Ajax: Fast Threshold Fully Homomorphic Encryption without Noise Flooding

Zhenkai Hu [*, †] Haofei Liang[*] Xiao Wang[‡] Xiang Xie [§, ¶, ✉] Kang Yang[†, ✉]
Yu Yu[*, ‖] Wenhao Zhang[‡]

Abstract

Threshold fully homomorphic encryption (ThFHE) enables multiple parties to perform arbitrary computation over encrypted data, while the secret key is distributed across the parties. The main task of designing ThFHE is to construct threshold key-generation and decryption protocols for FHE schemes. Among existing FHE schemes, FHEW-like cryptosystems enjoy the advantage of fast bootstrapping and small parameters. However, known ThFHE solutions use the "noise-flooding" technique to realize threshold decryption, which requires either large parameters or switching to a scheme with large parameters via bootstrapping, leading to a slow decryption process. Besides, for key generation, existing ThFHE schemes either assume a generic MPC or a trusted setup, or incur noise growth that is linear in the number n of parties.

In this paper, we propose a fast ThFHE scheme Ajax, by designing threshold key-generation and decryption protocols for FHEW-like cryptosystems. In particular, for threshold decryption, we eliminate the need for noise flooding, and instead present a new technique called "mask-then-open" based on random double sharings over different rings, while keeping the advantage of small parameters. For threshold key generation, we show a simple approach to reduce the noise growth from n times to $max(0.038n, 2)$ times in the honest-majority setting, where at most $t = \lfloor (n-1)/2 \rfloor$ parties are corrupted. Our end-to-end implementation reports the running time 17.6 *s* and 0.9 *ms* (resp., 91.9 *s* and 4.4 *ms*) of generating a set of keys and decrypting a single ciphertext respectively, for $n = 3$ (resp., $n = 21$) parties under the network of 1 Gbps bandwidth and 1 *ms* ping time. Compared to the state-of-the-art implementation, our protocol improves the end-to-end performance of the threshold decryption protocol by a factor of at least $5.7\times \sim 283.6\times$ across different network latencies from $t = 1$ to $t = 13$. Our approaches can also be applied in other types of FHE schemes like BGV, BFV, and CKKS.

1 Introduction

Fully homomorphic encryption (FHE) [40, 64] enables one to perform arbitrary computation over encrypted data, and has found a variety of applications (e.g., delegation of heavy computation). For now, there are three kinds of concretely efficient FHE schemes: BGV/BFV [14, 15, 38], CKKS [21] and GSW/FHEW/TFHE [25, 36, 41]. Among them, FHEW-like cryptosystems (including TFHE-like schemes), e.g., [23–26, 36, 49–51, 53, 68], provide the advantage of fast boostrapping and small parameters.

Threshold fully homomorphic encryption (ThFHE) [3, 9, 40] extends FHE from a single party to multiple parties. Specifically, ThFHE allows n parties to secretly share a secret key sk and jointly generate the corresponding public key pk and evaluation key evk; then any set of at least $t+1$ parties can distributedly decrypt a given ciphertext correctly, where t is the corruption threshold (i.e., the maximum number of corrupted parties). ThFHE is a key cryptographic tool in numerous applications such as the delegation of computation over the data of multiple users [59], secure multi-party computation (MPC) (e.g., [3, 27, 39, 40, 44, 58, 65]) and universal thresholdizer [9, 20, 37]. Recently, NIST proposed the first call for designing and implementing multi-party threshold schemes, including ThFHE [16].

As for key generation, most of ThFHE schemes, e.g., [9, 10, 19, 20, 28, 37, 57, 59, 67], assume a generic MPC or a trusted setup to generate a set of FHE keys. In particular, it is prohibitively expensive to use a generic MPC to sample noises from the discrete Gaussian distribution [70, 73]. To achieve better efficiency, the main approach used in previous threshold key-generation protocols [2, 3, 50, 54, 55, 58, 65] lets every party individually generate samples of LWE (learning with errors) or RLWE (ring learning with errors), and then combines them to obtain the public key and evaluation key

[*] Shanghai Jiao Tong University. {zhenkaihu,lianghaofei,yyuu}@sjtu.edu.cn
[†] State Key Laboratory of Cryptology. yangk@sklc.org.
[‡] Northwestern University. wangxiao1254@gmail.com,wenhao.zhang@northwestern.edu
[§] East China Normal University.
[¶] Primus Labs. xiexiangiscas@gmail.com
[‖] Shanghai Qi Zhi Institute.

based on linear secret sharings. However, this approach incurs the noise growth linear in the number of parties n, which requires large parameters for a large n.

In terms of threshold decryption, existing ThFHE schemes, e.g., [2,3,9,10,18–20,37,54,55,57,59], first compute partial decryption shares of a ciphertext with the shares of a secret key, when using the "noise flooding" (a.k.a., "noise smudging") approach to guarantee security of the secret key, and then combine all partial decryption shares to recover the plaintext. Among them, the schemes [2,3,9,18–20,37,54,55] analyze the security of partial decryption shares with noise flooding based on statistical distance, which requires exponentially large flooding noises and thus an exponential ciphertext modulus for the decryption procedure. To reduce the magnitude of flooding noises, the recent work [10] performs the security analysis based on the Rényi divergence [6], which supports a polynomial modulus but only guarantees weak game-based security [59]. Later, two works [57,59] continue to adopt the analysis based on the Rényi divergence, but developed new ThFHE schemes with simulation-based security. Although the two works allow one to use a polynomial modulus for the decryption procedure, they require a super-polynomial modulus for the encryption process, and thus still need large parameters for the ThFHE scheme. [1]

To keep the advantage of small parameters in FHEW-like cryptosystems, prior work [30] proposed a different way to use noise flooding based on statistical distance. Specifically, before performing partial decryption, the parties bootstrap a ciphertext under small parameters to that under sufficiently large parameters (particularly, an exponential modulus). In this way, it is nice that the underlying FHEW-like cryptosystem can continue to adopt small parameters, but the additional bootstrapping operation under large parameters is very expensive. Very recently, a new approach called "noise-simulatable sanitization" for threshold decryption was proposed [67], and supports the usage of small parameters. However, this approach requires the assistance of an honest server to perform ciphertext sanitization, where the server cannot collude with the parties who perform threshold decryption; moreover, the sanitization of ciphertexts requires the randomized bootstrapping technique [11] that is computationally expensive. Both "noise-simulatable sanitization" and "transciphering" approach [5,56] require bootstrapping-like operations; the former additionally relies on a trusted server, while the latter requires modifications to the underlying FHE scheme. In addition, generic MPC can be directly applied to realize threshold decryption with small parameters, but it cannot achieve practical efficiency due to high rounds or large communication.

FHEW-like cryptosystems are a class of popular FHE schemes, and have been implemented and included in a lot of libraries such as [2,43,49,72]. A central challenge is to design threshold extensions of these schemes without modifying the underlying FHE algorithms, while retaining the efficiency benefits of small parameters. However, supporting such small parameters remains challenging, as most prior works either rely on large parameters or introduce significant overhead. This limitation motivates us to propose a new framework to enable distributed decryption with small parameters.

[1] The modulus can be reduced to polynomial by transciphering [5,56]. However, this approach not only has to modify the underlying FHE scheme but also requires homomorphic evaluation of a decryption circuit on a symmetric-key primitive.

1.1 Our Contributions

In this paper, we propose a new public-key ThFHE scheme named as Ajax by designing threshold key-generation and decryption protocols towards existing FHEW-like cryptosystems. Our protocols are proven secure in the standard simulation-based paradigm and can be securely integrated into bigger protocols. Our approach is general, and can also be applied to obtain the threshold extension of FHE schemes such as BGV, BFV and CKKS. In the key generation phase, we assume an honest-majority and semi-honest setting (t out of n parties), whereas in the decryption phase, we consider a dishonest-majority and setting (t out of $t+1$ parties). This corruption model is enjoyed by real-world applications, as users can decrypt ciphertexts even if some servers crash or some shares are missing. It has been used in threshold FHE [18] and a series of threshold ECDSA protocols such as [17,29,33,34,52], and is also deployed in practice. The protocols in the main body are focused on semi-honest. In the full version [47], we give an overview of how to extend the key generation and decryption protocols to active security. Specifically, we have the following results.

Threshold key generation with smaller noise growth. We propose a simple "noise-compressing" technique in the honest-majority setting that reduces the noise growth in the public key and evaluation key from a factor of n to $\max\left(\frac{\alpha}{\sigma}\cdot n, \frac{n}{n-t}\right)$. In the case where $n = 2t+1$, we have $n/(n-t) \approx 2$, and under our parameters this expression evaluates to $\max(0.038n, 2)$. Here, α denotes the minimal variance achievable for discrete Gaussian noise sampling in practical implementations, σ denotes the noise variance required for the ciphertext to achieve the targeted security level, n is the number of parties, and we have $\alpha/\sigma < 1$. We use this technique to design a new threshold key generation protocol, which can scale to a large number of parties.

Threshold decryption without noise flooding. We address the key technical challenge for threshold decryption: supporting small parameters in FHEW-like cryptosystems without sacrificing efficiency. Our technique avoids noise flooding, supports small parameters, and achieves practical efficiency. Together with additive secret sharing, we design a new threshold decryption protocol, such that any $t+1$ parties in n parties can distributedly decrypt ciphertexts. Our protocol requires a constant number of rounds overall, with two rounds in the on-

line phase. Our technique incurs more communication rounds but supports smaller parameters and achieves lower computational and communication costs compared to the noise flooding approach.

End-to-end implementation. We implemented the threshold key generation and decryption protocols of our ThFHE scheme[2] and evaluated their performance under different network bandwidths and latencies. For the cases of $n = 3$ and $n = 21$, our end-to-end implementation reports that threshold key generation protocol takes 17.6 *s* and 91.9 *s* (resp., 218.6 *s* and 908.0 *s*) in the LAN (resp., WAN) setting. In both cases (i.e., 2 and 11 parties running the threshold decryption protocol), the amortized [3] running time of decrypting a single ciphertext is 0.9 *ms* and 4.4 *ms* (resp., 5.4 *ms* and 42.4 *ms*) in the LAN (resp., WAN) setting. Compared to the state-of-the-art ThFHE implementation [30], our threshold decryption protocol improves the end-to-end performance by a factor of at least $5.7\times \sim 283.6\times$ across different network settings, when the corruption threshold is increased from $t = 1$ to $t = 13$. Note that the security guarantees of our protocol differ from those of [30]. However, since it represents the best available ThFHE implementation closely related to our work, we use it as a point of comparison.

2 Technical Overview

In this section, we give an overview of our techniques.

2.1 Threshold Key Generation

The main task of threshold key-generation protocols is to distributedly generate LWE or RLWE samples. The distributed generation of RLWE samples is totally similar to that of LWE samples, and thus, in the following, we take the distributed generation of LWE samples as an example. Without loss of generality, we assume that the LWE modulus q is prime. Although the protocols described in this paper are presented under the assumption that q is prime, our schemes also support cases where q is a power of two, with some modifications required in certain details, introduced in the full version [47].

Prior solution. Existing solutions [2, 3, 50, 54, 55, 58, 65] let each party P_i independently sample a share $\vec{s}_i$ from a secret-key distribution $\mathcal{D}_{\mathsf{sk}}^{\ell}$ and a noise e_i from a discrete Gaussian distribution χ_σ with a variance σ, and then make P_i send $b_i = \langle \vec{a}, \vec{s}_i \rangle + e_i \in \mathbb{Z}_q$ to all other parties where $\vec{a} \in \mathbb{Z}_q^{\ell}$ can be sampled uniformly by running a standard coin-tossing protocol with public output. Then, all parties can locally combine the $\{b_i\}$ to obtain an LWE sample, i.e., $b = \sum_{i\in[1,n]} b_i$ such that $b = \langle \vec{a}, \vec{s} \rangle + e$ where $\vec{s} = \sum_{i\in[1,n]} \vec{s}_i$ and $e = \sum_{i\in[1,n]} e_i$. The security is natural as each b_i is an independent LWE sample. However, the resulting LWE sample b involves a noise e that is blown up by n times, i.e., the variance of e is equal to $n \cdot \sigma$.

Our solution. We observe that it is unnecessary to let each party generate an independent LWE sample when the number of honest parties $n - t > 1$, and instead it is sufficient to guarantee that the aggregated value b is a secure LWE sample. Specifically, we let every party P_i sample e_i from a discrete Gaussian distribution $\chi_{\sigma'}$ with a variance $\sigma' = \sigma/(n-t)$. Let $\mathcal{H}$ and $\mathcal{C}$ denote the sets of honest parties and corrupted parties, respectively. When $t \approx n/2$, we have $n \cdot \sigma' \approx 2\sigma$, and thus the resulting noise $e = \sum_{i=1}^{n} e_i$ is reduced to $2\times$ compared to previous work.

Note that the resulting noise $e = \sum_{i\in\mathcal{H}} e_i + \sum_{i\in\mathcal{C}} e_i$ where $\sum_{i\in\mathcal{C}} e_i$ is learned by the adversary, and $\sum_{i\in\mathcal{H}} e_i$ is kept secret from the adversary. Since the sum of $n-t$ discrete Gaussian distributions with variance σ' is a discrete Gaussian distribution with variance $(n-t)\cdot\sigma' = \sigma$, $\sum_{i\in\mathcal{H}} e_i$ complies with the discrete Gaussian distribution χ_σ. In other words, $\sum_{i\in\mathcal{H}} e_i$ has the identical distribution as a normal noise sampled from the distribution χ_σ.

It is important to emphasize that in practice, it is impossible to perform discrete Gaussian sampling with an arbitrarily small variance σ'. We will defer the discussion of how we choose σ' and the constraints it imposes to Section 4.1.

Because the variance σ' of e_i is too small, we let P_i distribute e_i using Shamir secret sharing to obtain shares $[\![e_i]\!]$. Then all parties can locally compute $[\![e]\!] = \sum_{i\in[1,n]} [\![e_i]\!]$.

We do not let every party P_i sample a secret-key share $\vec{s}_i$ from $\mathcal{D}_{\mathsf{sk}}^{\ell}$, and choose to make all parties jointly produce a vector of degree-t Shamir sharings $[\![\vec{s}]\!]$. Then all parties can locally compute a degree-t Shamir sharing $[\![b]\!] = \langle \vec{a}, [\![\vec{s}]\!] \rangle + [\![e]\!]$, and open it to obtain b. The detailed protocols are given in Protocol Π_{LWE} (Figure 7) and Protocol Π_{RLWE} (in the full version [47]).

The remaining task is to generate $[\![\vec{s}]\!]$. If $\mathcal{D}_{\mathsf{sk}}$ is a uniform distribution over $\mathbb{Z}_q^{\ell}$, it is straightforward to generate a degree-t Shamir sharing $[\![\vec{s}]\!]$ by running a standard random-sharing generation protocol. Nevertheless, in FHE schemes, $\mathcal{D}_{\mathsf{sk}}$ is often a distribution defined over either $\{0,1\}$ or $\{-1,0,1\}$. For the case of $\{0,1\}$, all parties can run a known protocol [31] to generate random sharings of bits, which requires one multiplication of secret sharings for generating each random bit sharing. For the case of $\{-1,0,1\}$, all parties can produce two vectors of degree-t Shamir sharings $[\![\vec{s}_1]\!]$ and $[\![\vec{s}_2]\!]$ with $\vec{s}_1, \vec{s}_2 \in \{0,1\}^{\ell}$, and then locally compute $[\![\vec{s}]\!] = [\![\vec{s}_1]\!] - [\![\vec{s}_2]\!]$. It is easy to see that each component of $\vec{s}$ is equal to $-1, 1$ with probability $1/4$ and 0 with probability $1/2$; thus $\vec{s}$ complies with $\mathcal{D}_{\mathsf{sk}}$ defined over $\{-1,0,1\}$.

2.2 Threshold Decryption

A threshold decryption protocol enables any set of $t+1$ parties to distributedly decrypt one LWE or RLWE ciphertext.

[2] https://github.com/primus-labs/Ajax/

[3] By *amortized setting*, we mean that the reported results are averaged over a batch of 20,000 ciphertexts in distributed decryption, i.e., the average time per ciphertext is measured.

As such, we take the distributed decryption of an LWE ciphertext as an example, as the distributed decryption of an RLWE ciphertext is similar. Before executing the protocol, any $t+1$ parties can locally convert a vector of degree-t Shamir sharings $[\![\vec{s}]\!]$ into a vector of additive secret sharings $\langle\vec{s}\rangle$ using the standard approach. We w.l.o.g. assume that an LWE ciphertext $(\vec{a},b)$ with $b=\langle\vec{a},\vec{s}\rangle+e+\Delta\cdot m$ is defined over a power-of-two modulus q, where $m\in\mathbb{Z}_p$ for some power-of-two modulus p such that $p\,|\,q$ and $\Delta=q/p$. Otherwise, we can perform the modulus switching to satisfy the conditions.

Prior solutions based on noise flooding. A series of ThFHE schemes such as [2,3,9,10,18–20,37,54,55,57,59] let all parties first compute partial decryption shares with noise flooding and then combine them into recover a plaintext $m\in\mathbb{Z}_p$. Specifically, the basic idea behind the noise-flooding approach is to let each party P_i generate a flooding noise E_i and then locally compute $\langle\mathrm{v}\rangle=b-\langle\vec{a},\langle\vec{s}\rangle\rangle+E_i$ [4]. Since $\sum_{i=1}^{n}E_i$ is sufficiently large to hide the original noise e in the ciphertext, the parties can open $\langle\mathrm{v}\rangle$ to a specified receiver P_R without leaking the information of the secret key $\vec{s}$. While this approach has the advantage of being rather simple, it has the disadvantage of requiring the ratio between the fooding noise E and the size of the ciphertext noise e to be superpolynomial in the security parameter. This, in turn, requires the LWE problem to be secure with a superpolynomial modulus-to-noise ratio, which weakens security and requires larger LWE parameters to compensate. The noise-flooding approach typically requires a single round of communication in the online phase of decryption. Noah's Ark [30] proposes an efficient and robust[5] asynchronous threshold decryption protocol that is secure against malicious adversaries under a corruption threshold of $t=(n-1)/3$. It adapts the noise-flooding approach by performing bootstrapping (called Switch-n-Squash in [30]) in the online decryption phase: homomorphic evaluation uses a smaller modulus, which is then lifted to a larger modulus at decryption time, where bootstrapping simultaneously reduces the ciphertext noise. This preserves the efficiency of the FHE computation, supports a wider range of parameter choices, and does not increase the number of communication rounds in the online phase.

Our solution. We present a "mask-then-open" approach to remove the noise term inside the additive secret sharing, and eliminate the need for noise flooding altogether. Recall that the decryption process would compute

$$\left\lceil\frac{p}{q}(b-\langle\vec{a},\vec{s}\rangle)\right\rfloor=\left\lceil\frac{p}{q}(\Delta\cdot m+e)\right\rfloor.$$

We let the parties first perform the partial decryption by locally computing an additive secret sharing $\langle\Delta\cdot m+e\rangle=b-\langle\vec{a},\langle\vec{s}\rangle\rangle$. Since q is a multiple of Δ, we have

$$(\Delta\cdot m+e \mod q) \mod\Delta=e \pmod\Delta.$$

Therefore, we let the parties locally compute $\langle e\rangle^{\Delta}=\langle\Delta\cdot m+e\rangle \mod\Delta$, i.e., every party takes its share modulo Δ. Henceforth, we use $\langle x\rangle^{m}$ to denote an additive secret sharing over a ring $\mathbb{Z}_m$.

Then, we let the parties run an efficient protocol to jointly generate a random double sharing $(\langle r\rangle^{\Delta},\langle r\rangle^{q})$ with $r\in\mathbb{Z}_{\Delta}$, where two sharings are defined over two different rings $\mathbb{Z}_{\Delta}$ and $\mathbb{Z}_q$. Now, we can mask $\langle e\rangle^{\Delta}$ with $\langle r\rangle^{\Delta}$, i.e., the parties locally compute $\langle e+r\rangle^{\Delta}=\langle e\rangle^{\Delta}+\langle r\rangle^{\Delta}$, and then open $\langle e+r\rangle^{\Delta}$ to let all parties obtain $e+r \mod\Delta$. Next, we convert $\langle e\rangle$ back to $\mathbb{Z}_q$ by letting the parties locally compute $\langle e\rangle^{q}=e+r-\langle r\rangle^{q}$. Finally, the parties can locally compute $\langle\Delta\cdot m\rangle^{q}=\langle\Delta\cdot m+e\rangle^{q}-\langle e\rangle^{q}$, and open it to the specified receiver P_R who can recover m from $\Delta\cdot m$.

Note that $e+r \pmod\Delta$ may not be equal to the integer $e+r$. To ensure the decryption correctness, we need to carefully analyze the probability that $e+r \mod\Delta=e+r$. Since r is uniform in $[-\Delta/2,\Delta/2)$, we have $e+r\in[-\Delta/2+e,\Delta/2+e)$. Then the probability that $e+r\notin[-\Delta/2,\Delta/2)$ is $|e|/\Delta$. When $|e|/\Delta$ is sufficiently small, the decryption-failure probability becomes negligible.

Comparison and limitation. The advantages of basic noise flooding are its simplicity and the fact that it requires only a single communication round. Because most existing basic noise flooding schemes are either purely theoretical (e.g., [9]) or use different FHE schemes such as BGV/BFV [2,3], a precise communication comparison is difficult. As a reference point, we hypothetically instantiate a noise flooding approach with the same TFHE scheme as in our work [8] and a 30-bit statistical security parameter. In this setting, both approaches incur 128 bits of online communication per party, but noise flooding uses a single round with lower computational cost, whereas our protocol uses two rounds and has higher online computation. Moreover, basic noise flooding schemes typically have no offline phase, while Ajax requires an offline phase to generate random masks; in return, Ajax supports a broader range of FHE parameters, allowing for smaller moduli, making the FHE evaluation phase significantly more efficient than that in basic noise-flooding-based schemes (with 80–128-bit moduli and RLWE modulus degree 2048).

To the best of our knowledge, [30] is the only work that implemented a threshold scheme for FHEW/TFHE-like cryptosystems; we therefore compare against it despite incomparable security models. [30] provides robustness and malicious security with an online phase consisting of a single communication round, whereas our scheme targets an all-but-one corrupt, semi-honest setting but avoids bootstrapping and is computationally lighter. In the online phase, [30] requires

[4] E_i can be viewed as an additive secret sharing of $E=\sum E_i$. In some works, rather than having each party independently sample its own noise, all parties jointly generate a noise E, and each party receives a secret share $[\![E]\!]$. The underlying idea in both approaches is to use a large noise to mask the smaller noise.

[5] It guarantees correct decryption even in the presence of malicious parties.

$384t$ bits of communication, while our implementation uses $128t$ bits (theoretically $100t$ bits), where t is the threshold. We cannot precisely compare the offline cost since [30] omits these details. In our protocol, the offline communication is $O(n^2k^2)$ or $O(nk)$ bits (n is the number of parties and k is the ring dimension), depending on the implementation of Beaver-triple generation, while [30] reports an offline cost of $O(nk)$ bits. In summary, compared with [30], Ajax reduces online computation and bandwidth in the decryption phase and improves end-to-end performance by $5.7\times$–$283.6\times$ for $t = 1$ to $t = 13$ across different network latencies, at the cost of one extra online round, larger offline communication, and a slightly weaker statistical security level (about 30 bits vs. roughly 40 bits).

Despite supporting larger parameter ranges and imposing weaker constraints on the modulus than both the traditional noise-flooding approach and its refinement in [30], our protocol still shares several limitations of noise-flooding-based designs. In particular, the noise level after homomorphic evaluation continues to influence the decryption procedure, including the choice of the decryption modulus and the resulting decryption correctness probability. Although our scheme uses MPC to remove the ciphertext noise before reconstruction, the overall correctness probability of decryption remains sensitive to the underlying noise growth and to the parameters of the FHE scheme.

3 Preliminaries

Notation. We use λ and ρ to denote the computational and statistical security parameters, respectively. We use κ to denote the number of LWE samples in the public key; that is, an LWE public key consists of pairs $\{(\vec{a}_i, b_i)\}_{i=0}^{\kappa-1}$. For $a, b \in \mathbb{N}$, we write $[a,b] = \{a, \ldots, b\}$ and $[a,b) = \{a, \ldots, b-1\}$. For a set S, we use $x \leftarrow\!\!\$\, S$ to denote sampling x from S uniformly at random; For a distribution $\mathcal{D}$, we use $x \leftarrow \mathcal{D}$ to denote sampling x according to $\mathcal{D}$. We use n and t to denote the total number of parties and the corruption threshold (i.e., the maximum number of corrupted parties), respectively. For the sake of simplicity, we assume that $n = 2t+1$ in the honest-majority setting, even if our protocols support the case of $n = 2t$. We use $\mathcal{H}$ and $\mathcal{C}$ to denote the set of honest parties and that of corrupted parties, respectively.

We write p for the plaintext modulus, and (ℓ, q) and (N, Q) for the dimension and ciphertext modulus in the LWE-based and RLWE-based settings, respectively. We assume, w.l.o.g., that q and Q are prime and that p is a power of two.

We write $\mathcal{R} = \mathbb{Z}[X]/(X^N+1)$ and $\mathcal{R}_Q = \mathcal{R}/Q\mathcal{R}$. We denote by $\vec{x}$ a vector in $\mathbb{Z}_q^\ell$ and by $\boldsymbol{x}$ a ring element in $\mathcal{R}_Q$ (equivalently, a vector in $\mathbb{Z}_Q^N$). We use x_i to denote the i-th component of a vector $\vec{x}$ or $\boldsymbol{x}$ with x_0 the first entry. For two vectors $\vec{x}$ and $\vec{y}$, we use $\langle \vec{x}, \vec{y} \rangle$ to denote their inner product. We denote by $\boldsymbol{x} \circ \boldsymbol{y}$ the multiplication of two ring elements $\boldsymbol{x}$ and $\boldsymbol{y}$ in $\mathcal{R}_Q$. The ℓ_∞ norm of a value x or a ring element $\boldsymbol{x}$ is denoted by $\|x\|_\infty$ or $\|\boldsymbol{x}\|_\infty$. By $\lfloor \cdot \rceil$, we denote the round function. A secret key distribution is denoted by $\mathcal{D}_{\mathsf{sk}}$, which is either the uniform distribution in $\{0,1\}$, or the distribution defined over $\{-1,0,1\}$ where choosing $-1, 1$ with probability $1/4$ and 0 with probability $1/2$. We use $\mathcal{D}_{\mathsf{sk}}^\ell$ to denote the distribution of length-ℓ vectors where each component is sampled from $\mathcal{D}_{\mathsf{sk}}$.

An additive secret sharing of a secret $x \in \mathbb{Z}_m$ is denoted by $\langle x \rangle^m$. Here, if $\mathbb{Z}_m$ is not explicitly specified, the superscript m is removed. We use $[\![x]\!]^q$ to denote a degree-t Shamir secret sharing over a finite field $\mathbb{Z}_q$, where the superscript q is removed, if $\mathbb{Z}_q$ is not explicitly specified. A vector of degree-t Shamir secret sharings over $\mathbb{Z}_q$ is written as $[\![\vec{x}]\!]^q$ or $[\![\boldsymbol{x}]\!]^q$. We use $[x]$ as a unified notation for linear secret sharings (e.g., additive secret sharings and Shamir secret sharings).

3.1 Secret Sharing Schemes

Shamir secret sharing scheme. Let $\mathbb{F}$ be a finite field with $|\mathbb{F}| > n$, and $\alpha_1, \ldots, \alpha_n$ be n distinct non-zero elements in $\mathbb{F}$, Shamir secret sharing scheme [66] is described as follows:

- $\mathsf{Share}(x)$: A dealer samples a random polynomial $f(\cdot) \in \mathbb{F}[X]$ of degree at most t, such that $f(0) = x$. Then, the dealer sends $f(\alpha_i) \in \mathbb{F}$ to P_i for each $i \in [1,n]$. A degree-t Shamir sharing on secret x is denoted by $[\![x]\!]$.

- $\mathsf{Rec}([\![x]\!], P_R)$: P_R receives the shares from any $t+1$ parties, and then uses the $t+1$ shares to reconstruct the secret $x = f(0)$ using Lagrange interpolation.

Additive secret sharing scheme. Let $\mathbb{R}$ be any-sized ring (e.g., $\mathbb{Z}_{2^k}$ for some $k \in \mathbb{N}$ or $\mathbb{Z}_q$ for a prime q). The additive secret sharing scheme with $t = n-1$ is shown as follows:

- $\mathsf{Share}(x)$: A dealer samples $\langle x \rangle_i \leftarrow\!\!\$\, \mathbb{R}$ for $i \in [1,n]$ such that $\sum_{i\in[1,n]} \langle x \rangle_i = x \in \mathbb{R}$. Then, the dealer sends the share $\langle x \rangle_i$ to every party P_i. An additive secret sharing on x is denoted by $\langle x \rangle$.

- $\mathsf{Rec}(\langle x \rangle, P_R)$: P_R receives the shares from all n parties, and reconstructs $x := \sum_{i\in[1,n]} \langle x \rangle_i \in \mathbb{R}$.

The Open protocol can always be constructed by running the Rec protocol n times, and thus we omit the description of Open of the Shamir and the additive secret sharing schemes. We use $\mathsf{Share}(x, [x]_J)$ to procedure samples $n - |J|$ shares in $\mathbb{F}$ ($\mathbb{R}$) uniformly such that all shares reconstruct the secret x, and outputs the shares for a subset $[1,n]\backslash J$ of parties for Shamir (additive) secret sharing scheme. Note that any $t+1$ shares of a degree-t Shamir sharing can be locally converted into an additive secret sharing with $t+1$ parties (see, e.g., [62]).

Functionality $\mathcal{F}_{\mathsf{Rand}}$

This functionality is parameterized by a finite field $\mathbb{F}$, and interacts with $P_1, \ldots, P_n$ and the adversary as follows:

1. Sample $r \leftarrow\!\!\$\, \mathbb{F}$, and receive a set $[r]_C$ of the shares of corrupted parties from the adversary.
2. Run $\mathsf{Share}(r, [r]_C)$ to obtain the shares of honest parties in $\mathcal{H}$, and send the shares to honest parties.

Figure 1: Functionality for generating random sharings.

Functionality $\mathcal{F}_{\mathsf{Coin}}$

This functionality is parameterized by a finite set S, and interacts with parties $P_1, \ldots, P_n$ as follows:

- Sample $r \leftarrow\!\!\$\, S$ and output r to all parties.

Figure 2: Functionality for coin tossing.

3.2 Ideal Functionalities

3.2.1 Ideal Functionalities for Generating Random Sharings and Random Coins

We adopt the standard ideal functionality $\mathcal{F}_{\mathsf{Rand}}$ shown in Figure 1 for generating random sharings over a field $\mathbb{F}$. We mainly use $\mathcal{F}_{\mathsf{Rand}}$ in the honest-majority setting; an instantiation of this functionality can be found in [32].

If the randomness r is sampled from $\{0,1\}$ and the shares are still defined over a finite field $\mathbb{F}$ (or a general ring $\mathbb{R}$), we use $\mathcal{F}_{\mathsf{RandBit}}$ to denote the ideal functionality for producing sharings of random bits. In the honest-majority setting (i.e., $t = (n-1)/2$), we adopt the protocol $\Pi^{field}_{\mathsf{RandBit}}$ in [31] to generate degree-t Shamir sharings of random bits. In the dishonest-majority setting (i.e., $t = n-1$), we modify $\Pi^{field}_{\mathsf{RandBit}}$ for generating additive secret sharings of random bits over $\mathbb{Z}_{2^k}$, where Shamir secret sharings over a Galois ring are replaced with additive secret sharings. The protocol $\Pi^{field}_{\mathsf{RandBit}}$ and modified protocol $\Pi^{ring}_{\mathsf{RandBit}}$ are described in the full version [47].

We also use a standard ideal functionality $\mathcal{F}_{\mathsf{Coin}}$ shown in Figure 2 to generate public random coins. For our usage, the set S in $\mathcal{F}_{\mathsf{Coin}}$ is $\mathbb{Z}_q^\ell$ or $\mathcal{R}_Q$. Our protocols mainly call $\mathcal{F}_{\mathsf{Coin}}$ in the honest-majority setting. In this setting, $\mathcal{F}_{\mathsf{Coin}}$ can be efficiently realized by calling $\mathcal{F}_{\mathsf{Rand}}$ and then opening either ℓ or N random degree-t Shamir sharings. The communication can be compressed by calling $\mathcal{F}_{\mathsf{Rand}}$ to generate a random seed and then using a random oracle to generate random coins in either $\mathbb{Z}_q^\ell$ or $\mathcal{R}_Q$.

Functionality $\mathcal{F}_{\mathsf{Mult}}$

This functionality is parameterized by a field $\mathbb{F}$. It interacts with parties $P_1, \ldots, P_n$ and the adversary as follows:

1. Receive the shares from all parties, and reconstruct the secrets $x, y \in \mathbb{F}$. Then compute $z := x \cdot y \in \mathbb{F}$.
2. Receive a set $[z]_C$ of the shares of corrupted parties from the adversary. Then, run $\mathsf{Share}(z, [z]_C)$ to obtain the shares of honest parties in $\mathcal{H}$, and send these shares to honest parties.

Figure 3: Functionality for multiplication of secret sharings over a finite field.

3.2.2 Ideal Functionality for Multiplications of Secret Sharings

We define a standard ideal functionality $\mathcal{F}_{\mathsf{Mult}}$ for multiplications of secret sharings, which is shown in Figure 3. It covers existing linear secret sharings in both honest-majority and dishonest-majority settings, e.g., Shamir secret sharings and additive secret sharings.

For two ring elements $\boldsymbol{a}, \boldsymbol{b} \in \mathcal{R}_Q$ written in vector form, we use $\circ$ to denote their ring multiplication, i.e., $\boldsymbol{c} = \boldsymbol{a} \circ \boldsymbol{b} \in \mathcal{R}_Q$. Therefore, given two vectors of degree-t Shamir sharings on $\boldsymbol{a}, \boldsymbol{b} \in \mathcal{R}_Q$, the degree-$t$ Shamir sharings of $\boldsymbol{c} = \boldsymbol{a} \circ \boldsymbol{b} \in \mathcal{R}_Q$ can be securely computed by calling $\mathcal{F}_{\mathsf{Mult}}$ over $\mathbb{Z}_Q$ multiple times in parallel, and then using the linear property of Shamir secret sharings to locally compute additions of Shamir sharings.

In the Shamir secret sharing setting, the multiplication of shared values can be carried out directly using the DN protocol [32]. In the dishonest-majority setting, our protocol invokes $\mathcal{F}_{\mathsf{Mult}}$ for multiplications of additive secret sharings over a ring $\mathbb{Z}_{2^k}$. For semi-honest security, all parties generate random multiplication triples over $\mathbb{Z}_{2^k}$ in the offline phase, and then use the Beaver approach [7] to compute multiplications of additive secret sharings in the online phase. The online protocol requires one round (or two rounds), and takes communication of $O(n^2 k)$ bits (or $O(nk)$ bits). The rounds and communication cost in the offline phase depend on which type of protocols is adopted. Our implementation adopts a computation-cheap and communication-large protocol. Specifically, all parties run an oblivious transfer (OT) extension protocol to generate random OTs, whose communication is sublinear in the number of random OTs, if the OT extension protocols (e.g., [12,13,45,63,71]) in the PCG framework are used. We adopt the state-of-the-art OT-extension implementation [71] to achieve not only small communication but also fast computation. Then, random OTs are consumed to compute random multiplication triples over $\mathbb{Z}_{2^k}$, using the Gilboa multiplication technique [42,48]. This protocol needs two rounds without counting the rounds in the reusable setup

phase, and takes communication of $O(n^2k^2)$ bits for each triple. To achieve smaller communication at the cost of more expensive computation, we can use a somewhat homomorphic encryption scheme (supporting one level of multiplications) to generate random multiplication triples over $\mathbb{Z}_{2^k}$, following prior works, e.g., [22,58]. This approach requires three rounds and communication of $O(nk)$ bits per triple.

3.3 Fully Homomorphic Encryption

3.3.1 FHE Definition

We define the syntax of FHE as follows:

Definition 1. *A fully homomorphic encryption (FHE) scheme* $\mathsf{FHE} = (\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec}, \mathsf{Eval})$ *consists of the following probabilistic polynomial-time algorithms:*

- $\mathsf{params} \leftarrow \mathsf{Setup}(1^\lambda)$*: On input a security parameter* λ*, the setup algorithm outputs a set of public parameters* params*, which is an implicit input to the following algorithms unless otherwise specified. In particular,* params *defines the plaintext space* $\mathcal{M}$ *and the ciphertext space* $\mathcal{T}$*.*
- $(\mathsf{sk}, \mathsf{pk}, \mathsf{evk}) \leftarrow \mathsf{KeyGen}(\mathsf{params})$*: On input* params*, the key-generation algorithm outputs a secret key* sk*, a public key* pk *and a public evaluation key* evk*.*
- $c \leftarrow \mathsf{Enc}(\mathsf{pk}, m)$*: On input a public key* pk *and a message* $m \in \mathcal{M}$*, the encryption algorithm outputs a ciphertext* c*.*
- $c' \leftarrow \mathsf{Eval}(\mathsf{evk}, f, c_1, \ldots, c_n)$*: On input a evaluation key* evk*, a function* $f : \mathcal{M}^n \to \mathcal{M}$*, a set of ciphertexts* $c_1, \ldots, c_n$*, the homomorphic evaluation algorithm outputs a ciphertext* c'*.*
- $m \leftarrow \mathsf{Dec}(\mathsf{sk}, c)$*: On input a secret key* sk *and a ciphertext* $c \in \mathcal{T}$*, the decryption algorithm outputs a message* m*.*

For correctness, we require for each $\lambda \in \mathbb{N}$, $n \geq 1$, $f : \mathcal{M}^n \to \mathcal{M}$ and $m_1, \ldots, m_n \in \mathcal{M}$, for any $(\mathsf{sk}, \mathsf{pk}, \mathsf{evk}) \leftarrow \mathsf{KeyGen}(\mathsf{Setup}(1^\lambda))$, we have with overwhelming probability

$$\mathsf{Dec}(\mathsf{sk}, \mathsf{Eval}(\mathsf{evk}, f, \mathsf{Enc}(\mathsf{pk}, m_1), \ldots, \mathsf{Enc}(\mathsf{pk}, m_n))) = f(m_1, \ldots, m_n).$$

In this paper, we aim to design threshold key-generation and decryption protocols for FHEW-like cryptosystems. A detailed description of the FHEW-like system is provided in the full version [47].

3.4 Threshold Fully Homomorphic Encryption

In the general threshold setting, threshold fully homomorphic encryption (ThFHE) enables n parties to jointly generate the FHE keys and any $t+1$ parties of n parties to decrypt ciphertexts. Even though t parties out of n parties are corrupted, the keys are kept secret and the ciphertexts are CPA secure.

Functionality $\mathcal{F}_{\mathsf{KeyGen}}$

This functionality is parameterized by a set of parameters params, and interacts with parties $P_1, \ldots, P_n$ and the adversary as follows.

1. Run $\mathsf{FHE.KeyGen}(\mathsf{params})$ to get $(\mathsf{sk}, \mathsf{pk}, \mathsf{evk})$.
2. Receive a set $[\mathsf{sk}]_{\mathcal{C}}$ of the shares of corrupted parties from the adversary. Then, run $\mathsf{Share}(\mathsf{sk}, [\mathsf{sk}]_{\mathcal{C}})$ to obtain the shares of honest parties in $\mathcal{H}$, and then send the shares to honest parties.
3. Send $(\mathsf{pk}, \mathsf{evk})$ to all parties and the adversary.

Figure 4: Functionality for threshold key generation of FHE.

Functionality $\mathcal{F}_{\mathsf{Dec}}$

This functionality interacts with any $t+1$ parties (w.l.o.g., $P_1, \ldots, P_{t+1}$) and a receiver P_R who obtains the decryption output, as well as the adversary as follows:

1. Receive the shares on sk from all parties, and reconstruct the secret key sk.
2. Receive a ciphertext c from all parties, and then run $\mathsf{FHE.Dec}(\mathsf{sk}, c)$ to obtain a message m.
3. Output m to P_R, and send c to the adversary.

Figure 5: Functionality for threshold decryption of FHE.

ThFHE is the same as FHE, except that there exists two efficient protocols Π_{KeyGen} and Π_{Dec} to securely compute algorithms KeyGen and Dec, respectively. As in prior works, we only consider the public-key setting, and thus Enc can be locally computed without the need of running any protocol. We adopt the ideal/real paradigm to prove security of protocols Π_{KeyGen} and Π_{Dec}, which enables ThFHE to be securely integrated into bigger protocols, e.g., a secure multi-party computation (MPC) protocol. Following prior work [30], we define two ideal functionalities $\mathcal{F}_{\mathsf{KeyGen}}$ and $\mathcal{F}_{\mathsf{Dec}}$, which are shown in Figure 4 and Figure 5, respectively. We will prove that Π_{KeyGen} and Π_{Dec} securely realize $\mathcal{F}_{\mathsf{KeyGen}}$ and $\mathcal{F}_{\mathsf{Dec}}$.

4 Threshold FHE with Semi-honest Security

This section describes our new ThFHE scheme Ajax with semi-honest security. In particular, for FHEW-like cryptosystems, two concretely efficient threshold key-generation and decryption protocols are shown in detail. Before giving the two protocols, we present two efficient distributed protocols, which enables n parties to jointly generate LWE and RLWE

Functionality $\mathcal{F}_{\mathsf{LWE}}$

This functionality is parameterized by an integer q, a dimension ℓ, an integer h and a noise distribution χ_σ with a variance σ, and interacts with $P_1, \ldots, P_n$ and the adversary as follows:

1. Receive the shares of $[\vec{s}]^q$ from all parties, and reconstruct the secret key $\vec{s} \in \mathbb{Z}_q^\ell$.
2. Sample $\vec{a} \leftarrow\!\!\$\, \mathbb{Z}_q^\ell$, and send it to all parties.
3. Sample a noise $e \leftarrow \chi_\sigma$, and receive a set $[e]_C^q$ of the shares of corrupted parties from the adversary. Then, run $\mathsf{Share}(e, [e]_C^q)$ to obtain the shares of honest parties in $\mathcal{H}$, and send the shares to honest parties.
4. Receive an error $d \in \mathbb{Z}_q$ with $\|d\|_\infty \le h$ from the adversary, and computes $b := \langle \vec{a}, \vec{s} \rangle + e + d \in \mathbb{Z}_q$.
5. Receive a set $[b]_C^q$ of the shares of corrupted parties from the adversary. Then, run $\mathsf{Share}(b, [b]_C^q)$ to obtain the shares of honest parties in $\mathcal{H}$, and send the shares to honest parties.

Figure 6: Functionality for generating sharings of LWE.

samples. These distributed protocols may be of independent interest for other applications such as distributed key generation of BGV, BFV, CKKS and lattice-based public-key encryption. Without loss of generality, we assume that the moduli q, Q for LWE and RLWE are primes. In this way, we are able to use the standard Shamir secret sharing scheme to design threshold key-generation protocols. The extension from a prime to power-of-two modulus is briefly introduced in the full version [47].

4.1 Distributed Protocols for Generating LWE and RLWE Samples

4.1.1 Distributed Protocol for Generating LWE Samples

The ideal functionality $\mathcal{F}_{\mathsf{LWE}}$, which generates a shared LWE ciphertext, is defined in Figure 6, with its instantiation given by Protocol Π_{LWE} in Figure 7. The ideal functionality $\mathcal{F}_{\mathsf{RLWE}}$ and protocol Π_{RLWE} are analogous to $\mathcal{F}_{\mathsf{LWE}}$ and Π_{LWE}; we defer their full description to the full version [47].

Lemma 1. *Protocol Π_{LWE} (Figure 7) securely realizes functionality $\mathcal{F}_{\mathsf{LWE}}$ (Figure 6) with the variance σ' in the presence of a semi-honest adversary who corrupts at most $t = (n-1)/2$ parties in the $\mathcal{F}_{\mathsf{Coin}}$-hybrid model.*

The proof of Lemma 1 is given in the full version [47]. When using PRG, the communication complexity of Π_{LWE} is approximately $n/2$ elements over $\mathbb{F}_q$ with 1 round per party.

Protocol Π_{LWE}

Inputs: Parties $P_1, \ldots, P_n$ hold a prime q, an LWE dimension ℓ and a vector of degree-t Shamir sharings $[\![\vec{s}]\!]^q$. Let $\chi_{\sigma'}$ be a discrete Gaussian distribution with variance σ'.

Generating LWE samples: $P_1, \ldots, P_n$ jointly generate an LWE sample with a noise distribution χ_σ as follows:

1. All parties call $\mathcal{F}_{\mathsf{Coin}}$ to produce a random vector $\vec{a} \in \mathbb{Z}_q^\ell$.
2. Every party P_i samples a noise $e_i \leftarrow \chi_{\sigma'}$ with $\sigma' = \sigma/(n-t)$, and then runs $\mathsf{Share}(e_i)$ to let all parties obtain a degree-t Shamir sharing $[\![e_i]\!]^q$.
3. All parties locally compute $[\![b]\!]^q := \langle \vec{a}, [\![\vec{s}]\!]^q \rangle + \sum_{i=1}^n [\![e_i]\!]^q$, and then **output** $(\vec{a}, [\![b]\!]^q)$.

Figure 7: Protocol with semi-honest security for distributed generation of LWE samples in the $\mathcal{F}_{\mathsf{Coin}}$-hybrid model.

4.1.2 Generating RLWE Samples

The distributed generation of RLWE samples mirrors the LWE case. We replace scalar Shamir sharings with vector-valued (coefficient-wise) Shamir sharings and realize ring multiplications via matrix operations in the coefficient representation. We therefore omit further details here. The protocol and functionality are presented in the full version [47].

We now explain why it suffices for each party to sample noise with variance $\sigma' = \sigma/(n-t)$ to ensure that the resulting LWE and RLWE ciphertexts achieve 128-bit security. Here, we assume that ciphertexts with noise variance σ already meet the 128-bit security level.

4.1.3 Gaussian Distribution

During the execution of Π_{LWE} and Π_{RLWE}, each party independently samples noise from a Gaussian distribution. Due to the additivity property of independent Gaussians, the sum of these noise terms is also Gaussian, with mean and variance equal to the sums of the individual means and variances, respectively. Specifically, let $X \sim \mathcal{N}(\mu_X, \sigma_X)$ and $Y \sim \mathcal{N}(\mu_Y, \sigma_Y)$ be two independent Gaussian random variables. Then:

$$X + Y \sim \mathcal{N}(\mu_X + \mu_Y, \sigma_X + \sigma_Y).$$

Applying this property, the combined noise from the $(n-t)$ honest parties has variance, $(n-t) \cdot \sigma' = (n-t) \cdot \frac{\sigma}{n-t} = \sigma$, which matches the required variance to achieve security. In real-world systems, it is impossible to perform discrete Gaussian sampling with an arbitrarily small variance σ. Due to the limitations of finite-precision and tailcut, the sampled noise distribution deviates from the ideal distribution.

In our implementation, we use the cumulative distribution table (CDT) sampling method proposed in [60] to sample noise. CDT sampling draws from a target discrete Gaussian distribution by precomputing a monotone table of cumulative probabilities. To sample, we first draw a uniform random value $r \in [0,1)$, then find the smallest index i in the cumulative distribution table such that the entry is at least r, and take i as the (unsigned) sample. For a zero-mean discrete Gaussian, finally assign a random sign to i by symmetry. Using Lemma 2 from [1], we estimate the relative error of the sampled noise and thereby determine the minimal variance α achievable for discrete Gaussian sampling under our parameter setting.

Definition 2 ([1]). *The relative error between P and Q is defined as*

$$\delta_{\mathrm{RE}}(P,Q) := \max_{x\in S} \delta_{\mathrm{RE}}(P(x),Q(x)) = \max_{x\in S} \frac{|P(x)-Q(x)|}{P(x)}.$$

Moreover, the statistical distance between P and Q is bounded as $\Delta(P,Q) \le \frac{1}{2}\delta_{\mathrm{RE}}(P,Q)$. Let τ be the tailcut parameter of Gaussian sampling implementation, so that the probability $\Pr[X=x]$ is set to zero for all $|x| \ge \tau\cdot\sigma$.

Lemma 2 ([1]). *Let $\bar{D}_s$ be the output distribution of CDT sampling, where s is the scale. Let $S := [\lfloor -\tau s \rfloor, \lceil \tau s \rceil] \cap \mathbb{Z}$ be the truncated support, and let p be the floating-point precision, i.e., the number of bits in the mantissa, then*

$$\delta_{\mathrm{RE}}(D_s,\bar{D}_s) \le \frac{\rho_s(\mathbb{Z})-\rho_s(S)}{2} + \max_{x\in S}\frac{\mathrm{cdf}_s(x)}{D_s(x)}\cdot 2^{-p}.$$

Under our setting, the relative error is bounded by:

$$\delta_{\mathrm{RE}}(D_s,\bar{D}_s) \le \frac{e^{-T^2/2s^2}}{1-e^{-(2T+1)/2s^2}} + \frac{1}{\rho_s(T-1)}\cdot 2^{-p}$$

Based on the above formula, we choose a value s such that $\delta_{\mathrm{RE}}(D_s,\bar{D}_s) \le 2^{-128}$, and then set α to this value of s. The detailed proof and analysis are provided in Appendix A.1 and Appendix A.2. We adopt the same tailcut parameter as in [35], setting $\tau = 12$. The floating-point precision is set to $p = 256$ bits. Under this configuration, when the scale and α are set to $s = 0.7$, the relative error between the resulting discrete Gaussian distribution and the ideal distribution is about $2^{-136.8}$. If $\sigma' = \sigma/(n-t)$ and σ' also satisfied $\sigma' \ge \alpha$, each party samples noise with variance σ', the total noise variance sums to $n\cdot\frac{\sigma}{n-t} \approx 2\sigma$. When the number of parties becomes large such that $\sigma/(n-t) < \alpha$, each party instead samples noise with variance α, the total noise variance becomes $n\cdot\alpha$ rather than 2σ (prior works are $n\cdot\sigma$).

4.2 Threshold Key Generation for FHEW-like Cryptosystems

In the key generation, we need to generate the parties' private key sk, public key pk, bootstrapping key bk, and key switching key ksk. Since pk, bk, and ksk are all related to sk, we first introduce how to generate the parties' sk.

Functionality $\mathcal{F}_{\mathsf{GenSK}}$

This functionality is parameterized by two integers q, Q as well as a secret-key distribution $\mathcal{D}_{\mathsf{sk}}$ that is defined over either $\{0,1\}$ or $\{-1,0,1\}$ for our use case. It interacts with parties $P_1,\ldots,P_n$ and the adversary as follows:

1. Sample $\vec{s} \leftarrow \mathcal{D}_{\mathsf{sk}}^{\ell}$ and $\boldsymbol{z} \leftarrow \mathcal{D}_{\mathsf{sk}}^{N}$.
2. Receive three sets of the shares of corrupted parties from the adversary, i.e., $[\vec{s}]_C^q$, $[\vec{s}]_C^Q$ and $[\boldsymbol{z}]_C^Q$.
3. Run $\mathsf{Share}(\vec{s},[\vec{s}]_C^q)$, $\mathsf{Share}(\vec{s},[\vec{s}]_C^Q)$ and $\mathsf{Share}(\boldsymbol{z},[\boldsymbol{z}]_C^Q)$ to obtain the shares of honest parties in $\mathcal{H}$, and then send the shares to honest parties.

Figure 8: Functionality for generating the secret sharings of secret keys of FHE.

4.2.1 Distributed Protocol for Generating Secret Keys

The ideal functionality for generating sk, denoted as $\mathcal{F}_{\mathsf{Gensk}}$, is shown in Figure 8. It outputs the parties' private key $\mathsf{sk} = ([\![\vec{s}]\!]^q, [\![\vec{s}]\!]^Q)$, along with the bootstrapping encryption key $[\![\boldsymbol{z}]\!]^Q$. Here, $\vec{s} = (s_1,\ldots,s_\ell) \in \mathcal{D}_{\mathsf{sk}}^{\ell}$ and $\boldsymbol{z} = (z_1,\ldots,z_N) \in \mathcal{D}_{\mathsf{sk}}^{N}$ are vectors of shared secret elements. A detailed instantiation of $\mathcal{F}_{\mathsf{GenSK}}$ is given in the full version [47]. A brief overview is provided below.

To generate the private key $\mathsf{sk} = ([\![\vec{s}]\!]^q, [\![\vec{s}]\!]^Q)$, all parties produce sharings $([\![s_i]\!]^q, [\![s_i]\!]^Q)$ for each $i \in [\ell]$ by emulating bitwise XOR over the field. To generate the bootstrapping encryption key $[\![\boldsymbol{z}]\!]^Q$, the parties generate sharings $[\![z_i]\!]^Q$ for each $i \in [N]$ using Protocol $\Pi_{\mathsf{RandBit}}^{field}$ detailed in the full version [47].

4.2.2 The Main Protocol for Threshold Key Generation

With the protocols described above, we now introduce our key generation protocol Π_{KeyGen}, as shown in Figure 9. We use κ to denote the number of LWE samples in the public key; that is, an LWE public key consists of pairs $\{(\vec{a}_i,b_i)\}_{i=0}^{\kappa-1}$, and in our setting we take $\kappa = \ell\cdot\lceil\log_2 q\rceil$; ℓ is the LWE dimension, N is the RLWE dimension, $\vec{s}$ is the LWE secret key, and $\boldsymbol{z}$ is the RLWE secret key. Moreover, L_{bk} and L_{ksk} denote the gadget lengths of the bootstrapping key and the key switching key, respectively, while B_{bk} and B_{ksk} denote their corresponding gadget bases.

- All parties follow the steps outlined in the protocols, call $\mathcal{F}_{\mathsf{GenSK}}$ to generate the degree-t Shamir sharings of a secret key $(\vec{s},\boldsymbol{z})$: $[\![\vec{s}]\!]^q, [\![\vec{s}]\!]^Q$ and $[\![\boldsymbol{z}]\!]^Q$.
- To generate public key pk, all parties using the protocol Π_{LWE} generate $(\vec{a}_i,[\![b_i]\!]^q), i \in [\kappa]$, then they run $\mathsf{Open}([\![b_i]\!]^q)$ to get b_i. Let $\mathsf{pk} = \{(\vec{a}_i,b_i)\}_{i\in[\kappa]}$.

Protocol Π_{KeyGen}

Inputs: $P_1,\ldots,P_n$ hold a set of public parameters $\mathsf{params} = (p,q,Q,\ell,N,\kappa,\sigma_{\mathsf{pk}},\sigma_{\mathsf{bsk}},\sigma_{\mathsf{ksk}},B_{\mathsf{bk}},B_{\mathsf{ksk}},\mathcal{D}_{\mathsf{sk}})$.

Distributed generation of the public key and secret key: All parties $P_1,\ldots,P_n$ do the following:

1. All parties call $\mathcal{F}_{\mathsf{GenSK}}$ to produce the degree-t Shamir sharings of a secret key $(\vec{s},\boldsymbol{z})$: $[\![\vec{s}]\!]^q, [\![\vec{s}]\!]^Q$ and $[\![\boldsymbol{z}]\!]^Q$. where $\vec{s} \in \mathcal{D}_{\mathsf{sk}}^{\ell}$ and $\boldsymbol{z} \in \mathcal{D}_{\mathsf{sk}}^{N}$.
2. For each $i \in [0,\kappa)$, the parties call $\mathcal{F}_{\mathsf{LWE}}$ with the input $[\![\vec{s}]\!]^q$ to obtain $(\vec{a}_i, [\![b_i]\!]^q)$, and then run $\mathsf{Open}([\![b_i]\!]^q)$ to get $b_i \in \mathbb{Z}_q$.
3. All parties **output** $\mathsf{pk} := \{(\vec{a}_i,b_i)\}_{i=0}^{\kappa-1}$ and $\mathsf{sk} = (\vec{s},\boldsymbol{z})$.

Distributed generation of a bootstrapping key: The parties $P_1,\ldots,P_n$ execute as follows:

3. All parties set $[\![\boldsymbol{m}_i]\!]^Q = ([\![s_i]\!]^Q, 0, ..., 0)$ for each $i \in [0,\ell)$, where the constant coefficient of $\boldsymbol{m}_i \in \mathcal{R}_Q$ is s_i and the other coefficients of $\boldsymbol{m}_i$ are 0.
4. For each $i \in [0,\ell)$ and $j \in [0, L_{\mathsf{bk}})$ where $L_{\mathsf{bk}} = \lceil \log_{B_{\mathsf{bk}}} Q \rceil$,
   (a) All parties call $\mathcal{F}_{\mathsf{RLWE}}$ with the input $[\![\boldsymbol{z}]\!]^Q$ to obtain $(\boldsymbol{a}_{i,j}, [\![\boldsymbol{b}_{i,j}]\!]^Q)$.
   (b) The parties update $[\![\boldsymbol{b}_{i,j}]\!]^Q := [\![\boldsymbol{b}_{i,j}]\!]^Q + (B_{\mathsf{bk}})^j \cdot [\![\boldsymbol{m}_i]\!]^Q$, and then run $\mathsf{Open}([\![\boldsymbol{b}_{i,j}]\!]^Q)$ to get $\boldsymbol{b}_{i,j} \in \mathcal{R}_Q$.
   (c) All parties set $\mathsf{RLWE}_{\boldsymbol{z},Q}((B_{\mathsf{bk}})^j \cdot \boldsymbol{m}_i) := (\boldsymbol{a}_{i,j}, \boldsymbol{b}_{i,j})$.
5. For each $i \in [0,\ell)$, all parties set
$$\mathsf{RLWE}'_{\boldsymbol{z},Q,B_{\mathsf{bk}}}(\boldsymbol{m}_i) = (\mathsf{RLWE}_{\boldsymbol{z},Q}(\boldsymbol{m}_i), \mathsf{RLWE}_{\boldsymbol{z},Q}(B_{\mathsf{bk}} \cdot \boldsymbol{m}_i), \ldots, \mathsf{RLWE}_{\boldsymbol{z},Q}((B_{\mathsf{bk}})^{L_{\mathsf{bk}}-1} \cdot \boldsymbol{m}_i)).$$
6. For each $i \in [0,\ell)$, the parties call $\mathcal{F}_{\mathsf{Mult}}$ N times on the input $(-[\![\boldsymbol{z}]\!]^Q, [\![\boldsymbol{m}_i]\!]^Q)$ to compute $[\![-\boldsymbol{z} \circ \boldsymbol{m}_i]\!]^Q$. The parties also perform Step (4) and Step (5) to obtain $\mathsf{RLWE}'_{\boldsymbol{z},Q,B_{\mathsf{bk}}}(-\boldsymbol{z} \circ \boldsymbol{m}_i)$.
7. All parties set $\mathsf{RGSW}_{\boldsymbol{z},Q,B_{\mathsf{bk}}}(s_i) = \big(\mathsf{RLWE}'_{\boldsymbol{z},Q,B_{\mathsf{bk}}}(-\boldsymbol{z} \circ \boldsymbol{m}_i), \mathsf{RLWE}'_{\boldsymbol{z},Q,B_{\mathsf{bk}}}(\boldsymbol{m}_i)\big)$, and then **output** $\mathsf{bk} = (\mathsf{RGSW}_{\boldsymbol{z},Q,B_{\mathsf{bk}}}(s_0), \mathsf{RGSW}_{\boldsymbol{z},Q,B_{\mathsf{bk}}}(s_1), ..., \mathsf{RGSW}_{\boldsymbol{z},Q,B_{\mathsf{bk}}}(s_{\ell-1}))$.

Distributed generation of a key-switching key: All parties $P_1,\ldots,P_n$ perform the following steps:

8. For each $i \in [0,N)$ and $j \in [0, L_{\mathsf{ksk}})$ where $L_{\mathsf{ksk}} = \lceil \log_{B_{\mathsf{ksk}}} Q \rceil$,
   (a) All parties call $\mathcal{F}_{\mathsf{LWE}}$ with the input $[\![\vec{s}]\!]^Q$ to get $(\vec{a}_{i,j}, [\![b_{i,j}]\!]^Q)$.
   (b) The parties update $[\![b_{i,j}]\!]^Q := [\![b_{i,j}]\!]^Q + (B_{\mathsf{ksk}})^j \cdot [\![z_i]\!]^Q$, and then run $\mathsf{Open}([\![b_{i,j}]\!]^Q)$ to obtain $b_{i,j} \in \mathbb{Z}_Q$.
   (c) All parties define $\mathsf{LWE}_{\vec{s},Q}((B_{\mathsf{ksk}})^j \cdot z_i) = (\vec{a}_{i,j}, b_{i,j})$.
9. For each $i \in [0,N)$, all parties set
$$\mathsf{LWE}'_{\vec{s},Q,B_{\mathsf{ksk}}}(z_i) = (\mathsf{LWE}_{\vec{s},Q}(z_i), \mathsf{LWE}_{\vec{s},Q}(B_{\mathsf{ksk}} \cdot z_i), \ldots, \mathsf{LWE}_{\vec{s},Q}((B_{\mathsf{bk}})^{L_{\mathsf{ksk}}-1} \cdot z_i)).$$
10. The parties **output** $\mathsf{ksk} = (\mathsf{LWE}'_{\vec{s},Q,B_{\mathsf{ksk}}}(z_0), \mathsf{LWE}'_{\vec{s},Q,B_{\mathsf{ksk}}}(z_1), ..., \mathsf{LWE}'_{\vec{s},Q,B_{\mathsf{ksk}}}(z_{N-1}))$.

Figure 9: Semi-honest protocol for threshold key generation of FHE in the $(\mathcal{F}_{\mathsf{GenSK}}, \mathcal{F}_{\mathsf{LWE}}, \mathcal{F}_{\mathsf{RLWE}}, \mathcal{F}_{\mathsf{Mult}})$-hybrid model. This protocol jointly generates the public key *pk*, secret key *sk*, bootstrapping key *bk*, and key switching key *ksk*.

- To generate the bootstrapping key bk, the parties follow the TFHE bootstrapping key generation process. For each $[\![s_i]\!]^Q$, parties compute $\mathsf{RGSW}_{z,Q,B_{\mathsf{bk}}}(s_i)$, and construct $\mathsf{bk} = (\mathsf{RGSW}_{z,Q,B_{\mathsf{bk}}}(s_0), \ldots, \mathsf{RGSW}_{z,Q,B_{\mathsf{bk}}}(s_{\ell-1}))$, as described in Step 3 of Π_{KeyGen}.
- To generate the key switching key ksk, the parties follow the TFHE key switching key generation process. They compute $\mathsf{ksk} = (\mathsf{LWE}'_{\vec{s},Q,B_{\mathsf{ksk}}}(z_0), \ldots, \mathsf{LWE}'_{\vec{s},Q,B_{\mathsf{ksk}}}(z_{N-1}))$, as outlined in Step 4 of Π_{KeyGen}.

Theorem 1. *Protocol Π_{KeyGen} (Figure 9) securely realizes functionality $\mathcal{F}_{\mathsf{KeyGen}}$ (Figure 4) with the variance σ' in the presence of a semi-honest adversary who corrupts at most $t = (n-1)/2$ parties in the $(\mathcal{F}_{\mathsf{GenSK}}, \mathcal{F}_{\mathsf{LWE}}, \mathcal{F}_{\mathsf{RLWE}}, \mathcal{F}_{\mathsf{Mult}})$-hybrid model.*

The proof of Theorem 1 can be found in the full version [47]. The communication complexity of generating the public key and secret key is $O(\ell n + \kappa n)$ elements over $\mathbb{F}_q$ and $O(N)$ elements over $\mathbb{F}_Q$ per party. The communication complexity of generating the bootstrapping key is $O(\ell N n L_{bk})$ elements over $\mathbb{F}_Q$ per party. The communication complexity of generating the key-switching key is $O(NnL_{ksk})$ elements over $\mathbb{F}_Q$ per party.

4.3 Threshold Decryption Protocol

We now present our distributed decryption protocol Π_{Dec}, which is shown in Figure 10. In the technical overview we assumed $p \mid q$ for clarity. However, since we utilize Shamir secret sharing over a field for ease of implementation and description, q is not divisible by p in practice, the technique described in the technical overview cannot be applied directly. This is because each party's additive share is distributed over $\mathbb{Z}_q$, and their sum takes the form $\Delta \cdot m + e + i \cdot q$.

When reducing this sum modulo Δ, we encounter a problem: since q is not divisible by p, it follows that q is also not divisible by Δ. Therefore,

$$\Delta \cdot m + e + i \cdot q \pmod{\Delta} = e + i \cdot q \pmod{\Delta} \neq e \pmod{\Delta}.$$

To preserve correctness, we need to perform modulus switching, changing the moduli from (q, Δ) to (η, Φ) such that η is divisible by Φ. The detailed correctness proof and noise analysis are deferred to the full version [47]. We provide a brief proof and analysis below.

Parties perform an additional modulus switching to their additive secret share $\langle \Delta \cdot m + e \rangle$ to change the modulus from q to η, where η is chosen such that $p \mid \eta$ (since p is a power of two, we can simply set $\eta = 2^{\lfloor \log_2 q \rfloor} = 52$), as shown in Step 1 of Π_{Dec}. This operation introduces additional noise; however, the added noise is very small compared to the original noise e. Let e' denote the resulting total noise after modulus switching and Φ denote η/p. After modulus switching, the parties obtain the sharing $\langle \Phi \cdot m + e' \rangle^{\eta}$. Each party can then locally reduce $\langle \Phi \cdot m + e' \rangle^{\eta}$ modulo Φ to extract an additive secret sharing $\langle e' \rangle^{\Phi}$. The remaining steps are identical to those described in the technical overview and are detailed in Protocol Π_{Dec}.

The correctness of the protocol depends on whether $e' + r \bmod \Phi = e' + r$, where $\Phi = \eta/p$. Since r is sampled uniformly from the interval $[-\Phi/2, \Phi/2)$, the sum $e' + r$ is uniformly distributed over the interval $[-\Phi/2 + e',\ \Phi/2 + e')$. Thus, the probability that $e' + r \bmod \Phi \neq e' + r$ is bounded by $|e'|/\Phi$. Consequently, if $|e'|/\Phi$ is negligible, then the probability that threshold decryption succeeds, which is $1 - |e'|/\Phi$, is overwhelming. Note that e' also follows a Gaussian distribution. Therefore, the decryption error probability is in fact given by the integral of the product of the Gaussian probability density function of e' and the factor $(1 - |e'|/\Phi)$ over the interval $[-\Phi/2, \Phi/2)$. Let the variances of e' be σ', and let $f_{\sigma'}(x)$ denote the probability density function (PDF) of e'. The

Protocol Π_{Dec}

Inputs: Any $t+1$ parties (w.l.o.g., $P_1, \ldots, P_{t+1}$) and a receiver P_R (who obtains the decryption output) are given a prime q, an LWE dimension ℓ and a parameter η such that $p \mid \eta$. Let $\Phi = \eta/p$. The parties $P_1, \ldots, P_{t+1}$ hold a vector of additive secret sharings $\langle \vec{s} \rangle^q$ and a ciphertext $c = (\vec{a}, b)$ in the form of $b = \langle \vec{a}, \vec{s} \rangle + e + \Delta \cdot m \in \mathbb{Z}_q$ with $\Delta = \lfloor q/p \rceil$ and $m \in \mathbb{Z}_p$.

Offline phase: The parties $P_1, \ldots, P_{t+1}$ call $\mathcal{F}_{\mathsf{DoubleRings}}$ to generate a pair of random double sharings $(\langle r \rangle^{\eta}, \langle r \rangle^{\Phi})$.

Online phase: The parties $P_1, \ldots, P_{t+1}$ and receiver P_R execute as follows:

1. All parties (i.e., $P_1, \ldots, P_{t+1}$) locally compute
$$\langle u \rangle^q := b - \langle \vec{a}, \langle \vec{s} \rangle^q \rangle = \langle \Delta \cdot m + e \rangle^q,$$
$$\langle u' \rangle^{\eta} := \langle \Phi \cdot m + e' \rangle^{\eta} := \left\lfloor \frac{\eta}{q} \langle u \rangle^q \right\rceil,$$
$$\langle e' \rangle^{\Phi} := \langle \Phi \cdot m + e' \rangle^{\eta} \pmod{\Phi}.$$
2. All parties locally compute $\langle v \rangle^{\Phi} := \langle e' \rangle^{\Phi} + \langle r \rangle^{\Phi}$, P_1 run $\mathsf{Rec}(\langle v \rangle^{\Phi}, P_1)$ to get $v = e' + r$.
3. All parties locally compute $\langle e' \rangle^{\eta} := v - \langle r \rangle^{\eta}$ over $\mathbb{Z}_{\eta}$ and $\langle \Phi \cdot m \rangle^{\eta} := \langle u' \rangle^{\eta} - \langle e' \rangle^{\eta}$.
4. All parties and P_R run $\mathsf{Rec}(\langle \Phi \cdot m \rangle^{\eta}, P_R)$ to let P_R obtain $\Phi \cdot m \in \mathbb{Z}_{\eta}$. Then, P_R locally computes $m \in \mathbb{Z}_p$ from $\Phi \cdot m$.

Figure 10: Semi-honest protocol for threshold decryption of FHE in the $\mathcal{F}_{\mathsf{DoubleRings}}$-hybrid model.

overall decryption success probability is given by integrating $f_{\sigma'}(x)$ over the valid range $x \in [-\Phi/2, \Phi/2)$, weighted by the corresponding success probability $1 - \frac{|x|}{\Phi}$:

$$\Pr[\Pi_{\mathsf{Dec}}(\mathsf{param}, ct, [s]^q) = m] = \sum_{x=-\Phi/2}^{\Phi/2-1} \left(1 - \frac{|x|}{\Phi}\right) f_{\sigma'}(x).$$

We adopt the same decryption error probability magnitude as in [36], setting it to 2^{-30}. With our parameters (q a 53-bit prime and $\ell = 2048$), the decryption error rate is about $2^{-30.27}$. Although our scheme takes a 2048-dimensional LWE ciphertext as input, its efficiency is comparable to TFHE with a 1024-dimensional input, since we first apply key switching to reduce the dimension to 1024. Consequently, in both bootstrapping modes (see the full version [47]), the blind rotation operates on the 1024 dimension. If the decryption correctness probability is still insufficient, we also provide a correctness amplification technique as an extension; see the full version [47].

Theorem 2. *Protocol Π_{Dec} (Figure 10) securely realizes functionality $\mathcal{F}_{\mathsf{Dec}}$ (Figure 5) in the presence of a semi-honest adversary who corrupts up to t of the $t+1$ parties in the $\mathcal{F}_{\mathsf{DoubleRings}}$-hybrid model, and the statistical distance between Π_{Dec} and secret key decryption to the ciphertext (with the form of $\vec{a} \cdot \vec{s} + \Phi \cdot m + e \mod \eta$) is bounded by*

$$|1 - \mathrm{erf}\left(\frac{\Delta/2}{\sqrt{2} \cdot \sigma}\right) - \sum_{x=-\Phi/2}^{\Phi/2-1} \left(1 - \frac{|x|}{\Phi}\right) f_\sigma(x)| < 2^{-\rho}$$

where σ is the variance of the noise e in the ciphertext, $f_\sigma(x)$ is the probability density function of the Gaussian distribution with variance σ, and Φ is the modulus switching parameter, ρ is statistical security parameter.

Note that the variance σ of $f_\sigma(x)$ in Theorem 2 is taken after modulus switching. Thus, increasing Φ (i.e., choosing a larger η) amplifies the ciphertext noise but has a minor effect on the decryption correctness probability. We choose η close to q to minimize the noise introduced by modulus switching.

If the decryption correctness error is around 2^{-30}, the statistical distance between encrypt-then-decrypt and direct plaintext decryption is also around 2^{-30}. We adopt the same level of statistical distance, setting the statistical security parameter to $\rho = 30$. The proof of Theorem 2 is postponed to the full version [47].

The total communication complexity of Π_{Dec} in the online phase is $\log_2(\Phi \cdot \eta)$ bits with 2 rounds per party (100 bits in our setting). The communication complexity of Π_{Dec} in the offline phase depends on the specific implementation. The method we adopt, along with its associated communication cost, is described in Section 3.2.2.

Functionality $\mathcal{F}_{\mathsf{DoubleRings}}$

This functionality is parameterized by two rings $\mathbb{Z}_\eta$ and $\mathbb{Z}_\Phi$ with $\Phi \mid \eta$, and interacts with parties $P_1, \ldots, P_{t+1}$ and the adversary as follows:

1. Sample a random element $r \in \mathbb{Z}_\Phi$, and receive two sets $\langle r \rangle_C^\eta$ and $\langle r \rangle_C^\Phi$ of the shares of corrupted parties.
2. Run $\mathsf{Share}(r, \langle r \rangle_C^\eta)$ and $\mathsf{Share}(r, \langle r \rangle_C^\Phi)$ over respective rings $\mathbb{Z}_\eta$ and $\mathbb{Z}_\Phi$ to obtain the shares of honest parties in $\mathcal{H}$. Then, send the shares to honest parties.

Figure 11: Functionality for generating random double sharings over two different rings.

Protocol $\Pi_{\mathsf{DoubleRings}}$

Inputs: $P_1, \ldots, P_{t+1}$ are given two parameters η, Φ such that $\Phi \mid \eta$ and both of η, Φ are power-of-two. Let $m = \log \Phi$.

1. For each $i \in \{0, \ldots, \log \Phi - 1\}$, all parties call $\mathcal{F}_{\mathsf{RandBit}}$ to generate a random bit sharing $\langle b_i \rangle^\eta$.
2. All parties locally compute $\langle r \rangle^\eta := \sum_{i \in [0,m)} 2^i \cdot \langle b_i \rangle^\eta + \sum_{i \in [m, \log \eta)} 2^i \cdot \langle b_{m-1} \rangle^\eta$ and $\langle r \rangle^\Phi := \langle r \rangle^\eta \pmod{\Phi}$.
3. All parties **output** the random double sharing $(\langle r \rangle^\eta, \langle r \rangle^\Phi)$.

Figure 12: Semi-honest protocol for generating random double sharings over different rings in the $\mathcal{F}_{\mathsf{RandBit}}$-hybrid model.

4.3.1 Random Sharings over Two Different Rings

$\mathcal{F}_{\mathsf{DoubleRings}}$ is responsible for generating additive secret sharings $(\langle r \rangle^\eta, \langle r \rangle^\Phi)$, shown in Figure 11. The implementation of $\mathcal{F}_{\mathsf{DoubleRings}}$ is detailed in Protocol $\Pi_{\mathsf{DoubleRings}}$ (Figure 12).

The approach begins by generating shared random bit $\langle b \rangle$ over $\mathbb{Z}_\eta$ by $\mathcal{F}_{\mathsf{RandBit}}$, the instantiation of $\mathcal{F}_{\mathsf{RandBit}}$ can be found in the full version [47]. Then, $\log \Phi$ shared additive bits are combined to generate additive secret sharing $\langle r \rangle^\eta \in \mathbb{Z}_\eta$, where r is uniformly random in $[-\Phi/2, \Phi/2)$. Since $\Phi \mid \eta$, we can directly reduce modulo Φ to simultaneously obtain additive secret sharings $\langle r \rangle^\Phi \in \mathbb{Z}_\Phi$.

Lemma 3. *Protocol $\Pi_{\mathsf{DoubleRings}}$ (Figure 12) securely realizes functionality $\mathcal{F}_{\mathsf{DoubleRings}}$ (Figure 11) in the presence of a semi-honest adversary who corrupts up to t of the $t+1$ parties in the $\mathcal{F}_{\mathsf{RandBit}}$-hybrid model.*

The proof of Lemma 3 is given in the full version [47].

Corruption Threshold	Network	Noah's Online	Noah's Total	Our's Online	Our's Total
1	LAN	242.44	242.44	0.003	0.855
1	WAN	291.79	291.79	0.037	5.405
3	LAN	243.41	None	0.005	1.499
3	WAN	294.86	None	0.081	13.464
13	LAN	259.58	None	0.045	5.660
13	WAN	309.39	None	0.235	54.499

Table 1: End-to-end distributed decryption time comparison (ms) with [30]. Total time for $t > 1$ is not reported in [30].

5 Performance Evaluation

5.1 Parameter Selection

For a more efficient instantiation, we minimize the LWE dimension ℓ and the ring dimension N. We deploy a 53-bit modulus and a dimension of 2048 for both key switching and bootstrapping. We set the modulus $p = 2^4$ to store the message concatenated with the carry. The LWE public key security parameter $\kappa = \ell \cdot \lceil \log_2 q \rceil$. We report the detailed parameters in Appendix B Table 2. The noise variance σ refers to the overall variance after aggregating the noise generated by all parties. The above parameters provide 131-bit computational security verified by the LWE estimator [61], and bound the threshold decryption failure probability to 2^{-30} with overwhelming probability, with the same error probability magnitude as in [36].

5.2 Experimental Setup

We implement our ThFHE scheme Ajax using the Primus-FHE library [49] for FHEW-like cryptosystem and EMP-toolkit [69] for triples generation. Our implementation is developed using Rust and C++. All experiments are conducted on a cluster of AWS `c6i.2xlarge` instances, with each party running on a separate instance of 8 vCPUs and 16 GiB RAM. Our protocol utilizes multi-threading but the performance is still limited when scaling up the number of parties.

For network configuration, we use the default AWS network settings as our local setting, with approximately 0.1 ms round-trip time (RTT) and 10 Gbps bandwidth between instances. To simulate different network conditions, we use the `tc` command to adjust the RTT and bandwidth: 1 ms RTT with 1 Gbps bandwidth for local area network (LAN) setting, and 100 ms RTT with 100 Mbps bandwidth for wide area network (WAN) setting.

5.3 Performance of Threshold Key Generation

Although [4] includes implementations of ThFHE for CKKS and BGV/BFV schemes, we were unable to find any implementation of ThFHE for the TFHE scheme. No other comparable implementations were identified to enable a meaningful performance comparison. Therefore, we report only the breakdown of our key generation time across different phases. The complete end-to-end threshold key generation time is provided in the full verison. We also evaluate the impact of varying the number of parties on the performance of the Ajax threshold key generation protocol under different network conditions, shown in the full version [47]. The main bottleneck lies in generating the bootstrapping key. In our experimental results, the communication cost for generating the bootstrapping key accounts for about 99% of the total communication in the key generation phase. Detailed communication statistics are provided in the full version [47].

5.4 Performance of Threshold Decryption

Table 1 summarizes the threshold decryption performance of Ajax compared to [30]; we cite performance results directly from their paper. For $t = 1$, total times are reported; for $t = 3$ and $t = 13$, only the online phase is provided, and the offline time is marked as None. Their scheme employs noise flooding and a Switch-and-Squash procedure, causing runtime to vary across parameter sets, we report their fastest configuration. Our results are averaged over 20,000 batches. To the best of our knowledge, [30] is the only prior work that designs and implements a concretely efficient threshold TFHE scheme. In terms of security, our protocol is incomparable with [32]. For distributed decryption, while [30] assumes a corruption threshold $t = (n-1)/3$ but is secure against malicious adversaries, our protocol assumes $t = n-1$ but works in the semi-honest setting. For key generation, while [30] does not provide a protocol, we give a detailed protocol with only $max(0.038n, 2)\times$ noise growth. We acknowledge that comparing the performance between our protocol and [30] is not fair, as the security is incomparable. However, it may still be reasonable for scenarios that focus on the practicability and ignore which security model is used. Furthermore, [30] is the only prior work similar to ours.

We report the detailed results for the average threshold decryption time over 20,000 ciphertexts in the full version [47], illustrating the impact of varying the number of parties on the performance of Ajax batch threshold decryption protocol in the online phase under different network conditions. The results are presented in the full version [47].

Acknowledgements

The work of Yu Yu is supported by the National Natural Science Foundation of China (Grant Nos. 62125204 and 92270201) and Innovation Program for Quantum Science and Technology (No. 2021ZD0302901 / 2021ZD0302902).

Ethical Considerations

This work presents theoretical and algorithmic improvements to Threshold Fully Homomorphic Encryption (ThFHE), a cryptographic primitive that enables distributed computation over encrypted data. It does not involve human subjects, personal data, or interactions with deployed systems. The primary stakeholders of this research are: (i) the servers and organizations that deploy our threshold decryption protocol, (ii) the end users who rely on such systems for secure computation over encrypted data, and (iii) the data owners whose data may be encrypted under the FHE public keys, when they are not the same as the servers or end users. Our security model and protocols are designed to protect the confidentiality and integrity of these stakeholders' data and computations within clearly stated assumptions.

We prove the security of our protocols against semi-honest adversaries, who follow the prescribed steps but may try to extract additional information from the protocol transcript. Under our chosen parameters, the computational security level is 128 bits and the statistical security level is 30 bits. Finally, our open-source implementation is intended solely to validate the correctness and efficiency of our design; it has not undergone any security hardening or side-channel analysis, and should not be used as-is in real-world deployments.

We acknowledge several potential negative impacts and limitations:

- Misuse and over-reliance. Stronger cryptographic tools can be misused by malicious actors to hide harmful activity, and an incomplete understanding of the security model may lead deployers to overestimate the protection provided by our protocol.
- Implementation vulnerabilities. Any real-world implementation may contain bugs, side channels, or misconfigurations that fall outside our formal model and could compromise security.
- Model limitations. Our analysis does not cover all threat scenarios (e.g., asynchronous networks, dishonest majority in KeyGen). Attacks exploiting such gaps are outside our formal guarantees.
- Denial-of-Service (DoS). Our protocols involve computationally intensive cryptographic operations. An adversary might attempt to abuse these operations to mount DoS attacks on deployed systems.

To mitigate these risks, we explicitly document our security assumptions and adversary models, highlight the above limitations in the paper, and avoid making claims beyond what is proven. We encourage implementers to follow best practices for secure software development, side-channel hardening, and rate-limiting to reduce DoS risk and to integrate our constructions only within systems whose threat models match our assumptions. Within these bounds, and in line with the conference ethics guidelines, we believe that this research has been conducted responsibly.

The implementation uses publicly available open-source libraries, all of which were used in compliance with their respective licenses. We have read the conference ethics guidelines and submission instructions; we believe this research was conducted ethically; and this appendix provides the required, clearly marked ethics statement.

Open Science

To support reproducibility and in compliance with the USENIX Security open-science policy, we provide all artifacts necessary to evaluate the contributions of this paper.

Artifacts Provided

1. **Source Code.** Implementation of our improved Threshold Fully Homomorphic Encryption (ThFHE) protocols:
   - `thfhe/src/`: Core ThFHE protocol implementation.
   - `mpc/src/`: Multi-party computation schemes, including Shamir and additive secret sharing.
   - `network/src/`: Basic network communication and I/O components.
   - `thfhe/triples/ole/`: Code (using EMP) to generate Beaver triples over $\mathbb{Z}_{2^k}$.
2. **Experiment Scripts.** Supporting scripts for executing experiments:
   - `thfhe/batch/`: Scripts required to run batch experiments.
3. **Documentation.** `README.md` detailing setup instructions, required dependencies, compilation commands, and steps to reproduce all experimental results.

Access

All artifacts are available at the time of submission via the following **link**: `[https://doi.org/10.5281/zenodo.17963095]`. We commit to maintaining public availability of the artifacts.

References

[1] Carlos Aguilar-Melchor and Thomas Ricosset. Cdt-based gaussian sampling: From multi to double precision. *IEEE Transactions on Computers*, 67(11):1610–1621, 2018.

[2] Ahmad Al Badawi, Jack Bates, Flavio Bergamaschi, David Bruce Cousins, Saroja Erabelli, Nicholas Genise, Shai Halevi, Hamish Hunt, Andrey Kim, Yongwoo Lee, Zeyu Liu, Daniele Micciancio, Ian Quah, Yuriy Polyakov, Saraswathy R.V., Kurt Rohloff, Jonathan Saylor, Dmitriy Suponitsky, Matthew Triplett, Vinod Vaikuntanathan, and Vincent Zucca. OpenFHE: Open-source fully homomorphic encryption library. In *Proceedings of the 10th Workshop on Encrypted Computing & Applied Homomorphic Cryptography – WAHC'22*, pages 53–63. ACM, 2022.

[3] Gilad Asharov, Abhishek Jain, Adriana López-Alt, Eran Tromer, Vinod Vaikuntanathan, and Daniel Wichs. Multiparty computation with low communication, computation and interaction via threshold FHE. In David Pointcheval and Thomas Johansson, editors, *EUROCRYPT 2012*, volume 7237 of *LNCS*, pages 483–501. Springer, Berlin, Heidelberg, April 2012.

[4] Ahmad Al Badawi, Andreea Alexandru, Jack Bates, Flavio Bergamaschi, David Bruce Cousins, Saroja Erabelli, Nicholas Genise, Shai Halevi, Hamish Hunt, Andrey Kim, Yongwoo Lee, Zeyu Liu, Daniele Micciancio, Carlo Pascoe, Yuriy Polyakov, Ian Quah, Saraswathy R. V., Kurt Rohloff, Jonathan Saylor, Dmitriy Suponitsky, Matthew Triplett, Vinod Vaikuntanathan, and Vincent Zucca. OpenFHE: Open-source fully homomorphic encryption library. Cryptology ePrint Archive, Report 2022/915, 2022.

[5] Youngjin Bae, Jung Hee Cheon, Jaehyung Kim, Jai Hyun Park, and Damien Stehlé. HERMES: Efficient ring packing using MLWE ciphertexts and application to transciphering. In Helena Handschuh and Anna Lysyanskaya, editors, *CRYPTO 2023, Part IV*, volume 14084 of *LNCS*, pages 37–69. Springer, Cham, August 2023.

[6] Shi Bai, Tancrède Lepoint, Adeline Roux-Langlois, Amin Sakzad, Damien Stehlé, and Ron Steinfeld. Improved security proofs in lattice-based cryptography: Using the Rényi divergence rather than the statistical distance. *Journal of Cryptology*, 31(2):610–640, April 2018.

[7] Donald Beaver. Efficient multiparty protocols using circuit randomization. In Joan Feigenbaum, editor, *CRYPTO'91*, volume 576 of *LNCS*, pages 420–432. Springer, Berlin, Heidelberg, August 1992.

[8] Loris Bergerat, Anas Boudi, Quentin Bourgerie, Ilaria Chillotti, Damien Ligier, Jean-Baptiste Orfila, and Samuel Tap. Parameter optimization and larger precision for (T)FHE. *Journal of Cryptology*, 36(3):28, July 2023.

[9] Dan Boneh, Rosario Gennaro, Steven Goldfeder, Aayush Jain, Sam Kim, Peter M. R. Rasmussen, and Amit Sahai. Threshold cryptosystems from threshold fully homomorphic encryption. In Hovav Shacham and Alexandra Boldyreva, editors, *CRYPTO 2018, Part I*, volume 10991 of *LNCS*, pages 565–596. Springer, Cham, August 2018.

[10] Katharina Boudgoust and Peter Scholl. Simple threshold (fully homomorphic) encryption from LWE with polynomial modulus. In Jian Guo and Ron Steinfeld, editors, *ASIACRYPT 2023, Part I*, volume 14438 of *LNCS*, pages 371–404. Springer, Singapore, December 2023.

[11] Florian Bourse and Malika Izabachène. Plug-and-play sanitization for TFHE. Cryptology ePrint Archive, Report 2022/1438, 2022.

[12] Elette Boyle, Geoffroy Couteau, Niv Gilboa, Yuval Ishai, Lisa Kohl, Nicolas Resch, and Peter Scholl. Correlated pseudorandomness from expand-accumulate codes. In Yevgeniy Dodis and Thomas Shrimpton, editors, *CRYPTO 2022, Part II*, volume 13508 of *LNCS*, pages 603–633. Springer, Cham, August 2022.

[13] Elette Boyle, Geoffroy Couteau, Niv Gilboa, Yuval Ishai, Lisa Kohl, Peter Rindal, and Peter Scholl. Efficient two-round OT extension and silent non-interactive secure computation. In Lorenzo Cavallaro, Johannes Kinder, XiaoFeng Wang, and Jonathan Katz, editors, *ACM CCS 2019*, pages 291–308. ACM Press, November 2019.

[14] Zvika Brakerski. Fully homomorphic encryption without modulus switching from classical GapSVP. In Reihaneh Safavi-Naini and Ran Canetti, editors, *CRYPTO 2012*, volume 7417 of *LNCS*, pages 868–886. Springer, Berlin, Heidelberg, August 2012.

[15] Zvika Brakerski, Craig Gentry, and Vinod Vaikuntanathan. (Leveled) fully homomorphic encryption without bootstrapping. In Shafi Goldwasser, editor, *ITCS 2012*, pages 309–325. ACM, January 2012.

[16] Luís T. A. N. Brandão and René Peralta. NISTIR 8214C 2pd: NIST first call for multi-party threshold schemes (second public draft). NIST Internal Report, 2025. https://csrc.nist.gov/pubs/ir/8214/c/2pd.

[17] Ran Canetti, Rosario Gennaro, Steven Goldfeder, Nikolaos Makriyannis, and Udi Peled. UC non-interactive, proactive, threshold ECDSA with identifiable aborts. In

Jay Ligatti, Xinming Ou, Jonathan Katz, and Giovanni Vigna, editors, *ACM CCS 2020*, pages 1769–1787. ACM Press, November 2020.

[18] Yijia Chang and Songze Li. Arbitrary-threshold fully homomorphic encryption with lower complexity. The 34th USENIX Security Symposium, 2025.

[19] Sylvain Chatel, Christian Mouchet, Ali Utkan Sahin, Apostolos Pyrgelis, Carmela Troncoso, and Jean-Pierre Hubaux. PELTA - shielding multiparty-FHE against malicious adversaries. In Weizhi Meng, Christian Damsgaard Jensen, Cas Cremers, and Engin Kirda, editors, *ACM CCS 2023*, pages 711–725. ACM Press, November 2023.

[20] Jung Hee Cheon, Wonhee Cho, and Jiseung Kim. Improved universal thresholdizer from iterative shamir secret sharing. *Journal of Cryptology*, 38(15), 2025.

[21] Jung Hee Cheon, Andrey Kim, Miran Kim, and Yong Soo Song. Homomorphic encryption for arithmetic of approximate numbers. In Tsuyoshi Takagi and Thomas Peyrin, editors, *ASIACRYPT 2017, Part I*, volume 10624 of *LNCS*, pages 409–437. Springer, Cham, December 2017.

[22] Jung Hee Cheon, Dongwoo Kim, and Keewoo Lee. MHz2k: MPC from HE over $\mathbb{Z}_{2^k}$ with new packing, simpler reshare, and better ZKP. In Tal Malkin and Chris Peikert, editors, *CRYPTO 2021, Part II*, volume 12826 of *LNCS*, pages 426–456, Virtual Event, August 2021. Springer, Cham.

[23] Ilaria Chillotti, Nicolas Gama, Mariya Georgieva, and Malika Izabachène. Faster fully homomorphic encryption: Bootstrapping in less than 0.1 seconds. In Jung Hee Cheon and Tsuyoshi Takagi, editors, *ASIACRYPT 2016, Part I*, volume 10031 of *LNCS*, pages 3–33. Springer, Berlin, Heidelberg, December 2016.

[24] Ilaria Chillotti, Nicolas Gama, Mariya Georgieva, and Malika Izabachène. Faster packed homomorphic operations and efficient circuit bootstrapping for TFHE. In Tsuyoshi Takagi and Thomas Peyrin, editors, *ASIACRYPT 2017, Part I*, volume 10624 of *LNCS*, pages 377–408. Springer, Cham, December 2017.

[25] Ilaria Chillotti, Nicolas Gama, Mariya Georgieva, and Malika Izabachène. TFHE: Fast fully homomorphic encryption over the torus. *Journal of Cryptology*, 33(1):34–91, January 2020.

[26] Ilaria Chillotti, Damien Ligier, Jean-Baptiste Orfila, and Samuel Tap. Improved programmable bootstrapping with larger precision and efficient arithmetic circuits for TFHE. In Mehdi Tibouchi and Huaxiong Wang, editors, *ASIACRYPT 2021, Part III*, volume 13092 of *LNCS*, pages 670–699. Springer, Cham, December 2021.

[27] Ashish Choudhury, Jake Loftus, Emmanuela Orsini, Arpita Patra, and Nigel P. Smart. Between a rock and a hard place: Interpolating between MPC and FHE. In Kazue Sako and Palash Sarkar, editors, *ASIACRYPT 2013, Part II*, volume 8270 of *LNCS*, pages 221–240. Springer, Berlin, Heidelberg, December 2013.

[28] Siddhartha Chowdhury, Sayani Sinha, Animesh Singh, Shubham Mishra, Chandan Chaudhary, Sikhar Patranabis, Pratyay Mukherjee, Ayantika Chatterjee, and Debdeep Mukhopadhyay. Efficient threshold FHE with application to real-time systems. Cryptology ePrint Archive, Report 2022/1625, 2022.

[29] Hila Dahari-Garbian, Ariel Nof, and Luke Parker. Trout: Two-round threshold ecdsa from class groups. In *Proceedings of the 2025 ACM SIGSAC Conference on Computer and Communications Security*, pages 380–393, 2025.

[30] Morten Dahl, Daniel Demmler, Sarah El Kazdadi, Arthur Meyre, Jean-Baptiste Orfila, Dragos Rotaru, Nigel P. Smart, Samuel Tap, and Michael Walter. Noah's ark: Efficient threshold-fhe using noise flooding. In *Proceedings of the 11th Workshop on Encrypted Computing & Applied Homomorphic Cryptography – WAHC'23*, pages 35–46. ACM, 2023.

[31] Ivan Damgård, Matthias Fitzi, Eike Kiltz, Jesper Buus Nielsen, and Tomas Toft. Unconditionally secure constant-rounds multi-party computation for equality, comparison, bits and exponentiation. In Shai Halevi and Tal Rabin, editors, *TCC 2006*, volume 3876 of *LNCS*, pages 285–304. Springer, Berlin, Heidelberg, March 2006.

[32] Ivan Damgård and Jesper Buus Nielsen. Scalable and unconditionally secure multiparty computation. In Alfred Menezes, editor, *CRYPTO 2007*, volume 4622 of *LNCS*, pages 572–590. Springer, Berlin, Heidelberg, August 2007.

[33] Jack Doerner, Yashvanth Kondi, Eysa Lee, and abhi shelat. Threshold ECDSA from ECDSA assumptions: The multiparty case. In *2019 IEEE Symposium on Security and Privacy*, pages 1051–1066. IEEE Computer Society Press, May 2019.

[34] Jack Doerner, Yashvanth Kondi, Eysa Lee, and Abhi Shelat. Threshold ecdsa in three rounds. In *2024 IEEE Symposium on Security and Privacy (SP)*, pages 3053–3071. IEEE, 2024.

[35] Léo Ducas, Alain Durmus, Tancrède Lepoint, and Vadim Lyubashevsky. Lattice signatures and bimodal gaussians. Cryptology ePrint Archive, Paper 2013/383, 2013.

[36] Léo Ducas and Daniele Micciancio. FHEW: Bootstrapping homomorphic encryption in less than a second. In Elisabeth Oswald and Marc Fischlin, editors, *EUROCRYPT 2015, Part I*, volume 9056 of *LNCS*, pages 617–640. Springer, Berlin, Heidelberg, April 2015.

[37] Ehsan Ebrahimi and Anshu Yadav. Strongly secure universal thresholdizer. In Kai-Min Chung and Yu Sasaki, editors, *ASIACRYPT 2024, Part III*, volume 15486 of *LNCS*, pages 207–239. Springer, Singapore, December 2024.

[38] Junfeng Fan and Frederik Vercauteren. Somewhat practical fully homomorphic encryption. Cryptology ePrint Archive, Report 2012/144, 2012.

[39] Radhika Garg, Kang Yang, Jonathan Katz, and Xiao Wang. Scalable mixed-mode MPC. In *2024 IEEE Symposium on Security and Privacy*, pages 523–541. IEEE Computer Society Press, May 2024.

[40] Craig Gentry. Fully homomorphic encryption using ideal lattices. In Michael Mitzenmacher, editor, *41st ACM STOC*, pages 169–178. ACM Press, May / June 2009.

[41] Craig Gentry, Amit Sahai, and Brent Waters. Homomorphic encryption from learning with errors: Conceptually-simpler, asymptotically-faster, attribute-based. In Ran Canetti and Juan A. Garay, editors, *CRYPTO 2013, Part I*, volume 8042 of *LNCS*, pages 75–92. Springer, Berlin, Heidelberg, August 2013.

[42] Niv Gilboa. Two party RSA key generation. In Michael J. Wiener, editor, *CRYPTO'99*, volume 1666 of *LNCS*, pages 116–129. Springer, Berlin, Heidelberg, August 1999.

[43] Shruthi Gorantala, Rob Springer, Sean Purser-Haskell, William Lam, Royce Wilson, Asra Ali, Eric P. Astor, Itai Zukerman, Sam Ruth, Christoph Dibak, Phillipp Schoppmann, Sasha Kulankhina, Alain Forget, David Marn, Cameron Tew, Rafael Misoczki, Bernat Guillen, Xinyu Ye, Dennis Kraft, Damien Desfontaines, Aishe Krishnamurthy, Miguel Guevara, Irippuge Milinda Perera, Yurii Sushko, and Bryant Gipson. A general purpose transpiler for fully homomorphic encryption. Cryptology ePrint Archive, Paper 2021/811, 2021. https://eprint.iacr.org/2021/811.

[44] S. Dov Gordon, Feng-Hao Liu, and Elaine Shi. Constant-round MPC with fairness and guarantee of output delivery. In Rosario Gennaro and Matthew J. B. Robshaw, editors, *CRYPTO 2015, Part II*, volume 9216 of *LNCS*, pages 63–82. Springer, Berlin, Heidelberg, August 2015.

[45] Xiaojie Guo, Kang Yang, Xiao Wang, Wenhao Zhang, Xiang Xie, Jiang Zhang, and Zheli Liu. Half-tree: Halving the cost of tree expansion in COT and DPF. In Carmit Hazay and Martijn Stam, editors, *EUROCRYPT 2023, Part I*, volume 14004 of *LNCS*, pages 330–362. Springer, Cham, April 2023.

[46] James Howe, Ayesha Khalid, Ciara Rafferty, Francesco Regazzoni, and Máire O'Neill. On practical discrete gaussian samplers for lattice-based cryptography. *IEEE Transactions on Computers*, 67(3):322–334, 2016.

[47] Zhenkai Hu, Haofei Liang, Xiao Wang, Xiang Xie, Kang Yang, Yu Yu, and Wenhao Zhang. Ajax: Fast threshold fully homomorphic encryption without noise flooding. Cryptology ePrint Archive, Paper 2025/1834, 2025.

[48] Marcel Keller, Emmanuela Orsini, and Peter Scholl. MASCOT: Faster malicious arithmetic secure computation with oblivious transfer. In Edgar R. Weippl, Stefan Katzenbeisser, Christopher Kruegel, Andrew C. Myers, and Shai Halevi, editors, *ACM CCS 2016*, pages 830–842. ACM Press, October 2016.

[49] Primus Labs. Primus-fhe: Fully homomorphic encryption framework. https://github.com/primus-labs/primus-fhe, 2024. Accessed: 2024-12-18.

[50] Yongwoo Lee, Daniele Micciancio, Andrey Kim, Rakyong Choi, Maxim Deryabin, Jieun Eom, and Donghoon Yoo. Efficient FHEW bootstrapping with small evaluation keys, and applications to threshold homomorphic encryption. In Carmit Hazay and Martijn Stam, editors, *EUROCRYPT 2023, Part III*, volume 14006 of *LNCS*, pages 227–256. Springer, Cham, April 2023.

[51] Zeyu Liu and Yunhao Wang. Amortized functional bootstrapping in less than 7 ms, with $\tilde{O}(1)$ polynomial multiplications. In Jian Guo and Ron Steinfeld, editors, *ASIACRYPT 2023, Part VI*, volume 14443 of *LNCS*, pages 101–132. Springer, Singapore, December 2023.

[52] Yingjie Lyu, Zengpeng Li, Hong-Sheng Zhou, and Xudong Deng. Threshold ecdsa in two rounds. In *Proceedings of the 2025 ACM SIGSAC Conference on Computer and Communications Security*, pages 2937–2950, 2025.

[53] Daniele Micciancio and Yuriy Polyakov. Bootstrapping in FHEW-like cryptosystems. In *Proceedings of the 9th on Workshop on Encrypted Computing & Applied Homomorphic Cryptography – WAHC'21*, pages 17–28. ACM, 2021.

[54] Christian Mouchet, Elliott Bertrand, and Jean-Pierre Hubaux. An efficient threshold access-structure for rlwe-based multiparty homomorphic encryption. *Journal of Cryptology*, 36(10), 2023.

[55] Christian Mouchet, Juan Ramón Troncoso-Pastoriza, Jean-Philippe Bossuat, and Jean-Pierre Hubaux. Multiparty homomorphic encryption from ring-learning-with-errors. *PoPETs*, 2021(4):291–311, October 2021.

[56] Michael Naehrig, Kristin Lauter, and Vinod Vaikuntanathan. Can homomorphic encryption be practical? In *Proceedings of the 3rd ACM Workshop on Cloud Computing Security Workshop – CCSW'11*, pages 113–124. ACM, 2011.

[57] Hiroki Okada and Tsuyoshi Takagi. Low communication threshold FHE from standard (module-)LWE. Cryptology ePrint Archive, Paper 2025/409, 2025.

[58] Emmanuela Orsini, Nigel P. Smart, and Frederik Vercauteren. Overdrive2k: Efficient secure MPC over $\mathbb{Z}_{2^k}$ from somewhat homomorphic encryption. In Stanislaw Jarecki, editor, *CT-RSA 2020*, volume 12006 of *LNCS*, pages 254–283. Springer, Cham, February 2020.

[59] Alain Passelègue and Damien Stehlé. Low communication threshold fully homomorphic encryption. In Kai-Min Chung and Yu Sasaki, editors, *ASIACRYPT 2024, Part I*, volume 15484 of *LNCS*, pages 297–329. Springer, Singapore, December 2024.

[60] Chris Peikert. An efficient and parallel gaussian sampler for lattices. In *Annual Cryptology Conference*, pages 80–97. Springer, 2010.

[61] Rachel Player. *Parameter selection in lattice-based cryptography*. PhD thesis, Royal Holloway, University of London, 2018.

[62] Antigoni Polychroniadou and Yifan Song. Constant-overhead unconditionally secure multiparty computation over binary fields. In Anne Canteaut and François-Xavier Standaert, editors, *EUROCRYPT 2021, Part II*, volume 12697 of *LNCS*, pages 812–841. Springer, Cham, October 2021.

[63] Srinivasan Raghuraman, Peter Rindal, and Titouan Tanguy. Expand-convolute codes for pseudorandom correlation generators from LPN. In Helena Handschuh and Anna Lysyanskaya, editors, *CRYPTO 2023, Part IV*, volume 14084 of *LNCS*, pages 602–632. Springer, Cham, August 2023.

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

[65] Dragos Rotaru, Nigel P. Smart, Titouan Tanguy, Frederik Vercauteren, and Tim Wood. Actively secure setup for SPDZ. *Journal of Cryptology*, 35(1):5, January 2022.

[66] Adi Shamir. How to share a secret. *Communications of the Association for Computing Machinery*, 22(11):612–613, November 1979.

[67] Nigel P. Smart and Michael Walter. Error-simulatable sanitization for TFHE and applications. Cryptology ePrint Archive, Paper 2025/275, 2025.

[68] Ruida Wang, Yundi Wen, Zhihao Li, Xianhui Lu, Benqiang Wei, Kun Liu, and Kunpeng Wang. Circuit bootstrapping: Faster and smaller. In Marc Joye and Gregor Leander, editors, *EUROCRYPT 2024, Part II*, volume 14652 of *LNCS*, pages 342–372. Springer, Cham, May 2024.

[69] Xiao Wang, Alex J. Malozemoff, and Jonathan Katz. EMP-toolkit: Efficient MultiParty computation toolkit. https://github.com/emp-toolkit, 2016.

[70] Chengkun Wei, Ruijing Yu, Yuan Fan, Wenzhi Chen, and Tianhao Wang. Securely sampling discrete gaussian noise for multi-party differential privacy. In Weizhi Meng, Christian Damsgaard Jensen, Cas Cremers, and Engin Kirda, editors, *ACM CCS 2023*, pages 2262–2276. ACM Press, November 2023.

[71] Kang Yang, Chenkai Weng, Xiao Lan, Jiang Zhang, and Xiao Wang. Ferret: Fast extension for correlated OT with small communication. In Jay Ligatti, Xinming Ou, Jonathan Katz, and Giovanni Vigna, editors, *ACM CCS 2020*, pages 1607–1626. ACM Press, November 2020.

[72] Zama. TFHE-rs: A Pure Rust Implementation of the TFHE Scheme for Boolean and Integer Arithmetics Over Encrypted Data, 2022. https://github.com/zama-ai/tfhe-rs.

[73] Ruiyu Zhu, Changchang Ding, and Yan Huang. Practical mpc+ fhe with applications in secure multi-partyneural network evaluation. *Cryptology ePrint Archive*, 2020.

A Discrete Gaussian

A.1 Discrete Gaussian Sampling

Definition 3. *(Discrete Gaussian). Let $\mu, s \in \mathbb{R}$ with $s > 0$. The discrete Gaussian distribution with location μ and scale s is denoted $\mathcal{N}_{\mathbb{Z}}(\mu, s^2)$. It is a probability distribution supported on the integers and defined by*

$$\forall x \in \mathbb{Z}, \quad \mathbb{P}_{X \leftarrow \mathcal{N}_{\mathbb{Z}}(\mu,s^2)}[X = x] = \frac{e^{-(x-\mu)^2/2s^2}}{\sum_{y \in \mathbb{Z}} e^{-(y-\mu)^2/2s^2}}.$$

Note that we consider $\mu = 0$; in this case, the distribution is symmetric and centered at 0. This is the natural discrete analogue of the continuous Gaussian (which has density $\frac{1}{\sqrt{2\pi\sigma^2}} e^{-x^2/2s^2}$ at $x \in \mathbb{R}$).

Extensive research [1,46,60] has been devoted to designing discrete Gaussian samplers that are both more accurate and more efficient. However, regardless of the specific sampling algorithm, finite-precision arithmetic imposes a fundamental lower bound on the achievable accuracy in practical implementations. As the scale parameter s decreases, the gap between the sampled distribution and the ideal discrete Gaussian becomes increasingly pronounced.

For example, when the target standard deviation is $\sigma = 0.1$, the theoretical probability of sampling the value 1 is approximately 2^{-70}. If the sampling algorithm operates with only 64 bits of precision, the probability of sampling 1 becomes effectively zero. As discussed in our protocol, this implies a practical lower bound on the standard deviation of discrete Gaussian noise, which depends on both the sampling technique and the numerical precision used in the implementation. Our implementation follows the CDT sampling method proposed in [60].

In real-world systems, due to the limitations of finite-precision arithmetic, the sampled noise distribution deviates from the ideal discrete Gaussian. This deviation primarily arises from tail truncation. Ideally, the discrete Gaussian assigns nonzero probability to every integer, regardless of its magnitude. However, in practical implementations, the probability $\Pr[X = x]$ is set to zero for all $|x| \geq \tau \cdot \sigma$, effectively truncating the tail; this introduces a truncation error. [1] analyzes the error introduced by the CDT algorithm relative to the ideal discrete Gaussian distribution. We leverage their results to quantify the deviation between our implemented noise distribution and the ideal distribution under our chosen parameters.

In floating-point arithmetic the relative error of a number $\bar{x}$, which is the floating-point approximation with a p-bits mantissa of a real number x, is defined as

$$\delta_{\mathrm{RE}}(x, \bar{x}) := \frac{|x - \bar{x}|}{|x|}.$$

[1] extends the notion of relative error to arbitrary pairs of distributions P and Q defined over the same support $S := \mathrm{Supp}(P) = \mathrm{Supp}(Q)$:

Definition 4. *The relative error between P and Q is defined as*

$$\delta_{\mathrm{RE}}(P, Q) := \max_{x \in S} \delta_{\mathrm{RE}}(P(x), Q(x)) = \max_{x \in S} \frac{|P(x) - Q(x)|}{P(x)}.$$

Moreover, the statistical distance between P and Q is bounded as $\Delta(P, Q) \leq \frac{1}{2} \delta_{\mathrm{RE}}(P, Q)$.

Let $\rho_s(x) = e^{-x^2/2s^2}$, then the discrete Gaussian distribution D_s for any integer x, as

$$D_s(x) := \frac{\rho_s(x)}{\rho_s(\mathbb{Z})}.$$

Define $\mathrm{cdf}_{s,c}$ as the cumulative distribution function (cdf) of $D_{s,c}$:

$$\mathrm{cdf}_{s,c}(x) := \sum_{i=-\infty}^{x} D_{s,c}(i).$$

Lemma 4 ([1]). *Let $\bar{D}_s$ be the output distribution of CDT sampling, where s is the standard deviation. Let $S := [\lfloor -\tau s \rfloor, \lceil \tau s \rceil] \cap \mathbb{Z}$ be the truncated support, and let p be the floating-point precision, i.e., the number of bits in the mantissa, then*

$$\delta_{\mathrm{RE}}(D_s, \bar{D}_s) \leq \frac{\rho_s(\mathbb{Z}) - \rho_s(S)}{2} + \max_{x \in S} \frac{\mathrm{cdf}_s(x)}{D_s(x)} \cdot 2^{-p}.$$

Note,

$$\begin{aligned}
&\frac{\rho_s(\mathbb{Z}) - \rho_s(S)}{2} + \max_{x \in S} \frac{\mathrm{cdf}_s(x)}{D_s(x)} \cdot 2^{-p} \\
&= \frac{\sum_{x \in \mathbb{Z}} \rho_s(x) - \sum_{x \in S} \rho_s(x)}{2} + \max_{x \in S} \frac{\sum_{i=-\infty}^{x} \frac{\rho_s(i)}{\rho_s(\mathbb{Z})}}{\frac{\rho_s(x)}{\rho_s(\mathbb{Z})}} \cdot 2^{-p} \\
&= \frac{\sum_{x=-\infty}^{\lfloor -\tau s \rfloor - 1} \rho_s(x) + \sum_{x=\lceil \tau s + 1 \rceil}^{\infty} \rho_s(x)}{2} + \max_{x \in S} \frac{\sum_{i=-\infty}^{x} \rho_s(i)}{\rho_s(x)} \cdot 2^{-p} \\
&\leq \sum_{x=\lceil \tau s + 1 \rceil}^{\infty} \rho_s(x) + \frac{1}{\rho_s(\lceil \tau s \rceil)} \cdot 2^{-p}
\end{aligned}$$

Let $T = \lceil \tau s \rceil + 1$. Because the function $\rho_s(x)$ is monotonically decreasing for $x \geq T$. We observe the ratio between successive terms:

$$\frac{\rho_s(T+1)}{\rho_s(T)} = \frac{e^{-(T+1)^2/2s^2}}{e^{-T^2/2s^2}} = e^{-[(T+1)^2 - T^2]/2s^2} = e^{-(2T+1)/2s^2}$$

Let us define:

$$r := e^{-(2T+1)/2s^2}, \quad \text{with } r \in (0, 1)$$

Note for all $k \geq 0$,

$$\rho_s(T+k+1)/\rho_s(T+k) = e^{-(2T+2k+1)/2s^2} \leq r.$$

So we have

$$\begin{aligned}\rho_s(T+k) \leq & \rho_s(T+k-1)\cdot r \\ \leq & \rho_s(T+k-2)\cdot r^2 \\ \leq & \rho_s(T)\cdot r^k.\end{aligned}$$

Therefore, the tail sum can be bounded by a geometric series:

$$\begin{aligned}&\sum_{x=\lceil \tau s+1\rceil}^{\infty} \rho_s(x) \leq \sum_{k=0}^{\infty} \rho_s(T)\cdot r^k = \rho_s(T)\cdot \sum_{k=0}^{\infty} r^k \\ &\leq \frac{\rho_s(T)}{1-r} = \frac{e^{-T^2/2s^2}}{1-e^{-(2T+1)/2s^2}}\end{aligned}$$

then,

$$\delta_{\mathrm{RE}}(D_s, \bar{D}_s) \leq \frac{e^{-T^2/2s^2}}{1-e^{-(2T+1)/2s^2}} + \frac{1}{\rho_s(T-1)}\cdot 2^{-p}$$

We adopt the same tailcut parameter as in [35], setting $\tau = 12$. The floating-point precision is set to $p = 256$ bits. Under this configuration, when the scale is set to $s = 0.7$, the relative error between the resulting discrete Gaussian distribution and the ideal distribution is about $2^{-136.8}$.

A.2 Truncation Error in Gaussian Additivity

Recall the additivity property of Gaussian distributions: Let $X \sim \mathcal{N}(\mu_X, \sigma_X)$ and $Y \sim \mathcal{N}(\mu_Y, \sigma_Y)$ be two independent Gaussian random variables. Then:

$$X + Y \sim \mathcal{N}(\mu_X + \mu_Y, \sigma_X + \sigma_Y).$$

The proof of the additivity property of the Gaussian distribution: Let $X \perp Y$ be independent random variables, and define $Z = X + Y$. Then,

$$\begin{aligned}\mathrm{Var}[Z] =& \mathrm{Var}[X+Y] \\ =& \mathbb{E}\left[(X+Y)^2\right] - (\mathbb{E}[X]+\mathbb{E}[Y])^2 \\ =& \mathbb{E}[X^2] + 2\mathbb{E}[XY] + \mathbb{E}[Y^2] \\ & - \left(\mathbb{E}[X]^2 + 2\mathbb{E}[X]\mathbb{E}[Y] + \mathbb{E}[Y]^2\right).\end{aligned}$$

Rearranging, we obtain

$$= \underbrace{\mathbb{E}[X^2] - \mathbb{E}[X]^2}_{\mathrm{Var}[X]} + \underbrace{\mathbb{E}[Y^2] - \mathbb{E}[Y]^2}_{\mathrm{Var}[Y]} + 2\left(\mathbb{E}[XY] - \mathbb{E}[X]\mathbb{E}[Y]\right).$$

Since X and Y are independent, the cross term vanishes, i.e., $\mathbb{E}[XY] = \mathbb{E}[X]\mathbb{E}[Y]$. Therefore,

$$\mathrm{Var}[X+Y] = \mathrm{Var}[X] + \mathrm{Var}[Y].$$

In practice, Gaussian sampling is performed with truncation. Note that truncation does not break the independence of Gaussian samples. Thus, for discrete Gaussians $X \sim \mathcal{N}(0, \sigma_X^2)$ and $Y \sim \mathcal{N}(0, \sigma_Y^2)$, suppose the actual sampled distributions are $X' \sim \mathcal{N}(0, \sigma_X'^2)$ and $Y' \sim \mathcal{N}(0, \sigma_Y'^2)$. Then it follows that

$$X' + Y' \sim \mathcal{N}(0, \sigma_X'^2 + \sigma_Y'^2).$$

Assumed the statistical distance $\Delta(X, X') = SD(X, X') \leq \delta_1$, $\Delta(Y, Y') = SD(Y, Y') \leq \delta_2$. The upper bounded of statistical distance between $X+Y$ and $X'+Y'$ is given by:

$$\begin{aligned}\Delta(X+Y, X'+Y') \leq & \Delta(X+Y, X'+Y) + \Delta(X'+Y, X'+Y') \\ \leq & \Delta(X, X') + \Delta(Y, Y') \leq \delta_1 + \delta_2.\end{aligned}$$

In the context of our protocol, let $X_1, \ldots, X_n$ be independent discrete Gaussian variables and let $X_1', \ldots, X_n'$ be their (possibly truncated) sampled counterparts such that for all $i \in [n]$, $\Delta(X_i, X_i') \leq \delta$. Then,

$$\Delta\left(\sum_{i=1}^{n} X_i,\ \sum_{i=1}^{n} X_i'\right) \ \leq\ \sum_{i=1}^{n} \Delta\big(X_i, X_i'\big) \ \leq\ n\delta.$$

Under our parameters, each party's sampled discrete Gaussian deviates from the ideal by at most $2^{-136.8}$ in statistical distance. Thus, the distance between the sum of their noises and the ideal discrete Gaussian is bounded by $n \cdot 2^{-136.8}$, where n is the number of parties.

B Parameters Selection

LWE/RLWE		**Key Switching**	
ℓ, N	2048	L_{ksk}	52
q, Q	9007199254614017	B_{ksk}	2
σ_{lwe}, σ_{rlwe}	12.96	σ_{ksk}	$11.56 \cdot 2^{52}$
Bootstrapping		**Other**	
L_{bk}	8	κ	$\ell \cdot \lceil \log_2 q \rceil$
B_{bk}	2^6	p	2^4
σ_{bs}	12.96	λ	128

Table 2: Parameters selection for 128-bit computational security and 30-bit statistical security.