Threshold OPRF from Threshold Additive HE

Animesh Singh[1], Sikhar Patranabis[2], and Debdeep Mukhopadhyay[1]

[1]Indian Institute of Technology, Kharagpur
[1]IBM Research, India

June 26, 2024

Abstract

An oblivious pseudorandom function (OPRF) is a two-party protocol in which a party holds an input and the other party holds the PRF key, such that the party having the input only learns the PRF output and the party having the key would not learn the input. Now, in a threshold oblivious pseudorandom function (TOPRF) protocol, a PRF key $\mathbf{K}$ is initially shared among T servers. A client can obtain a PRF value by interacting with $t(\leq T)$ servers but is unable to compute the same with up to $(t-1)$ servers. In this paper, we present a practically efficient homomorphic encryption (HE)-based post-quantum secure TOPRF protocol. Our proposed approach, which is based on a novel use of threshold HE, is agnostic of the underlying PRF and outperforms existing fully homomorphic encryption (FHE)-based approaches for TOPRF computation by several orders of magnitude in terms of running time. The FHE-based approaches require *bootstrapping*, a computationally extensive operation, and the primary bottleneck for evaluating large-depth circuits. Whereas, our proposed approach is based on a multi-party computation (MPC) protocol that uses a threshold additive HE scheme based on Regev's cryptosystem (J'ACM 2009) alternative to FHE-based approaches. Concretely, we show a novel replacement of bootstrapping required in traditional FHE schemes by a threshold additive HE-based interactive protocol that performs *masked* decryption followed by table *look-ups*, jointly performed by a group of servers holding secret shares of the HE decryption key.

Finally, We present a practical validation of our approach by realizing an AES-based TOPRF with an evaluation time of *less than 1 second* on consumer-grade server(s).

1 Introduction

Oblivious PRF. Oblivious Pseudorandom Functions (OPRF) [FIPR05, NR04] are interactive schemes between a server with a description of a PRF alongwith its key, and a user holding an input message. At the end of the interaction, the user learns the output of the PRF evaluated on the user's input message, whereas the server learns nothing about the user's input. OPRFs have numerous applications including private set intersection [AES03, HFH99, FIPR05, HL08, DCT10], password protocols [FK00, JKK14, JKKX17], searchable encryption [FIPR05, CJJ$^+$13, JJK$^+$13], file de-duplication [BKR13], pseudonymization [CL17], and more, and have served as a basis for other primitives such as Private Information Retrieval and Oblivious

Transfer. The general constructions of OPRFs proposed in the literature are either based on Oblivious Transfer (OT) or secure MPC. In this work, we proposed a novel approach to constructing a threshold OPRF using any symmetric key ciphers as PRF and evaluating the PRF circuit homomorphically using an HE scheme.

Threshold Oblivious PRF. A Threshold Oblivious Pseudorandom Function (TOPRF) [JKKX17, AMMM18] protocol requires the secret PRF key $\mathbf{k}$ to be distributed among T servers and for PRF evaluation on the client's input the client has to interact with at least $t(\leq T)$ number of servers to get the PRF output. Furthermore, a collusion of at most $(t-1)$ servers learns no information about the PRF input. TOPRF has many potential applications such as password-based threshold authentication protocol [AMMM18], password-protected secret sharing [JKK14, JKKX17], distributed password-authenticated symmetric encryption [DHL20, DHL22], threshold private set intersection [LZQ23] etc. TOPRF was introduced by Jarecki et al.[JKKX17], in which authors propose a simple TOPRF protocol called 2HashTDH and prove its security under the Gap Threshold One-More Diffie-Hellman (Gap-TOMDH) assumption in the random oracle model. They also show that Gap-TOMDH is hard in the generic group model. Several follow-up works [AMMM18, HAP18, DHL20, DHL22, LZQ23], have been proposed in this regard but all of these works are based on quantum unsafe assumptions such as variants of Diffie-Hellman. In this work, we present the first practically efficient quantum-safe TOPRF protocol based on LWE assumption [Reg09]. The construction of our proposed TOPRF protocol uses threshold additive HE for distributed evaluation of the underlying pseudorandom function.

Homomorphic Encryption (HE). Cloud computing technologies [Hay08, WVLY$^+$10] enable offloading complex computations on large datasets to third-party servers, but this raises security concerns, especially for sensitive data like medical records. Ensuring compliance and privacy necessitates securing data at rest, in transit, and during computation. While traditional encryption addresses data security at rest and in transit but fails to protect data during computation.

Homomorphic Encryption (HE) resolves this problem by enabling computation on encrypted data. An HE scheme allows a third party (e.g., cloud, service provider) to perform certain computable functions on encrypted data while preserving the functionality and format of the encrypted data. An additive HE scheme allows addition operation on the encrypted data. For sample messages m_1 and m_2, one can obtain $Enc(m_1 + m_2)$ by using $Enc(m_1)$ and $Enc(m_2)$ without knowing m_1 and m_2 explicitly, where $Enc(\cdot)$ denotes the encryption function. Goldwasser-Micali (GM) [GM19] proposed the first probabilistic public key encryption (PKE) scheme which is additively homomorphic but only for binary numbers. After a few years, Benaloh [Ben94] proposed another additively homomorphic encryption (AHE) scheme by extending the GM cryptosystem to encrypt the message as a block instead of bit by bit. Then Paillier [Pai99] introduced a novel probabilistic AHE scheme based on the composite residuosity problem [Jag12], which is similar to quadratic and higher residuosity problems [ZMI88] used in GM and Benaloh cryptosystems. A detailed survey on several AHE schemes can be found in [AAUC18]. But all these AHE schemes are not resistant to quantum attacks until in 2009 Oded Regev [Reg09] introduced a lattice-based HE scheme that is additively homomorphic but can be extended to support a fully homomorphic capability with *bootstrapping*, a novel technique proposed by Gentry [Gen09]. After this breakthrough result by Gentry, a significant body of research works [SS10, CNT12, DM15a, CGGI20, CGBH$^+$18, FSK$^+$21] were proposed to focus on

Table 1: Comparison between our basic OPRF structure with our final optimized TOPRF protocol.

Structure	**Flavor**	**Comp. time**	**Comm. bandwidth**
Naïve FHE-based OPRF	Non-distributed	470 *secs*	0.62 MB
Our optimized TOPRF	Threshold (2-party)	0.96 *sec*	5.12 MB

building practically efficient fully homomorphic encryption (FHE) systems.

Multi-key FHE. Another actively studied cryptographic primitive for secure computation is multi-key FHE (MKFHE) which enables computations to be performed by multiple parties simultaneously. Multi-party computation (MPC) provides promising solutions for secure computation with efficient techniques and performances. The first practical MPC primitives have been proposed in [GBOW88, Yao86], and since then it has been widely studied in the cryptography community. In this approach, two or more parties participate in an interactive protocol to compute a function on their private inputs, where only the function's output is revealed to the parties. In recent years, MKFHE has been intensively used to achieve a practical round-optimal secure MPC [BJMS18, MW16] protocols. An MKFHE scheme allows multiple users to participate in a commonly outsourced computation over a cloud server. López-Alt et al. [LATV12] first proposed a generalized notion of MKFHE scheme, capable of performing arithmetic operations on ciphertexts encrypted under different keys. Chen et al. [CCS19] extended this work to achieve a low-latency MKFHE scheme over Torus-FHE (TFHE) [CGGI20], which is based on Learning with error (LWE) assumption [Reg09] and its ring variant (RLWE) [LPR10]. A recent work by Klemsa et al. [KÖA23] proposed an efficient MKFHE scheme with low noise growth and linear complexity achieving a faster bootstrapping mechanism.

Threshold (Multi-key) FHE. While FHE resolves the crucial problem of computation on encrypted data, the decryption key should be stored securely to get any real benefit out of it. In traditional threshold cryptographic approaches [Sha79, DF90, DSDFY94] the decryption key is shared among multiple servers (say T) to avoid a "single point of failure" and a threshold number of them (say $t \leq T$) can collaborate to recompute the decryption key. However, this defies the purpose as a single compromise at the decryption server, during a key reconstruction, would reveal the key entirely. An ideal solution must have the decryption key distributed *at all time*. Generally, this is achieved by a Threshold FHE scheme [AJLA$^+$12, MW16, BGG$^+$18, CCK23], where the decryption is performed jointly by any threshold number of parties without reconstructing the key at any one place. In particular, in a t-out-of-T threshold FHE scheme, parties compute partial decryption with their key shares and send them over to the decryptor, who, once obtains t such partial decryptions in total, combines them to get the message.

In an MKFHE scheme, two or more parties (say $\mathcal{T}$) want to jointly compute a function in an outsourced fashion. Each of the involved parties encrypts their input with their keys and outsources the computation over a remote server. During decryption, all $\mathcal{T}$ parties compute partial decryption of the resultant ciphertext and broadcast among themselves. After combining $\mathcal{T}$ partial decryptions, each party can compute the output of the computation. Hence, the decryption procedure in a multi-key setting inherently follows a threshold construction with a $\mathcal{T}$-out-of-$\mathcal{T}$ threshold structure.

1.1 Our Contribution

Now, we briefly discuss our main theoretical contributions and the additional constructions required to build an efficient threshold oblivious PRF (TOPRF). Our TOPRF can be constructed using any symmetric-key cipher, but we use AES to demonstrate our TOPRF construction.

AES Block Cipher. AES [DR99] is a NIST standardized widely used symmetric-key block cipher that operates on fixed-size blocks of data, i.e., it takes fixed-sized input (e.g., 128 bits) and key (e.g., 128, 192, or 256 bits) and outputs a fixed-sized ciphertext. AES employs a substitution-permutation network (SPN) structure, consisting of multiple rounds of substitution and permutation operations. In each round, AES applies four main operations: byte substitution using a nonlinear substitution box (S-box), shifting rows, mixing columns using matrix operations, and adding a round key derived from the encryption key. The number of rounds varies depending on the key size: 10 rounds for a 128-bit key, 12 rounds for a 192-bit key, and 14 rounds for a 256-bit key.

Homomorphic Evaluation of AES. Our work uses the Bristol format (designed by the authors of SCALE-MAMBA [HHNZ19]) of AES circuits, which are represented with Boolean gates consisting of XORs, ANDs, and NOTs. The Bristol formats are Boolean representations of circuits which are considered to be FHE [BGG+18, Gen09, BGV14] and MPC [WRK17, BELO16] friendly. In MPC protocols, the Garbled circuits [KS08] offer better efficiency compared to secret-sharing-based implementations [MW16, LATV12] due to the free XOR and constant round operations provided by Garbled circuits. Whereas, our primary focus is on the HE-based evaluation of AES circuit on the encrypted inputs. We start our OPRF construction with a two-party protocol in which one party (client) encrypts its input using an FHE scheme and the other party (server) homomorphically evaluates the PRF circuit using its key and encrypted client's input. In this case, we use TFHE [CGGI20], a widely used FHE scheme for Boolean circuits, that allows evaluating an arbitrary binary gate on encrypted data followed by a low-latency bootstrapping operation. Hence, the TFHE scheme provides better computational efficiency for Boolean circuits ($\mathcal{C} : \{0,1\}^\star \to \{0,1\}$) compared to other FHE schemes like BGV [BGV14], BFV [Bra12] and CKKS [CKKS17].

Now, our initial OPRF construction is achieved as follows, the client encrypts AES message ($\mathbf{aes}_{msg}$) and the server encrypts its AES key ($\mathbf{aes}_{key}$) with TFHE public key (PK) and the server having sufficient computational resources evaluates the AES circuit homomorphically. After the homomorphic evaluation of AES on the encrypted data, it sends back the AES output in an encrypted form to the client; who then can decrypt with TFHE secret key ($\mathbf{SK}$) and retrieve the output of AES evaluation. Using TFHE the homomorphic evaluation of AES-128 (i.e., 128-bit $\mathbf{aes}_{key}$ and $\mathbf{aes}_{msg}$) takes approximately 470 seconds or, around 8 minutes in a standard computing platform. Such a huge evaluation time occurred due to the large computation overhead of the bootstrapping operation, required after every Boolean gate (except NOT gate) evaluation to reduce the noise in the resultant ciphertext, helping to retain correctness in further computation. Each bootstrapping operation takes around 13 milliseconds and the Bristol format of AES-128 has roughly $32,000$ Boolean gates, thus the homomorphic evaluation incurs a huge execution time of 470 seconds. This situation becomes even much worse (taking more than an hour) in the case of multi-key TFHE, in which bootstrapping takes around 0.27 seconds considering only 2 parties. With this motivation, we propose a novel approach to evaluate any Boolean circuit in which

Table 2: LUT for Boolean $\otimes$ for the masked bit $\{r_0^i, r_1^i\}$

$LUT_{\otimes}^{r_0^i, r_1^i}$	$\mathsf{Enc}_{pk}\left((0 \oplus r_0^i) \otimes (0 \oplus r_1^i)\right)$	$\mathsf{Enc}_{pk}\left((0 \oplus r_0^i) \otimes (1 \oplus r_1^i)\right)$
	$\mathsf{Enc}_{pk}\left((1 \oplus r_0^i) \otimes (0 \oplus r_1^i)\right)$	$\mathsf{Enc}_{pk}\left((1 \oplus r_0^i) \otimes (1 \oplus r_1^i)\right)$

no bootstrapping is required at any point in the evaluation process using a threshold HE scheme and achieves an evaluation time of AES-128 in less than 1 second. Note that, in our construction, we perform *no bootstrapping*, and therefore we require only an additively homomorphic encryption (HE) scheme. In particular, we showcase an efficient construction that uses an additive HE with an MPC protocol to achieve fully homomorphic capability that can homomorphically evaluate any Boolean circuit without performing bootstrapping.

Our TOPRF using Threshold Additive HE. Here, we present a brief discussion on our proposed TOPRF construction using threshold additive HE. Despite evaluating the AES on a single high-end server, we use T number of public servers, relatively lower-end computational resources (ref. Section 3.2 for a detailed discussion). A trusted dealer samples a public key and secret key pair $(\mathsf{pk}, \mathbf{sk})$ for an additive HE scheme. In particular, we use homomorphic encryption from Regev's Cryptosystem [Reg09] which is additively homomorphic and based on LWE assumption (ref. Appendix A for a formal definition of LWE). Now, the secret key $\mathbf{sk}$ is distributed among the T servers $(S_1, \ldots, S_T)$ such that the ℓ^{th} server gets the secret share $\mathbf{sk}_\ell$, $\forall \ell \in \{1, \ldots, T\}$ and the public key pk is provided to the client. Apart from the secret shares, the trusted dealer also samples L number of random-bit pairs $\left(\{r_0^0, r_1^0\}, \ldots \{r_0^{L-1}, r_1^{L-1}\}\right) \in \{0,1\}^{2L}$, where L be the total number of Boolean gates (XORs and ANDs) in the AES-based PRF circuit. For each $\{r_0^i, r_1^i\}_{i \in \{0, \ldots, L\}}$ pair the trusted dealer constructs a look-up table (LUT) based on the Boolean gate at the i^{th} iteration of the PRF circuit. Table 2 shows a LUT structure for "$\otimes$" (which can be either Boolean XOR or AND), the Boolean operation at the i^{th} stage while evaluating the PRF circuit. Here, $\oplus$ denotes XOR operation on unencrypted/clear data, and $\mathsf{Enc}_{\mathsf{pk}}(\cdot)$ denotes LWE encryption under the public key pk. Now, the trusted dealer sends the random bits in encrypted form, i.e., $\left(\{\mathbf{cr}_0^0, \mathbf{cr}_1^0\}, \ldots, \{\mathbf{cr}_0^{L-1}, \mathbf{cr}_1^{L-1}\}\right)$, where $\mathbf{cr}_j^i = \mathsf{Enc}_{\mathsf{pk}}(r_j^i)$, $\forall j \in \{0, 1\}$ to one of the T servers, say S_1 and L LUTs to the other $(T-1)$ servers, i.e., $(S_2, \ldots, S_T)$.

Now, at the very beginning of our PRF evaluation, the client encrypts its input $\mathbf{aes}_{msg}$ using pk and sends the ciphertexts to S_1; the server also encrypts the PRF key $\mathbf{aes}_{key}$ using pk. As mentioned earlier, we use the Bristol representation of the PRF circuit, which consists of only AND, XOR, and NOT gates. The PRF circuit is now homomorphically evaluated by the T servers in the following way – Let us assume at the i^{th} iteration the following operation is to be performed, $\mathbf{ct}^i = \mathsf{HomAND}(\mathbf{ct}_0^i, \mathbf{ct}_1^i)$, which denotes homomorphic AND gate over encrypted data/ciphertexts $\mathbf{ct}_0^i, \mathbf{ct}_1^i$. But in an FHE scheme based on TFHE, the homomorphic AND requires bootstrapping, which is costly and incurs significant overhead in circuit evaluation. To avoid bootstrapping operations, the server S_1 masks the input ciphertexts $(\mathbf{ct}_0^i, \mathbf{ct}_1^i)$ with the encryption of random bits, i.e., with $(\mathbf{cr}_0^i, \mathbf{cr}_1^i)$ by performing homomorphic XOR operation as, $\hat{\mathbf{ct}}_j^i = \mathsf{HomXOR}_{wb}(\mathbf{ct}_j^i, \mathbf{cr}_j^i)$, $\forall j \in \{0, 1\}$, here $\mathsf{HomXOR}_{wb}(\cdot)$ represents homomorphic XOR operation without bootstrapping. Note that, in TFHE at most one Boolean gate can be evaluated without performing a bootstrapping operation (for more detailed analysis ref. Appendix A.2), because a bootstrapping operation [GHS12] is nothing but homomorphic decryption followed by adding a fresh noise into the evaluated ciphertext. Next, S_1 computes the partial decryption of $\hat{\mathbf{ct}}_j^i = (\hat{b}_j^i, \hat{\mathbf{a}}_j^i)$, $\forall j \in \{0,1\}$ with its secret share $\mathbf{sk}_1$ as $\hat{\gamma}_j^i = \hat{b}_j^i - \langle \hat{\mathbf{a}}_j^i, \mathbf{sk}_1 \rangle + e_{sm}^{i,j}$, here $\langle \cdot, \cdot \rangle$ denotes vector-dot product and $e_{sm}^{i,j}$ be a smudging noise [BGG$^+$18, CSS$^+$22, BS23], required to statistically hide the encryption noise. These partial decryptions $\left(\hat{\gamma}_j^i, \hat{\mathbf{a}}_j^i\right)$, $\forall j \in \{0,1\}$ are then sent to other $(T-1)$ servers for performing threshold decryption. After performing the threshold decryption operation, the servers $(S_2, \ldots, S_T)$ retrieve the underlying plaintexts of $\hat{\mathbf{ct}}_j^i$, i.e., $\hat{m}_j^i = m_j^i \oplus r_j^i$, where m_j^i be the plaintext of $\mathbf{ct}_j^i$, $\forall j \in \{0,1\}$. One of these $(T-1)$ servers then queries the i^{th} LUT as $\mathbf{ct}_i^\star = LUT_\wedge^{r_0^i, r_1^i}(\hat{m}_0^i, \hat{m}_1^i)$, here $\wedge$ denotes AND operation. $\mathbf{ct}_i^\star$ is then sent back to S_1 as response and S_1 sets $\mathbf{ct}^i = \mathbf{ct}_i^\star$, which finally completes the evaluation of the target operation, i.e., $\mathsf{HomAND}(\cdot)$ at the i^{th} iteration. In Figure 2 we present a pictorial overview of the execution phase of our TOPRF protocol for a single Boolean operation.

Observe that to compute $\mathsf{HomAND}(\mathbf{ct}_0^i, \mathbf{ct}_1^i)$ no bootstrapping is performed; instead, we incur only two rounds of communication alongwith a few lightweight computations at both the server's end. During communication ciphertexts are transmitted, which have a size of approximately 2 KB; hence to transmit 3 ciphertexts (2 from S_1 and 1 to S_1) we incur a total of around 6 KB of data for one homomorphic Boolean computation. Considering 1 GB/sec LAN connectivity between the public servers, it only requires 6 μs of time which is significantly lower compared to one bootstrapping cost of 13 ms, considering the single-key version of FHE. While the communication cost may appear negligible, it will result in significant overhead when considering the entire AES-128 circuit, comprising approximately $32,000$ Boolean gates. Therefore, to enhance the efficiency of our TOPRF construction, we provide several optimization techniques which include reducing communication overhead by approximately 99% between the public servers. In the subsequent sections, we gradually discuss our different optimization tools to construct the final TOPRF architecture that can be evaluated in less than 1 sec using AES as the baseline PRF. Table 1 compares the basic FHE-based OPRF framework with our optimized TOPRF protocol (considering $T = 2$ parties), demonstrating a substantial reduction in PRF evaluation time with minimal communication overhead.

1.2 Related Works

In this section, we discuss some more related works on OPRF. Naor and Reingold [NR04] first showed an interactive and oblivious evaluation of a PRF, where a client with input x obtains $PRF_K(x)$ for a function $PRF_K(\cdot)$ that is contributed by a server. Freedman et al. [FIPR05] later denoted such two-party protocol as an oblivious pseudorandom function OPRF and demonstrated how OPRFs were useful for obliviously searching a database. Several years later, Hazay and Lindell [HL10] made a noteworthy discovery that shows that OPRF and private set intersection (PSI) are closely related, elucidating how their inherent obliviousness property facilitates the design of protocols aimed at safeguarding confidential data such as passwords, search queries, identities, digital footprints, etc. Over the preceding decade, OPRFs have emerged as a pivotal component in various applications including oblivious keyword search [FIPR05, KKRT16], PSI [KS08, JL09, JL10], password-protected secret sharing (PPSS/TPASS) [JKK14, BJSL11, JKKX17, BFH$^+$20, DHL20], private information retrieval (PIR) [FIPR05], password-authenticated key exchange (PAKE) [JKK14, JKX18], single sign-on (SSO) with privacy [BFH$^+$20], cloud key management [JKR19], de-duplication systems [CDCGS19], secure pattern matching [FHV18] and "untraceable" contact tracing [KRS$^+$19]. We refer to the work [CHL22] for a thorough discussion on several OPRF constructions proposed in the literature. Everspaugh et al. [ECS$^+$15] introduces a partially oblivious PRF (POPRF) in the context of the password hardening system. It extends OPRF functionality to include a public input/tag *tag* for the PRF evaluation. A client learns the $PRF_K(tag, x)$ output, where *tag* is known by both server and client, and the private input x remains hidden. But this POPRF construction relies on bilinear pairings which has inefficient performance. After that Tyagi et al.[TCR$^+$22] introduce an efficient POPRF construction that combines aspects of the 2HashDH OPRF of Jarecki et al.[JKK14] with the Dodis-Yampolskiy (DY) verifiable random function [DY05]. Jarecki et al.[JKR18] propose an efficient threshold POPRF construction based on Diffie-Hellman (DH) that can accommodate very efficient elliptic curve groups and present a range of applications for building more secure and reliable key management systems. All of the above-mentioned OPRF constructions are based on quantum unsafe assumptions and the most efficient quantum-safe OPRF construction was proposed by Albrecht et al.[ADDG23], where the authors constructed a POPRF from lattice assumptions and low-depth (weak) PRF [BIP$^+$18]. However, our present work provides an efficient threshold PRF (TOPRF) protocol, which can evaluate any arbitrary depth PRF circuit obliviously in a distributed fashion while relying on LWE, quantum-safe assumption.

2 Preliminaries & Background

Here, we introduce some notations used throughout this paper and present some preliminary background on the cryptographic primitives used in this work.

2.1 Notations

We use lower-case bold letter "**a**" to denote a vector and upper-case bold letter "**A**" to denote a matrix. We write $x \overset{\$}{\leftarrow} \chi$ to represent that an element x is sampled uniformly randomly from a set/distribution χ and with $a \leftarrow b$ we denote the value of b is assigned to a. For $a, b \in \mathbb{Z}$ such that $a, b \geq 0$, we denote by $[a]$ and $[a, b]$ the set of integers lying between 1 and a (both inclusive), and the set of integers lying between a and b (both inclusive). We use $\langle \cdot, \cdot \rangle$ to denote vector-dot product operation. We refer to $\lambda \in \mathbb{N}$ as the security parameter and denote by $poly(\lambda)$ and $negl(\lambda)$ any generic (unspecified) polynomial function and negligible function in λ, respectively.[1] We use (t, T) as a shorthand notation for t-out-of-T structure.

2.2 Threshold Pseudorandom Function (TOPRF)

An $(\mathcal{X}, \mathcal{R})$-threshold oblivious pseudorandom function TOPRF [AMMM18] is a tuple of four PPT algorithms (TOPRF.Setup, TOPRF.Encode, TOPRF.Eval, TOPRF.Combine) that satisfies the consistency property below.

- $(\{K\}, pp) \leftarrow \mathsf{TOPRF.Setup}(1^\kappa, n, t)$. It generates n secret key shares $K_1, K_2, \ldots, K_n$ and public parameters pp. Share K_i is given to i^{th} party.

[1]Note that, a function $f : \mathbb{N} \rightarrow \mathbb{N}$ is said to be negligible in λ if for every positive polynomial p, $f(\lambda) < 1/p(\lambda)$ when λ is sufficiently large.

- $c \leftarrow \mathsf{TOPRF.Encode}(x, \rho)$. It generates an encoding c of $x \in \mathcal{X}$ using randomness $\rho \in \mathcal{R}$.
- $z_i \leftarrow \mathsf{TOPRF.Eval}(K_i, c)$. It generates shares of TOPRF value from an encoding. Party i computes the i^{th} share z_i from c by running $\mathsf{TOPRF.Eval}(\cdot)$ with K_i and c.
- $h/\perp \leftarrow \mathsf{TOPRF.Combine}(x, \{(i, z_i)\}_{i \in \mathcal{S}}, \rho)$. It combines the shares received from parties in the set $\mathcal{S}$ using randomness ρ to generate a value h. If the algorithm fails, its output is denoted by $\perp$.

Consistency: For all $\kappa \in \mathbb{N}$, any $n, t \in \mathbb{N}$ such that $t \leq n$, all $(\{K_i\}_{i\in[n]}, pp)$ generated by $\mathsf{TOPRF.Setup}(1^\kappa, n, t)$ any value $x \in \mathcal{X}$, any randomness $\rho, \rho' \in \mathcal{R}$, and any two sets $\mathcal{S}, \mathcal{S}' \subseteq [n]$ of size at least t, if $c \leftarrow \mathsf{TOPRF.Encode}(x, \rho)$, $c' \leftarrow \mathsf{TOPRF.Encode}(x, \rho')$, $z_i \leftarrow \mathsf{TOPRF.Eval}(K_i, c)$ for $i \in \mathcal{S}$, and $z'_j \leftarrow \mathsf{TOPRF.Eval}(K_j, c')$ for $j \in \mathcal{S}'$, then $\mathsf{TOPRF.Combine}(x, \{(i, z_i)\}_{i\in\mathcal{S}}, \rho) = \mathsf{TOPRF.Combine}(x, \{(j, z'_j)\}_{j\in\mathcal{S}'}, \rho') \neq \perp$.

Security Properties: A TOPRF protocol satisfies two properties, *unpredictability* and *obliviousness*. *Unpredictability* ensures that it must be difficult to predict the TOPRF output on a random value, and the *obliviousness* property ensures that the random value itself is hard to guess even if the TOPRF output is available. For a more detailed and formal description of a TOPRF security properties we refer to PASTA by Agrawal et al. [AMMM18].

2.3 Additive Homomorphic Encryption (AHE)

In this work, we use an Additive Homomorphic Encryption (AHE) scheme by extending Regev's cryptosystem [Reg09], which is defined as follows– an AHE scheme is a tuple of five PPT (Probabilistic Polynomial Time) algorithms ($\mathsf{AHE.Setup}$, $\mathsf{AHE.KeyGen}$, $\mathsf{AHE.Enc}$, $\mathsf{AHE.Dec}$, $\mathsf{AHE.Eval}$).

- $\mathsf{params}_a \leftarrow \mathsf{AHE.Setup}(1^\lambda)$: Given a security parameter λ, it sets lattice dimension n, a modulus q, key space $\mathcal{K}$, and noise distribution $\mathcal{E}$ and returns a public parameter $\mathsf{params}_a = \{n, q, \mathcal{K}, \mathcal{E}\}$.
- $(\mathbf{sk}, \mathsf{pk}) \leftarrow \mathsf{AHE.KeyGen}(\mathsf{params}_a)$: It takes the public parameter params_a as input and sample the secret key $\mathbf{sk} \overset{\$}{\leftarrow} \mathcal{K}^n$. Generate a set of public keys $\mathsf{pk} = \{\mathsf{pk}_1, \ldots, \mathsf{pk}_z\}$, where each $\mathsf{pk}_i = \langle \mathbf{a}_i, \mathbf{sk} \rangle + e_i$, with $\mathbf{a}_i \overset{\$}{\leftarrow} U(\mathbb{Z}_q^n)$ (uniform distribution over $\mathbb{Z}_q^n$) and $e_i \overset{\$}{\leftarrow} \mathcal{E}$ be the encryption noise, $\forall i \in [z]$.
- $\mathbf{ct} \leftarrow \mathsf{AHE.Enc}(m, \mathsf{pk})$: Given a message bit $m \in \{0, 1\}$ and the public key set pk, returns a ciphertext $\mathbf{ct} = (b, \mathbf{a})$, where, $b = \sum_{j\in\mathcal{H}} \mathsf{pk}_j + \lfloor \frac{q}{2} \rfloor \cdot m \pmod{q} \in \mathbb{Z}_q^{n+1}$, and $\mathbf{a} = \sum_{j\in\mathcal{H}} \mathbf{a}_j$, where $\mathcal{H}$ be a random subset of pk.
- $m^\star \leftarrow \mathsf{AHE.Dec}(\mathbf{ct}^\star, \mathbf{sk})$: Given an evaluated ciphertext $\mathbf{ct}^\star$ and the secret key $\mathbf{sk}$, returns a message bit $m^\star = \lfloor (\frac{2}{q})[\langle \mathbf{ct}^\star, \mathbf{sk} \rangle]_q \rfloor \pmod{2} \in \{0, 1\}$.
- $\mathbf{ct}^\star \leftarrow \mathsf{AHE.Add}(\mathbf{ct}_1, \mathbf{ct}_2, \mathsf{pk})$: Given two ciphertexts $\mathbf{ct}_1 = \mathsf{AHE.Enc}(m_1, \mathsf{pk})$, and $\mathbf{ct}_2 = \mathsf{AHE.Enc}(m_2, \mathsf{pk})$, the public key pk, evaluate $\mathbf{ct}^\star = (\mathbf{ct}_1 + \mathbf{ct}_2) \bmod q$, such

that

$$\begin{aligned}\langle \mathbf{ct}^\star, \mathbf{sk}\rangle =& \langle \mathbf{ct}_1, \mathbf{sk}\rangle + \langle \mathbf{ct}_2, \mathbf{sk}\rangle \bmod q \\ =& \lfloor \frac{q}{2} \rfloor \cdot (m_1 + m_2) + e^\star \bmod q\end{aligned}$$

where, $e^\star = (e_1 + e_2)$, e_1, e_2 are the encryption noise in $\mathbf{ct}_1, \mathbf{ct}_2$ respectively. Now, according to the Regev's encryption if $e^\star < \frac{1}{2}\lfloor \frac{q}{2} \rfloor$, $(m_1 + m_2) \bmod 2 \leftarrow \mathsf{AHE.Dec}(\mathbf{ct}^\star, \mathbf{sk})$. Therefore, homomorphic addition in Regev's cryptosystem actually returns addition modulo 2, in other words, Boolean XOR of m_1, m_2.

2.4 Fully Homomorphic Encryption (FHE)

A Fully Homomorphic Encryption (FHE) scheme is a tuple of five PPT (Probabilistic Polynomial Time) algorithms ($\mathsf{FHE.Setup}$, $\mathsf{FHE.KeyGen}$, $\mathsf{FHE.Enc}$, $\mathsf{FHE.Dec}$, $\mathsf{FHE.Eval}$). These PPT algorithms are similar to that of an AHE scheme; except it supports an additional operation called "bootstrapping" [GHS12] after $\mathsf{FHE.Eval}$ that allows evaluation of circuits to arbitrary depth.

- $\mathsf{params}_f \leftarrow \mathsf{FHE.Setup}(1^\lambda)$: Given a security parameter λ, this function returns a public parameter params_f.
- $(\mathbf{SK}, \mathsf{PK}) \leftarrow \mathsf{FHE.KeyGen}(\mathsf{params}_f)$: It takes the public parameter params_f as input and generates a secret key $\mathbf{SK}$ and the corresponding public key PK.
- $\mathbf{CT} \leftarrow \mathsf{FHE.Enc}(m, \mathsf{PK})$: Given a message bit $m \in \{0,1\}$ and the public key PK, returns a ciphertext $\mathbf{CT} \in \{0,1\}^\star$.
- $m \leftarrow \mathsf{FHE.Dec}(\mathbf{CT}^\star, \mathbf{SK})$: Given an evaluated ciphertext $\mathbf{CT}^\star$ and the secret key $\mathbf{SK}$, returns a message bit $m \in \{0,1\}$.
- $\mathbf{CT}^\star \leftarrow \mathsf{FHE.Eval}(\{\mathbf{CT}_i\}_{i\in\ell}, \mathsf{PK}, \mathcal{C})$: Given ℓ ciphertexts $\{\mathbf{CT}_i\}_{i\in\ell}$, the public key PK and a circuit $\mathcal{C} : \{0,1\}^\ell \to \{0,1\}^\star$. Evaluate the input circuit $\mathcal{C}$ over the input ciphertexts and return an evaluated ciphertext $\mathbf{CT}^\star$.

IND-CPA security: For any $(\mathsf{PK}, \mathbf{SK}) \leftarrow \mathsf{FHE.KeyGen}(1^\lambda)$, for any messages $m_0, m_1 \in \mathcal{M}$, and for any probabilistic polynomial-time (PPT) adversary $\mathcal{A}$, letting $\mathbf{CT}_0 \leftarrow \mathsf{FHE.Enc}(m_0, \mathsf{PK})$ and $\mathbf{CT}_1 \leftarrow \mathsf{FHE.Enc}(m_1, \mathsf{PK})$,

$$\begin{aligned}|\Pr[\mathcal{A}(\mathsf{PK}, m_0, m_1, \mathbf{CT}_0) = 1] - \Pr[\mathcal{A}(\mathsf{PK}, m_0, m_1, \mathbf{CT}_1)] = 1| \\ \leq negl(\lambda).\end{aligned}$$

Correctness: The homomorphism of an FHE scheme ensures correctness. For any (Boolean) function $f : \{0,1\}^\ell \to \{0,1\}^\star \in \mathcal{F}$ and any sequence of ℓ messages $m_1, \ldots, m_\ell$, letting $(\mathsf{PK}, \mathbf{SK}) \leftarrow \mathsf{FHE.KeyGen}(1^\lambda)$, and $\mathbf{CT}_i \leftarrow \mathsf{FHE.Enc}(m_i, \mathsf{PK})$ for each $i \in [\ell]$, we have the following,

$$\begin{aligned}\Pr[\mathsf{FHE.Dec}(\mathsf{FHE.Eval}(\mathbf{CT}_1, \ldots \mathbf{CT}_\ell, \mathsf{PK}, f)) \neq f(m_1, \ldots, m_\ell)] \\ \leq negl(\lambda)\end{aligned}$$

Compactness: There exists a polynomial $poly(\lambda)$ such that, for any (Boolean) function $f : \{0,1\}^\ell \to \{0,1\}^\star \in \mathcal{F}$ and any sequence of ℓ messages $m_1, \ldots, m_\ell$, letting $(\mathsf{PK}, \mathbf{SK}) \leftarrow \mathsf{FHE.KeyGen}(1^\lambda)$, and $\mathbf{CT}_i \leftarrow \mathsf{FHE.Enc}(m_i, \mathsf{PK})$ for each $i \in [\ell]$, we have

$$|\mathbf{ct}^\star \leftarrow \mathsf{FHE.Eval}(\mathbf{CT}_1, \ldots \mathbf{CT}_\ell, \mathsf{PK}, f))| \leq poly(\lambda),$$

where $poly(\lambda)$ is independent of size of f and the number ℓ of inputs.

Although we discuss the IND-CPA security, compactness, and correctness properties for an FHE scheme, we can argue the same for an AHE scheme as defined in Section 2.3. Most of the FHE schemes are based on some lattice-based hard problems and among those Learning with Errors (LWE) [Reg09] and its ring variant Ring-LWE (RLWE) [LPR10] are widely used to construct efficient FHE schemes. The encryption procedure in an FHE generally uses LWE or RLWE assumptions, while the bootstrapping procedure requires an additional RGSW assumption. All these different types of lattice-based hardness assumptions are formally defined and discussed in Appendix A.1.

2.5 Our Threshold HE Scheme

In this work, we construct our TOPRF using a threshold HE scheme. As mentioned in the introduction that our TOPRF construction only requires a threshold AHE scheme rather than an FHE. Therefore, in this section, we formally define an LWE-based threshold AHE scheme that follows the Regev cryptosystem [Reg09]. However, we argue that the thresholdization procedure can trivially be extended to an FHE scheme because both the AHE and FHE have a similar encryption and decryption procedure; only the homomorphic evaluation is different due to the bootstrapping, which does not influence the threshold decryption operation. We define our threshold AHE (ThAHE) scheme below, where the key-generation, encryption, decryption, and evaluation operations are similar to the AHE scheme defined earlier.

- $\mathsf{params} \leftarrow \mathsf{ThAHE.Setup}(1^\lambda)$: Given a security parameter λ, it sets lattice dimension $n = n(\lambda)$, a modulus $q = q(\lambda)$, key space $\mathcal{K} = \{0,1\}$, and noise distribution $\mathcal{G}_\alpha$, a Gaussian distribution with standard deviation α. Returns a public parameter $\mathsf{params} = \{n, q, \mathcal{K}, \mathcal{G}_\alpha\}$.
- $(\mathbf{sk}, \mathsf{pk}) \leftarrow \mathsf{ThAHE.KeyGen}(\mathsf{params})$: It takes the public parameter params as input and sample the secret key $\mathbf{sk} \xleftarrow{\$} \mathcal{K}^n$. Generate a set of public keys $\mathsf{pk} = \{\mathsf{pk}_1, \ldots, \mathsf{pk}_z\}$, where each $\mathsf{pk}_i = \langle \mathbf{a}_i, \mathbf{sk} \rangle + e_i$, with $\mathbf{a}_i \xleftarrow{\$} U(\mathbb{Z}_q^n)$ (uniform distribution over $\mathbb{Z}_q^n$) and $e_i \xleftarrow{\$} \mathcal{G}_\alpha$ be the encryption noise, $\forall i \in [z]$.
- $\mathbf{ct} \leftarrow \mathsf{ThAHE.Enc}(m, \mathsf{pk})$: Given a message bit $m \in \{0,1\}$ and the public key set pk, returns a ciphertext $\mathbf{ct} = (b, \mathbf{a})$, where, $b = \sum_{j \in \mathcal{H}} \mathsf{pk}_j + \lfloor \frac{q}{2} \rfloor \cdot m \pmod{q} \in \mathbb{Z}_q^{n+1}$, and $\mathbf{a} = \sum_{j \in \mathcal{H}} \mathbf{a}_j$, $\mathcal{H}$ be a random subset of pk.
- $\mathbf{ct}^\star \leftarrow \mathsf{ThAHE.Add}(\mathbf{ct}_1, \mathbf{ct}_2, \mathsf{pk})$: Given two ciphertexts $\mathbf{ct}_1 = \mathsf{ThAHE.Enc}(m_1, \mathsf{pk})$, and $\mathbf{ct}_2 = \mathsf{ThAHE.Enc}(m_2, \mathsf{pk})$, the public key pk, evaluate $\mathbf{ct}^\star = (\mathbf{ct}_1 + \mathbf{ct}_2) \mod q$, such that

$$\begin{aligned}\langle \mathbf{ct}^\star, \mathbf{sk} \rangle =& \langle \mathbf{ct}_1, \mathbf{sk} \rangle + \langle \mathbf{ct}_2, \mathbf{sk} \rangle \bmod q \\ =& \lfloor \frac{q}{2} \rfloor \cdot (m_1 + m_2) + e^\star \bmod q\end{aligned}$$

Here, $e^\star = (e_1 + e_2)$, e_1, e_2 are the encryption noise in $\mathbf{ct}_1, \mathbf{ct}_2$ respectively.

- $m^\star \leftarrow \mathsf{ThAHE.Dec}(\mathbf{ct}^\star, \mathbf{sk})$: Given an evaluated ciphertext $\mathbf{ct}^\star$ and the secret key $\mathbf{sk}$, returns a message bit $m^\star = \lfloor (\frac{2}{q})[\langle \mathbf{ct}^\star, \mathbf{sk}\rangle]_q \rfloor \pmod 2 \in \{0, 1\}$.

- $\gamma_j \leftarrow \mathsf{ThAHE.PartDec}(\mathbf{sk}^j, \mathbf{ct})$: It takes a secret key share $\mathbf{sk}^j$ for the j^{th} party and a ciphertext $\mathbf{ct} = (b, \mathbf{a})$ as input and returns a partially decrypted ciphertext $\gamma_j = \langle \mathbf{a}, \mathbf{sk}^j \rangle + e^j_{sm}$. Here a Gaussian noise e^j_{sm} is added with the partial decryption γ_j to statistically hide the LWE secret key $\mathbf{sk}$ and LWE noise e. In the literature, this e^j_{sm} is known as "smudging noise" and generally, e^j_{sm} is sampled from a Gaussian distribution $\mathcal{G}_\sigma$ with a standard deviation σ.

  Now, σ has to be atleast polynomially larger [BS23, CSS+22] than α, to statistically hide the LWE noise e.

- $\phi \leftarrow \mathsf{ThAHE.Combine}(\{\gamma_j\}_{j\in[T]}, \mathbf{ct})$: It takes $t(\leq T)$ number of partial decryptions $\{\gamma_j\}_{i\in[t]}$ from T parties and the ciphertext $\mathbf{ct} = (b, \mathbf{a})$ as input and compute a phase as

$$\phi = b - \sum_{j=1}^{t} \gamma_j = b - \sum_{j=1}^{t} \left(\langle \mathbf{a}, \mathbf{sk^j} \rangle + e^j_{sm} \right),$$
$$= m + e - \sum_{j=1}^{t} e^j_{sm}.$$

- $m \leftarrow \mathsf{ThAHE.Decode}(\phi)$: On input the phase ϕ this function performs a rounding operation to remove the noises and returns the message m, i.e., if $\phi > 0$ returns 1 otherwise returns 0.

3 Our Proposal: TOPRF using Threshold AHE

In this section, we provide a detailed description of our proposed TOPRF construction using the threshold decryption procedure of an AHE scheme. We extend our threshold decryption algorithm from the work by Chowdhury et al. [CSS+22], that uses LISSS [DT06] to distribute the secret key **sk** among T servers following a t-out-of-T threshold access structure[1]. We extended the LISSS secret sharing scheme into our ThAHE scheme to distribute the secret key **sk** among T servers; and every j^{th} server gets the j^{th} share of **sk**, $\forall j \in [T]$.

3.1 Basic Construction of OPRF using FHE

First, we describe how an Oblivious PRF (OPRF) can be constructed with an FHE scheme using a naïve approach and gradually motivate our threshold OPRF construction by showcasing a technique (one of our primary contributions) in which our TOPRF construction only requires an AHE scheme rather than an FHE by leveraging the thresholdization mechanism of an AHE over multiple servers.

Homomorphic PRF evaluation using FHE. We use AES-128 as our PRF and evaluate on encrypted inputs using an FHE scheme over a cloud server. This approach provides a

[1]Any $t \leq T$ number of parties can collaborate to perform the final decryption.

basic OPRF construction using FHE. We note that in our present construction, we use a Boolean representation of AES circuit (ref. Appendix A.3), but we want to point out that, our OPRF can be implemented using any PRF circuit represented in Boolean format.

Now, evaluating AES over encrypted inputs is a challenging task, especially due to the large computation overhead of FHE bootstrapping. We use TFHE [CGGI20] scheme to evaluate the AES-128 circuit represented in the Bristol format [bri], which uses only the following Boolean gates: XORs, ANDs and NOTs. A trivial solution is that a trusted key-dealer samples an FHE public key PK and a secret key **SK**, and provides PK to the client and the computing server and **SK** only to the client. Then the client having AES input message $\mathbf{aes}_{msg}$, encrypts $\mathbf{aes}_{msg}$ with PK and outsources the encrypted data to the server for homomorphic evaluation. Now, the server having the AES key $\mathbf{aes}_{key}$, encrypts it with PK and performs each Boolean gate of the AES circuit homomorphically using FHE computation using functions from the TFHE library[1]. After evaluation of all the gates, the encrypted AES output is sent back to the client; who then can decrypt using **SK** to remove the encryption layer of FHE and retrieves the AES output.

Performance analysis. The Bristol format of AES-128 consists of $25,124$ XOR, $6,800$ AND and $1,692$ NOT gates, and among these gates, XOR and AND gates require a bootstrapping operation (as, NOT gates do not incur any noise growth). Each gate-bootstrapping takes approximately 13 *ms* in the TFHE library on a standard computing platform. Consequently, AES-128 evaluation takes around 470 *secs*, which is quite impractical in real-world scenarios. Now, we gradually discuss our proposed approach to reduce the homomorphic evaluation time of AES using threshold AHE (ThAHE) scheme.

3.2 Threshold Construction of TOPRF using ThAHE

Evaluating any Boolean circuit homomorphically in an encrypted domain requires bootstrapping that allows circuit evaluation up to arbitrary depth. However, this bootstrapping operation is the primary bottleneck of any homomorphic evaluation due to its large computational cost. One of the primary goals of this work is to reduce the number of bootstrapping operations during homomorphic evaluation of Boolean circuits. In the TFHE scheme, every gate evaluation (except NOT gate) is followed by a bootstrapping operation. A bootstrapping operation is homomorphic decryption followed by adding a fresh noise into the ciphertext, which means, at most one Boolean gate can be evaluated without performing a bootstrapping operation (for more detailed analysis ref. Appendix A.2). We use this observation to construct our efficient TOPRF protocol. Before describing our proposed TOPRF construction using a ThAHE scheme, we present our modified definition of TOPRF (basic definition is presented in Section 2) to prove the security and consistency of our TOPRF protocol.

*Our Threshold Oblivious Pseudorandom Function (*TOPRF′*)* Our modified definition of TOPRF follows the same definition of the original TOPRF [AMMM18] which is mentioned in Section 2 except the following two algorithms TOPRF′.Eval and TOPRF′.Combine.

- $z'_{enc} \leftarrow \mathsf{TOPRF}'.\mathsf{Eval}(\{K_i\}_{i \in \mathcal{S}}, c)$. The evaluation of our modified TOPRF′ is not a function rather it is a protocol that is performed jointly by n parties. After execution of the protocol, each party gets z'_{enc}, the encrypted output of the PRF evaluation.

[1]https://github.com/tfhe/tfhe.git

Table 3: LUTs for Boolean AND ($\wedge$) and XOR ($\oplus$) for the masked bit $\{r_0^i, r_1^i\}$

$LUT_{\wedge}^{r_0^i, r_1^i}$	$\mathbf{ct}_{((0 \oplus r_0^i),(0 \oplus r_1^i))} = \mathsf{ThAHE.Enc}\left((0 \oplus r_0^i) \wedge (0 \oplus r_1^i), \mathsf{pk}\right)$	$\mathbf{ct}_{((0 \oplus r_0^i),(1 \oplus r_1^i))} = \mathsf{ThAHE.Enc}\left((0 \oplus r_0^i) \wedge (1 \oplus r_1^i), \mathsf{pk}\right)$
	$\mathbf{ct}_{((1 \oplus r_0^i),(0 \oplus r_1^i))} = \mathsf{ThAHE.Enc}\left((1 \oplus r_0^i) \wedge (0 \oplus r_1^i), \mathsf{pk}\right)$	$\mathbf{ct}_{((1 \oplus r_0^i),(1 \oplus r_1^i))} = \mathsf{ThAHE.Enc}\left((1 \oplus r_0^i) \wedge (1 \oplus r_1^i), \mathsf{pk}\right)$

$LUT_{\oplus}^{r_0^i, r_1^i}$	$\mathbf{ct}_{((0 \oplus r_0^i),(0 \oplus r_1^i))} = \mathsf{ThAHE.Enc}\left((0 \oplus r_0^i) \oplus (0 \oplus r_1^i), \mathsf{pk}\right)$	$\mathbf{ct}_{((0 \oplus r_0^i),(1 \oplus r_1^i))} = \mathsf{ThAHE.Enc}\left((0 \oplus r_0^i) \oplus (1 \oplus r_1^i), \mathsf{pk}\right)$
	$\mathbf{ct}_{((1 \oplus r_0^i),(0 \oplus r_1^i))} = \mathsf{ThAHE.Enc}\left((1 \oplus r_0^i) \oplus (0 \oplus r_1^i), \mathsf{pk}\right)$	$\mathbf{ct}_{((1 \oplus r_0^i),(1 \oplus r_1^i))} = \mathsf{ThAHE.Enc}\left((1 \oplus r_0^i) \oplus (1 \oplus r_1^i), \mathsf{pk}\right)$

- $h/\perp \leftarrow \mathsf{TOPRF}'.\mathsf{Combine}(z'_{enc}, \rho, \{K_i\}_{i \in \mathcal{S}})$. Our TOPRF combination is just a threshold decryption operation that is performed by each party in the set $\mathcal{S}$ using randomness ρ with their corresponding secret key shares K_i to generate the final decrypted value h. If the algorithm fails, its output is denoted by $\perp$.

Consistency and security properties. The consistency of our modified TOPRF' is similar to the original definition with an added dependency on the correctness of threshold decryption operation. Whereas, security is now dependent on the hardness of the encryption scheme used alongwith the secret sharing of PRF output over multiple parties as done in the former case. The intermediate results of PRF evaluation are available to all the participating parties but in an encrypted format; thus not accessible to any of the parties, only the final output is known after performing the combination operation.

Constructing TOPRF using ThAHE. Following our modified definition we now discuss the proposed TOPRF construction. We leverage the criteria of evaluating one binary gate without bootstrapping into our efficient TOPRF construction. In our TOPRF construction a party holds the PRF input $\mathbf{aes}_{msg}$, whereas the server's PRF key $\mathbf{aes}_{key}$ is distributed among T servers, say $(S_1, \ldots, S_T)$. Now, our TOPRF protocol has the following phases–

Pre-processing phase. At the very beginning of our proposed TOPRF protocol a trusted dealer samples a secret key and public key pair $(\mathbf{sk}, \mathsf{pk}) \leftarrow \mathsf{ThAHE.KeyGen}(\mathsf{params}_a)$ for an ThAHE scheme, and distributes the secret key $\mathbf{sk}$ into T shares $(\mathbf{sk}_1, \ldots, \mathbf{sk}_T)$ using LISSS [DT06] while following a t-out-of-T threshold access structure and provides them to T servers such that each server S_i gets $\mathbf{sk}_i$, $\forall i \in [T]$ and any t-sized subset of $\{S_1, \ldots, S_T\}$ can perform threshold decryption. The public key pk is then provided to the client.

The trusted dealer also samples L pairs of random bits $\{(r_0^0, r_1^0), \ldots, (r_0^{L-1}, r_1^{L-1})\} \overset{\$}{\leftarrow} \{0,1\}^L$, where L be the total number of XOR and AND gates in AES-128 Boolean circuit. It then encrypts these random bits as $\mathbf{cr}_j^i \leftarrow \mathsf{ThAHE.Enc}(r_j^i, \mathsf{pk})$, $\forall i \in [L], j \in \{0,1\}$. For each (r_0^i, r_1^i) pair, the trusted dealer constructs a look-up table $LUT_{\otimes}^{r_0^i, r_1^i}$ based on the operation $\otimes$ at the i^{th} iteration of AES circuit evaluation. Table 3 shows LUTs for Boolean AND (denoted as $\wedge$) and XOR (denoted as $\oplus$) gate. Finally, returns the encrypted set $\{(\mathbf{cr}_0^0, \mathbf{cr}_1^0), \ldots, (\mathbf{cr}_0^{L-1}, \mathbf{cr}_1^{L-1})\}$ to one of the servers, say S_1 and the L number of LUTs are provided to the other servers $(S_2, \ldots, S_T)$. In Figure 1 we present a pictorial description of our entire pre-processing/offline phase, performed once at the beginning of our TOPRF protocol.

Remark on key generation step. The key generation step of a threshold HE scheme is generally achieved in two different ways. One approach uses a "top-down" technique [BGG$^+$18, JRS17] in which a trusted key dealer generates a public key-secret key pair, and then distributes the secret key shares among the parties. An alternative approach alleviates the need for a trusted key dealer by allowing the parties to generate their secret key-public key pairs. Then,

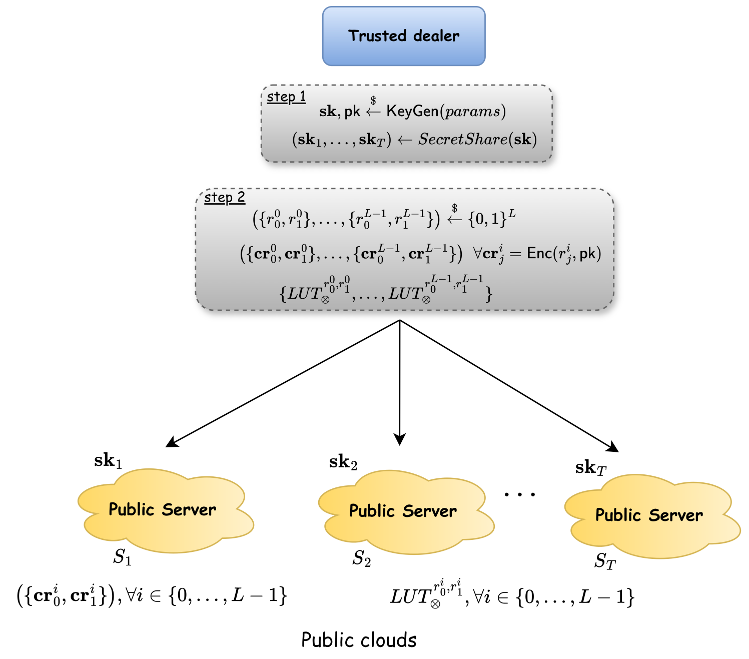
Trusted dealer
step 1
$\mathbf{sk}, \mathsf{pk} \xleftarrow{\$} \mathsf{KeyGen}(params)$
$(\mathbf{sk}_1, \ldots, \mathbf{sk}_T) \leftarrow SecretShare(\mathbf{sk})$
step 2
$(\{r_0^0, r_1^0\}, \ldots, \{r_0^{L-1}, r_1^{L-1}\}) \xleftarrow{\$} \{0,1\}^L$
$(\{\mathbf{cr}_0^0, \mathbf{cr}_1^0\}, \ldots, \{\mathbf{cr}_0^{L-1}, \mathbf{cr}_1^{L-1}\})\ \forall \mathbf{cr}_j^i = \mathsf{Enc}(r_j^i, \mathsf{pk})$
$\{LUT_{\otimes}^{r_0^0, r_1^0}, \ldots, LUT_{\otimes}^{r_0^{L-1}, r_1^{L-1}}\}$
$\mathbf{sk}_1$
$\mathbf{sk}_2$
$\mathbf{sk}_T$
Public Server
Public Server
Public Server
$\ldots$
S_1
S_2
S_T
$(\{\mathbf{cr}_0^i, \mathbf{cr}_1^i\}), \forall i \in \{0, \ldots, L-1\}$
$LUT_{\otimes}^{r_0^i, r_1^i}, \forall i \in \{0, \ldots, L-1\}$
Public clouds

Figure 1: Pre-processing/offline phase of our proposed TOPRF construction

it uses a "bottom-up" technique [STH+23, KJY+20] to generate the common public key. In contrast, our approach adopts a top-down strategy for key generation. Nonetheless, we can accommodate the bottom-up method for distributed key generation in our threshold AHE scheme, leveraging the technique proposed by Sugizaki et al. [STH+23]. This adaptation incurs extra costs in key generation and homomorphic computation time, thereby notably impacting efficiency.

Execution phase. After the pre-processing phase, the client gets the public key pk, which is then used by the client to encrypt its 128-bit input $\mathbf{aes}_{msg} \in \{0,1\}^{128}$ and generate a set of input ciphertexts as $\mathbf{ct}_k = \mathsf{AES.Enc}(\mathbf{aes}_{msg}[k], \mathsf{pk})$, $\forall k \in \{0, \ldots, 127\}$. Now, the client outsources these encrypted set $\{\mathbf{ct}_0, \ldots, \mathbf{ct}_{127}\}$ to one of the servers, say S_1 which holds the 128-bit PRF key $\mathbf{aes}_{key} \in \{0,1\}^{128}$. S_1 also encrypts the key with pk and generates the set $\{\mathbf{ct}'_0, \ldots, \mathbf{ct}'_{127}\}$, where $\mathbf{ct}'_k = \mathsf{AES.Enc}(\mathbf{aes}_{key}[k], \mathsf{pk})$, $\forall k \in \{0, \ldots, 127\}$.

Assume, during the homomorphic evaluation of the AES-128 circuit, at the i^{th} iteration an AND gate needs to be evaluated on two input ciphertexts $\mathbf{ct}_0^i$ and $\mathbf{ct}_1^i$, i.e., $\mathbf{ct}^i \leftarrow \mathsf{HomAND}\left(\mathbf{ct}_0^i, \mathbf{ct}_1^i\right)$, where $\mathsf{HomAND}(\cdot)$ denotes homomorphic AND operation on encrypted data and returns an encrypted output and $\mathbf{ct}^i$ be the resultant ciphertexts at the i^{th} iteration. Generally, any homomorphic gate evaluation over encrypted inputs is followed by a bootstrapping operation to retain the noise level under a pre-specified threshold of correct decryption in the resultant ciphertext. But as mentioned earlier, our TOPRF construction requires no bootstrapping; consequently, we only rely on a threshold AHE scheme. To evaluate $\mathbf{ct}^i \leftarrow \mathsf{HomAND}\left(\mathbf{ct}_0^i, \mathbf{ct}_1^i\right)$ using a threshold AHE scheme, S_1 first mask the input

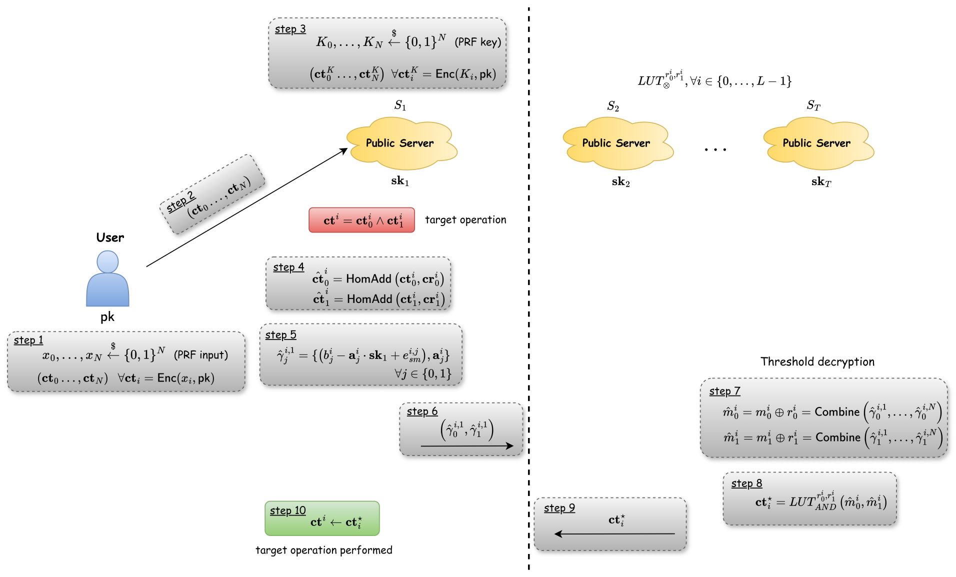
step 3
$K_0, \dots, K_N \xleftarrow{\$} \{0,1\}^N$ (PRF key)
$(\mathbf{ct}_0^K \dots, \mathbf{ct}_N^K)$ $\forall \mathbf{ct}_i^K = \mathsf{Enc}(K_i, \mathsf{pk})$
$LUT_{\otimes}^{r_0^i, r_1^i}, \forall i \in \{0, \dots, L-1\}$
S_1
Public Server
$\mathbf{sk}_1$
S_2
Public Server
$\mathbf{sk}_2$
$\dots$
S_T
Public Server
$\mathbf{sk}_T$
step 2
$(\mathbf{ct}_0 \dots, \mathbf{ct}_N)$
$\mathbf{ct}^i = \mathbf{ct}_0^i \wedge \mathbf{ct}_1^i$
target operation
User
pk
step 4
$\hat{\mathbf{ct}}_0^i = \mathsf{HomAdd}(\mathbf{ct}_0^i, \mathbf{cr}_0^i)$
$\hat{\mathbf{ct}}_1^i = \mathsf{HomAdd}(\mathbf{ct}_1^i, \mathbf{cr}_1^i)$
step 1
$x_0, \dots, x_N \xleftarrow{\$} \{0,1\}^N$ (PRF input)
$(\mathbf{ct}_0 \dots, \mathbf{ct}_N)$ $\forall \mathbf{ct}_i = \mathsf{Enc}(x_i, \mathsf{pk})$
step 5
$\hat{\gamma}_j^{i,1} = \{(b_j^i - \mathbf{a}_j^i \cdot \mathbf{sk}_1 + e_{sm}^{i,j}), \mathbf{a}_j^i\}$
$\forall j \in \{0,1\}$
Threshold decryption
step 7
$\hat{m}_0^i = m_0^i \oplus r_0^i = \mathsf{Combine}(\hat{\gamma}_0^{i,1}, \dots, \hat{\gamma}_0^{i,N})$
$\hat{m}_1^i = m_1^i \oplus r_1^i = \mathsf{Combine}(\hat{\gamma}_1^{i,1}, \dots, \hat{\gamma}_1^{i,N})$
step 6
$(\hat{\gamma}_0^{i,1}, \hat{\gamma}_1^{i,1})$
step 8
$\mathbf{ct}_i^\star = LUT_{AND}^{r_0^i, r_1^i}(\hat{m}_0^i, \hat{m}_1^i)$
step 10
$\mathbf{ct}^i \leftarrow \mathbf{ct}_i^\star$
target operation performed
step 9
$\mathbf{ct}_i^\star$

Figure 2: Execution phase of our proposed TOPRF construction

ciphertexts $\mathbf{ct}_0^i$ and $\mathbf{ct}_1^i$ with the encrypted random-bits $\mathbf{cr}_0^i$ and $\mathbf{cr}_1^i$ respectively, as follows

$$\hat{\mathbf{ct}}_0^i \leftarrow \mathsf{ThAHE.Add}(\mathbf{ct}_0^i, \mathbf{cr}_0^i, \mathsf{pk}),$$
$$\hat{\mathbf{ct}}_1^i \leftarrow \mathsf{ThAHE.Add}(\mathbf{ct}_1^i, \mathbf{cr}_1^i, \mathsf{pk}).$$

Note that, one addition operation does not require a bootstrapping to retain the correctness in the resultant ciphertext (discussed in Appendix A.2). S_1 then computes the partial decryption $\hat{\gamma}_{j,1}^i$ using its secret share $\mathbf{sk}_1$, as

$$\hat{\gamma}_{j,1}^i \leftarrow \mathsf{ThAHE.PartDec}\left(\hat{\mathbf{ct}}_j^i, \mathbf{sk}_1\right),$$

here $\hat{\mathbf{ct}}_j^i = (\hat{b}_j^i, \hat{\mathbf{a}}_j^i)$, $\forall j \in \{0,1\}$. Now, these partial decryptions $\left(\hat{\gamma}_{0,1}^i, \hat{\gamma}_{1,1}^i\right)$ of masked ciphertexts $\hat{\mathbf{ct}}_0^i, \hat{\mathbf{ct}}_1^i$ are forwarded to $(S_2, \dots, S_T)$ for performing the threshold decryption. After receiving the partial decryptions $\left(\hat{\gamma}_{0,1}^i, \hat{\gamma}_{1,1}^i\right)$, the servers $(S_2, \dots, S_T)$ compute their partial decryptions with their corresponding secret shares, i.e., $\hat{\gamma}_{j,k}^i \leftarrow \mathsf{ThAHE.PartDec}\left(\hat{\gamma}_{1,1}, \mathbf{sk}_k\right)$, $\forall k \in \{2, \dots, T\}$. Here, $\hat{\gamma}_{j,k}^i$ refers the partial decryption computed by server S_k using the its secret share $\mathbf{sk}_k$. While all the partial decryptions are computed by all the servers, the final combination is performed by taking all such partial decryptions $\left(\hat{\gamma}_{j,1}^i, \dots, \hat{\gamma}_{j,t}^i\right)$ to compute the final decrypted plaintext, as

$$\hat{m}_j^i \leftarrow \mathsf{ThAHE.Combine}\left(\{\hat{\gamma}_{j,k}^i\}_{k \in [t]}\right), \ \forall j \in \{0,1\}.$$

Observe that, these decrypted results $\hat{m}_0^i, \hat{m}_1^i$ are masked with the random bits r_0^i, r_1^i respectively, i.e., $\hat{m}_j^i = m_j^i \oplus r_j^i$, where m_j^i is the underlying plaintext of $\mathbf{ct}_j^i$, $\forall j \in \{0,1\}$. Therefore, $\hat{m}_0^i, \hat{m}_1^i$ are *information theoretically secure* under the random bits r_0^i, r_1^i.

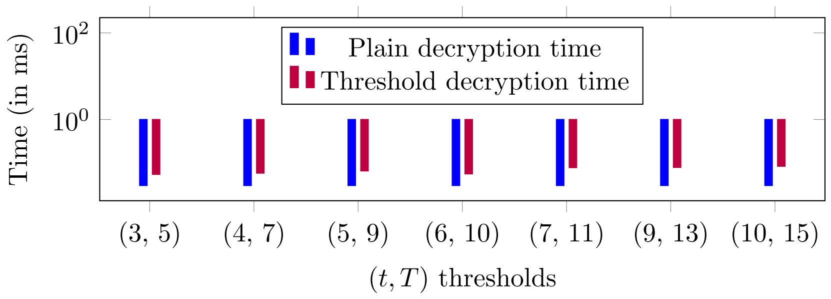

Figure 3: Time comparison between plain decryption and threshold decryption for our threshold AHE scheme

Now, these masked plaintexts are queried on the LUT at i^{th} stage under the random bits r_0^i, r_1^i, i.e., $LUT_{\wedge}^{r_0^i, r_1^i}\left(\hat{m}_0^i, \hat{m}_1^i\right)$. Recall that, we assume to evaluate an AND ($\wedge$) gate at the i^{th} stage. The $LUT_{\wedge}^{r_0^i, r_1^i}\left(\hat{m}_0^i, \hat{m}_1^i\right)$ then returns $\mathbf{ct}_{((\hat{m}_0^i \oplus r_0^i),(\hat{m}_1^i \oplus r_1^i))}$, which is an encryption of $\left(\hat{m}_0^i \oplus r_0^i\right) \wedge \left(\hat{m}_1^i \oplus r_1^i\right)$. Then the resultant ciphertext $\mathbf{ct}_{((\hat{m}_0^i \oplus r_0^i),(\hat{m}_1^i \oplus r_1^i))}$ after LUT operation is sent back to the server S_1 as a result of $\mathsf{HomAND}\left(\mathbf{ct}_0^i, \mathbf{ct}_1^i\right)$; and finally S_1 sets $\mathbf{ct}^i \leftarrow \mathbf{ct}_{((\hat{m}_0^i \oplus r_0^i),(\hat{m}_1^i \oplus r_1^i))}$.

Similarly, for an $\mathbf{ct}^l \leftarrow \mathbf{ct}_0\mathsf{HomXOR}(\mathbf{ct}_0^l, \mathbf{ct}_1^l)$, i.e., for a homomorphic XOR operation at some l^{th} iteration, the servers perform all the operations in the execution phase except the LUT query step, in which after performing the threshold decryption the servers make a query on the LUT for the XOR ($\oplus$) operation, i.e., $LUT_{\oplus}^{r_0^l, r_1^l}\left(\hat{m}_0^l, \hat{m}_1^l\right)$ and retrieve encryption of $\left(\hat{m}_0^l \oplus r_0^l\right) \oplus \left(\hat{m}_1^l \oplus r_1^l\right)$. After that, this resultant ciphertext $\mathbf{ct}_{((\hat{m}_0^l \oplus r_0^l),(\hat{m}_1^l \oplus r_1^l))}$ of LUT operation is sent back to S_1 and it sets $\mathbf{ct}^l \leftarrow \mathbf{ct}_{((\hat{m}_0^l \oplus r_0^l),(\hat{m}_1^l \oplus r_1^l))}$. Figure 2 demonstrates our proposed TOPRF protocol in a step-by-step fashion for the reader's convenience.

3.3 Performance analysis

We note here that, for evaluating one Boolean gate either XOR or AND, our proposed methodology based on a threshold AHE scheme requires only one round of communication during threshold decryption performed by $(S_2, \ldots, S_T)$ and two independent arithmetic operations (without bootstrapping) by server S_1. It turned out that the cost of single bootstrapping is significantly more computationally expensive than the above combination of threshold decryptions and arithmetic operations. Our threshold decryption operation takes approximately 0.03 *ms* (ignoring communication latency, which is discussed in the subsequent sections) of time while the cost of homomorphic operation without bootstrapping is negligible (detailed analysis is presented in our experimental results in Section 4). Figure 3 shows that the time required for our threshold decryption operation is only slightly higher than that of plain decryption using an AHE scheme. In other words, our proposed threshold decryption procedure incurs only minimal overhead over the plain decryption algorithm. Using our proposed threshold AHE-based approach, we achieve around 0.96 *secs* of evaluation time[1] instead of 470 *secs*, considering Therefore, we achieve more than $470\times$ improvement in computational cost for evaluating our proposed TOPRF protocol.

3.4 Reducing communication during our TOPRF evaluation

Until now, we have discussed optimization techniques for the homomorphic execution time of our TOPRF using ThAHE scheme. Although we have achieved significant improvement in PRF evaluation time, it has an additional communication overhead, because during PRF evaluation, masked ciphertexts are communicated for threshold decryption (although performed in one round). For every Boolean gate evaluation, partial decryptions of 2 masked ciphertexts are communicated between $t(\leq T)$ servers. This requires less than T communications of 2 ciphertexts. Now, in our LWE-based AHE scheme, the size of each ciphertext is approximately 2 KB. Hence, for 31924 many Boolean gates (considering only XOR and AND gates) for an AES-128 circuit, a total of $2 \times 31924 \times t \times 2$ KB $= 127,696t$ KB $\approx 2^{17}t$ KB, or $128t$ MB data is communicated. Again in the response phase, one ciphertext is sent from $(S_2, \ldots, S_T)$ to S_1 after querying the LUT. This requires a total of 31924×2 KB $\approx 2^{16}$ KB, or 64 MB data is transferred. Hence, the overall communication bandwidth for evaluating an entire AES-128 circuit is $128t + 64$ MB, or $64(2t+1)$ MB. Assuming a 1 GB/s LAN connectivity between the public servers, we achieve $0.0625(2t+1)$ *secs* communication delay. For a smaller $t \leq 4$, the communication latency is comparatively lower with respect to our TOPRF evaluation time of 0.96 *secs*. Therefore, in this section, we provide several optimization techniques to further reduce the total communication delay between the public servers to support a sufficiently large t value.

Reducing communication bandwidth from S_1 to others. Now, we describe a technique to reduce the ciphertext size transferred from S_1 to the rest of the servers, i.e., $(S_2, \ldots, S_T)$. As already mentioned, the ciphertext generated from our LWE-based encryption has a size of approximately 2 KB, this is because the random vector $\mathbf{a}$ inside a ciphertext $\mathbf{ct} = (b, \mathbf{a})$. This random vector $\mathbf{a}$ is a n dimensional vector with each element of size $\log_2 q$ *bits*, and the b component is only $\log_2 q$ *bits*. Therefore, the size of one ciphertext is approximately equal to the size of $\mathbf{a}$. Here, $n = 512$ is the dimension of the LWE secret, and $q = 2^{32}$ is the ciphertext modulus. This parameter set is chosen according to the TFHE library where the Albrecht, Player, and Scott's Lattice Estimator [APS15] provides the estimated complexity of attacking using these parameters is roughly 2^{128} operations. Therefore, the size of $\mathbf{a}$ is $n \log_2 q = 512 \times 32$ *bits* or, 2 KB.

Hence, to reduce the size of a ciphertext, we need to reduce the size of the random vector $\mathbf{a}$. Assume, at the i^{th} stage $\mathbf{ct}_0^i$ and $\mathbf{ct}_1^i$ are the input ciphertexts of some homomorphic operation. These ciphertexts are masked with random bits r_0^i and r_1^i as follows, $\hat{\mathbf{ct}}_0^i = \mathsf{ThAHE.Add}\left(\mathbf{ct}_0^i, \mathbf{cr}_0^i, \mathsf{pk}\right)$ and $\hat{\mathbf{ct}}_1^i = \mathsf{ThAHE.Add}\left(\mathbf{ct}_1^i, \mathbf{cr}_1^i\right)$, where $\mathbf{cr}_j^i$, is the encryption of the random bit r_j^i, $\forall j \in \{0,1\}$. Now, for $\hat{\mathbf{ct}}_j^i = (\hat{b}_j^i, \hat{\mathbf{a}}_j^i)$, we have $\hat{b}_j^i = \langle \hat{\mathbf{a}}_j^i, \mathbf{sk} \rangle + \hat{m}_j^i + \hat{e}_j^i$, $\forall j \in \{0,1\}$. As discussed in Section 3.2, the noise $\hat{e}_j^i = 2 \cdot (e_j^i + er_j^i)$, here e_j^i and er_j^i are the noises inside $\mathbf{ct}_j^i$ and $\mathbf{cr}_j^i$, respectively $\forall j \in \{0,1\}$. The basis of our first reduction in communication bandwidth relies on the gap between minimum (α_{min}) and maximum (α_{max})

[1]Our homomorphic addition without bootstrapping takes approximately 0.15 μs, and a LUT look-up time is considered to be negligible.

standard deviations of Gaussian noise distributions used in our ThAHE scheme[1] from which the noises in the ciphertexts are sampled. α_{min} represents the minimum standard deviation required for security against quantum attacks and α_{max} represents the maximum standard deviation required for decryption correctness for an LWE ciphertext (ref. Table 6 for the parameter values). In our proposed scheme, we sample the noises in input ciphertexts from the Gaussian distribution with standard deviation α_{min}. From the property of Gaussian distribution with mean 0, we can approximately compute (with 99.994% probability) the values of the sampled noises with the following bound, $|e| \leq 4 \cdot \alpha_{min}$. Following this, we can infer that $|e^i_j|, |er^i_j| \leq 4 \cdot \alpha_{min}$ and $|\hat{e}^i_j| \leq 16 \cdot \alpha_{min}$. Now, from correctness of decryption we must have $|\hat{e}^i_j| \leq 4 \cdot \alpha_{max}$. We now use these results to optimize the communication delay from S_1 to $(S_2, \ldots, S_T)$. As most of the communication bandwidth is consumed by the random vector $\hat{\mathbf{a}}^i_j$ inside an LWE ciphertext $\hat{\mathbf{ct}}^i_j$. Therefore, to reduce the size of $\mathbf{a}$, we discard some bits from $\hat{\mathbf{a}}^i_j$ without affecting the decryption correctness performed by $(S_2, \ldots, S_T)$. This raises the following question,

how many least significant bits can we remove from each component of $\hat{\mathbf{a}}^i_j$ so that $\hat{\mathbf{ct}}^i_j$ decrypts to $\hat{m}^i_j$?

Let us assume we discard least significant k-bits from each of the component of $\hat{\mathbf{a}}^i_j$, i.e., $\hat{a}^i_{j,t}$, $\forall t \in \{0, \ldots, n-1\}$ of $\hat{\mathbf{a}}^i_j = (\hat{a}^i_{j,0}, \ldots, \hat{a}^i_{j,n-1})$. Now, we denote the least significant k-bits of $\hat{a}^i_{j,t}$ as $\hat{\mathbf{a}}^{i,\ell}_{j,t}$ and the rest of the most significant bits are denoted as $\hat{\mathbf{a}}^{i,m}_{j,t}$. Therefore, each component of the random vector $\hat{\mathbf{a}}^i_j$ can be represented as $\hat{a}^i_{j,t} = \hat{a}^{i,m}_{j,t} \cdot 2^k + \hat{a}^{i,\ell}_{j,t}$. Now, we only communicate $(\hat{b}^i_j, \hat{\mathbf{a}}^{i,m}_j)$, where $\hat{\mathbf{a}}^{i,m}_j = (\hat{a}^{i,m}_{j,0}, \ldots, \hat{a}^{i,m}_{j,n-1})$, $\forall j \in \{0,1\}$ as the masked ciphertexts from S_1. Now, the threshold decryption operation over $\hat{\mathbf{ct}}^i_j$ is performed as follows,

$$\begin{aligned}
\hat{\Phi}^i_j &= \hat{b}^i_j - \langle(\hat{\mathbf{a}}^{i,m}_j \cdot 2^k), \mathbf{sk}\rangle, \\
&= \hat{b}^i_j - \langle(\hat{\mathbf{a}}^i_j - \hat{\mathbf{a}}^{i,\ell}_j), \mathbf{sk}\rangle, \\
&= \hat{b}^i_j - \langle\hat{\mathbf{a}}^i_j, \mathbf{sk}\rangle + \langle\hat{\mathbf{a}}^{i,\ell}_j, \mathbf{sk}\rangle, \\
&= \hat{m}^i_j + \hat{e}^i_j + \langle\hat{\mathbf{a}}^{i,\ell}_j, \mathbf{sk}\rangle = \hat{m}^i_j + \hat{e}^{i,\prime}_j.
\end{aligned}$$

For correctness of decryption the new noise $\hat{e}^{i,\prime}_j = \hat{e}^i_j - \langle\hat{\mathbf{a}}^{i,\ell}_j, \mathbf{sk}\rangle$ must satisfy the following condition $|\hat{e}^{i,\prime}_j| \leq 4 \cdot \alpha_{max}$. Hence, $|\hat{e}^i_j + \langle\hat{\mathbf{a}}^{i,\ell}_j, \mathbf{sk}\rangle| \leq 4 \cdot \alpha_{max}$. We have mentioned already that $|\hat{e}^i_j| \leq 16 \cdot \alpha_{min}$. Now, we need to compute the maximum (absolute) value of $\langle\hat{\mathbf{a}}^{i,\ell}_j, \mathbf{sk}\rangle$, which turned out to be $n \times 2^{k-1}$ because every component of $\hat{\mathbf{a}}^{i,\ell}_j$ is a k-bit integer and $\mathbf{sk} \in \{0,1\}^n$. Therefore, we now have the following,

$$\begin{aligned}
|\hat{e}^i_j + \langle\hat{\mathbf{a}}^{i,\ell}_j, \mathbf{sk}\rangle| &\leq 4 \cdot \alpha_{max}, \\
16 \cdot \alpha_{min} + n \cdot 2^{k-1} &\leq 4 \cdot \alpha_{max}, \\
2^{k-1} &\leq \frac{4}{n}(\alpha_{max} - 4 \cdot \alpha_{min}), \\
k &\leq \log_2(\frac{4}{n}(\alpha_{max} - 4 \cdot \alpha_{min})) + 1.
\end{aligned}$$

[1]The Gaussian noise distributions consider their means as 0

Hence, at most $k = \log_2(\frac{4}{n}(\alpha_{max} - 4 \cdot \alpha_{min})) + 1$ least significant bits can be discarded from every component of the random vector $\hat{\mathbf{a}}_j^i$ without affecting the decryption correctness.

In this work, we are using a 32-bit modulus for our ThAHE scheme, i.e., the modulus $q = 2^{32}$ and the noises $|\hat{e}_j^{i,\prime}| \leq 4 \cdot \alpha_{max} \cdot 2^{32}$ and $|\hat{e}_j^i| \leq 16 \cdot \alpha_{min} \cdot 2^{32}$. Substituting these values in the above equation results in $k \leq 20$. Therefore, at most 20 least significant bits can be discarded from all the components in $\hat{\mathbf{a}}_j^i$, $\forall j \in \{0, 1\}$. Each component $\hat{a}_{j,t}^i$, $\forall t \in \{0, \ldots, n-1\}$ of $\hat{\mathbf{a}}_j^i = (\hat{a}_{j,0}^i, \ldots, \hat{a}_{j,n-1}^i)$ is represented as a 32-bit integer in our ThAHE scheme. After applying our optimization each component of $\hat{\mathbf{a}}_j^i$ becomes a 12-bit integer, thus achieving around 62.5% reduction in communication bandwidth during communication from S_1 to the rest of the servers $(S_2, \ldots, S_T)$.

Alternative method of communication reduction using PRG. We now present a separate solution for reducing the LWE ciphertext size. We adapt this technique from Chen et al. [CDKS21], in which a pseudo-random number generator $\mathcal{G} : \{0,1\}^\lambda \to \mathbb{Z}_q^n$, where $\lambda = 128$ be our security parameter. Since the second component of a LWE ciphertext $\mathbf{a}$ is purely random over $\mathbb{Z}_q^n$, we can modify our encryption algorithm such that it samples a seed θ and takes it as the input of a pseudo-random number generator $\mathcal{G}$ to generate $\mathbf{a} \leftarrow \mathcal{G}(\theta)$. As a result, a ciphertext can be represented as (b, θ), and this variant remains semantically secure in the random oracle model [CDKS21]. Therefore, the communication cost per LWE ciphertext is now only $\lambda + \log_2 q$ *bits*.

Now, recall from our TOPRF construction in Section 3.2, that we need to mask the input ciphertexts $\mathbf{ct}_j^i = (b_j^i, \mathbf{a}_j^i)$ with encryption of random bits $\mathbf{cr}_j^i = (br_j^i, \mathbf{ar}_j^i)$, and compute the resultant ciphertext $\hat{\mathbf{ct}}_j^i \leftarrow \mathsf{ThAHE.Add}\left(\mathbf{ct}_j^i, \mathbf{cr}_j^i\right)$, $\forall j \in \{0, 1\}$. Note that, we only have the seeds corresponding to $(\mathbf{a}_j^i, \mathbf{ar}_j^i)$, i.e., $(\theta_j^i, \theta_j^{i\prime})$ for the input ciphertexts $\mathbf{ct}_j^i, \mathbf{cr}_j^i$ as they are freshly generated by the client/trusted dealer. After performing the masking the randomness in the resultant ciphertext $\hat{\mathbf{ct}}_j^i = (\hat{b}_j^i, \hat{\mathbf{a}}_j^i)$ is $\hat{\mathbf{a}}_j^i = \mathbf{a}_j^i + \mathbf{ar}_j^i$. Hence, to get the seed $\hat{\theta}_j^i$ corresponding to $\hat{\mathbf{a}}_j^i$, we require a "additive" seed homomorphic PRG [BLMR13] that takes $(\theta_j^i, \theta_j^{i\prime})$ as input and returns $\hat{\theta}_j^i$. But, in the literature, we only have *almost* seed homomorphic PRGs from LWR assumption [BLMR13], in which the resultant seed $\hat{\theta}_j^i$ incurs a noise in the PRG output, i.e.,

$$\mathcal{G}_{LWR}^{add}\left(\theta_j^i, \theta_j^{i\prime}\right) = \mathcal{G}_{LWR}^{add}\left(\theta_j^i\right) + \mathcal{G}_{LWR}^{add}\left(\theta_j^{i\prime}\right) + \mathbf{e}_{LWR},$$

where, $\mathcal{G}_{LWR}^{add} : \mathbb{Z}_m^p \to \mathbb{Z}_n^q$, be the almost seed homomorphic PRG with $q < p$ and $m < n$ and $\mathbf{e}_{LWR} \in \{0, 1, 2\}^n$. In particular, the noise $\mathbf{e}_{LWR}$ is inherent to these PRGs; this is discussed in detail by Peter Scholl in [Sch18]. However, with certain techniques leveraging the concept of noise gap between the required security and correctness, we can remove this error during threshold decryption. Despite this complex method, we can simply transfer the seeds corresponding to the input ciphertexts, i.e., $\theta_j^i, \theta_j^{i\prime}$, from which the randomness in the masked ciphertexts can be computed as $\hat{\mathbf{a}}_j^i \leftarrow \mathcal{G}(\theta_j^i) + \mathcal{G}(\theta_j^{i\prime})$. Consequently, our masked ciphertexts are now of the form $\hat{\mathbf{ct}}_j^i = (\hat{b}_j^i, \theta_j^i, \theta_j^{i\prime})$, $\forall j \in \{0, 1\}$. With this approach, the size of the masked ciphertexts is now $2\lambda + \log_2 q$ instead of $(n+1)\log_2 q$ *bits* and consequently, we achieve around 98.24% reduction in the overall communication bandwidth from S_1 to the other servers.

Reducing communication bandwidth from others to S_1. We now extend the above-mentioned ciphertext compression technique to decrease the communication bandwidth for

Table 4: Modified LUT for Boolean AND ($\wedge$) for the masked bit $\{r_0^i, r_1^i\}$

$LUT_{\wedge}^{r_0^i, r_1^i}$	$\mathbf{ct}_{((0 \oplus r_0^i),(0 \oplus r_1^i))} = (b_{00}^i, \theta_{00}^i)$	$\mathbf{ct}_{((0 \oplus r_0^i),(1 \oplus r_1^i))} = (b_{01}^i, \theta_{01}^i)$
	$\mathbf{ct}_{((1 \oplus r_0^i),(0 \oplus r_1^i))} = (b_{10}^i, \theta_{10}^i)$	$\mathbf{ct}_{((1 \oplus r_0^i),(1 \oplus r_1^i))} = (b_{11}^i, \theta_{11}^i)$

Table 5: LUT for Boolean AND for the masked bit $\{r_0^i = 1, r_1^i = 0\}$

$LUT_{\wedge}^{1,0}$	ThAHE.Enc $((0 \oplus 1) \wedge (0 \oplus 0), \mathsf{pk})$ = ThAHE.Enc$(0, \mathsf{pk})$	ThAHE.Enc $((0 \oplus 1) \wedge (1 \oplus 0), \mathsf{pk})$ = ThAHE.Enc$(1, \mathsf{pk})$
	ThAHE.Enc $((1 \oplus 1) \wedge (0 \oplus 0), \mathsf{pk})$ = ThAHE.Enc$(0, \mathsf{pk})$	ThAHE.Enc $((1 \oplus 1) \wedge (1 \oplus 0), \mathsf{pk})$ = ThAHE.Enc$(0, \mathsf{pk})$

the response ciphertext to S_1. After performing threshold decryption, the masked plaintext bits are used to query on an LUT corresponding to a Boolean operation. Each of these LUTs consists of four ciphertexts for four possible values of the masking bits (r_0^i, r_1^i), and these LUTs are constructed by the trusted dealer. Hence, during LUT generation, the seeds corresponding to the ciphertexts are stored instead of using the random vectors. For example, the modified LUT for Boolean AND ($\wedge$) is shown in Table 4, where in place of random vectors the seeds are stored as part of the ciphertexts, such that $\mathbf{a}_{uv}^i = \mathcal{G}(\theta_{uv}^i)$. corresponding to $\mathbf{ct}_{((u \oplus r_0^i),(v \oplus r_1^i))}$, $\forall u, v \in \{0,1\}^2$. Therefore, as a response to S_1 the rest of the servers send the compressed ciphertext $\mathbf{ct}_{((u \oplus r_0^i),(v \oplus r_1^i))} = (b_{uv}^i, \theta_{uv}^i)$, which has a size of $\lambda + \log_2 q$ *bits*; thus achieves approximately 99.02% reduction in the communication delay during response phase from $(S_2, \dots, S_T)$ to S_1.

Analysis on communication bandwidth using our proposed optimizations. Here, we analyze the performance gain using our ciphertext compression techniques. As discussed earlier, the total communication bandwidth required for our initial construction of TOPRF for a (t, T) threshold decryption is $64(2t + 1)$ MB or, $128(2t + 1)$ MB based on single-key or 2-key variant of ThAHE scheme. Now, with 31924 many Boolean gates for an AES-based PRF circuit, we compute our optimized communication bandwidth as follows,

- for communicating ciphertexts from S_1 to $(S_2, \dots, S_T)$ we achieve approximately 98.24% reduction in ciphertexts sizes, hence the communication cost is now $128t \times \frac{1.76}{100}$ MB or, $2.25t$ MB, instead of $128t$ MB, required to transfer the masked ciphertexts in our initial construction.
- for the response phase 64 MB data needs to be sent from $(S_2, \dots, S_T)$ to S_1, but applying our 99.02% reduction technique, our communication cost in the response phase is reduced to $64 \times \frac{0.98}{100}$, or 0.627 MB.

Therefore, the total communication bandwidth required for homomorphically evaluating an AES-based PRF circuit is approximately $2.25t + 0.63$ MB. Assuming a 1 GB/s LAN connectivity between the public servers, the time required for total communication is now $2.25t + 0.63$ *ms*, or $\approx 2.25t$ *ms* for large values of t. Observe that, the optimized communication bandwidth grows linearly in t and is negligible (for reasonable values of t) in comparison to the TOPRF evaluation time of 0.96 *secs*. Therefore, we can enlarge the threshold value t up to 400 servers to have a comparable delay compared to our TOPRF evaluation time. In Figure 4 we present a comparison between the total bandwidth required for optimized and un-optimized versions of TOPRF for multiple values of t.

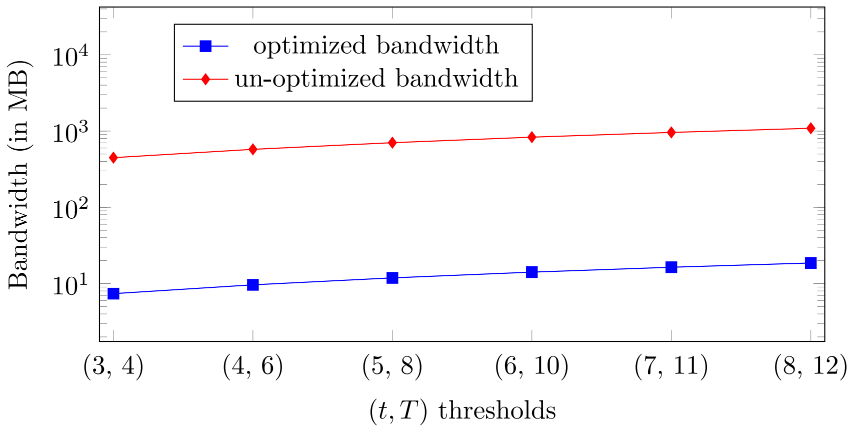

Figure 4: Comparison between optimized and un-optimized communication bandwidth for different values of (t, T)

3.5 Correctness analysis of our TOPRF evaluation

Here, we prove the correctness of a PRF circuit evaluation using our proposed TOPRF construction. Recall from Section 3.2, at the beginning of our TOPRF protocol, a trusted key dealer distributes the decryption key **sk** among T servers and generates encryptions of some random bits and LUTs, that are used to evaluate a Boolean PRF circuit homomorphically and securely. Now, in the execution phase of one of the T serves say, S_1 masks the input ciphertexts of a Boolean gate by performing homomorphic addition with the encrypted random bits. We sample the initial noise during encryption from the lower-bounded noise distribution, thus one addition (or, XOR) operation in an AHE (or, FHE) scheme requires no noise reduction technique such as bootstrapping because the noise in the resultant ciphertext remains within the required threshold (discussed in Appendix A.2). Thus after the threshold decryption performed by $(S_2, \ldots, S_T)$, the servers get the required plaintext bits. Using this they query an LUT corresponding to the Boolean gate that needs to be evaluated. Now, our correctness of the entire TOPRF evaluation relies on the query phase of LUT operation, i.e., the LUT must return the desired encryption corresponding to the output of the target Boolean gate that needs to be evaluated.

We prove the correctness of the LUT query phase using an example, suppose, for random bits (r_0^i, r_1^i) at the i^{th}, the LUT for Boolean AND is $LUT_{\wedge}^{r_0^i, r_1^i}$. Let us assume the input bits at the i^{th} be $m_0^i = 0$ and $m_1^i = 1$ and the random masking bits be $r_0^i = 1$ and $r_1^i = 0$. After homomorphic addition the masked plaintexts are $\hat{m}_0^i = m_0^i \oplus r_0^i = 0 \oplus 1 = 1$ and $\hat{m}_1^i = m_1^i \oplus r_1^i = 1 \oplus 0 = 1$. The $LUT_{\wedge}^{r_0^i, r_1^i}$ for $r_0^i = 1$ and $r_1^i = 0$ is shown in Table 5. After the threshold decryption $(S_2, \ldots, S_T)$ retrieves $\hat{m}_0^i = 1$ and $\hat{m}_1^i = 1$ and query with $LUT_{\wedge}^{1,0}\left(\hat{m}_0^i, \hat{m}_1^i\right) = LUT_{\wedge}^{1,0}(1, 1)$. This returns the ciphertext at position $(1, 1)$ in the LUT as shown in Table 5, which outputs an encryption of 0 which is indeed the correct evaluation of $m_0^i \wedge m_1^i = 0 \wedge 1 = 0$.

Similarly, we can prove the correctness for Boolean XOR operation with different values of random masking bits and input plaintexts.

Table 6: Parameters used in experimental setting

Parameter	**Value**
n (LWE dimension)	512
α_{min} (Minimum standard deviation of LWE noise)	$\sqrt{\frac{2}{\pi}} \times 2^{-15}$
α_{max} (Maximum standard deviation of LWE noise)	$\sqrt{\frac{2}{\pi}} \times 2^{-6}$
σ (Standard deviation of smudging noise)	0.0312

3.6 Security analysis of our TOPRF protocol

Our TOPRF construction relies on a set of T cloud servers. Among T servers, one has the encrypted PRF key K and encrypted masking bits. The decryption key **sk** is distributed amongst the rest of the $(T-1)$ server. Now, our security analysis ensures that our TOPRF protocol is secure under maximal corruption up to $(T-1)$ servers and satisfies the "unpredictability" and "obliviousness" properties. Let $\mathcal{S}_c$ be a set of servers corrupted by a semi-honest PPT adversary $\mathcal{A}$ and for maximal corruption $|\mathcal{S}_c| = (T-1)$.

Theorem 1. *Our TOPRF protocol satisfies* unpredictability/pseudo-randomness *property under the maximal corruption of* $(T-1)$ *servers by a semi-honest PPT adversary* $\mathcal{A}$.

Proof. Suppose $\mathcal{A}$ corrupt up to $(T-1)$ servers, i.e., $|\mathcal{S}_c| = T$. In such case, $\mathcal{A}$ has access to the masked plaintext bits after completion of the threshold decryption protocol. But these plaintexts are information-theoretically secure due to the homomorphic masking operation by one of the T servers at the beginning of the protocol. Thus, the original plaintexts are hidden in the view of $\mathcal{A}$. However, during the PRF circuit evaluation, due to the homomorphic property of the underlying AHE scheme, the intermediate inputs/results remain encrypted, and $\mathcal{A}$ is unable to perform decryption with $(T-1)$ secret key shares according to the property of threshold LISSS [DT06] scheme. Therefore, the intermediate results are inaccessible to $\mathcal{A}$, and consequently, the output of PRF evaluation remains indistinguishable from random. □

Theorem 2. *Our TOPRF protocol satisfies* obliviousness *property under the maximal corruption of* $(T-1)$ *servers by a semi-honest PPT adversary* $\mathcal{A}$.

Proof. Our TOPRF protocol satisfies the obliviousness property under a similar argument as claimed in the proof of unpredictability, i.e., although $\mathcal{A}$ corrupt up to $(T-1)$ servers, $\mathcal{A}$ is unable to retrieve the client's PRF input x or the server's PRF key K, because they are encrypted under an AHE scheme at the beginning of the protocol and during PRF evaluation the intermediate results remain encrypted due to homomorphic property of the encryption scheme. Thus neither the PRF input nor the PRF key is accessible to $\mathcal{A}$. This proves the obliviousness property of our TOPRF protocol. □

Although we provide an informal description of our security analysis, a formal proof of security of our proposed TOPRF protocol is discussed in Appendix A.4.

Algorithm 1 Software implementation of ThAHE with (t, T)-threshold decryption

```
Input: $\mathsf{msg}_1 \in \{0,1\}$, $\mathsf{msg}_2 \in \{0,1\}$, $t \in \mathbb{N}$, $T \in \mathbb{N}$, $\mathcal{S}_t \subseteq \mathcal{S}$ s.t $|\mathcal{S}_t| = t$ and $|\mathcal{S} = T|$
Output: $\mathsf{outp} \leftarrow \mathsf{msg}_1 \oplus \mathsf{msg}_2$
1: $(\mathbf{LweSk}, \mathsf{LwePk}) \leftarrow \mathsf{LweKeyGen}(\mathsf{LweParams})$
2: $\mathsf{cipher}_1 \leftarrow \mathsf{AsymEncrypt}(\mathsf{msg}_1, \mathsf{LwePk})$
3: $\mathsf{cipher}_2 \leftarrow \mathsf{AsymEncrypt}(\mathsf{msg}_2, \mathsf{LwePk})$
4: $\mathsf{result\_cipher} \leftarrow \mathsf{HomAdd}(\mathsf{cipher}_1, \mathsf{cipher}_2)$
5: $\mathsf{ShareSecret}(t, T, \mathbf{LweSk})$        ▷ Now all parties get their key shares. Each party $i \in \mathcal{S}$
   calculates $outp$ on its own.
6: $\mathsf{outp} \leftarrow \mathsf{thresholdDecrypt}(\mathsf{result\_cipher}, \mathcal{S}_t, t, T, i)$
7: return outp
```

4 Implementation and Experimental Results

In this section, we describe the implementation of (t, T)-threshold decryption of our ThAHE scheme on an x86-based computing platform.

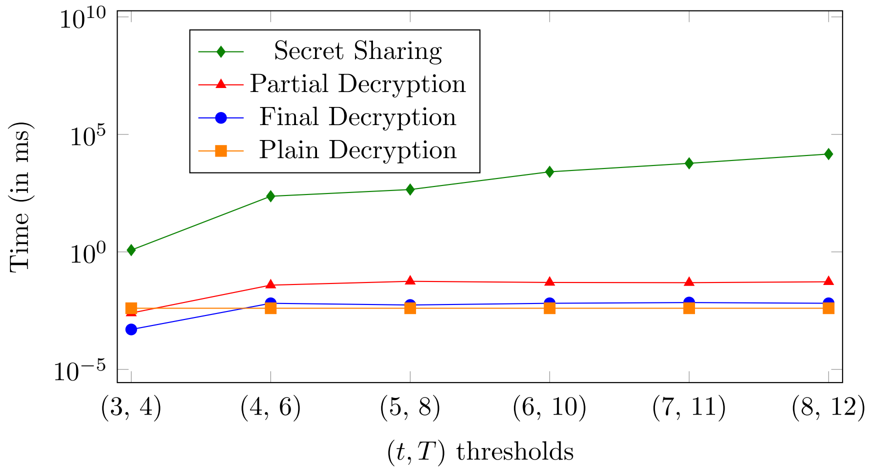

Figure 5: Secret sharing, threshold decryption, and plain decryption time. Note that the y-axis is in logarithmic scale.

4.1 Implementation Details

Our ThAHE scheme is implemented by following the Regev cryptosystem [Reg09], where the secret key **LweSK** and public key LwePK is generated based on LWE hardness assumption, thus taking the same parameter set from TFHE [CGGI20] library and our smudging noise parameter σ is chosen from [CSS+22], which is a TFHE-based threshold FHE scheme achieving polynomial modulus-to-noise ratio. According to Albrecht, Player, and Scott's Lattice Estimator [APS15] with the estimated complexity of attacking using these parameters is approximately 2^{128} operations. We then modified the decryption algorithms to support threshold decryption and for the key generation, we used a set of parameters as mentioned

Table 7: Comparison with some existing OPRF constructions.

Scheme	PRF	Assumption	Quantum-safe	Obliviousness	Threshold
TCRSTW21 [TCR+22]	DY variant	OM-Gap-sDH	X	✓	X
MPRSY20 [MPR+20]	DY	q-DHI	X	X	Distributed
ECSJR15 [ECS+15]	HashDH	OMBD-DH	X	✓	✓
JKKX17 [JKKX17]	2HashDH	Gap-OMDH	X	X	✓
BFHLY19 [BFH+20], DHL22 [DHL20]	2HashDH	Gap-OMBDH	X	✓	Distributed
JKR18 [JKR18]	Any	Gap-OMDH	X	✓	✓
ADDG23 [ADDG23]	Low-depth [BIP+18]	LWE	✓	✓	X
Ours	Any	LWE	✓	✓	✓

in Table 6. After that, the threshold secret sharing is performed by a trusted cloud server with sufficient computational resources. Subsequently, homomorphic evaluations are performed on encrypted data stored on the cloud server. In Algorithm 1 we briefly provide the implementation steps of our ThAHE scheme. Now, all these algorithms are implemented on a computing platform with an Intel(R) Xeon(R) CPU E5-2690 v4 CPU (2.60GHz clock-frequency), 28 physical cores, and 64GB RAM.

Remark on parameter choices. Considering the recent quantum attack on LWE [Che24] our present understanding is that the TFHE scheme is not immediately impacted due to a smaller modulus(q) to noise(e) ratio relative to n; however, the exact implications are yet to be analyzed in detail. Hence, for our threshold AHE scheme, our parameter set chooses $q = 2^{32}$, $n = 512$, $e = 2^{17}$ as in the original TFHE scheme. These parameters yield a ratio $\frac{q}{e} = 2^{15} \approx n^{1.67}$, considerably smaller than the attack requirement of $n^{4.5}$ as mentioned in by Yilei Chen [Che24].

4.2 Experimental Evaluation

Here, we discuss the experimental evaluation performance of our proposed ThAHE scheme.

Threshold Decryption vs. Plain Decryption. Now, we present our performance result using two 128-bit messages msg_1 and msg_2. Now, the user/client encrypts msg_1 and msg_2 with its public key LwePk using our defined AsymEncrypt function to produce cipher_1 and cipher_1. Now, we use our HomAddd function to perform homomorphic addition operation without a bootstrapping and produce resultant ciphertext result_cipher. We then perform (t, T)-threshold decryption on result_cipher according to the LISSS [CSS+22] secret sharing scheme. Some of the concrete sets of parameters used in our experiments are taken from the TFHE library as listed in Table 6.

Figure 5 shows the secret sharing time, partial decryption time, and final decryption time on a standard computing platform in terms of milliseconds. Figure 5 also shows that the time required for threshold decryption is only slightly higher than that of plain decryption using our ThAHE scheme. In other words, our proposed threshold decryption procedure incurs only minimal overhead over the plain decryption algorithm specified in the original Regev's scheme [Reg09]

Comparison with some previous works on TOPRF. Here, we discuss comparisons between our TOPRF protocol with some of the previous works on oblivious PRFs. Table 7 briefly presents these comparisons. Some of the previous OPRF constructions [TCR+22, MPR+20] uses Dodis-Yampolskiy (DY) PRF [DY05] and they rely on One-More Gap Strong Diffie-

Hellman Inversion (OM-Gap-sDH) or q-Diffie-Hellman Inversion (q-DHI) assumption. Few other works [ECS$^+$15, JKKX17, BFH$^+$20, DHL20] use Hashed Diffie-Hellman (HashDH) PRF under the idealized assumption that the hash function produces uniformly random elements from a group. These works rely on One-More Bilinear Decisional Diffie-Hellman (OMBD-DH) or, Gap One-More Diffie-Hellman (OMDH) assumption in the random-oracle model. Among these works, very few of them [ECS$^+$15, JKR18] support both the *obliviousness* and *thresholdization* property, while relying on quantum-unsafe assumptions such as variants of Diffie-Hellman. Whereas, in this work, we propose the first TOPRF protocol based on the quantum-safe LWE assumption.

5 Conclusion

In this work, we propose a novel threshold oblivious PRF (TOPRF) construction using a distributed cloud architecture that uses a threshold decryption paradigm using an LWE-based Regev's encryption scheme to perform an efficient homomorphic evaluation of symmetric cryptosystems using only an additive HE scheme. In particular, we use a Boolean representation of the AES-128 circuit, evaluated on an efficient additive encryption scheme for Boolean circuits. Furthermore, we propose several optimization techniques to reduce the online computation time and communication bandwidth between the cloud servers. These optimizations help to evaluate an AES-128 circuit in less than 1 second in the encrypted domain and achieve an efficient TOPRF construction using symmetric cryptosystems. The distributed cloud architecture emerges as a new research domain for optimizing FHE schemes and making them usable in practical applications.

References

[AAUC18] Abbas Acar, Hidayet Aksu, A Selcuk Uluagac, and Mauro Conti. A survey on homomorphic encryption schemes: Theory and implementation. *ACM Computing Surveys (Csur)*, 51(4):1–35, 2018.

[ADDG23] Martin R Albrecht, Alex Davidson, Amit Deo, and Daniel Gardham. Crypto dark matter on the torus: Oblivious prfs from shallow prfs and fhe. *Cryptology ePrint Archive*, 2023.

[AES03] Rakesh Agrawal, Alexandre Evfimievski, and Ramakrishnan Srikant. Information sharing across private databases. In *Proceedings of the 2003 ACM SIGMOD international conference on Management of data*, pages 86–97, 2003.

[AJLA$^+$12] 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 *Advances in Cryptology–EUROCRYPT 2012: 31st Annual International Conference on the Theory and Applications of Cryptographic Techniques, Cambridge, UK, April 15-19, 2012. Proceedings 31*, pages 483–501. Springer, 2012.

[AMMM18] Shashank Agrawal, Peihan Miao, Payman Mohassel, and Pratyay Mukherjee. Pasta: password-based threshold authentication. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, pages 2042–2059, 2018.

[APS15] Martin R Albrecht, Rachel Player, and Sam Scott. On the concrete hardness of learning with errors. *Journal of Mathematical Cryptology*, 9(3):169–203, 2015.

[BELO16] Aner Ben-Efraim, Yehuda Lindell, and Eran Omri. Optimizing semi-honest secure multiparty computation for the internet. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, pages 578–590, 2016.

[Ben94] Josh Benaloh. Dense probabilistic encryption. In *Proceedings of the workshop on selected areas of cryptography*, pages 120–128, 1994.

[BFH$^+$20] Carsten Baum, Tore Frederiksen, Julia Hesse, Anja Lehmann, and Avishay Yanai. Pesto: proactively secure distributed single sign-on, or how to trust a hacked server. In *2020 IEEE European Symposium on Security and Privacy (EuroS&P)*, pages 587–606. IEEE, 2020.

[BGG$^+$18] Dan Boneh, Rosario Gennaro, Steven Goldfeder, Aayush Jain, Sam Kim, Peter MR Rasmussen, and Amit Sahai. Threshold cryptosystems from threshold fully homomorphic encryption. In *Annual International Cryptology Conference*, pages 565–596. Springer, 2018.

[BGV14] Zvika Brakerski, Craig Gentry, and Vinod Vaikuntanathan. (leveled) fully homomorphic encryption without bootstrapping. *ACM Transactions on Computation Theory (TOCT)*, 6(3):1–36, 2014.

[BIP$^+$18] Dan Boneh, Yuval Ishai, Alain Passelègue, Amit Sahai, and David J Wu. Exploring crypto dark matter: New simple prf candidates and their applications. In *Theory of Cryptography Conference*, pages 699–729. Springer, 2018.

[BJMS18] Saikrishna Badrinarayanan, Aayush Jain, Nathan Manohar, and Amit Sahai. Threshold multi-key fhe and applications to round-optimal mpc. *IACR Cryptol. ePrint Arch*, 580:2018, 2018.

[BJSL11] Ali Bagherzandi, Stanislaw Jarecki, Nitesh Saxena, and Yanbin Lu. Password-protected secret sharing. In *Proceedings of the 18th ACM conference on Computer and Communications Security*, pages 433–444, 2011.

[BKR13] Mihir Bellare, Sriram Keelveedhi, and Thomas Ristenpart. Message-locked encryption and secure deduplication. In *Advances in Cryptology–EUROCRYPT 2013: 32nd Annual International Conference on the Theory and Applications of Cryptographic Techniques, Athens, Greece, May 26-30, 2013. Proceedings 32*, pages 296–312. Springer, 2013.

[BLMR13] Dan Boneh, Kevin Lewi, Hart Montgomery, and Ananth Raghunathan. Key homomorphic prfs and their applications. In *Annual Cryptology Conference*, pages 410–428. Springer, 2013.

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

[bri] KU Leuven COSIC. SCALE-MAMBA, 2019. https://github.com/ KULeuven-COSIC/SCALE-MAMBA.

[BS23] Katharina Boudgoust and Peter Scholl. Simple threshold (fully homomorphic) encryption from lwe with polynomial modulus. *Cryptology ePrint Archive*, 2023.

[CCK23] Jung Hee Cheon, Wonhee Cho, and Jiseung Kim. Improved universal thresholdizer from threshold fully homomorphic encryption. *Cryptology ePrint Archive*, 2023.

[CCS19] Hao Chen, Ilaria Chillotti, and Yongsoo Song. Multi-key homomorphic encryption from tfhe. In *International Conference on the Theory and Application of Cryptology and Information Security*, pages 446–472. Springer, 2019.

[CDCGS19] Jan Camenisch, Angelo De Caro, Esha Ghosh, and Alessandro Sorniotti. Oblivious prf on committed vector inputs and application to deduplication of encrypted data. In *International conference on financial cryptography and data security*, pages 337–356. Springer, 2019.

[CDKS21] Hao Chen, Wei Dai, Miran Kim, and Yongsoo Song. Efficient homomorphic conversion between (ring) lwe ciphertexts. In *International Conference on Applied Cryptography and Network Security*, pages 460–479. Springer, 2021.

[CGBH$^+$18] Hao Chen, Ran Gilad-Bachrach, Kyoohyung Han, Zhicong Huang, Amir Jalali, Kim Laine, and Kristin Lauter. Logistic regression over encrypted data from fully homomorphic encryption. Cryptology ePrint Archive, Report 2018/462, 2018. `https://eprint.iacr.org/2018/462`.

[CGGI20] Ilaria Chillotti, Nicolas Gama, Mariya Georgieva, and Malika Izabachène. Tfhe: fast fully homomorphic encryption over the torus. *Journal of Cryptology*, 33(1):34–91, 2020.

[Che24] Yilei Chen. Quantum algorithms for lattice problems. *Cryptology ePrint Archive*, 2024.

[CHL22] Sílvia Casacuberta, Julia Hesse, and Anja Lehmann. Sok: oblivious pseudorandom functions. In *2022 IEEE 7th European Symposium on Security and Privacy (EuroS&P)*, pages 625–646. IEEE, 2022.

[CJJ$^+$13] David Cash, Stanislaw Jarecki, Charanjit Jutla, Hugo Krawczyk, Marcel-Cătălin Roşu, and Michael Steiner. Highly-scalable searchable symmetric encryption with support for boolean queries. In *Advances in Cryptology–CRYPTO 2013: 33rd Annual Cryptology Conference, Santa Barbara, CA, USA, August 18-22, 2013. Proceedings, Part I*, pages 353–373. Springer, 2013.

[CKKS17] Jung Hee Cheon, Andrey Kim, Miran Kim, and Yongsoo Song. Homomorphic encryption for arithmetic of approximate numbers. In *International conference on the theory and application of cryptology and information security*, pages 409–437. Springer, 2017.

[CL17] Jan Camenisch and Anja Lehmann. Privacy for distributed databases via (un) linkable pseudonyms. *Cryptology ePrint Archive*, 2017.

[CNT12] Jean-Sébastien Coron, David Naccache, and Mehdi Tibouchi. Public key compression and modulus switching for fully homomorphic encryption over the integers. In David Pointcheval and Thomas Johansson, editors, *EUROCRYPT 2012*, volume 7237 of *LNCS*, pages 446–464. Springer, Heidelberg, April 2012.

[CSS$^+$22] 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*, 2022.

[DCT10] Emiliano De Cristofaro and Gene Tsudik. Practical private set intersection protocols with linear complexity. In *Financial Cryptography and Data Security: 14th International Conference, FC 2010, Tenerife, Canary Islands, January 25-28, 2010, Revised Selected Papers 14*, pages 143–159. Springer, 2010.

[DF90] Yvo Desmedt and Yair Frankel. Threshold cryptosystems. in (brassard, gilles, hrsg.): Advances in cryptology-crypto š89. jgg. 435 in lecture notes in computer science, 1990.

[DHL20] Poulami Das, Julia Hesse, and Anja Lehmann. Dpase: Distributed password-authenticated symmetric encryption. *Cryptology ePrint Archive*, 2020.

[DHL22] Poulami Das, Julia Hesse, and Anja Lehmann. Dpase: Distributed password-authenticated symmetric-key encryption, or how to get many keys from one password. In *Proceedings of the 2022 ACM on Asia Conference on Computer and Communications Security*, pages 682–696, 2022.

[DM15a] 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, Heidelberg, April 2015.

[DM15b] Léo Ducas and Daniele Micciancio. Fhew: bootstrapping homomorphic encryption in less than a second. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 617–640. Springer, 2015.

[DR99] Joan Daemen and Vincent Rijmen. Aes proposal: Rijndael. 1999.

[DSDFY94] Alfredo De Santis, Yvo Desmedt, Yair Frankel, and Moti Yung. How to share a function securely. In *Proceedings of the twenty-sixth annual ACM symposium on Theory of computing*, pages 522–533, 1994.

[DT06] Ivan Damgård and Rune Thorbek. Linear integer secret sharing and distributed exponentiation. In *International Workshop on Public Key Cryptography*, pages 75–90. Springer, 2006.

[DY05] Yevgeniy Dodis and Aleksandr Yampolskiy. A verifiable random function with short proofs and keys. In *International Workshop on Public Key Cryptography*, pages 416–431. Springer, 2005.

[ECS$^+$15] Adam Everspaugh, Rahul Chaterjee, Samuel Scott, Ari Juels, and Thomas Ristenpart. The pythia {PRF} service. In *24th USENIX Security Symposium (USENIX Security 15)*, pages 547–562, 2015.

[FHV18] Sebastian Faust, Carmit Hazay, and Daniele Venturi. Outsourced pattern matching. *International Journal of Information Security*, 17(3):327–346, 2018.

[FIPR05] Michael J Freedman, Yuval Ishai, Benny Pinkas, and Omer Reingold. Keyword search and oblivious pseudorandom functions. In *TCC*, volume 3378, pages 303–324. Springer, 2005.

[FK00] Warwick Ford and Burton S Kaliski. Server-assisted generation of a strong secret from a password. In *Proceedings IEEE 9th International Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprises (WET ICE 2000)*, pages 176–180. IEEE, 2000.

[FSK$^+$21] Axel Feldmann, Nikola Samardzic, Aleksandar Krastev, Srini Devadas, Ron Dreslinski, Karim Eldefrawy, Nicholas Genise, Christopher Peikert, and Daniel Sanchez. F1: A Fast and Programmable Accelerator for Fully Homomorphic Encryption (Extended Version). *arXiv preprint arXiv:2109.05371*, 2021.

[GBOW88] S Goldwasser, M Ben-Or, and A Wigderson. Completeness theorems for non-cryptographic fault-tolerant distributed computing. In *Proc. of the 20th STOC*, pages 1–10, 1988.

[Gen09] Craig Gentry. Fully homomorphic encryption using ideal lattices. In *Proceedings of the forty-first annual ACM symposium on Theory of computing*, pages 169–178, 2009.

[GHS12] Craig Gentry, Shai Halevi, and Nigel P Smart. Fully homomorphic encryption with polylog overhead. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 465–482. Springer, 2012.

[GM19] Shafi Goldwasser and Silvio Micali. Probabilistic encryption & how to play mental poker keeping secret all partial information. In *Providing sound foundations for cryptography: on the work of Shafi Goldwasser and Silvio Micali*, pages 173–201. 2019.

[HAP18] Yotam Harchol, Ittai Abraham, and Benny Pinkas. Distributed ssh key management with proactive rsa threshold signatures. In *Applied Cryptography and Network Security: 16th International Conference, ACNS 2018, Leuven, Belgium, July 2-4, 2018, Proceedings 16*, pages 22–43. Springer, 2018.

[Hay08] Brian Hayes. Cloud computing, 2008.

[HFH99] Bernardo A Huberman, Matt Franklin, and Tad Hogg. Enhancing privacy and trust in electronic communities. In *Proceedings of the 1st ACM conference on Electronic commerce*, pages 78–86, 1999.

[HHNZ19] Marcella Hastings, Brett Hemenway, Daniel Noble, and Steve Zdancewic. Sok: General purpose compilers for secure multi-party computation. In *2019 IEEE symposium on security and privacy (SP)*, pages 1220–1237. IEEE, 2019.

[HL08] Carmit Hazay and Yehuda Lindell. Efficient protocols for set intersection and pattern matching with security against malicious and covert adversaries. In *Theory of Cryptography: Fifth Theory of Cryptography Conference, TCC 2008, New York, USA, March 19-21, 2008. Proceedings 5*, pages 155–175. Springer, 2008.

[HL10] Carmit Hazay and Yehuda Lindell. Efficient protocols for set intersection and pattern matching with security against malicious and covert adversaries. *Journal of cryptology*, 23(3):422–456, 2010.

[Jag12] Tibor Jager. The generic composite residuosity problem. In *Black-Box Models of Computation in Cryptology*, pages 49–56. Springer, 2012.

[JJK$^+$13] Stanislaw Jarecki, Charanjit Jutla, Hugo Krawczyk, Marcel Rosu, and Michael Steiner. Outsourced symmetric private information retrieval. In *Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security*, pages 875–888, 2013.

[JKK14] Stanislaw Jarecki, Aggelos Kiayias, and Hugo Krawczyk. Round-optimal password-protected secret sharing and t-pake in the password-only model. In *Advances in Cryptology–ASIACRYPT 2014: 20th International Conference on the Theory and Application of Cryptology and Information Security, Kaoshiung, Taiwan, ROC, December 7-11, 2014, Proceedings, Part II 20*, pages 233–253. Springer, 2014.

[JKKX17] Stanisław Jarecki, Aggelos Kiayias, Hugo Krawczyk, and Jiayu Xu. Toppss: cost-minimal password-protected secret sharing based on threshold oprf. In *Applied Cryptography and Network Security: 15th International Conference, ACNS 2017, Kanazawa, Japan, July 10-12, 2017, Proceedings 15*, pages 39–58. Springer, 2017.

[JKR18] Stanislaw Jarecki, Hugo Krawczyk, and Jason Resch. Threshold partially-oblivious prfs with applications to key management. *Cryptology ePrint Archive*, 2018.

[JKR19] Stanislaw Jarecki, Hugo Krawczyk, and Jason Resch. Updatable oblivious key management for storage systems. In *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*, pages 379–393, 2019.

[JKX18] Stanislaw Jarecki, Hugo Krawczyk, and Jiayu Xu. Opaque: an asymmetric pake protocol secure against pre-computation attacks. In *Advances in Cryptology–EUROCRYPT 2018: 37th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Tel Aviv, Israel, April 29-May 3, 2018 Proceedings, Part III 37*, pages 456–486. Springer, 2018.

[JL09] Stanisław Jarecki and Xiaomin Liu. Efficient oblivious pseudorandom function with applications to adaptive ot and secure computation of set intersection. In *Theory of Cryptography: 6th Theory of Cryptography Conference, TCC 2009, San Francisco, CA, USA, March 15-17, 2009. Proceedings 6*, pages 577–594. Springer, 2009.

[JL10] Stanisław Jarecki and Xiaomin Liu. Fast secure computation of set intersection. In *Security and Cryptography for Networks: 7th International Conference, SCN 2010, Amalfi, Italy, September 13-15, 2010. Proceedings 7*, pages 418–435. Springer, 2010.

[JRS17] Aayush Jain, Peter MR Rasmussen, and Amit Sahai. Threshold fully homomorphic encryption. *Cryptology ePrint Archive*, 2017.

[KJY$^+$20] Eunkyung Kim, Jinhyuck Jeong, Hyojin Yoon, Younghyun Kim, Jihoon Cho, and Jung Hee Cheon. How to securely collaborate on data: Decentralized threshold he and secure key update. *IEEE Access*, 8:191319–191329, 2020.

[KKRT16] Vladimir Kolesnikov, Ranjit Kumaresan, Mike Rosulek, and Ni Trieu. Efficient batched oblivious prf with applications to private set intersection. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, pages 818–829, 2016.

[KÖA23] Jakub Klemsa, Melek Önen, and Yavuz Akın. A practical tfhe-based multi-key homomorphic encryption with linear complexity and low noise growth. *Cryptology ePrint Archive*, 2023.

[KRS$^+$19] Daniel Kales, Christian Rechberger, Thomas Schneider, Matthias Senker, and Christian Weinert. Mobile private contact discovery at scale. In *28th USENIX Security Symposium (USENIX Security 19)*, pages 1447–1464, 2019.

[KS08] Vladimir Kolesnikov and Thomas Schneider. Improved garbled circuit: Free xor gates and applications. In *Automata, Languages and Programming: 35th International Colloquium, ICALP 2008, Reykjavik, Iceland, July 7-11, 2008, Proceedings, Part II 35*, pages 486–498. Springer, 2008.

[LATV12] Adriana López-Alt, Eran Tromer, and Vinod Vaikuntanathan. On-the-fly multiparty computation on the cloud via multikey fully homomorphic encryption. In *Proceedings of the forty-fourth annual ACM symposium on Theory of computing*, pages 1219–1234, 2012.

[LPR10] Vadim Lyubashevsky, Chris Peikert, and Oded Regev. On ideal lattices and learning with errors over rings. In *Annual international conference on the theory and applications of cryptographic techniques*, pages 1–23. Springer, 2010.

[LZQ23] Feng-Hao Liu, En Zhang, and Leiyong Qin. Efficient multiparty probabilistic threshold private set intersection. In *Proceedings of the 2023 ACM SIGSAC Conference on Computer and Communications Security*, pages 2188–2201, 2023.

[MPR$^+$20] Peihan Miao, Sarvar Patel, Mariana Raykova, Karn Seth, and Moti Yung. Two-sided malicious security for private intersection-sum with cardinality. In *Annual International Cryptology Conference*, pages 3–33. Springer, 2020.

[MW16] Pratyay Mukherjee and Daniel Wichs. Two round multiparty computation via multi-key fhe. In *Advances in Cryptology–EUROCRYPT 2016: 35th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Vienna, Austria, May 8-12, 2016, Proceedings, Part II 35*, pages 735–763. Springer, 2016.

[NR04] Moni Naor and Omer Reingold. Number-theoretic constructions of efficient pseudo-random functions. *Journal of the ACM (JACM)*, 51(2):231–262, 2004.

[Pai99] Pascal Paillier. Public-key cryptosystems based on composite degree residuosity classes. In *International conference on the theory and applications of cryptographic techniques*, pages 223–238. Springer, 1999.

[Reg09] Oded Regev. On lattices, learning with errors, random linear codes, and cryptography. *Journal of the ACM (JACM)*, 56(6):1–40, 2009.

[Sch18] Peter Scholl. Extending oblivious transfer with low communication via key-homomorphic prfs. In *Public-Key Cryptography–PKC 2018: 21st IACR International Conference on Practice and Theory of Public-Key Cryptography, Rio de Janeiro, Brazil, March 25-29, 2018, Proceedings, Part I 21*, pages 554–583. Springer, 2018.

[Sha79] Adi Shamir. How to share a secret. *Communications of the ACM*, 22(11):612–613, 1979.

[SS10] Damien Stehlé and Ron Steinfeld. Faster fully homomorphic encryption. In Masayuki Abe, editor, *ASIACRYPT 2010*, volume 6477 of *LNCS*, pages 377–394. Springer, Heidelberg, December 2010.

[STH+23] Yukimasa Sugizaki, Hikaru Tsuchida, Takuya Hayashi, Koji Nuida, Akira Nakashima, Toshiyuki Isshiki, and Kengo Mori. Threshold fully homomorphic encryption over the torus. In *European Symposium on Research in Computer Security*, pages 45–65. Springer, 2023.

[TCR+22] Nirvan Tyagi, Sofía Celi, Thomas Ristenpart, Nick Sullivan, Stefano Tessaro, and Christopher A Wood. A fast and simple partially oblivious prf, with applications. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 674–705. Springer, 2022.

[WRK17] Xiao Wang, Samuel Ranellucci, and Jonathan Katz. Authenticated garbling and efficient maliciously secure two-party computation. In *Proceedings of the 2017 ACM SIGSAC conference on computer and communications security*, pages 21–37, 2017.

[WVLY+10] Lizhe Wang, Gregor Von Laszewski, Andrew Younge, Xi He, Marcel Kunze, Jie Tao, and Cheng Fu. Cloud computing: a perspective study. *New generation computing*, 28(2):137–146, 2010.

[Yao86] Andrew Chi-Chih Yao. How to generate and exchange secrets. In *27th Annual Symposium on Foundations of Computer Science (sfcs 1986)*, pages 162–167. IEEE, 1986.

[ZMI88] Yuliang Zheng, Tsutomu Matsumoto, and Hideki Imai. Cryptographic applications of 7th-residuosity problem with 7 an odd integer. *Yokohama National University, Japan*, 1988.

A Some more Preliminary Definitions

Here, we describe some additional definitions used in the paper.

A.1 LWE, RLWE & RGSW

The FHE scheme presented in [CGGI20] is based on the Learning with Error (LWE) problem and its ring variant (RLWE). In this section, we will briefly discuss these two problems. Instead of working over integer modulo q i.e., $\mathbb{Z}/q\mathbb{Z}$ or over the ring $\mathbb{Z}[X]/\left(X^Z+1\right)$ modulo q in the ring variant, the TFHE scheme works over the real Torus $\mathbb{T} = \mathbb{R} \bmod 1$ and over $T = \mathbb{T}[X]/\left(X^Z+1\right)$, the the set of cyclotomic polynomials over $\mathbb{T}$ for a power-of-two integer Z.

LWE. A LWE sample is a pair $(b, \mathbf{a}) \in \mathbb{T}^{n+1}$, where $\mathbf{a}$ is sampled from a uniform distribution over $\mathbb{T}^n$ and $b = \langle \mathbf{a}, \mathbf{sk} \rangle + e$. Here, $\mathbf{sk}$ is the LWE secret key, sampled from an uniform key distribution $\mathcal{K} = \{0,1\}^n$ and the LWE noise e sampled from a Gaussian distribution $\mathcal{G}_\alpha$ with standard deviation $\alpha > 0$.

RLWE. Similarly, a RLWE sample is a pair of polynomials $(b, a) \in T^2$, where a is sampled uniformly from T and $b = a \cdot z + e \pmod 1$. Here, "$\cdot$" denotes polynomial multiplication and z is the RLWE secret which is an integer polynomial of degree Z sampled from a key distribution ψ on $R = \mathbb{Z}[X]/\left(X^Z + 1\right)$, the set of cyclotomic polynomials over $\mathbb{Z}$. Now, the error polynomial e is sampled from a Gaussian distribution with standard deviation $\beta > 0$. We will set ψ as the uniform distribution on the set of polynomials of R with binary coefficients in $\{0, 1\}$.

Now, we will define two problems for both LWE and RLWE:

- *Decision problem:* For a fixed LWE secret $\mathbf{s}$ (resp. RLWE secret z), distinguish the uniform distribution over $\mathbb{T}^{n+1}$ (resp. T^2) from the LWE (resp. RLWE) samples.
- *Search problem:* Given arbitrary many samples from LWE (resp. RLWE) distribution, find the secret $\mathbf{s}$ (resp. z).

RGSW. We can define the RGSW samples as the Torus variant of RGSW [DM15b] samples in the same way in LWE and RLWE. For a fixed RLWE secret $z \in R$, we define a RGSW samples as $\mathbf{C} = \mathbf{Z} + \mu \cdot \mathbf{G}_2$, where each line of the matrix $\mathbf{Z} \in T^{d\times 2}$ is a RLWE encryption of 0, $\mathbf{G}_2$ is the gadget matrix and the message $\mu \in R$ is an integer polynomial. RGSW samples are homomorphic with respect to addition and internal multiplication.

A.2 Evaluating One Boolean gate without Bootstrapping

In this section, we discuss why in the TFHE scheme a homomorphic Boolean gate evaluation requires no bootstrapping. That means, that although TFHE supports gate-bootstrapping, we can evaluate at most one Boolean gate without performing a bootstrapping. Here, we will elaborate on this reason using XOR gate in from TFHE as an example. Let us assume, $\mathbf{ct}_1 = (b_1, \mathbf{a}_1)$ and $\mathbf{ct}_2 = (b_2, \mathbf{a}_2)$ are two LWE ciphertexts encrypting $m_1, m_2 \in \{0, 1\}$ respectively. According to the TFHE library, the plaintexts m_1 and m_2 are scaled in the range $[-\frac{1}{8}, \frac{1}{8}]$. Assume, Δ be the scaling function that maps $\Delta(0) = -\frac{1}{8}$ and $\Delta(1) = \frac{1}{8}$. Hence, we have

$$b_1 = \langle \mathbf{a}_1, \mathbf{sk}\rangle + \Delta(m_1) + e_1, \quad b_2 = \langle \mathbf{a}_2, \mathbf{sk}\rangle + \Delta(m_2) + e_2$$

where, $\mathbf{sk} \in \{0, 1\}^n$ be the secret key and $e_1, e_2 \in \mathcal{G}_\alpha$ are the encryption noise sampled from a Gaussian distribution with standard deviation α. For decryption correctness the noises should satisfy $\|e_1\|_\infty, \|e_1\|_\infty < \frac{1}{16}$. We now demonstrate the homomorphic XOR operation between $\mathbf{ct}_1$ and $\mathbf{ct}_2$,

$$\mathsf{HomXOR}(\mathbf{ct}_1, \mathbf{ct}_2) = \left(\frac{1}{4}, \mathbf{0}\right) + 2 \cdot (\mathbf{ct}_1 + \mathbf{ct}_2).$$

Let, $\mathbf{ct}^\star = (b^\star, \mathbf{a}^\star)$ be the output of $\mathsf{HomXOR}(\mathbf{ct}_1, \mathbf{ct}_2)$. Hence, we have

$$\mathbf{a}^\star = 2 \cdot (\mathbf{a}_1 + \mathbf{a}_2),$$

$$b^\star = \frac{1}{4} + \langle \mathbf{a}^\star, \mathbf{sk}\rangle + 2 \cdot (\Delta(m_1) + \Delta(m_2)) + 2 \cdot (e_1 + e_2).$$

Let us analyze what will happen if we perform decryption of $\mathbf{ct}^\star$ without performing the bootstrapping over $\mathbf{ct}^\star$. To decrypt $\mathbf{ct}^\star$, first compute a phase $\phi = b^\star - \langle \mathbf{a}^\star, \mathbf{sk}\rangle \pmod 1$, now if $\phi > 0$ return 1, otherwise return 0. Consider the following analysis of the decryption operation with $m_1 = 0, m_2 = 0$, i.e., $\Delta(m_1) = -\frac{1}{8}, \Delta(m_2) = -\frac{1}{8}$ and the encryption noises with the same sign, i.e., either $0 \leq e_1, e_2 < \frac{1}{16}$, or $-\frac{1}{16} < e_1, e_2 \leq 0$,

$$\begin{aligned}
\phi &= \frac{1}{4} + 2 \cdot (\Delta(m_1) + \Delta(m_2)) + 2 \cdot (e_1 + e_2), \\
&= \frac{1}{4} + 2 \cdot \left(-\frac{1}{8} - \frac{1}{8}\right) + 2 \cdot (e_1 + e_2), \\
&= -\frac{1}{4} + 2 \cdot (e_1 + e_2), \\
&= -\frac{1}{4} + \delta \quad \left[0 \leq \delta < \frac{1}{4}, \text{ for } 0 \leq e_1, e_2 < \frac{1}{16}\right], \\
&= \delta' \quad \left[-\frac{1}{4} < \delta' < 0 \text{ for both } m_1 = 0, m_2 = 0\right].
\end{aligned}$$

Now, with another extreme scenario when both the noises are negative,

$$\begin{aligned}
&= -\frac{1}{4} + 2 \cdot (e_1 + e_2), \\
&= -\frac{1}{4} + \delta \quad \left[0 \geq \delta > -\frac{1}{4}, \text{ for } -\frac{1}{16} < e_1, e_2 \leq 0\right], \\
&= \delta' \quad \left[-\frac{1}{2} < \delta' < -\frac{1}{4} \text{ for both } m_1 = 0, m_2 = 0\right]
\end{aligned}$$

In both of these cases, $-\frac{1}{2} < \phi < 0$, i.e., $\phi \pmod 1 = \phi < 0$, hence the decryption of XOR upon $m_1 = 0$ and $m_2 = 0$ is 0, which is a correct decryption. Now, consider when both the plaintexts are $m_1 = m_2 = 1$, thus $\Delta(m_1) = \Delta(m_2) = \frac{1}{8}$,

$$\begin{aligned}
\phi &= \frac{1}{4} + 2 \cdot (\Delta(m_1) + \Delta(m_2)) + 2 \cdot (e_1 + e_2), \\
&= \frac{1}{4} + 2 \cdot \left(\frac{1}{8} + \frac{1}{8}\right) + 2 \cdot (e_1 + e_2), \\
&= \frac{1}{4} + \frac{1}{2} + 2 \cdot (e_1 + e_2), \\
&= \frac{1}{4} + \frac{1}{2} + \delta \quad \left[0 \leq \delta < \frac{1}{4}, \text{ for } 0 \leq e_1, e_2 < \frac{1}{16}\right], \\
&= \delta' \quad \left[\frac{3}{4} < \delta' < 1 \text{ for both } m_1 = 1, m_2 = 1\right].
\end{aligned}$$

Now, with the other extreme case when both the noises are negative,

$$\begin{aligned}
&= \frac{1}{4} + \frac{1}{2} + 2 \cdot (e_1 + e_2), \\
&= \frac{1}{4} + \frac{1}{2} + \delta \quad \left[0 \geq \delta > -\frac{1}{4}, \text{ for } -\frac{1}{16} < e_1, e_2 \leq 0\right], \\
&= \delta' \quad \left[\frac{1}{2} < \delta' < \frac{3}{4} \text{ for both } m_1 = 1, m_2 = 1\right].
\end{aligned}$$

In both of these cases, $\frac{1}{2} < \phi < 1$, i.e., $\phi \pmod 1 = \phi - 1 < 0$ hence the decryption of XOR upon $m_1 = 1$ and $m_2 = 1$ is 0, which is a correct decryption. We now consider the case when both the plaintexts are of opposite sign, without loss of generality consider $m_1 = 0, m_2 = 1$, and $\Delta(m_1) = -\frac{1}{8}$ and $\Delta(m_2) = \frac{1}{8}$,

$$\begin{aligned}
\phi &= \frac{1}{4} + 2 \cdot (\Delta(m_1) + \Delta(m_2)) + 2 \cdot (e_1 + e_2), \\
&= \frac{1}{4} + 2 \cdot \left(-\frac{1}{8} + \frac{1}{8}\right) + 2 \cdot (e_1 + e_2), \\
&= \frac{1}{4} + 2 \cdot (e_1 + e_2), \\
&= \frac{1}{4} + \delta \quad \left[0 \leq \delta < \frac{1}{4}, \text{ for } 0 \leq e_1, e_2 < \frac{1}{16}\right], \\
&= \delta' \quad \left[\frac{1}{4} < \delta' < \frac{1}{2} \text{ for both } m_1 = 0, m_2 = 1\right].
\end{aligned}$$

Again considering the other extreme choices when both the noises are negative,

$$\begin{aligned}
&= \frac{1}{4} + 2 \cdot (e_1 + e_2), \\
&= \frac{1}{4} + \delta \quad \left[0 \geq \delta > -\frac{1}{4}, \text{ for } -\frac{1}{16} < e_1, e_2 \leq 0\right], \\
&= \delta' \quad \left[0 < \delta' < \frac{1}{4}\right].
\end{aligned}$$

In both of these cases, $0 < \phi < \frac{1}{2}$, i.e., $\phi \pmod 1 = \phi > 0$; hence the decryption of XOR upon $m_1 = 0$ and $m_2 = 1$ is 1, which is a correct decryption. So far, we have demonstrated that the extreme choices of encryption noises i.e., when they are of the same sign we can achieve correct decryption without the bootstrapping operation and it is trivial to show similar cases when noises are of the opposite sign; that is the value of ϕ in each of the above cases will remain in the required range. A similar approach as mentioned above can also be achieved for homomorphic AND operation on $\mathbf{ct}_1$ and $\mathbf{ct}_2$, i.e., $\mathsf{HomAND}(\mathbf{ct}_1, \mathbf{ct}_2) = \left(-\frac{1}{8}, \mathbf{0}\right) + (\mathbf{ct}_1 + \mathbf{ct}_2)$.

A.3 Bristol Representation of AES

In this section, we will have a closer look into the Bristol representation of AES-128 specifically, presented in Listing 1. A Bristol Format file starts with a line that defines the number of

Boolean gates and the number of wires in the Boolean circuit. The second line symbolizes the first two numbers as the number of input wires to the function with its corresponding number of output wires. From the third line onwards, each line represents a binary gate and its input/output specifications. The first two values represent the number of input and output respectively. The next two values are input wire numbers, which are fed as the inputs of the binary gate; after evaluation, the result is written in the next wire number, which is the output wire. For example, consider the AES-128 circuit presented in Listing 1.

For example, in the 1^{st} line, '33616' and '33872' denote the number of gates and the wires respectively. In 2^{nd} line, the first two values i.e., '128' and '128' refer to the size of the two inputs, and the third value '128' is the size of the output. Now, in 3^{rd} line the last entry represents XOR operation taking the third and the fourth values i.e., '226' and '229' as input wires, and the result after homomorphic XOR operation is written in the output wire number '33736'. The first two values '2' and '1' denote the number of inputs and the number of outputs. The Bristol Formats are known to be TFHE-friendly as it is represented using only binary gates.

Listing 1: Bristol Format of AES-128

```
33616 33872
128 128 128

2 1 226 229 33736 XOR
2 1 178 50 33663 XOR
          ⋮
2 1 32888 33064 33031 AND
1 1 32898 33141 INV
          ⋮
2 1 268 269 257 XOR
2 1 256 257 33749 XOR
```

A.4 Security Analysis of TOPRF Protocol

Here, we present a detailed analysis of security by proving both the unpredictability and obliviousness properties of our proposed TOPRF protocol. We define several hybrids describing the different views of a semi-honest PPT adversary and show indistinguishability between the hybrids to prove the security of the underlying TOPRF protocol. In the hybrids, with the term "simulated" key shares, we refer to shares of the zero-vector $\vec{\mathbf{0}}$.

Proof of unpredictability/pseudo-randomness. Now, we prove the unpredictability property of our TOPRF using some hybrids defined as follows,

Hybrid$_0$–

1. On input the security parameter λ, the adversary $\mathcal{A}$ outputs an access structure $\mathbb{A} \in \{S_1, \dots, S_T\}$.
2. Run $\mathsf{ThAHE.KeyGen}(params)$ and returns $(\mathsf{pk}, \mathbf{sk}_1, \dots, \mathbf{sk}_T)$. Now, $\mathcal{A}$ is given pk.

3. $\mathcal{A}$ outputs a set $\mathcal{S}_c \subset \{S_1, \ldots, S_T\}$ such that $\mathcal{S}_c \notin \mathbb{A}$. The adversary is given $\{\mathbf{sk}_j\}_{j \in \mathcal{S}_c}$.

4. $\mathcal{A}$ also has access to $\{\mathbf{ct}_i\}_{i\in Q} \leftarrow \{\mathsf{ThAHE.Enc}(z_i, \mathsf{pk})\}_{i\in Q}$, where Q is a set containing the client's (PRF) input, the server's (PRF) key, and the random masking bits, i.e., $|Q| = 2(N + L)$, N be the size of the underlying PRF input/key and L be the number of random-bit pairs required for masking. Therefore, the adversary has encryptions of the PRF input, PRF key, and the random masking bits.

5. $\mathcal{A}$ is handed over $\gamma_j^i \leftarrow \mathsf{ThAHE.PartDec}(\mathbf{ct}_i, \mathsf{sk}_j)$, $\forall i \in [Q], j \in \mathcal{S}_c$.

6. $\mathcal{A}$ also gets the masked plaintext bits $\{\hat{m}_0^i, \hat{m}_1^i\}_{i\in[L]}$ after the threshold decryption operation performed as a part of our TOPRF protocol.

7. Finally, $\mathcal{A}$ is provided with the original PRF output $\mathsf{out} \in \{0,1\}^N$, after performing the final threshold decryption.

Briefly saying the $\mathbf{Hybrid}_0$ represents the view of $\mathcal{A}$ that includes the real secret key shares and encryptions of the original PRF input and key.

$\underline{\mathbf{Hybrid}_1}$– This hybrid is the same as $\mathbf{Hybrid}_0$ except for the following steps,

3. $\mathcal{A}$ outputs a set $\mathcal{S}_c \subset \{S_1, \ldots, S_T\}$ such that $\mathcal{S}_c \notin \mathbb{A}$. The adversary is given the simulated secret shares $\{\mathbf{sk}'_j\}_{j\in\mathcal{S}_c}$.

5. The adversary $\mathcal{A}$ is now given the simulated partial decryptions $\gamma_j^{i\prime} \leftarrow \mathsf{ThAHE.PartDec}(\mathbf{ct}_i, \mathsf{sk}'_j)$, $\forall i \in [Q], j \in \mathcal{S}_c$.

The $\mathbf{Hybrid}_1$ represents the view of $\mathcal{A}$ that comprises the simulated secret key shares, i.e., shares of $\vec{\mathbf{0}}$ and ciphertexts corresponding to the original PRF input and key.

$\underline{\mathbf{Hybrid}_2}$– This hybrid is the same as $\mathbf{Hybrid}_1$ except for the following steps,

4. $\mathcal{A}$ is now provided with $\{\mathbf{ct}'_i\}_{i\in Q} \leftarrow \{\mathsf{ThAHE.Enc}(0, \mathsf{pk})\}_{i\in Q^K}$, where Q^K denotes the PRF key, and $|Q^K| = N$. Therefore, encryption of the PRF key is now replaced with random encryptions of 0.

The $\mathbf{Hybrid}_2$ represents the view of $\mathcal{A}$ that includes the simulated secret key shares, i.e., shares of $\vec{\mathbf{0}}$ and random ciphertexts (encryptions of 0) in place of the encryptions of original PRF key.

$\underline{\mathbf{Hybrid}_3}$– This hybrid is the same as $\mathbf{Hybrid}_2$ except for the following steps,

7. $\mathcal{A}$ is now provided with random PRF output $\mathsf{out}' \in \{0,1\}^N$, sampled uniformly from the output space of the PRF.

Indistinguishability argument. The only difference in the view of the adversary $\mathcal{A}$ from $\mathbf{Hybrid}_0$ to $\mathbf{Hybrid}_1$ is that the partial decryptions $\gamma_j^{i\prime}$ are generated by simulated shares using a maximal invalid share set $\mathcal{S}_c$ of the secret key shares rather than by the real partial decryptions γ_j^i with access to the real secret key shares. By the properties of LISSS scheme [DT06], these two hybrids, i.e., $\mathbf{Hybrid}_0$ and $\mathbf{Hybrid}_1$ are statistically indistinguishable. Now, considering the hybrids $\mathbf{Hybrid}_1$ and $\mathbf{Hybrid}_2$, the only change in the view of

$\mathcal{A}$ is that the ciphertexts corresponding the PRF key are random encryptions of 0 rather than the original PRF/server's key. According to the IND-CPA security [JRS17, BGG$^+$18] of our underlying homomorphic encryption scheme, the view of $\mathcal{A}$ in $\mathbf{Hybrid}_1$ and $\mathbf{Hybrid}_2$ are computationally indistinguishable. Finally, the indistinguishability between the hybrids $\mathbf{Hybrid}_2$ and $\mathbf{Hybrid}_3$ relies on the pseudorandomness property of the underlying PRF (in our case, AES), i.e., the output of the PRF evaluation is computationally indistinguishable from random. Therefore, we claim that for a semi-honest PPT adversary $\mathcal{A}$ these four hybrids are indistinguishable. Hence, in the view of $\mathcal{A}$ the output of our TOPRF is indistinguishable from random, this proves the unpredictability property of our TOPRF protocol.

Proof of obliviousness. We prove the obliviousness property of our TOPRF in a similar way using some hybrids defined as follows,

$\underline{\mathbf{Hybrid}_0}$–

1. On input the security parameter λ, the adversary $\mathcal{A}$ outputs an access structure $\mathbb{A} \in \{S_1, \ldots, S_T\}$.
2. Run $\mathsf{ThAHE.KeyGen}(params)$ and returns $(\mathsf{pk}, \mathbf{sk}_1, \ldots, \mathbf{sk}_T)$. $\mathcal{A}$ is given pk.
3. $\mathcal{A}$ outputs a set $\mathcal{S}_c \subset \{S_1, \ldots, S_T\}$ such that $\mathcal{S}_c \notin \mathbb{A}$. The adversary is given $\{\mathbf{sk}_j\}_{j \in \mathcal{S}_c}$.
4. $\mathcal{A}$ also has access to $\{\mathbf{ct}_i\}_{i \in Q} \leftarrow \{\mathsf{ThAHE.Enc}(z_i, \mathsf{pk})\}_{i \in Q}$, where Q is a set containing the client's (PRF) input, the server's (PRF) key, and the random masking bits, i.e., $|Q| = 2(N + L)$, N is the underlying PRF input/key size and L is the number of random-bit pairs required for masking. Therefore, the adversary has encryptions of the PRF input, PRF key, and the random masking bits.
5. $\mathcal{A}$ is now handed over $\gamma_j^i \leftarrow \mathsf{ThAHE.PartDec}(\mathbf{ct}_i, \mathsf{sk}_j)$, $\forall i \in [Q], j \in \mathcal{S}_c$.
6. $\mathcal{A}$ also gets the masked plaintext bits $\{\hat{m}_0^i, \hat{m}_1^i\}_{i \in [L]}$ after the threshold decryption operation performed in our TOPRF protocol.
7. Finally, $\mathcal{A}$ is provided with the original PRF output $\mathsf{out} \in \{0,1\}^N$, after performing the final threshold decryption.

The $\mathbf{Hybrid}_0$ is similar to the previous case, which represents the view of $\mathcal{A}$ that includes the real secret key shares and encryptions of the original PRF input and key.

$\underline{\mathbf{Hybrid}_1}$– This hybrid is the same as $\mathbf{Hybrid}_0$ except for the following steps,

3. $\mathcal{A}$ outputs a set $\mathcal{S}_c \subset \{S_1, \ldots, S_T\}$ such that $\mathcal{S}_c \notin \mathbb{A}$. $\mathcal{A}$ is given the simulated secret shares $\{\mathbf{sk}'_j\}_{j \in \mathcal{S}_c}$.
5. $\mathcal{A}$ is now provided with the simulated partial decryptions $\gamma_j^{i\prime} \leftarrow \mathsf{ThAHE.PartDec}(\mathbf{ct}_i, \mathsf{sk}'_j)$, $\forall i \in [Q], j \in \mathcal{S}_c$.

The $\mathbf{Hybrid}_1$ represents the view of $\mathcal{A}$ that includes the simulated secret key shares, i.e., shares of $\vec{\mathbf{0}}$ and ciphertexts corresponding to the original PRF input and key.

$\underline{\mathbf{Hybrid}_2}$– This hybrid is the same as $\mathbf{Hybrid}_1$ except for the following steps,

4. $\mathcal{A}$ is provided with $\{\mathbf{ct}'_i\}_{i\in Q} \leftarrow \{\mathsf{ThAHE.Enc}(0, \mathsf{pk})\}_{i\in Q^x}$, where Q^x denotes the PRF input, and $|Q^x| = N$. Therefore, the encryptions of the PRF input are now replaced with random encryptions of 0.

The $\mathbf{Hybrid}_2$ represents the view of $\mathcal{A}$ that includes the simulated secret key shares and random ciphertexts (encryptions of 0) in place of the PRF input.

Indistinguishability argument. The only change in the view of $\mathcal{A}$ from $\mathbf{Hybrid}_0$ to $\mathbf{Hybrid}_1$ is that the partial decryptions $\gamma_j^{i\prime}$ are generated with simulated secret shares using a maximal invalid share set $\mathcal{S}_c$ rather than by the real partial decryptions γ_j^i with access to the real secret key shares. Similarly, we can argue that according to the properties of LISSS scheme [DT06], $\mathbf{Hybrid}_0$ and $\mathbf{Hybrid}_1$ are statistically indistinguishable. Now, considering $\mathbf{Hybrid}_1$ and $\mathbf{Hybrid}_2$, the only change in the view of the adversary $\mathcal{A}$ is that the ciphertexts corresponding to the PRF input are random encryptions of 0 rather than the original PRF/client's input. According to the IND-CPA security [JRS17, BGG$^+$18] of our underlying HE scheme, the view of $\mathcal{A}$ in $\mathbf{Hybrid}_1$ and $\mathbf{Hybrid}_2$ are computationally indistinguishable. Therefore, for a semi-honest PPT $\mathcal{A}$ it is hard to guess the PRF input from the encrypted data, this shows the obliviousness property of our TOPRF protocol.