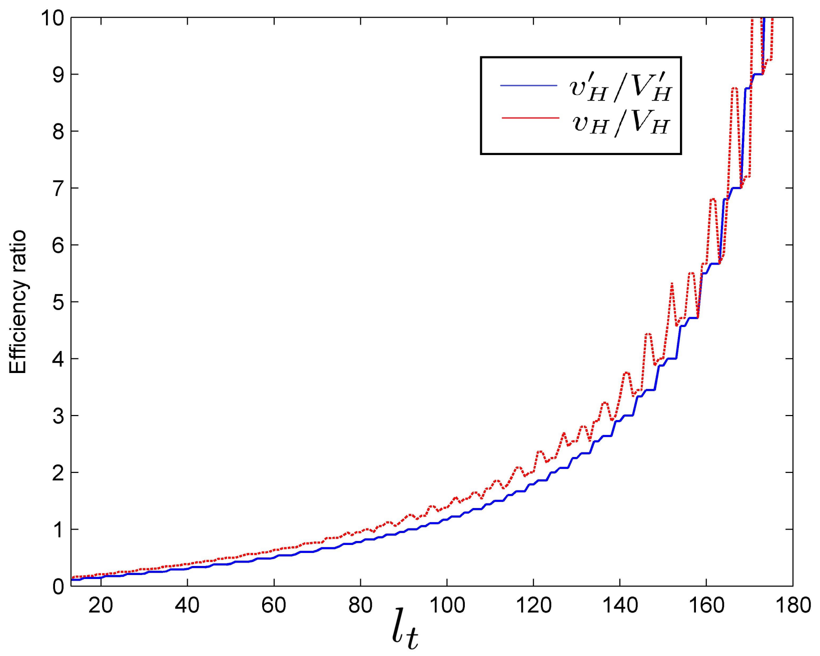

Fig. 5. Efficiency ratio of the ErrorBoot over the normal head-on approach, measured by the number of uni-bootstraps: v_H/V_H and v'_H/V'_H at $l_q = 200$.

In Figure 5, the bootstrapping efficiency of different approaches changes as the plaintext sizes changes. For ciphertext modulo $l_q = 200$, the plaintext size $l_t = 90$ is a cut-off point: when $l_t > 90$, the ErrorBoot approach is more efficiency. To ensure a fair comparison, we stop the ErrorBoot when the resulting error is reduced to the same magnitude as using normal head-on bootstrapping method.

Test	l_q	l_t	V		Runtime [ms]		Speedup
			v'_H	V'_H	Head-on	ErrorBoot	
I	17	10	3	1	3,953	627	× 6.3
II	22	16	5	1	3,953	653	×6.05
III	27	20	5	1	3,991	666	×5.99
IV	29	22	6	1	4,763	671	× 7.09

Table 1. Tail error block bootstrapping tests with parameters from [25].

7.1 Experimental results on run-time cost

$l_q = 29$, $l_B = 6$:

$l_t = 18$: $v = 6$ or 7: as long as as $v < 2^4$, $2\sqrt{v}E_B < 2^9$, so $l_I \geq 9$, for small initial error $l_I + 2 \geq 11$.

$l_t = 18$: maximal size supporting effective backup bootstrapping using plaintext bootstrapping starting with small initial error

19: maximal size supporting effective backup bootstrapping using plaintext bootstrapping

20: maximal size supporting effective backup bootstrapping using error bootstrapping

21: maximal size supporting sign bit extraction using tail-up approach; also maximal size supporting effective backup bootstrapping using LSB precursor and error bootstrapping.

22: maximal size supporting sign bit extraction using LSB precursor.

Time-cost comparison: tail-up approach vs head-on approach:

The numbers of uni-bootstraps in the tail-up approach and the head-on approach, in both the regular case and the greedy case, are respectively: v_T, v'_T, v_H, v'_H.

For fixed d, when l_t tends to infinity, the ratio $v_H : v_T$ tends to $(d+2) : (2d)$, so does $v'_H : v'_T$. When $d = 2, 3, 4, 5$, the limit of the ratio is $1, 0.83, 0.75, 0.7$ respectively. The lower bound of the limit is 0.5.

Fig. 6 shows the reduction rates on the number of uni-bootstraps of the head-on approach over the tail-up approach: $1 - v_H/v_T$ and $1 - v'_H/v'_T$, at $d = 4$. Starting from about $l_t = 25$, the reduction rates oscillate slightly around the value 25%.

When l_t is fixed but d varies, Fig. 7 presents the runtime save percentages of the head-on approach over the tail-up approach at $l_t = 200$. When $d = 20$, the upper bound of twice speedup is almost reached.

The experiments are conducted on a laptop with Intel(R) Core(TM) i5-7500 3.40 GHz CPU and 16GB RAM. The NTL library is not installed. The Palisade library is the MinGW64 g++ 11.2.0 version v1.11.5. By experiments, $l_Q = 60, l_q = 29$ are the maximal values of the two parameters supported by FHEW/TFHE in the Palisade library.

In [25], the largest precision supported is $l_t = 22$. Since the purpose of the tests in [25] is to back up only the sign bit, every time a block is bootstrapped, the modulus number of the leftover plaintext is reduced by ciphertext modulus switch, so that l_Q, l_q, l_t can be decreased accordingly. For backup-purposed bootstrapping, these parameters cannot change, because all the plaintext blocks must be concatenated after bootstrapping.

We choose the following parameters that are the same as in [25]:

$$l_n = 9, \quad l_N = 11, \quad l_Q = 60, \quad B_g = 2^{14}. \tag{7.1}$$

By experiments, under the above common parameters,

- $l_t = 20$ is the largest value supporting $E_{\text{fin}} < \Delta/2$, where E_{fin} is the refreshed error of the output ciphertext after the whole bootstrapping;
- $l_t = 18$ is the largest value supporting $E_{\text{fin}} \leq E_{\text{init}}/2$, when $E_{\text{init}} = \Delta/4$.

Since we test backup bootstrapping, the run-time in our experiments is longer than that in [25]. In our tests, each uni-bootstrap takes about 0.8s, and the overwhelming majority of the run-time is taken by the uni-bootstraps.

As to parameter d, in [25] it is 3 ($3 + 2 = 5$ is the block size there), while in our tests we try both 3 and 4. When $d = 4$, some experimental results are presented in Table 2 for regular block bootstrapping, and in Table 3 for greedy block bootstrapping.

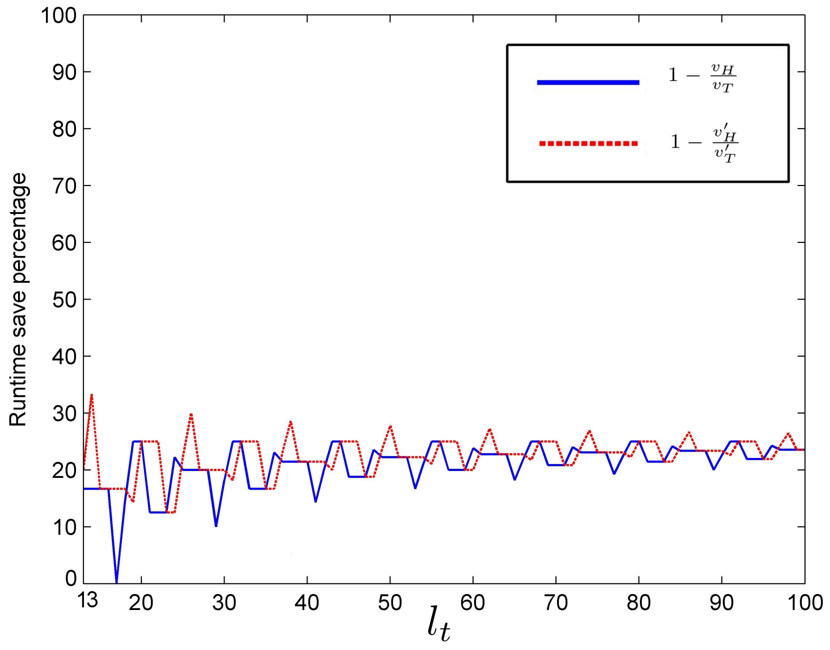

Fig. 6. Reduction rates on the number of uni-bootstraps: $1 - v_H/v_T$ and $1 - v'_H/v'_T$ of the head-on approach over the tail-up approach at $d = 4$.

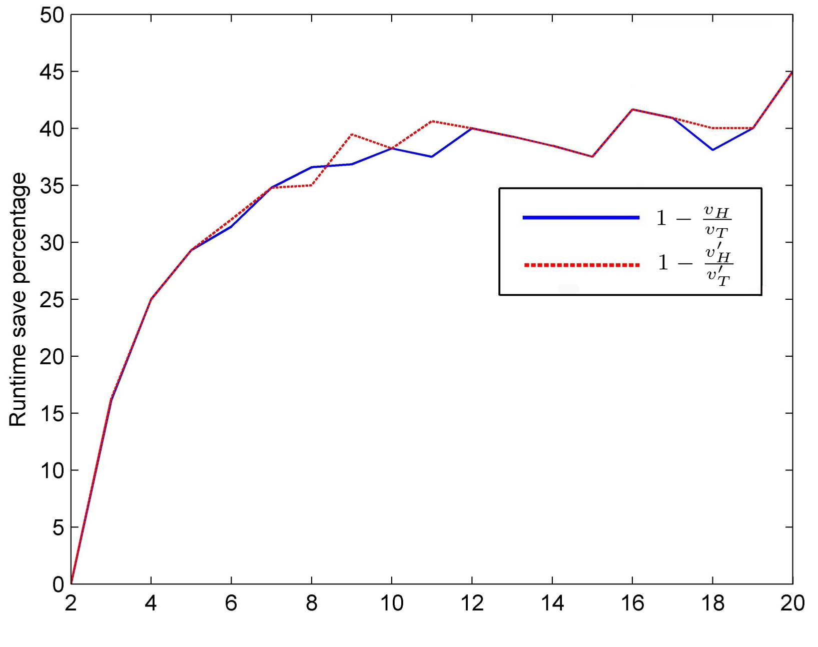

Fig. 7. Reduction rates on the number of uni-bootstraps: $1 - v_H/v_T$ and $1 - v'_H/v'_T$ of the head-on approach over the tail-up approach at $l_t = 200$.

Test	l_q	l_t	V		Runtime [ms]		Runtime save
			V_H	V_T	head-on	tail-up	
I	22	16	5	6	3,953	4,825	18.0%
II	23	17	6	6	4,625	4,831	4.26%
III	25	18	6	7	4,654	5,632	17.3%
IV	27	19	6	8	4,689	6,431	27.0%
V	29	20	6	8	4,706	6,447	27.0%

Table 2. Block bootstrapping tests with $d = 4$.

Test	l_q	l_t	V'		Runtime [ms]		Runtime save
			V'_H	V'_T	head-on	tail-up	
VI	22	16	5	6	3,953	4,825	18.0%
VII	23	17	5	6	3,971	4,831	17.8%
VIII	25	18	5	6	3,982	4,840	17.7%
IX	27	19	6	7	4,689	5,651	17.0%
X	29	20	6	8	4,706	6,447	27.0%

Table 3. Greedy block bootstrapping tests with $d = 4$.

Remark : According to [25], the runtime of Homsign function with parameters $l_q = 22$ and $l_t = 16$ is $1,658$ ms. The reason why our time is significantly higher than that of [25] is because that our aim is exact bootstrapping, not just recovery of sign bits. Based on the idea of [25], we test the tail-up bootstrapping with $l_q = 22, l_t = 16$, $d = 3$. The runtime is ...

For fixed n, the parameters l, k of $E_{\text{MDS}} \le 2^l(1 - 2^{-k})$ are determined by the failure probability and verified by simulation. In Subsection 2.3, it was shown that e_{MDS} can be taken as a Gaussian distribution with standard deviation (2.21), so that the failure probability of bound E_{MDS} for e_{MDS} can be calculated by (2.23). The following are some results on parameter $l = \lceil \log E_{\text{MDS}} \rceil$:

When $l_n = 9$,

- $l = 5, k = 4$ has failure probability 1.86×10^{-8};
- $l = 6, k = 5$ has failure probability 3.08×10^{-31}.

When $l_n = 10$,

- $l = 5, k = 4$ has failure probability 6.97×10^{-5};
- $l = 6, k = 5$ with failure probability 2.03×10^{-16}.

In simulation of e_{MDS} by the sum of i.d.d. uniformly random variables in $[-1/2, 1/2]$, 100,000 tests are made for each $l_n \in \{9, 10\}$. The results are the following:

- When $l_n = 9$, then in all the tests, $|e_{\text{MDS}}| \le 21.9 < 24 = 2^5 - 2^3$, indicating the smaller bound $l = 5$, $k = 2$. On the other hand, the theoretical failure probability of $|e_{\text{MDS}}| > 24$ is 6.8×10^{-6}.
- when $l_n = 10$, then $|e_{\text{MDS}}| < 33.19 <= 48 = 2^6 - 2^4$, indicating the smaller bound $l = 6$, $k = 2$. On the other hand, the theoretical failure probability of $|e_{\text{MDS}}| > 48$ is 1.97×10^{-10}.

Both theoretical failure probability and simulation tests support $l = 5$ when $l_n = 9$. This is equivalent to $d = l_N - l - 2 = 4$, instead of $d = 3$ in [25]. In [29], when $l_n = 11$, $l_N = 12$ or 13, then $d = 4$ or 5, indicating $l = 6$. If the secret key is uniform ternary, then the failure probability is pretty high for $l = 6$.

When $E_{\text{init}} = \Delta/4$, all the inequality constraints are satisfied.

If v is the number of uni-bootstraps in the head-on approach, then asymptotically $v \times 2d/(d + 2)$ is the number of uni-bootstraps in the tail-up approach. With the same initial error bound E_{init}, the ratio of the refreshed errors by the two approaches is asymptotically $\sqrt{v} : \sqrt{v \times 2d/(d+2)} = \sqrt{(d+2)/(2d)}$. The upper bound of the ratio is $1/\sqrt{2}$.

In general, the *bootstrapping efficiency* is the relative error decrease per uni-bootstrap. In the heuristic case, the efficiency of a bootstrapping method consisting of v uni-bootstraps is

$$F(v) := (1 - E_{\text{fin}}/E_{\text{init}}) \times (1/v) = (1 - \sqrt{v}E_B/E_{\text{init}}) \times (1/v). \tag{7.2}$$

For a practical bootstrapping on a ciphertext with input error e_{init} and output error e_{fin}, the bootstrapping efficiency is

$$F := (1 - |e_{\text{fin}}/e_{\text{init}}|) \times (1/T), \tag{7.3}$$

where T is the total time cost.

when $d = 4$ and $E_{\text{init}} = \Delta/4$. By experiments, $E_B \sim N(0, 16)$, and $E_{\text{fin}} = \sqrt{v}E_B$. Let $X_i \sim N(0, 16)$. Then $e_{\text{fin}} = \sum_{i \in [v]} X_i$. In order for $Pr(|e_{\text{fin}}| > E_{\text{fin}}) \leq 2 \times 10^{-9}$ $(\approx 2 - 2\text{erf}(6/\sqrt{2}))$, theoretically

$$E_{\text{fin}} = 6 \times 16\sqrt{v} = 96\sqrt{v} \tag{7.4}$$

suffices.

In experiments, we calculated $|e_{\text{fin}}| = |\sum_{i \in [v]} X_i|$ for each of $v \in \{5, 6, 7, 8\}$ 10,000 times. The occurred results are all within the theoretical bound (7.4):

– For $v = 5$, $|e_{\text{fin}}| \leq 166.1 < 214 \approx E_{\text{fin}}$, with average 28.6, and median 24.2;
– for $v = 6$, $|e_{\text{fin}}| \leq 180.1 < 235 \approx E_{\text{fin}}$, with average 31.2, and median 26.5;
– for $v = 7$, $|e_{\text{fin}}| \leq 200.5 < 253 \approx E_{\text{fin}}$, with average 33.7, and median 28.6;
– for $v = 8$, $|e_{\text{fin}}| \leq 212.0 < 271 \approx E_{\text{fin}}$, with average 36.1, and median 30.5.

8 Conclusion

In this paper, we proposed a new approach to large-precision bootstrapping in FHEW/TFHE cryptosystem. It starts from the head block and backs up the blocks approximately, getting the accurate result at the end of the whole bootstrapping procedure. The algorithm is faster and at the same time outputs smaller refreshed error.

A future work on large-precision bootstrapping is programmable functional bootstrapping. In [25] a special function was investigated: the sign function defined on large-precision plaintexts. This function permits faster computing by reducing the input ciphertext modulus gradually. For more general programmable function defined on large-precision plaintexts, how to speed up corresponding functional bootstrapping is an important open problem.

Developing SIMD execution of FHEW/TFHE bootstrapping remains another open problem. 4 years ago, a method based on Nussbaumer transform [28] was proposed to deal with this problem. It has theoretical asymptotic analysis but no practical implementation. There is no more progress in this important direction by now.

References

1. Alperin-Sheriff, J. and Peikert, C. Faster bootstrapping with polynomial error. In: *CRYPTO 2014*, LNCS **8616**, pp. 297-314, 2014.
2. Brakerski, Z. Fully homomorphic encryption without modulus switching from classical GapSVP. In: *CRYPTO 2012*, LNCS **7417**, pp. 868-886, 2012.
3. Brakerski, Z., Gentry, C., and Vaikuntanathan, V. (Leveled) fully homomorphic encryption without bootstrapping. In: *ITCS 2012*, pp. 309-325, 2012.
4. FHEW with efficient multibit bootstrapping. In: *LATINCRYPT 2015*, LNCS **9230**, pp. 119-135, 2015.
5. Chen, H., Chillotti, I., and Song, Y. Improved bootstrapping for approximate homomorphic encryption. In: *EUROCRYPT 2019*, LNCS **11477**, pp. 34-54, 2019.
6. Cheon, J., Han, K., Kim, A., Kim, M., and Song, S. Bootstrapping for approximate homomorphic encryption. 2018. In: *EUROCRYPT 2018*, LNCS **10820**, pp. 360-384, 2018.
7. Chillotti, I., Gama, N., Georgieva, M., and Izabachène, M. Faster fully homomorphic encryption: bootstrapping in less than 0.1 seconds. In: *ASIACRYPT 2016*, LNCS **10031**, pp. 3-33, 2016.

8. Chillotti, I., Gama, N., Georgieva, M., and Izabachène, M. Improving TFHE: faster packed homomorphic operations and efficient circuit bootstrapping. In: *ASIACRYPT 2017*, LNCS **10624**, pp. 377-408, 2017.
9. Carpov, S., Izabachène, M., and Mollimard, V. New techniques for multi-value input homomorphic evaluation and applications. In: *CT-RSA 2019*, LNCS **11450**, pp. 106-126, 2019.
10. Chillotti, I., Joye, M., and Paillier, P. Programmable bootstrapping enables efficient homomorphic inference of deep neural networks. *Cryptology ePrint Archive Report* 2021/091, 2021.
11. Cheon, J., Kim, A., Kim, M., and Song, Y. Homomorphic encryption for arithmetic of approximate numbers. In: *ASIACRYPT 2017*, LNCS **10624**, pp. 409-437, 2017.
12. Chillotti, I., Ligier, D., Orfila, J.-B., and Tap, S. Improved programmable bootstrapping with larger precision and efficient circuits for TFHE. *Cryptology ePrint Archive Report* 2021/729, 2021.
13. Chen, L. and Zhang, Z. Bootstrapping fully homomorphic encryption with ring plaintexts within polynomial noise. In: *ProvSec 2017*, LNCS **10592**, pp. 285-304, 2017.
14. Ducas, L. and Micciancio, D. FHEW: bootstrapping homomorphic encryption in less than a second. In: *EUROCRYPT 2015*, LNCS **9056**, pp. 617-640, 2015.
15. Fan, J. and Vercauteran, F. Somewhat practical fully homomorphic encryption. *Cryptology ePrint Archive Report* 2012/144, 2012.
16. Gentry, C. Fully homomorphic encryption using ideal lattices. In: *STOC 2009*, pp. 169-178, 2009.
17. Guimarães, A., Borin, E., and Aranha, D.F. Revisiting the functional bootstrapping in TFHE. *IACR Transactions on Cryptographic Hardware and Embedded Systems* 2021(2): 229-253, 2021.
18. Gentry, C., Halevi, S., and Smart, N.P. Better Bootstrapping in fully homomorphic encryption. In: *PKC 2012*, LNCS **7293**, pp. 1-16, 2012.
19. Gentry, C., Sahai, A., and Waters, B. Homomorphic encryption from learning with errors: conceptually-simpler, asymptotically-faster, attribute-based. In: *CRYPTO 2013*, LNCS **8042**, pp. 75-92, 2013.
20. Hirosama, R., Abe, M., and Okamoto, T. Packing messages and optimizing bootstrapping in GSW-FHE. In: *PKC 2015*, LNCS **9020**, pp. 699-715, 2015.
21. Halevi, S. and Shoup, V. Bootstrapping for HElib. In: *EUROCRYPT 2015*, LNCS **9056**, pp.641-670, 2015.
22. Kim, A., Deryabin, M., Eom, J., Choi, R., Lee, Y., Ghang, W., and Yoo, D. General bootstrapping approach for RLWE-based homomorphic encryption. *Cryptology ePrint Archive Report* 2021/691, 2021.
23. FDFB: fully domain functional bootstrapping towards practical fully homomorphic encryption. *Cryptology ePrint Archive Report* 2021/1135, 2021.
24. Li, B. and Micciancio, D. On the security of homomorphic encryption on approximate numbers. In: *EUROCRYPT 2021*, pp. 648-677, 2021.
25. Liu, Z., Micciancio, D., and Polyakov, Y. Large-precision homomorphic sign evaluation using FHEW/TFHE bootstrapping. *Cryptology ePrint Archive Report* 2021/1137, 2021.
26. Liu, C., Wang, A., and Zheng, Z. Optimizing bootstrapping and evaluating large FHE gates in the LWE-based GSW-FHE. *Cryptology ePrint Archive Report* 2021/490, 2021.
27. Micciancio, D. and Polyakov, Y. Bootstrapping in FHEW-like cryptosystem. 2020. *Cryptology ePrint Archive Report* 2020/086, 2020.
28. Micciancio, D. and Sorrell, J. Ring packing and amortized FHEW bootstrapping. In: *ICALP 2018*, pp. 100:1-14, 2018.
29. Yang, Z., Xie, X., Shen, H., Chen, S., and Zhou, J. TOTA: fully homomorphic encryption with smaller parameters and stronger security. *Cryptology ePrint Archive Report* 2021/1347, 2021.

Appendix. Fundamental inequalities on backup block bootstrapping

A.1 Fundamental inequalities on tail-up plaintext bootstrapping: the first group

In plaintext bootstrapping, we always assume the number of uni-bootstraps $v > 1$.

Lemma 13 Under the error quality constraint $E_{\text{fin}} = \sqrt{v}E_B \le E_{\text{init}}/2$,

$$v < 2^{2(d_B - d_I)-1}. \tag{A.1}$$

In particular, if $v > 1$, then $d_B \ge d_I + 2$.

Proof. In power-of-2 binomial bounds, the error quality constraint is

$$\sqrt{v}2^{-d_B}(1-2^{-k_B}) \le 2^{-1-d_I}(1-2^{-k_I}). \tag{A.2}$$

So $\sqrt{v} < 2^{d_B-1-d_I}/(1-2^{-k_B}) \le 2^{d_B-d_I+1}/3$, and (A.1) follows. Q.E.D.

In the rest of this section, we assume $d_B \ge d_I + 2$.

The condition for tail-up bootstrapping is (4.4), namely $\sqrt{E_{\mathrm{MDS}}^2 + (E_{\mathrm{init}}^2 + E_B^2)N^2/(2\Delta)^2} < N/4$. In practice, for simplicity we can use the stronger condition in the following lemma for tail-up bootstrapping.

Lemma 14 In power-of-2 binomial bounds, (4.4) for tail-up bootstrapping is true if either (1) $d_I > 0$, or (2) $d_I = 0$, $k \le 2d$, $k_I \le 2\min(d, d_B)$.

Proof. In power-of-2 binomial bounds, (4.4) becomes

$$2^{-2d_I}(1-2^{-k_I})^2 + 2^{-2d_B}(1-2^{-k_B})^2 + 2^{-2d}(1-2^{-k})^2 < 1. \tag{A.3}$$

When $d_I > 0$, by $d_B > d_I > 0$ and $d > 0$, the inequality is always true.

When $d_I = 0$, then $d_B \ge 2$, and (A.3) becomes

$$2^{-2d_B}(1-2^{-k_B})^2 + 2^{-2d}(1-2^{-k})^2 - 2^{1-k_I} + 2^{-2k_I} < 0. \tag{A.4}$$

There are three cases:

Case 1. $d_B > d$. Then (A.4) requires $1 - k_I \ge -2d$. *i.e.*, $k_I \le 2d + 1$. If $k_I \le 2d$, then (A.4) is true.

Case 2. $d_B = d$. Then (A.4) requires $1 - k_I > -2d$, namely $k_I \le 2d$. If $k_I = 2d$, then (A.4) becomes

$$-2^{1-k_B} + 2^{-2k_B} - 2^{1-k} + 2^{-2k} + 2^{-2d} < 0. \tag{A.5}$$

When $k \le 2d$, then $1 - k > -2d$, and the above inequality is true.

Case 3. $d_B < d$. Then (A.4) requires $1 - k_I \ge -2d_B$, namely $k_I \le 2d_B + 1$. When $k_I \le 2d_B$, then (A.4) is true. Q.E.D.

For uniformly random ternary key $s \in \mathbb{Z}_n$ of the input ciphertext, the following inequality is usually satisfied:

$$k \le 2d, \quad i.e., \quad l_N \ge l + k/2 + 2. \tag{A.6}$$

So essentially Lemma 14 states that for small intial error, if $k_I \le 2\min(d, d_B)$, then the tail-up approach is possible. On the other hand, the condition provided in Lemma 14 is sufficient but not necessary. A full version of tail-up bootstrapping condition (4.4) in power-of-2 binomial bounds, will be given in Lemma 17 as a corollary of the following two lemmas.

The following lemma gives the maximal size of the first block when the initial error is large.

Lemma 15 Let $d_I > 0$. Let d_0 be the biggest integer satisfying (4.2), or equivalently,

$$2^{2d_0}E_{\mathrm{MDS}}^2(4/N)^2 + (E_{\mathrm{init}}^2 + E_B^2)(2/\Delta)^2 < 1. \tag{A.7}$$

Then $d_0 = d - 1$, except for the following cases where $d_0 = d$ (called greedy mode): either (1) $k \le 2d_I$, or (2) $k = 2d_I + 1$ and one of the following is true:

- $d_B = 2d_I + 1 \ge k_I$;
- $d_B = 2d_I + 1$ and $k_I = 2d_I + 2 \ge k_B$;
- $d_B \le 2d_I$ and $k_I \le 2(d_B - d_I)$;
- $k_I = 2(d_B - d_I) + 1$ and $2d_B = 3d_I$ and $k_B \le k_I$;
- $k_I = 2(d_B - d_I) + 1$ and $4d_I \ge 2d_B > 3d_I$ and $k_B \le 2(2d_I - d_B + 1)$;
- $k_I = 2(d_B - d_I) + 1$ and $2d_B < 3d_I$ and $k_B \le 2(d_B - d_I + 1)$;
- $d_B \ge 2d_I + 2 \ge k_I$.

Proof. In power-of-2 binomial bounds, (A.7) becomes

$$2^{-2d_I}(1-2^{-k_I})^2 + 2^{-2d_B}(1-2^{-k_B})^2 + 2^{-2(d-d_0)}(1-2^{-k})^2 < 1. \tag{A.8}$$

Obviously $0 \le d_0 \le d$. If $d_0 = d - 1$, then the left side of (A.8) $< 1/4 + 1/16 + 1/4 < 1$, so the inequality is true.

If $d_0 = d$, then (A.8) becomes

$$2^{-2d_I}(1-2^{-k_I})^2 + 2^{-2d_B}(1-2^{-k_B})^2 - 2^{1-k} + 2^{-2k} < 0. \tag{A.9}$$

So $k \le 2d_I + 1$. If $k \le 2d_I$, the above inequality is true. If $k = 2d_I + 1$, then (A.9) becomes

$$-2^{1-k_I} + 2^{-2k_I} + 2^{-2(d_B-d_I)}(1-2^{-k_B})^2 + 2^{-2-2d_I} < 0. \tag{A.10}$$

So $1 - k_I > -2 - 2d_I$ and $1 - k_I \ge -2(d_B - d_I)$, namely $k_I \le 2d_I + 2$ and $k_I \le 2(d_B - d_I) + 1$. There are three cases.

Case 1. $-2(d_B - d_I) = -2 - 2d_I$, namely $d_B = 2d_I + 1$. When $k_I \le 2d_I + 1$, (A.10) is true. When $k_I = 2d_I + 2$, (A.10) becomes $2^{-3-2d_I} - 2^{-k_B} + 2^{-1-2k_B} < 0$, which is true if and only if $k_B \le 2d_I + 2 = k_I$.

Case 2. $-2(d_B - d_I) > -2 - 2d_I$, namely $d_B \le 2d_I$. When $k_I \le 2(d_B - d_I)$, then (A.10) is true. When $k_I = 2(d_B - d_I) + 1$, (A.10) becomes

$$2^{-2(d_B-d_I+1)} - 2^{1-k_B} + 2^{-2k_B} + 2^{-2(2d_I-d_B+1)} < 0. \tag{A.11}$$

If $d_B - d_I + 1 = 2d_I - d_B + 1$, namely $2d_B = 3d_I$, then (A.11) is true if and only if $k_B \le 2(d_B - d_I + 1) - 1 = d_I + 1$. If $d_B - d_I + 1 > 2d_I - d_B + 1$, namely $2d_B > 3d_I$, then (A.11) is true if and only if $k_B \le 2(2d_I - d_B + 1)$. If $d_B - d_I + 1 < 2d_I - d_B + 1$, namely $2d_B < 3d_I$, then (A.11) is true if and only if $k_B \le 2(d_B - d_I + 1)$.

Case 3. $-2(d_B - d_I) < -2 - 2d_I$, namely $d_B \ge 2d_I + 2$. When $k_I \le 2d_I + 1$, (A.10) is true. When $k_I = 2d_I + 2$, (A.10) becomes $2^{-2d_I-2} + 2^{-2(d_B-2d_I-1)}(1-2^{-k_B})^2 < 1$, which is always true.

In the statement of the lemma, there are 7 cases, the first two of which are covered by Case 1, the last one is covered by Case 3, and the middle four are covered by Case 2. Q.E.D.

The following lemma gives the maximal size of the first block when the initial error is small.

Lemma 16 Let $d_I = 0$. Let d_0 be the biggest integer satisfying (A.7). Then integer d_0 exists if and only if either (1) $k_I \le 2d_B$, or (2) $k_I = 2d_B + 1$ and $k_B \le k_I + 1$. When d_0 exists, then $d_0 \le d - 1$, and in details, if $k_I \le 2d_B$, then $d_0 \in \{d - \lfloor k_I/2 \rfloor, d - \lfloor k_I/2 \rfloor - 1\}$, and $d_0 = d - \lfloor k_I/2 \rfloor$ if and only if one of the following is true:

1. $k_I = d_B$ is odd, and $k_I \ge k$;
2. $k_I = d_B = k - 1$ is odd, and $k \ge k_B$;
3. k_I is odd, $2d_B \ge k_I > d_B$, and $k < 2d_B + 2 - k_I$;
4. k_I is odd, $2d_B \ge k_I > d_B$, $k = 2d_B + 2 - k_I$, $4d_B = 3(k_I - 1)$, $k_B < 2(k_I - d_B)$;
5. k_I is odd, $2d_B \ge k_I > d_B$, $k = 2d_B + 2 - k_I$, $4d_B \ne 3(k_I - 1)$, $k_B \le \min(2k_I - 2d_B, 2d_B + 3 - k_I)$;
6. k_I is odd, $k_I < d_B$, $k \le k_I + 1$;
7. k_I is even, $k_I \le 2d_B - 2$;
8. $k_I = 2d_B$, $k = k_B \le 2d_B + 1$;
9. $k_I = 2d_B$, $k < k_B$, $k \le 2d_B$;
10. $k_I = 2d_B$, $k < k_B$, $k = 2d_B + 1$, $k_B \le 2k$;
11. $k_I = 2d_B$, $k > k_B$, $k_B \le 2d_B$;
12. $k_I = 2d_B$, $k > k_B$, $k_B = 2d_B + 1$, $k \le 2k_B$.

If $k_I = 2d_B + 1$ and $k_B \le k_I + 1$, then $d_0 \in \{d - d_B - \lfloor k_B/2 \rfloor, d - d_B - \lfloor k_B/2 \rfloor - 1\}$, and $d_0 = d - d_B - \lfloor k_B/2 \rfloor$ if and only if one of the following is true:

– k_B is odd, $k_I = 2k_B - 1$, $k \le k_B$;
– k_B is odd, $k_I \ge 2k_B$, $k \le k_B + 1$;
– k_B is odd, $k_I \le 2k_B - 2$, $k \le 2 + k_I - k_B$;
– k_B is even, $k_B \le k_I - 1$;
– $k_B = k_I + 1$, $k \le k_B$.

When $d_0 = d - d_B - \lfloor k_B/2 \rfloor$, then $d_0 = d - \lfloor (k_I + k_B - 1)/2 \rfloor$; when $d_0 = d - d_B - \lfloor k_B/2 \rfloor - 1$, then $d_0 = d - \lceil (k_I + k_B)/2 \rceil$.

In particular, $d_0 = d - 1$ if and only if one of the following is true:

– $k_I = 2$;
– $k_I = 3$, $d_B = 3$, $k \le 3$;

- $k_I = 3$, $d_B = 3$, $k = 4 \geq k_B$;
- $k_I = 3$, $d_B = 2$, $k = 2$;
- $k_I = 3$, $d_B = 2$, $k = 3$, $k_B = 2$;
- $k_I = 3$, $d_B \geq 4 \geq k$.

Proof. (A.8) is the power-of-2 binomial form of (A.7). Since $d_I = 0$, we have $d_B \geq 2$, and (A.8) becomes

$$2^{-2d_B}(1-2^{-k_B})^2 + 2^{-2(d-d_0)}(1-2^{-k})^2 - 2^{1-k_I} + 2^{-2k_I} < 0. \tag{A.12}$$

So $-2(d-d_0) \leq 1-k_I$, namely $d_0 \leq d-(k_I-1)/2$. In particular, $d_0 \leq d-1$.

We first prove the conditions on the existence of integer d_0. When $-2d_B > 1-k_I$, namely $k_I \geq 2d_B+2$, (A.12) is obviously false, so $k_I \leq 2d_B+1$. If $k_I \leq 2d_B$, then (A.12) is true for sufficiently small $d_0 < 0$. If $k_I = 2d_B+1$, (A.12) becomes

$$-2^{1-k_B} + 2^{-2k_B} + 2^{-2(d-d_0-d_B)}(1-2^{-k})^2 + 2^{-2-2d_B} < 0. \tag{A.13}$$

When $1-k_B \leq -2-2d_B$, namely $k_B \geq 2d_B+3$, the above inequality is false. When $1-k_B > -2-2d_B$, namely $k_B \leq 2d_B+2 = k_I+1$, (A.13) is true for sufficiently small $d_0 < 0$.

Before proving the conclusions on the exact value of d_0, we point out the following simple fact, whose proof is trivial:

$$\lfloor x/2 \rfloor = \lceil (x+1)/2 \rceil - 1, \quad \lfloor (x-1)/2 \rfloor = \lceil x/2 \rceil - 1, \quad \forall \mathbf{x} \in \mathbb{Z}. \tag{A.14}$$

In (A.12), the four terms on the left are divided into three groups: the group $2^{-2(d-d_0)}(1-2^{-k})^2$, which is positive; the group 2^{-2-2d_B}, also positive; and the group $-2^{1-k_I}+2^{-2k_I}$, negative. There are three cases between the first two groups.

Case 1. $-2(d-d_0) = 1-k_I$, namely k_I is odd, and $d_0 = d-(k_I-1)/2 = d-\lfloor k_I/2 \rfloor$. (A.12) becomes

$$2^{-(2d_B+1-k_I)}(1-2^{-k_B})^2 + 2^{-1-k_I} - 2^{1-k} + 2^{-2k} < 0. \tag{A.15}$$

There are three subcases.

Subcase 1.1. $-(2d_B+1-k_I) = -1-k_I$, namely $k_I = d_B$. (A.15) becomes

$$2^{-k_I}(1-2^{-k_B}+2^{-1-2k_B}) - 2^{1-k} + 2^{-2k} < 0. \tag{A.16}$$

So $-k_I \leq 1-k$, namely $k_I \geq k-1$. If $k_I \geq k$, the above inequality is true. If $k_I = k-1$, then (A.16) becomes $2^{-k_B} - 2^{-1-2k_B}) - 2^{-1-k} > 0$, which is true if and only if $k_B \leq k$.

Subcase 1.2. $-(2d_B+1-k_I) > -1-k_I$, namely $k_I > d_B$. (A.15) requires $-(2d_B+1-k_I) \leq 1-k$, namely $k \leq 2d_B+2-k_I$. So $2d_B+2-k_I \geq 2$, *i.e.*, $k_I \leq 2d_B$. When $k < 2d_B+2-k_I$, then (A.15) is true. When $k = 2d_B+2-k_I$, (A.15) becomes

$$-2^{1-k_B} + 2^{-2k_B} + 2^{-2(k_I-d_B)} + 2^{-(2d_B+3-k_I)} < 0. \tag{A.17}$$

(1) If $-2(k_I-d_B) = -(2d_B+3-k_I)$, namely $4d_B = 3(k_I-1)$, then (A.17) is true if and only if $1-k_B > 1-2(k_I-d_B)$, namely $k_B < 2(k_I-d_B)$. (2) If $-2(k_I-d_B) \neq -(2d_B+3-k_I)$, then (A.17) is true if and only if $1-k_B > \max(-2(k_I-d_B), -(2d_B+3-k_I))$, namely $k_B \leq \min(2k_I-2d_B, 2d_B+3-k_I)$.

Subcase 1.3. $-(2d_B+1-k_I) < -1-k_I$, namely $k_I < d_B$. Then (A.15) is true if and only if $-1-k_I < 1-k$, namely $k \leq k_I+1$.

Case 2. $-2(d-d_0) = -k_I$, namely k_I is even, and $d_0 = d-k_I/2 = d-\lfloor k_I/2 \rfloor$. Then $k_I \leq 2d_B$, and (A.12) becomes

$$2^{-(2d_B-k_I)}(1-2^{-k_B})^2 + 2^{-k_I} - 2^{1-k} + 2^{-2k} - 1 < 0. \tag{A.18}$$

When $k_I \leq 2d_B-2$, (A.18) is obviously true.

When $k_I = 2d_B \geq 4$, (A.18) becomes

$$2^{-2d_B} - 2^{1-k_B} + 2^{-2k_B} - 2^{1-k} + 2^{-2k} < 0. \tag{A.19}$$

If $k = k_B$, the inequality is true if and only if $k \leq 2d_B+1$. If $k \neq k_B$, by symmetry, assume $k < k_B$, then (A.19) requires $-2d_B \leq 1-k$, namely $k \leq 2d_B+1$. If $k \leq 2d_B$, (A.19) is true. If $k = 2d_B+1 \leq k_B-1$, (A.19) becomes $-2^{1-k_B} + 2^{-2k_B} + 2^{-2k} < 0$, which is true if and only if $1-k_B > -2k$, namely $k_B \leq 2k$.

Notice that if d_0 does not satisfy Case 1, then neither does it satisfy Case 2. The converse is also true. Also notice that in both Case 1 and Case 2, $k_I \le 2d_B$.

Case 3. $-2(d-d_0) \le -1-k_I$, namely $d_0 \le d-(k_I+1)/2$. Then $d_0 \le d-2$. When $k_I \le 2d_B$, (A.12) is true, and when this happens, $d_0 = d - \lceil (k_I+1)/2 \rceil = d - \lfloor k_I/2 \rfloor - 1$. In other words, if $k_I \le 2d_B$, then either $d_0 = d - \lfloor k_I/2 \rfloor$, or $d_0 = d - \lfloor k_I/2 \rfloor - 1$, and the former is true if and only if d_0 satisfies Case 1 or Case 2.

In the following, we consider the situation where $k_I = 2d_B + 1$ and $k_B \le 2d_B + 2 = k_I + 1$. (A.12) becomes

$$-2^{1-k_B} + 2^{-2k_B} + 2^{k_I-1-2(d-d_0)}(1-2^{-k})^2 + 2^{-1-k_I} < 0. \tag{A.20}$$

So $k_I - 1 - 2(d-d_0) \le 1 - k_B$. There are three subcases.

Subcase 3.1. $k_I - 1 - 2(d-d_0) = 1 - k_B$, namely k_B is odd, and $d_0 = d - (k_I + k_B - 2)/2 = d - \lfloor (k_I + k_B - 1)/2 \rfloor = d - d_B - \lfloor k_B/2 \rfloor$. Then (A.20) becomes

$$-2^{1-k} + 2^{-2k} + 2^{-1-k_B} + 2^{-(2+k_I-k_B)} < 0. \tag{A.21}$$

(1) If $-1-k_B = -(2+k_I-k_B)$, namely $k_I = 2k_B - 1$, (A.21) is true if and only if $k \le k_B$. (2) If $-1-k_B > -(2+k_I-k_B)$, namely $k_I \ge 2k_B$, (A.21) is true if and only if $k \le k_B + 1$. (3) If $-1-k_B < -(2+k_I-k_B)$, namely $k_I \le 2k_B - 2$, (A.21) is true if and only if $k \le 2 + k_I - k_B$.

Subcase 3.2. $k_I - 1 - 2(d-d_0) = -k_B$, namely k_B is even, and $d_0 = d - (k_I + k_B - 1)/2 = d - \lfloor (k_I + k_B - 1)/2 \rfloor = d - d_B - \lfloor k_B/2 \rfloor$. (A.20) becomes

$$-1 - 2^{1-k} + 2^{-2k} + 2^{-k_B} + 2^{-(1+k_I-k_B)} < 0. \tag{A.22}$$

If $1 + k_I - k_B > 0$, namely $k_B \le k_I$, then (A.22) is true. If $1 + k_I - k_B = 0$, namely $k_B = k_I + 1$, (A.22) becomes $-2^{1-k} + 2^{-2k} + 2^{-k_B} < 0$, which is true if and only if $k \le k_B$.

Notice that if d_0 does not satisfy Case 1 and Case 2, then if it does not satisfy Subcase 3.1, nor does it satisfy Subcase 3.2. The converse is also true.

Subcase 3.3. $k_I - 1 - 2(d-d_0) \le -1 - k_B$, namely $d_0 \le d - (k_I + k_B)/2 = d - d_B - (k_B+1)/2$. Then $d_0 \le d - \lfloor (k_I + k_B)/2 \rfloor$. If $k_B \le k_I$, then (A.20) is true. If $k_B = k_I + 1$, (A.20) becomes $-2^{-1-k_I} + 2^{-2-2k_I} + 2^{k_I-1-2(d-d_0)}(1-2^{-k})^2 < 0$, which is true because $k_I - 1 - 2(d-d_0) \le -1 - k_B = -2 - k_I$.

So if d_0 does not satisfy any of Case 1, Case 2, Subcase 3.1, Subcase 3.2, then $d_0 = d - d_B - \lceil (k_B+1)/2 \rceil = d - d_B - \lfloor k_B/2 \rfloor - 1$.

In summary, there are four possible values of d_0: (a) $d - \lfloor k_I/2 \rfloor$, (b) $d - \lfloor k_I/2 \rfloor - 1$, (c) $d - d_B - \lfloor k_B/2 \rfloor$, (d) $d - d_B - \lfloor k_B/2 \rfloor - 1$. The former two values are taken when $k_I \le 2d_B$, where value (a) occurs in Case 1 and Case 2, and value (b) occurs in the beginning of Case 3. The latter two values are taken when $k_I = 2d_B + 1$, where value (c) occurs in Subcase 3.1 and Subcase 3.2, and value (d) occurs in Subcase 3.3.

In particular if $d_0 = d - 1$, then this is possible only when $k_I \le 3$ and $d - \lfloor k_I/2 \rfloor$. In Case 2, this is possible for $k_I = 2$ and all $d_B \ge 2$. In Subcase 1.1, this is possible for $k_I = 3$ only when $d_B = 3$, and either $k \le 3$, or $k = 4 \ge k_B$. In Subcase 1.2, this is possible for $k_I = 3$ only when $d_B = 2$, and either $k = 2$, or $k = 3$ and $k_B = 2$. In Subcase 1.3, this is possible for $k_I = 3$ only when $d_B \ge 4 \ge k$. Q.E.D.

The following lemma is a direct corollary of Lemma 16 on the existence of $d_0 \ge 0$ for large initial error.

Lemma 17 Let d_0 be the biggest integer satisfying (A.7). Then integer $d_0 \ge 0$ exists if and only if (1) either $k_I \le 2d_B$, or $k_I = 2d_B + 1$ and $k_B \le k_I + 1$, (2) one of the following is true:

- if $d_0 = d - \lfloor k_I/2 \rfloor$, then $k_I \le 2d + 1$;
- if $d_0 = d - \lfloor k_I/2 \rfloor - 1$, then $k_I \le 2d - 1$;
- if $k_I = 2d_B + 1$ and $d_0 = d - d_B - \lfloor k_B/2 \rfloor$, then $k_B \le 2(d - d_B) + 1$;
- if $k_I = 2d_B + 1$ and $d_0 = d - d_B - \lfloor k_B/2 \rfloor - 1$, then $k_B \le 2(d - d_B) - 1$.

In particular, if either $k_I \le \min(2d_B, 2d-1)$, or $k_I = 2d_B + 1 \le 2d - k_B$, then $d_0 \ge 0$ exists.

A.2 Fundamental inequalities on tail-up plaintext bootstrapping: the second group

Lemma 18 Let $d'_{i-1} > 0$, and let d_i be the biggest integer satisfying (4.39), namely

$$2^{2d_i} E_{\mathrm{MDS}}^2 (4/N)^2 + 2^{-2d'_{i-1}}((1+2^{-2})E_{\mathrm{init}}^2 - E_B^2)(2/\Delta)^2 < 1. \tag{A.23}$$

Then $d_i = d - 1$, except for the following cases where $d_i = d$ (called greedy mode):

1. $d'_{i-1} > (k-1)/2 - d_I$;
2. $d'_{i-1} = (k-1)/2 - d_I$, and one of the following is true:
   - $k_I = 2$;
   - $k_I = 3, k \geq 4$;
   - $k_I = 3, k = 3, d_I = 0, d_B = 2$.

Proof. In power-of-2 binomial bounds, (A.23) becomes

$$2^{-2(d-d_i)}(1-2^{-k})^2 + (1+2^{-2})\left(2^{-2(d'_{i-1}+d_I)}(1-2^{-k_I})^2 - 2^{-2(d'_{i-1}+d_B)}(1-2^{-k_B})^2\right) < 1. \tag{A.24}$$

So $d_i \leq d$. When $d - d_i = 1$, the left side of (A.24) $< 2^{-2} \times (1 + 5/4) < 1$, so (A.24) is true.

When $d - d_i = 0$, (A.24) becomes

$$-2^{1-k} + 2^{-2k} + (1+2^{-2})\left(2^{-2(d'_{i-1}+d_I)}(1-2^{-k_I})^2 - 2^{-2(d'_{i-1}+d_B)}(1-2^{-k_B})^2\right) < 0. \tag{A.25}$$

On the left side of the above inequality, the last term $> 2^{-2(d'_{i-1}+d_I)} \times (5/4) \times (3/4 - 1/4) > 2^{-2(d'_{i-1}+d_I)-1}$, so $1 - k \geq -2(d'_{i-1} + d_I)$, namely $d'_{i-1} \geq (k-1)/2 - d_I$.

When $d'_{i-1} > (k-1)/2 - d_I$, (A.25) is true. When $d'_{i-1} = (k-1)/2 - d_I > 0$, then $k \geq 2d_I + 3$ is odd, and (A.25) becomes

$$2^{-2} + 2^{-1-k} - 2^{1-k_I} - 2^{-1-k_I} + 2^{-2k_I} + 2^{-2-2k_I} - (1+2^{-2})2^{-2(d_B-d_I)}(1-2^{-k_B})^2 < 0. \tag{A.26}$$

So $-2 \leq 1 - k_I$, namely $k_I \leq 3$.

If $k_I = 2$, then (A.26) is true. If $k_I = 3$, (A.26) becomes

$$2^{-1-k} - 2^{-4} + 2^{-6} + 2^{-8} - (1+2^{-2})2^{-2(d_B-d_I)}(1-2^{-k_B})^2 < 0. \tag{A.27}$$

There are four cases.

Case 1. $k = 4$. (A.27) becomes

$$-2^{-6} + 2^{-8} - (1+2^{-2})2^{-2(d_B-d_I)}(1-2^{-k_B})^2 < 0, \tag{A.28}$$

which is true.

Case 2. $k > 4$. (A.27) is obviously true.

Case 3. $k = 3$. By $d'_{i-1} = (k-1)/2 - d_I > 0$, we get $d_I = 0$, $d_B \geq 2$, and (A.27) becomes

$$2^{-6} + 2^{-8} - (1+2^{-2})2^{-2d_B}(1-2^{-k_B})^2 < 0. \tag{A.29}$$

It requires $d_B \leq 3$. When $d_B = 3$, (A.29) is false. When $d_B = 2$, (A.29) is true.

Case 4. $k = 2$. Then $d_I = 0$, $d_B \geq 2$, and (A.27) becomes

$$2^{-4} + 2^{-6} + 2^{-8} - (1+2^{-2})2^{-2d_B}(1-2^{-k_B})^2 < 0, \tag{A.30}$$

which is always false. Q.E.D.

A.3 Fundamental inequalities on LSB precursor

In this section, we assume $d_B \geq d_I + 1$ only.

Lemma 19 In power-of-2 binomial bounds, (4.89), or equivalently,

$$2^{-2}E^2_{\mathrm{MDS}}(4/N)^2 + E^2_{\mathrm{init}}(2/\Delta)^2 < 1, \tag{A.31}$$

is true if and only if either (1) $d_I > 0$, or (2) $d_I = 0$, $k_I \leq 2d + 2$, or (3) $d_I = 0$, $k_I = 2d + 3$, $k \leq 2d + 4$.

Proof. In power-of-2 binomial bounds, (A.31) becomes

$$2^{-2-2d}(1-2^{-k})^2 + 2^{-2d_I}(1-2^{-k_I})^2 < 1. \tag{A.32}$$

When $d_I > 0$, (A.32) is true. When $d_I = 0$, (A.32) becomes

$$2^{-2-2d}(1-2^{-k})^2 - 2^{1-k_I} + 2^{-2k_I} < 0. \tag{A.33}$$

So $-2 - 2d \leq 1 - k_I$, namely $k_I \leq 2d + 3$.

When $k_I \leq 2d + 2$, (A.33) is true. When $k_I = 2d + 3$, (A.33) becomes $-2^{1-k} + 2^{-2k} + 2^{-4-2d} < 0$. It is true if and only if $1 - k > -4 - 2d$, namely $k \leq 2d + 4$. Q.E.D.

A.4 Fundamental inequalities on head-on plaintext bootstrapping: feasibility, and the first group

$d_B \geq d_I + 2$ is assumed throughout this paper.

Consider the inequality (5.56) for head-on bootstrapping, *i.e.*,

$$\sqrt{((5/4)E_{\text{init}}^2 - E_B^2)(2/\Delta)^2 + 2^{2d_e}E_{\text{MDS}}^2(4/N)^2} < 1. \tag{A.34}$$

Lemma 20 Let d_e be the biggest integer satisfying (A.34). Then integer d_e exists if and only if either $d_I > 0$, or $d_I = 0$ and $k_I \leq 3$. When d_e exists, if $d_I > 0$, then $d_e \in \{d, d-1\}$, and $d_e = d$ (called greedy mode) if and only if one of the following is true:

- $k \leq 2d_I$;
- $k = 2d_I + 1$, $k_I = 2$;
- $k = 2d_I + 1$, $k_I = 3$, $d_I > 1$;
- $k = 2d_I + 1$, $k_I = 3$, $d_I = 1$, $d_B \leq 3$.

If $d_I = 0$, then $k_I \leq 3$, $d_e \leq d - 1$, and $d_e = d - 1$ if and only if $k_I = 2$.

When $d_I = 0$ and $k_I = 3$, then $d_e \in \{d-2, d-3\}$, and $d_e = d - 2$ if and only if one of the following is true:

- $d_B = 2$;
- $d_B = 3$, $k \leq 3$;
- $d_B = 3$, $k = 4$, $k_B \geq 4$;
- $d_B \geq 4$, $k = 2$.

Proof. In power-of-2 binomial bounds, (A.34) becomes

$$2^{-2d_I}(1+2^{-2})(1-2^{-k_I})^2 - 2^{-2d_B}(1-2^{-k_B})^2 + 2^{-2(d-d_e)}(1-2^{-k})^2 < 1. \tag{A.35}$$

Obviously $d_e \leq d$. There are four cases.

Case 1. $d_I > 0$ and $d_e < d$. The left side of (A.35) $< 5/16 + 1/4 < 1$, so the inequality holds.

Case 2. $d_I = 0$ and $d_e = d$. The left side of (A.35) $> 5/4 \times 9/16 + 9/16 > 1$, so the inequality no longer holds.

Case 3. $d_I > 0$ and $d_e = d$. (A.35) becomes

$$2^{-2d_I}(1+2^{-2})(1-2^{-k_I})^2 - 2^{-2d_B}(1-2^{-k_B})^2 - 2^{1-k} + 2^{-2k} < 0, \tag{A.36}$$

so $-2d_I \leq 1 - k$, namely $k \leq 2d_I + 1$. When $k < 2d_I + 1$, the above inequality is true. When $k = 2d_I + 1$, the above inequality becomes

$$1 + 2^{-2d_I} - (2^2+1)(2^{1-k_I} - 2^{-2k_I}) - 2^{-2(d_B-d_I-1)}(1-2^{-k_B})^2 < 0. \tag{A.37}$$

Since $d_B - d_I - 1 \geq 1$, if $k_I \geq 4$, the last two terms on the left side of (A.37) $> -(1/2 + 1/8) - 1/4 > -1$, the inequality is false. So $k_I \leq 3$.

In (A.37), when $k_I = 2$, the inequality becomes

$$2^{-2d_I} - 2^{-2(d_B-d_I-1)}(1-2^{-k_B})^2 < 3/2 - 5/2^4, \tag{A.38}$$

which is obviously true. When $k_I = 3$, the inequality becomes

$$2^{-2d_I} - 2^{-2(d_B-d_I-1)}(1-2^{-k_B})^2 < 2^{-2} - 5/2^6. \tag{A.39}$$

If $d_I > 1$, the above inequality is true. If $d_I = 1$, the above inequality becomes $2^{-2(d_B-4)}(1-2^{-k_B})^2 > 5/4$, which is true if and only if $d_B \leq 3$.

Case 4. $d_I = 0$ and $d_e < d$. Then $d_B \geq 2$, and (A.35) becomes

$$(1+2^{-2})(1-2^{-k_I})^2 - 2^{-2d_B}(1-2^{-k_B})^2 + 2^{-2(d-d_e)}(1-2^{-k})^2 < 1. \tag{A.40}$$

If $k_I \geq 4$, the above inequality is false, because on the left side, the first term $\geq 1 + 2^{-3} - 2^{-5} + 2^{-8} - 2^{-10}$, while the last two terms $> -2^{-4}$.

If $k_I = 2$, inequality (A.40) can be simplified to

$$-2^{-2} - 2^{-4} + 2^{-6} - 2^{-2d_B}(1 - 2^{-k_B})^2 + 2^{-2(d-d_e)}(1 - 2^{-k})^2 < 0, \tag{A.41}$$

which is true because the last term $< 2^{-2}$.

If $k_I = 3$, (A.40) can be simplified to

$$-2^{-4} + 2^{-6} + 2^{-8} - 2^{-2d_B}(1 - 2^{-k_B})^2 + 2^{-2(d-d_e)}(1 - 2^{-k})^2 < 0. \tag{A.42}$$

There are the following subcases:

Subcase 4.1. $d_B = 2$. (A.42) becomes

$$-2 + 2^{-2} + 2^{-4} + 2^{1-k_B} - 2^{-2k_B} + 2^{-2(d-d_e-2)}(1 - 2^{-k})^2 < 0. \tag{A.43}$$

It is true if and only if $d - d_e - 2 \geq 0$.

Subcase 4.2. $d_B = 3$. (A.42) becomes

$$-2^{-4} + 2^{-6} - 2^{-6}(1 - 2^{-k_B})^2 + 2^{-8} + 2^{-2(d-d_e)}(1 - 2^{-k})^2 < 0. \tag{A.44}$$

which is obviously true when $d - d_e \geq 3$, and false when $d - d_e = 1$. When $d - d_e = 2$, (A.44) becomes

$$-2^{1-k} + 2^{-2k} + 2^{-4} + 2^{-1-k_B} - 2^{-2-2k_B} < 0. \tag{A.45}$$

It requires $k \leq 4$. If $k \leq 3$, (A.45) is true. If $k = 4$, (A.45) becomes $-2^{-4} + 2^{-8} + 2^{-1-k_B} - 2^{-2-2k_B} < 0$. It is true if and only if $k_B \geq 4$.

Subcase 4.3. $d_B \geq 4$. (A.42) requires $d - d_e \geq 2$. When $d - d_e > 2$, (A.42) is true. When $d - d_e = 2$, (A.42) becomes

$$2^{-2} + 2^{-4} - 2^{-2(d_B-2)}(1 - 2^{-k_B})^2 - 2^{1-k} + 2^{-2k} < 0. \tag{A.46}$$

Since $d_B - 2 \geq 2$, the above inequality is true if and only if $k = 2$. Q.E.D.

Lemma 20 implies the following result on the existence of integer $d_e \geq 0$:

Lemma 21 In power-of-2 binomial bounds, if $d \geq 3$, then $d_e \geq 0$ exists if and only if either $d_I > 0$, or $d_I = 0$ and $k_I \leq 3$.

Lemma 21 gives the most pessimistic condition on k_I for small initial error in the head-on approach. In practice if v is small, then the condition can be significantly improved. The most optimistic case is $v = 3$, for which the condition on k_I differs only slightly from the prerequisite condition (14) of the tail-up approach. Consider inequality (5.54) for $v = 3$, namely,

$$\sqrt{E_{\text{init}}^2 + 2E_B^2 + E_{\text{MDS}}^2 2^{2(d_e+2)} \Delta^2/(2N)^2} < \Delta/2. \tag{A.47}$$

Lemma 22 Let d_e be the biggest integer satisfying (A.47). When $d_I > 0$, then $d_e \in \{d, d-1\}$. When $d_I = 0$, then under the assumption $k_I \leq \min(2d_B - 1, 2d - 1)$, integer $d_e \geq 0$ exists.

Proof. In power-of-2 binomial bounds, (A.47) becomes

$$2^{-2d_I}(1 - 2^{-k_I})^2 + 2^{1-2d_B}(1 - 2^{-k_B})^2 + 2^{-2(d-d_e)}(1 - 2^{-k})^2 < 1. \tag{A.48}$$

Obviously $d_e \leq d$. When $d_I > 0$ and $d_e < d$, (A.48) holds.

When $d_I = 0$ and $d_e = 0$, then $d_B \geq 2$, and (A.48) becomes

$$-2^{1-k_I} + 2^{-2k_I} + 2^{1-2d_B}(1 - 2^{-k_B})^2 + 2^{-2d}(1 - 2^{-k})^2 < 0. \tag{A.49}$$

If $d_B \leq d$, then $k_I \leq 2d_B - 1$ by the assumption, so (A.49) is true. If $d_B > d$, then $k_I \leq 2d - 1$, and (A.49) is also true. Q.E.D.

Consider the inequalities in (5.13) on the first group, namely

$$\begin{aligned} &\sqrt{E_{\text{MDS}}^2 + 2E_B^2(2N/q)^2} \leq 2^l, \\ &\sqrt{2E_{\text{MDS}}^2 + E_B^2(2N/q)^2} < 2^l + 2^{l-1}. \end{aligned} \tag{A.50}$$

Lemma 23 Let $l_s = l_N - l_t \leq l$. Then (A.50) is true if and only if either $k < 2(d_B + l - l_s)$, or $k = 2(d_B + l - l_s)$ and $k_B \leq 2k + 1$. In particular, if $l_s \leq 0$, then (A.50) is true.

Proof. In power-of-2 binomial bounds,

$$E_B(2N/q) = 2^{-d_B}(1 - 2^{-k_B})(\Delta N/q) = 2^{-d_B}(1 - 2^{-k_B})N/t = 2^{l_s - d_B}(1 - 2^{-k_B}). \tag{A.51}$$

So (A.50) becomes

$$\begin{aligned} (1 - 2^{-k})^2 + 2^{1-2(d_B+l-l_s)}(1 - 2^{-k_B})^2 &\leq 1, \\ 2(1 - 2^{-k})^2 + 2^{-2(d_B+l-l_s)}(1 - 2^{-k_B})^2 &< 2 + 2^{-2}. \end{aligned} \tag{A.52}$$

After simplification, the second inequality becomes

$$2^{-2(d_B+l-l_s)}(1 - 2^{-k_B})^2 < 2^{-2} + 2^{2-k} - 2^{1-2k}. \tag{A.53}$$

Since $d_B + l - l_s \geq 2$, the above inequality is always true.

For the first inequality in (A.52), after simplification, it becomes

$$-2^{1-k} + 2^{-2k} + 2^{1-2(d_B+l-l_s)}(1 - 2^{-k_B})^2 \leq 0. \tag{A.54}$$

So $k \leq 2(d_B + l - l_s)$. When $k < 2(d_B + l - l_s)$, then (A.54) is true. In particular, if $l_s \leq 0$, then $k \leq l < 2(d_B + l - l_s)$, and (A.54) is true. When $k = 2(d_B + l - l_s)$, (A.54) becomes $2^{-1-2k} - 2^{1-k_B} + 2^{-2k_B} \leq 0$, which is true if and only if $k_B \leq 2k + 1$. Q.E.D.

Consider (5.20), namely

$$\begin{aligned} \sqrt{E_{\text{MDS}}^2 + 2E_B^2(2N/q)^2} &\leq N/t, \\ \sqrt{2E_{\text{MDS}}^2 + E_B^2(2N/q)^2} &< N/t. \end{aligned} \tag{A.55}$$

Lemma 24 If $l_s = l_N - l_t > l$, then (A.55) is true.

Proof. In power-of-2 binomial bounds, by $N/t = 2^{l_s}$ and (A.51), (A.55) becomes

$$\begin{aligned} 2^{-2(l_s-l)}(1 - 2^{-k})^2 + 2^{1-2d_B}(1 - 2^{-k_B})^2 &\leq 1, \\ 2^{1-2(l_s-l)}(1 - 2^{-k})^2 + 2^{-2d_B}(1 - 2^{-k_B})^2 &< 1. \end{aligned} \tag{A.56}$$

Since $2d_B \geq 4$, $2(l_s - l) \geq 2$, both inequalities are true. Q.E.D.

A.5 Fundamental inequalities on head-on plaintext bootstrapping: the second group

First consider inequality (5.44), namely

$$\sqrt{E_{\text{MDS}}^2 q^2/(2^{d'+1}N)^2 + E_B^2} \leq q/2^{d'+d_i+2}, \tag{A.57}$$

under the condition $l_t - d' > l_N$, *i.e.*, $d' + l_s < 0$.

Lemma 25 If $d' + l_s < 0$, then $d_i = d + 1$ is the biggest integer satisfying (A.57).

Proof. In power-of-2 binomial bounds, by $\Delta/2 = q/(2t) = q \times 2^{-1-l_t}$, (A.57) becomes

$$2^{-2(d+1-d_i)}(1 - 2^{-k})^2 + 2^{-2(l_t+d_B-d'-1-d_i)}(1 - 2^{-k_B})^2 \leq 1. \tag{A.58}$$

So $d_i \leq \min(d+1, l_t - d' + d_B - 1)$. Since $l_t - d' > l_N$, $d_B \geq 2$, we have $l_t - d' + d_B - 1 \geq l_N + 2 = d + l + 4 > d + 4$. So $d_i \leq d + 1$.

If $d_i \leq d$, then (A.58) is true. If $d_i = d+1$, then (A.58) becomes

$$-2^{1-k} + 2^{-2k} + 2^{-2(l_t+d_B-d'-d-2)}(1-2^{-k_B})^2 \leq 0. \tag{A.59}$$

It is true if $k \leq 2(d_B + l_t - d' - d - 2)$. Since $l_t - d' - d - 2 \geq l_N + 1 - (l_N - l) = l + 1$ and $k \leq l$, the requirement is satisfied. So (A.59) is always true. Q.E.D.

Next consider inequality (5.44), namely

$$2^{l_{si}}q/(2^{d'+1}N) + \sqrt{E_{\text{MDS}}^2 q^2/(2^{d'+1}N)^2 + E_B^2} \leq q/2^{d'+d_i+2}, \tag{A.60}$$

under the condition $d + 3 \leq l_t - d' \leq l_N$, *i.e.*, $d' + l_s \geq 0$ and $d' < l_t - d - 2$. Obviously in (A.60), $2^{l_{si}}q/(2^{d'+1}N) < q/2^{d'+d_i+2}$, namely $d_i \leq l_t - d' - 2$.

Lemma 26 When $l_t - 2 > d' \geq -l_s$, let $d_i \leq l_t - d' - 2$ be the biggest integer satisfying (A.60).

- If $d' + l_s < l$, then $d_i = d+1$ or d, and $d_i = d+1$ if and only if $d' + l_s < l - k$.
- If $d' + l_s \geq l$, then either $d_i = l_t - d' - 2 \in [1, d]$ (called greedy mode), or $d_i = l_t - d' - 3 > 0$. In details, $d_i = l_t - d' - 2$ if and only if either (1) $d' + l_s > l$, or (2) $d' + l_s = l$ and $k \leq 2d_B$, or (3) $d' + l_s = l$, $k = 2d_B + 1$, $k_B \leq k + 1$.

Proof. In power-of-2 binomial bounds, (A.60) becomes

$$2^{-2(d+1-d_i)}(1-2^{-k})^2 + 2^{-2(d_B+l_t-d'-1-d_i)}(1-2^{-k_B})^2 \leq (1-2^{-(l_t-d'-1-d_i)})^2. \tag{A.61}$$

Since $d_i \leq l_t - d' - 2$, on the right side of the above inequality, $l_t - d' - 1 - d_i \geq 1$. Since the right side < 1, $d_i \leq d+1$ on the left side.

First consider the case $d' + l_s < l$, namely $d' < l_t - d - 2$. Notice that $l_t - d' - 1 \geq 2$. When $d_i \leq d$, then $l_t - d' - d_i - 1 \geq 2$, the right side of (A.61) $\geq 9/16$, while the left side $< 2^{-2} + 2^{-8}$, the inequality is true. When $d_i = d+1$, (A.61) becomes

$$-2^{1-k} + 2^{-2k} + 2^{1-(l_t-d'-d-2)} - 2^{-2(l_t-d'-d-2)} + 2^{-2(l_t-d'-d-2+d_B)}(1-2^{-k_B})^2 \leq 0. \tag{A.62}$$

It is true if and only if $k < l_t - d' - d - 2 = l - l_s - d'$.

Next consider the case $l_t - 2 > d' \geq l - l_s = l_t - d - 2$. Then $3 \leq l_t - d' \leq d + 2$. Since $d_i \leq l_t - d' - 2 \leq d$, if $d_i < l_t - d' - 2$, then $l_t - d' - 1 - d_i \geq 2$, so (A.61) is true. If $d_i = l_t - d' - 2$, then (A.61) becomes

$$2^{-2(d+2+d'-l_t)}(1-2^{-k})^2 + 2^{-2d_B}(1-2^{-k_B})^2 \leq 1. \tag{A.63}$$

Since $2d_B \geq 4$, (A.63) requires $d + 2 + d' - l_t \geq 0$.

If $d + 2 + d' - l_t > 0$, namely $d' + l_s > l$, then (A.63) is true. If $d + 2 + d' - l_t = 0$, then (A.63) becomes

$$-2^{1-k} + 2^{-2k} + 2^{-2d_B}(1-2^{-k_B})^2 \leq 0. \tag{A.64}$$

It requires $k \leq 2d_B + 1$, and is true if $k \leq 2d_B$. When $k = 2d_B + 1$, (A.64) becomes $2^{-1-k} - 2^{1-k_B} + 2^{-2k_B} \leq 0$, which is true if and only if $k_B \leq k + 1$. Q.E.D.

A.6 Fundamental inequalities on error bootstrapping feasibility

Consider (6.7), namely

$$\sqrt{E_{\text{init}}^2(2N/\Delta)^2 + E_{\text{MDS}}^2} < N. \tag{A.65}$$

Lemma 27 In power-of-2 binomial bounds, (A.65) is true if and only if either (1) $d_I > 0$, or (2) $d_I = 0$, $k_I < 2d+5$, or (3) $d_I = 0$, $k_I = 2d + 5 \geq k - 1$.

Proof. In power-of-2 binomial bounds, (A.65) becomes

$$2^{-2d_I}(1-2^{-k_I})^2 + 2^{-4-2d}(1-2^{-k})^2 < 1. \tag{A.66}$$

When $d_I > 0$, the inequality is always true. When $d_I = 0$, the inequality becomes

$$-2^{1-k_I} + 2^{-2k_I} + 2^{-4-2d}(1-2^{-k})^2 < 0, \tag{A.67}$$

which requires $1 - k_I \geq -4 - 2d$, namely $k_I \leq 2d + 5$. When $k_I < 2d + 5$, (A.67) is true. When $k_I = 2d + 5$, (A.67) becomes $-2^{1-k} + 2^{-2k} + 2^{-1-k_I} < 0$, which is true if and only if $k \leq k_I + 1$. Q.E.D.

A.7 Fundamental inequalities on error bootstrapping: the first group

From now on, we assume $d_B \geq 2$.

First consider (6.21), namely

$$\sqrt{5(E_{\text{MDS}}^2 + E_B^2(2N/\Delta)^2)} \leq N/2^{d_0+1}. \tag{A.68}$$

Lemma 28 Let d_0 be the biggest integer satisfying (A.68).

1. If $d_B < d+2$, then $d_0 = d_B - 2$ or $d_B - 3$, and $d_0 = d_B - 2$ if and only if one of the following is true:
   - $k_B = 2$, $d > d_B - 1$;
   - $k_B = 2$, $d = d_B - 1$, $k \leq 5$;
   - $k_B = 3$, $d > d_B$.
2. If $d_B > d+2$, then $d_0 = d$ or $d-1$, and $d_0 = d$ if and only if one of the following is true:
   - $k = 2$, $d_B > d+1$;
   - $k = 2$, $d_B = d+1$, $k_B \leq 5$;
   - $k = 3$, $d_B > d+2$.
3. If $d_B = d+2$, then $d_0 = d-1 = d_B - 3$.

In particular, if $d > 1$, then $d_0 \geq 0$ exists if and only either $d_B \geq 3$, or $d_B = 2$ and $k_B \leq 3$.

Proof. In power-of-2 binomial bounds, (A.68) becomes

$$(1+2^{-2})\left\{2^{-2(d-d_0)}(1-2^{-k})^2 + 2^{-2(d_B-d_0-2)}(1-2^{-k_B})^2\right\} \leq 1. \tag{A.69}$$

So $d - d_0 \geq 0$ and $d_B - d_0 - 2 \geq 0$. By $(5/4) \times 2 \times 9/16 = 45/32 > 1$, we get that at most one of $d - d_0, d_B - d_0 - 2$ can be equal to 0. There are three cases.

Case 1. $d - d_0 > d_B - d_0 - 2 \geq 0$, namely $d_0 \leq d_B - 2$ and $d_B \leq d+1$. Then $d_0 \leq d - 1$.

When $d_0 < d_B - 2$, then $d - d_0 > d_B - d_0 - 2 > 0$, and (A.69) is true. When $d_0 = d_B - 2$, (A.69) becomes

$$(1+2^{-2})(1-2^{-k})^2 2^{-2(d-d_B+2)} + 2^{-2} - 2^{1-k_B} + 2^{-2k_B} - 2^{-1-k_B} + 2^{-2-2k_B} \leq 0. \tag{A.70}$$

So $k_B \leq 3$. Also notice that $d - d_B + 2 \geq 1$.

When $k_B = 2$, (A.70) becomes

$$(1+2^{-2})(1-2^{-k})^2 2^{-2(d-d_B+2)} - 2^{-2} - 2^{-4} + 2^{-6} \leq 0. \tag{A.71}$$

If $d - d_B + 2 > 1$, the first term of (A.71) $< (5/4) \times 2^{-4} < 2^{-2}$, the inequality is true. If $d - d_B + 2 = 1$, (A.71) becomes $-2^{1-k} - 2^{-1-k} + 2^{-2k} + 2^{-2-2k} + 2^{-4} \leq 0$, which is true if and only if $k \leq 5$.

When $k_B = 3$, (A.70) becomes

$$(1+2^{-2})(1-2^{-k})^2 2^{-2(d-d_B+2)} - 2^{-4} + 2^{-6} + 2^{-8} \leq 0. \tag{A.72}$$

If $d - d_B + 2 > 2$, then the above inequality is true. If $d - d_B + 2 = 2$, the above inequality becomes $-2^{1-k} - 2^{-1-k} + 2^{-2k} + 2^{-2-2k} + 2^{-1} + 2^{-2} + 2^{-4} \leq 0$, which is obviously false. If $d - d_B + 2 = 1$, the inequality is also false.

Case 2. $d - d_0 = d_B - d_0 - 2 \geq 1$, namely $d = d_B - 2$ and $d_0 \leq d_B - 3 = d - 1$. The left side of (A.69) $< (5/4) \times 2^{-2} \times 2 < 1$, so the inequality is true.

Case 3. $0 \leq d - d_0 < d_B - d_0 - 2$, namely $d_0 \leq d$ and $d \leq d_B - 3$. In (A.69), when d_B and $d+2$ are switched, at the same time k_B and k are switched, the inequality is invariant. By this $\mathbb{Z}_2$ symmetry, all conclusions drawn from Case 1 can be transferred to Case 3 after the switching. Q.E.D.

Next consider inequalities (6.24), (6.25), namely

$$\begin{aligned} 2^{p+1} + \sqrt{E_{\text{MDS}}^2 + 2E_B^2(2N/\Delta)^2)} &\leq N/2^{d_0+1}, \\ \sqrt{2E_{\text{MDS}}^2 + E_B^2(2N/\Delta)^2} &< 2^{p+1} + 2^p, \end{aligned} \tag{A.73}$$

under the assumption $0 \leq p \leq l_N - 3$. In the first inequality, obviously $p + 1 < l_N - d_0 - 1$, namely $d_0 < l_N - p - 2$.

Lemma 29 Let $p \in [0, l_N - 3]$ be the smallest integer such that integer $d_0 \geq 0$ exists in (A.73). Denote $l_0 = l_N - d_B$, and denote $l_p = l_N - 2 - p \in [1, l_N - 2]$. After p is fixed, let d_0 be the biggest integer satisfying the two inequalities. Then

– If $d + 2 < d_B$, *i.e.*, $l > l_0$, then $p = l - 1$, $d_0 = d - 1$ or d. $d_0 = d$ if and only if either $k < 2(d_B - d - 2)$, or $k = 2(d_B - d - 2) \geq k_B - 1$.
– When $d + 2 = d_B$, *i.e.*, $l = l_0$, if either $k = 2$, or $k = 3 = k_B + 1$, then $p = l - 1$, else $p = l$. In both cases, $d_0 = d - 1 = d_B - 3$.
– If $d + 2 > d_B$, *i.e.*, $l < l_0$, then $p = l_0 - 1$, and $d_0 = d_B - 3 < d - 1$.

In particular, $d_0 \geq 0$ if and only if $d_B \geq 3$.

Proof. In power-of-2 binomial bounds, (A.73) becomes

$$\begin{aligned} &2^{-2d-4}(1 - 2^{-k})^2 + 2^{1-2d_B}(1 - 2^{-k_B})^2 \leq (2^{-d_0-1} - 2^{-l_p-1})^2, \\ &2^{-2d-3}(1 - 2^{-k})^2 + 2^{-2d_B}(1 - 2^{-k_B})^2 \ \ < 2^{-2l_p-1}(1 + 2^{-3}). \end{aligned} \tag{A.74}$$

First, we consider the second inequality of (A.74), from which we get the biggest integer l_p. The second inequality can be written as

$$2^{-2(d+1-l_p)}(1 - 2^{-k})^2 + 2^{1-2(d_B - l_p)}(1 - 2^{-k_B})^2 < 1 + 2^{-3}. \tag{A.75}$$

So $-2(d + 1 - l_p) \leq 0$ and $1 - 2(d_B - l_p) \leq 0$, namely, $l_p \leq \min(d + 1, d_B - 1)$, or equivalently, $p \geq \max(l - 1, l_0 - 1)$. When $l_p < \min(d + 1, d_B - 1)$, (A.75) is true. When $l_p = \min(d + 1, d_B - 1)$, there are three cases.

Case 1. $d + 2 = d_B$, namely $l = l_0$. By $l_p = d + 1 = d_B - 1$, (A.75) becomes

$$(1 - 2^{-k_B})^2 < 2^{-2} + 2^{2-k} - 2^{1-2k}. \tag{A.76}$$

It requires $k \leq 3$, as the left side $\geq 9/16$. When $k = 2$, the inequality is true. When $k = 3$, the inequality becomes $2^{1-k_B} - 2^{-2k_B} > 2^{-2} + 2^{-5}$, which is true if and only if $k_B = 2$.

Case 2. $d + 2 < d_B$, namely $l > l_0$. By $l_p = d + 1$, (A.75) becomes $2^{-2(d_B - d - 2) - 1}(1 - 2^{-k_B})^2 < 2^{-3} + 2^{1-k} - 2^{-2k}$, which is always true because the left side $< 2^{-3}$.

Case 3. $d + 2 > d_B$, namely $l < l_0$. By $l_p = d_B - 1$, (A.75) becomes $2^{-2(d+2-d_B)}(1 - 2^{-k})^2 + 2^{-1}(1 - 2^{-k_B})^2 < 1 + 2^{-3}$, which is always true because the left side $< 3/4$.

In summary, when $d + 2 \neq d_B$, then $p = \max(l - 1, l_0 - 1)$. When $d + 2 = d_B$, if either $k = 2$, or $k = 3 = k_B + 1$, then $p = l - 1$, else $p = l$. That $0 \leq p \leq l_N - 3$ requires $l_0 \leq l_N - 2$, namely $d_B \geq 2$. That $l - 1 \leq l_N - 3$ is just $d \geq 0$, which is always true.

Next, we find the biggest integer $d_0 < l_N - p - 2 = l_p$ from the first inequality of (A.74) after the value of p is fixed. There are four cases according to the values of p.

Case 4. $d + 2 = d_B$, $p = l - 1$, $d_0 < l_p = d_B - 1 = d + 1$. The first inequality of (A.74) becomes

$$(1 - 2^{-k})^2 + 2(1 - 2^{-k_B})^2 \leq (2^{d_B - 1 - d_0} - 1)^2. \tag{A.77}$$

When $d_0 \leq d_B - 3 = d - 1$, (A.77) is true. When $d_0 = d_B - 2$, (A.77) is false.

Case 5. $d + 2 = d_B$, $p = l$, $d_0 < l_p = d_B - 2 = d$. The first inequality of (A.74) becomes

$$2^{-2}(1 - 2^{-k})^2 + 2^{-1}(1 - 2^{-k_B})^2 \leq (2^{d_B - d_0 - 2} - 1)^2, \tag{A.78}$$

which is always true for $d_0 = d_B - 3 = d - 1$, because the left side $< 3/4$.

Case 6. $d + 2 < d_B$, $p = l - 1$, $d_0 < l_p = d + 1$. The first inequality of (A.74) becomes

$$(1 - 2^{-k})^2 + 2^{-1-2(d_B - d - 3)}(1 - 2^{-k_B})^2 \leq (2^{d+1-d_0} - 1)^2. \tag{A.79}$$

When $d_0 \leq d - 1$, (A.79) is true. When $d_0 = d$, (A.79) becomes

$$2^{-1-2(d_B - d - 3)}(1 - 2^{-k_B})^2 \leq 2^{1-k} - 2^{-2k}. \tag{A.80}$$

It requires $-1 - 2(d_B - d - 3) \leq 1 - k$, namely $k \leq 2(d_B - d - 2)$, and is true if $k < 2(d_B - d - 2)$. If $k = 2(d_B - d - 2)$, (A.80) becomes $2^{1-k_B} - 2^{-2k_B} \geq 2^{-1-k}$, which is true if and only if $k_B \leq k + 1$.

Case 7. $d + 2 > d_B$, $p = l_0 - 1$, $d_0 < l_p = d_B - 1$. The first inequality of (A.74) becomes

$$2^{-2(d+2-d_B)}(1 - 2^{-k})^2 + 2(1 - 2^{-k_B})^2 \leq (2^{d_B - d_0 - 1} - 1)^2. \tag{A.81}$$

When $d_0 \leq d_B - 3$, (A.81) is true. When $d_0 = d_B - 2$, (A.81) is false. Q.E.D.

A.8 Fundamental inequalities on error bootstrapping: the second group

Let $d_i = l_N - l_i - 2$. Then (6.33) can be written as

$$\sqrt{E_B^2 + E_{\mathrm{MDS}}^2 \Delta^2/(2N \times 2^{d'})^2} \leq \Delta/2^{d'+d_i+2}. \tag{A.82}$$

Lemma 30 Let $d' \geq 0$, and let d_i be the biggest integer satisfying (A.82).

1. If $d + d' < d_B - 2$, then $d_i = d+1$ or d, and $d_i = d+1$ if and only if either $d + d' < d_B - (k+3)/2$, or $d + d' = d_B - (k+3)/2$ and $k_B \leq k+1$.
2. If $d + d' = d_B - 2$, then $d_i = d = d_B - d' - 2$.
3. If $d+d' > d_B - 2$, then $d_i = d_B - d' - 1$ or $d_B - d' - 2$, and $d_i = d_B - d' - 1$ if and only if either $d + d' > d_B + (k_B - 5)/2$, or $d + d' = d_B + (k_B - 5)/2$ and $k \leq k_B + 1$.
4. $d_i > 0$ if and only if one of the following is true:
   - $d + d' \leq d_B - 2$;
   - $d_B + d - 1 > d + d' > d_B + (k_B - 5)/2$;
   - $d + d' = d_B + (k_B - 5)/2 < d_B + d - 1$, $k \leq k_B + 1$;
   - $d_B - 2 < d + d' < d_B - 2 + \min(d, (k_B - 1)/2)$;
   - $d + d' = d_B + (k_B - 5)/2 < d_B + d - 2$, $k > k_B + 1$.

Proof. In power-of-2 binomial bounds, (A.82) becomes

$$2^{-2(d_B - d' - d_i - 1)}(1 - 2^{-k_B})^2 + 2^{-2(d - d_i + 1)}(1 - 2^{-k})^2 \leq 1. \tag{A.83}$$

Then $d_B - d' - d_i - 1 \geq 0$ and $d - d_i + 1 \geq 0$, namely $d_i \leq \min(d+1, d_B - d' - 1)$. If $d_i < \min(d+1, d_B - d' - 1)$, then (A.83) is true. If $d_i = \min(d+1, d_B - d' - 1)$, there are three cases.

Case 1. $d + 1 < d_B - d' - 1$. Then $d' < d_B - d - 2$, and $d_i = d + 1 \leq d_B - d' - 2$. (A.83) becomes

$$2^{-2(d_B - d' - d - 2)}(1 - 2^{-k_B})^2 - 2^{1-k} + 2^{-2k} \leq 0. \tag{A.84}$$

So $-2(d_B - d' - d - 2) \leq 1 - k$, namely $k \leq 2(d_B - d' - d - 2) + 1$. If $k < 2(d_B - d' - d - 2) + 1$, then (A.84) is true. If $k = 2(d_B - d' - d - 2) + 1$, (A.84) becomes $-2^{1-k_B} + 2^{-2k_B} + 2^{-1-k} \leq 0$, which holds if and only if $k_B \leq k + 1$.

Case 2. $d + 1 = d_B - d' - 1$. Then $d' = d_B - d - 2$, and $d_i = d + 1 = d_B - d' - 1$. (A.83) is obviously false.

Case 3. $d + 1 > d_B - d' - 1$. Then $d' > d_B - d - 2$, and$d_i = d_B - d' - 1 \leq d$. (A.83) becomes

$$-2^{1-k_B} + 2^{-2k_B} + 2^{-2(d - d_i + 1)}(1 - 2^{-k})^2 \leq 0. \tag{A.85}$$

So $k_B \leq 2(d - d_i + 1) + 1 = 2(d - d_B + d') + 5$. If $k_B < 2(d - d_B + d') + 5$, then (A.85) is true. If $k_B = 2(d - d_B + d') + 5$, (A.85) becomes $2^{-1-k_B} - 2^{1-k} + 2^{-2k} \leq 0$, which holds if and only if $k \leq k_B + 1$.

Finally, in Case 1 and Case 2, obviously $d_i > 0$. In Case 3, if $d_i = d_B - d' - 1$, then $d + d' < d_B + d - 1$ is sufficient for $d_i > 0$. If $d_i = d_B - d' - 2$, then either $d_B - 2 < d + d' < d_B - 2 + (k_B - 1)/2$, or $d + d' = d_B + (k_B - 5)/2$ and $k > k_B + 1$; in both cases, $d + d' < d_B + d - 2$ is sufficient for $d_i > 0$. Q.E.D.