Functional Oblivious Transfer with Applications in Privacy-Preserving Machine Learning

Aydin Abadi[⋆1], Mohammad Naseri[⋆⋆2]

[1] Newcastle University
[2] Flower Labs

Abstract. Oblivious Transfer (OT) is a fundamental cryptographic primitive introduced nearly four decades ago. OT allows a receiver to select and learn t out of n private messages held by a sender. It ensures that the sender does not learn which specific messages the receiver has chosen, while the receiver gains no information about the remaining $n - t$ messages. In this work, we introduce the notion of *functional OT* (FOT), for the first time. FOT adds a layer of security to the conventional OT by ensuring that the receiver *only learns a function* of the selected messages, rather than the t individual messages themselves. We propose several protocols that realize this concept. In particular, we propose concrete instantiations of FOT when the function to be executed on the selected message is mean, mode, addition, or multiplication. The schemes are efficient and unconditionally secure. We also propose a *non-trivial* protocol that supports arbitrary functions on the selected messages mainly using fully homomorphic encryption (FHE) and oblivious linear function evaluation, where the number of FHE invocations is constant $O(1)$ with respect to n. Our asymptotic and concrete cost analyses demonstrate the efficiency of our unconditionally secure FOT protocols. FOT can enhance the security of privacy-preserving machine learning, particularly in (i) K-Nearest Neighbors schemes and (ii) client selection in Federated Learning (FL).

1 Introduction

Oblivious Transfer (OT) [19, 44, 57] is a vital cryptographic primitive that allows a receiver to select and learn t out of n messages held by a sender, where $t \geq 1$ and $n > t$. In this setting, the sender must remain oblivious to which specific messages the receiver has chosen, while the receiver must gain no information about the remaining $n - t$ messages. OT has applications in various domains, including secure multi-party computation [7, 27, 60], FL [46, 58, 59], private banking [17], and zero-knowledge proof systems [26].

In this work, we introduce the notion of *functional oblivious transfer* (FOT). Conceptually, FOT enhances the security of conventional OT by enabling the receiver to learn only a *certain function of the messages* they select, rather than learning each selected message, while the sender remains as oblivious as in traditional OT, as shown in Figure 1. We formally define FOT and present several instantiations of it. Specifically, we first introduce a functional OT protocol, $\mathit{\Gamma}_{\text{FOT}_2}$, which relies on fully homomorphic encryption (FHE) and oblivious linear function evaluation (OLE). This protocol supports arbitrary functions on the selected messages. While, in theory, secure computation can be achieved entirely using FHE or functional encryption [12], the primary challenge lies in designing protocols that minimize reliance on these primitives due to their high computational overhead. Addressing this challenge, $\mathit{\Gamma}_{\text{FOT}_2}$ ensures that the number of FHE invocations remains constant with respect to the total number of messages, n (and it is linear with t). It is well-suited for scenarios where $n - t$ is very large.

Moreover, we present *efficient* and *scalable* functional OT protocols (e.g., $\mathit{\Gamma}_{\text{FOT}_3}$–$\mathsf{Mean}$ and $\mathit{\Gamma}_{\text{FOT}_3}$–$\mathsf{Mode}$) which do not use any public key-based primitives. They are unconditionally secure and, as a result, are inherently post-quantum secure, provided the parties communicate over an unconditionally secure channel. These protocols mainly use a new combination of several techniques, including permutation maps, one-time pads, and padding. They also rely on a third party assumed to be susceptible to corruption by a semi-honest

[⋆] aydin.abadi@ncl.ac.uk
[⋆⋆] mohammad@flower.ai

adversary. These protocols securely support various fundamental functions, namely, mode, mean, addition, and multiplication on the selected messages. We have implemented these efficient protocols and made their source code publicly available [2, 3]. Our cost evaluation indicates that they scale well for large values of t and n. For instance, $\mathit{\Gamma}_{\text{FOT}_3}$–Mean terminates in 1.4 seconds while $\mathit{\Gamma}_{\text{FOT}_3}$–Mode completes in 3.2 seconds, when $t = 65{,}536$ and $n = 1{,}048{,}576$. We also demonstrate how these protocols can be extended to enable a receiver to securely compute a function on selected messages from a distributed database maintained by multiple senders.

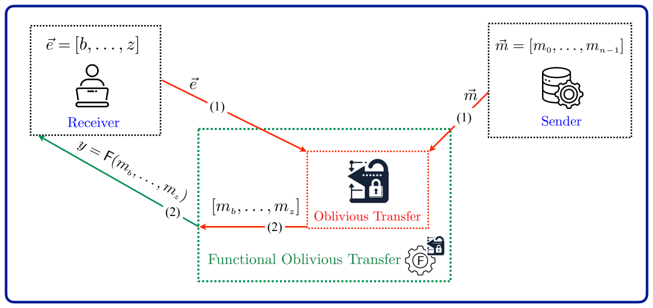
$\vec{e} = [b, \ldots, z]$
Receiver
$\vec{e}$
(1)
$\vec{m} = [m_0, \ldots, m_{n-1}]$
Sender
$\vec{m}$
(1)
$y = F(m_b, \ldots, m_z)$
(2)
$[m_b, \ldots, m_z]$
(2)
Oblivious Transfer
Functional Oblivious Transfer

Fig. 1: Showing the difference between conventional OT and functional OT. Here, $\vec{e}$ contains t elements.

FOT can be applied in at least two key scenarios: (i) in combination with scalable privacy-preserving K-NN search schemes [13, 50] to enable a receiver to efficiently but securely obtain only the prediction in K-NN algorithms, and (ii) in the client selection for FL [33,53,62] to allow a server to identify suitable clients without learning details of the clients' datasets or device characteristics while keeping its selection criteria hidden and imposing low overheads.

Summary of Contributions. The key contributions of this work are as follows:

– Formalization of functional oblivious transfer.
– Generic FHE-based FOT protocol.
– Efficient unconditionally secure constructions.
– Implementation and open-source release.
– Theoretical and concrete cost evaluation.
– Application of FOT to efficient privacy-preserving machine learning.

1.1 Structure of the Paper

The paper is structured as follows. Section 2 outlines key preliminaries, including notations and cryptographic foundations. Section 3 presents a formal definition of FOT. Section 4 presents a construction of FOT using FHE and oblivious linear function evaluation and proves the security of this construction. Section 5 introduces several FOT protocols that rely on only symmetric-key cryptography and proves the security of these protocols. Section 6 evaluates the overhead of these protocols. Section 7 discusses various applications of the proposed protocols. Section 8 reviews existing OT schemes. Finally, Section 9 concludes the paper by summarizing key contributions and suggesting directions for future work.

2 Preliminaries

2.1 Simulation-Based Security

In this paper, we use the simulation-based paradigm of secure multi-party computation [24] to define and prove the proposed protocol. Since we focus on the passive (semi-honest) adversarial model, we will restate the security definition within this context, after outlining the threat model. In this paper, by $\mathcal{X} \overset{c}{\equiv} \mathcal{Y}$ we mean that the two distributions $\mathcal{X}$ and $\mathcal{Y}$ are computationally indistinguishable.

Two-Party Computation. A two-party protocol Γ is captured by specifying a random process that maps a pair of inputs to a pair of outputs. Such a process is referred to as a functionality denoted by $f : \{0,1\}^* \times \{0,1\}^* \rightarrow \{0,1\}^* \times \{0,1\}^*$, where $f := (f_1, f_2)$. For every input pair (x, y), the output pair is a random variable $(f_1(x, y), f_2(x, y))$, such that the party with input x wishes to obtain $f_1(x, y)$ while the party with input y wishes to receive $f_2(x, y)$.

Security in the Presence of Passive Adversaries. In the passive adversarial model, the party corrupted by such an adversary correctly follows the protocol specification. In the simulation-based model, it is required that a party's "view" in a protocol's execution can be simulated given only its input and output. This implies that the parties learn nothing from the protocol's execution. Formally, in the two-party case, party i's view on input pair (x, y) is denoted by $\mathsf{View}_i^{\Gamma}(x, y)$ and equals $(w, r^i, m_1^i, ..., m_t^i)$, where $w \in \{x, y\}$ is the input of the i^{th} party, r_i is the outcome of this party's internal random coin tosses, and m_j^i represents the j^{th} message this party receives. The output of the i^{th} party during the execution of Γ on (x, y) is denoted by $\mathsf{Output}_i^{\Gamma}(x, y)$ and can be generated from its own view of the execution.

Definition 1. *Let f be the deterministic functionality defined above. Protocol Γ securely computes f in the presence of a passive probabilistic polynomial-time (PPT) adversary $\mathcal{A}$, if for every $\mathcal{A}$ in the real model, there exist PPT algorithms $(\mathsf{Sim}_1^{\Gamma}, \mathsf{Sim}_2^{\Gamma})$ such that:*

$$\{\mathsf{Sim}_1^{\Gamma}(x, f_1(x, y))\}_{x,y} \overset{c}{\equiv} \{\mathsf{View}_{1,\mathcal{A}}^{\Gamma}(x, y)\}_{x,y}$$
$$\{\mathsf{Sim}_2^{\Gamma}(y, f_2(x, y))\}_{x,y} \overset{c}{\equiv} \{\mathsf{View}_{2,\mathcal{A}}^{\Gamma}(x, y)\}_{x,y}$$

2.2 Notations and Assumptions

We denote an empty string by ϵ, a sender by $\mathcal{S}$, a receiver by $\mathcal{R}$, a third-party helper by $\mathcal{H}$, and an adversary by $\mathcal{A}$. We use $x \leftarrow v$ to denote the assignment of the value v to the variable x. We use $\overrightarrow{x} \leftarrow v$ to indicate appending the value v to the vector $\overrightarrow{x}$. We assume parties interact with each other through a secure channel; in particular, through an unconditionally secure communication channel [49] when they use the protocols presented in Section 5. By $\mathcal{A}_{\mathcal{I}}$ we mean the adversary that corrupts party $\mathcal{I}$. Also, U denotes a universe of messages $m_0, \ldots, m_l$. We define σ as the maximum bit size of messages in U, i.e., $\sigma = \mathsf{Max}(|m_1|, \ldots, |m_l|)$. For encryption (or encoding) we use a one-time pad defined over a finite field $\mathbb{F}_p$, where $\lambda = \log_2(p)$ is a sufficiently large security parameter. Shortly, we will explain how we should set its size. We define an algorithm $\mathsf{Find}(\overrightarrow{v}, j) \rightarrow indx$, that takes as input a vector $\overrightarrow{v}$ and a value j. If value j is in $\overrightarrow{v}$, it returns the index of j in $\overrightarrow{v}$; otherwise, it returns ϵ. By $\mathcal{X} \equiv \mathcal{Y}$ we mean $\mathcal{X}$ and $\mathcal{Y}$ are unconditionally indistinguishable. For a bit string b, by $b[i]$ we mean the i-th binary value of b, for $i \geq 0$.

Mode. The mode function Mode is a statistical function that returns the value(s) in a vector with the highest frequency of occurrence. Since Mode relies on a frequency function, we initially define the frequency function and then define Mode.

Definition 2 (Frequency). *Let $\overrightarrow{s} = [s_0, \ldots, s_{n-1}]$ be a vector over some domain U. The frequency function, for $\overrightarrow{s}$ and an element $\hat{s} \in S$, is defined as:*

$$\mathsf{Frequency}(\overrightarrow{s}, \hat{s}) = \Big|\{i \in [0, n-1] \mid s_i = \hat{s}\}\Big|$$

Definition 3 (Mode). *Let $\overrightarrow{s} = [s_0, \ldots, s_{n-1}]$ be a vector over domain U. Let $\mathsf{Frequency}$ be the frequency function formalized in Definition 2. The mode of $\overrightarrow{s}$ is the set of elements in $\overrightarrow{s}$ with the highest frequency. Formally:*

$$\mathsf{Mode}(\overrightarrow{s}) = \{\hat{s} \in \overrightarrow{s} \mid \mathsf{Frequency}(\overrightarrow{s}, \hat{s}) = \max_{s_j \in \overrightarrow{s}} \mathsf{Frequency}(\overrightarrow{s}, s_j)\}.$$

In this work, for the sake of simplicity, we assume Mode returns only a single value. In the concrete instantiation of Mode, we will use a padding technique to allow a party who decrypts a ciphertext to find out whether a correct key is used.

Padding Technique. The padding technique uses the idea of concatenating the binary representation of a message m_i with a string of γ zeros. We can define the padding function as:

$$\mathsf{Pad}(m_i, \gamma, p) \rightarrow m'_i = m_i \cdot 2^{\gamma} \bmod p$$

Given a padded value m'_i (after decryption), one can check whether the lower γ bits of m'_i are all zeros using the following check.

$$\mathsf{Check}_{\mathsf{Pad}}(m'_i, \gamma) = \begin{cases} \text{True}, & \text{if } m'_i \bmod 2^{\gamma} = 0 \\ \text{False}, & \text{otherwise.} \end{cases}$$

This check will allow a party to verify whether a correct key was used to decrypt a message. We can remove the pad from m'_i, by using the following function:

$$\mathsf{Unpad}(m'_i, \gamma, p) \rightarrow m_i = m'_i \cdot (2^{\gamma})^{-1} \bmod p$$

To ensure the finite field $\mathbb{F}_p$ can contain padded values, we set $\lambda > \sigma + \gamma$, when the padding is used. For instance, when $\sigma = 128$, it would suffice to set $\lambda = 256$. In this case, the probability of error (i.e., using a wrong key without being able to detect it) is 2^{-128}, which is negligible regarding λ.

Generic Subroutines for Secure Function Evaluation. For the sake of generality, we define three generic algorithms (Encode, $\mathsf{Evaluate}$, Decode) that will be used in the OT to help (securely) evaluate various functions on the response of senders. We will provide concrete instantiations of them in various cases, e.g., $\mathit{\Gamma}_{\text{FOT}_3}$–$\mathsf{Mean}$ and $\mathit{\Gamma}_{\text{FOT}_3}$–$\mathsf{Mode}$.

- $\mathsf{Encode}(\vec{m}, \vec{pp}, \vec{sk}) \rightarrow \vec{h}$: a probabilistic algorithm. It takes as input a vector of plaintext messages $\vec{m}$, a vector of public parameters $\vec{pp}$ that includes a description des_{F} of a function F to be evaluated on $\vec{m}$, and a vector of secret keys $\vec{sk}$. It encodes elements of $\vec{m}$ and returns the result $\vec{h}$.
- $\mathsf{Evaluate}(\vec{h}, \vec{pp}) \rightarrow \theta$: a deterministic algorithm. It takes a vector of encoded messages $\vec{h}$ and a vector of public parameters $\vec{pp}$. It returns an encoded evaluated value θ.
- $\mathsf{Decode}(\theta, \vec{pp}, \vec{sk}) \rightarrow \theta'$: a deterministic algorithm. It takes as input the encoded evaluated message θ, public parameters $\vec{pp}$, and the secret keys $\vec{sk}$. It returns a decoded evaluated message θ'.

2.3 Fully Homomorphic Encryption

A homomorphic encryption scheme is a (public-key) encryption scheme that allows arbitrary functions to be evaluated on a set of ciphertext [22]. It consists of four algorithms:

- $\mathsf{HE.KeyGeneration}(1^{\lambda}) \rightarrow (sk_{\mathsf{HE}}, pk_{\mathsf{HE}})$. A probabilistic algorithm that takes a security parameter as input. It returns a pair of private key sk_{HE} and public key pk_{HE}.
- $\mathsf{HE.Encrypt}(pk_{\mathsf{HE}}, m) \rightarrow h$. A probabilistic algorithm that takes pk_{HE} and a plaintext message m from the scheme's plaintext universe U. It returns a ciphertext h.
- $\mathsf{HE.Decrypt}(sk_{\mathsf{HE}}, h) \rightarrow m$. A deterministic algorithm that takes sk_{HE} and a ciphertext h. It returns a plaintext m.
- $\mathsf{HE.Evaluate}(\mathsf{F}, pk_{\mathsf{HE}}, h_0, \ldots, h_{n-1}) \rightarrow h'$. It takes a function F representation defined over U, the public key pk_{HE}, and n ciphertexts $h_0, \ldots, h_{n-1}$. It outputs a ciphertext h'.

We require that FHE satisfies IND-CPA security, see Appendix A for a formal definition. We define the computation complexity of $\mathsf{HE.Evaluate}(\mathsf{F}, pk_{\mathsf{HE}}, h_0, \ldots, h_{n-1})$ as $Comp(\mathsf{F}, n)$. In this work, we require a sender $\mathcal{S}$ to obliviously select t out of n messages that involve homomorphic addition and multiplication. To provide sufficient detail, we will use $\overset{H}{+}$ and $\overset{H}{\times}$ to denote homomorphic addition and multiplication respectively. In this work, we use FHE that works over integers for non-binary message spaces [41]. Specifically, we are interested in FHE whose message space is defined over a large prime number q, where $q \gg p$. In this case, for every element r in $\mathbb{F}_p$, there exists its multiplicative inverse r^{-1} in $\mathbb{F}_q$, i.e., $r \cdot r^{-1} \bmod q = 1$.

2.4 Oblivious Linear Function Evaluation

Oblivious linear function evaluation (OLE) is a two-party protocol that involves a sender and receiver [23]. In OLE, the sender has two inputs $a, b \in \mathbb{F}_p$ and the receiver has a single input, $c \in \mathbb{F}_p$. The protocol allows the receiver to learn only $s = a \cdot c + b \in \mathbb{F}_p$, while the sender learns nothing. Figure 2 presents the OLE's ideal functionality, $\mathcal{F}_{\mathsf{OLE}}$.

1. Upon receiving a message $(inputS, (a, b))$ from the sender with $a, b \in \mathbb{F}_p$, check that there is no stored tuple; if it fails, ignore that message. Store a and b and then transmit a message $(input)$ to $\mathcal{A}$.
2. Upon receiving a message $(inputR, c)$ from the receiver where $c \in \mathbb{F}_p$, check that there is no stored tuple; if the check fails, then ignore the message. Store c and then transmit a message $(input)$ to $\mathcal{A}$.
3. Upon receiving a message $(deliver, S)$ from $\mathcal{A}$, check whether (a, b) and c have been stored; if the check fails, ignore that message. Otherwise, send $(delivered)$ to the sender.
4. Upon receiving a message $(deliver, R)$ from $\mathcal{A}$, check whether (a, b) and c have been stored; if the check fails, ignore that message. Set $s = a \cdot c + b$ and send $(output, s)$ to the receiver.

Fig. 2: Ideal functionality $\mathcal{F}_{\mathsf{OLE}}$ for OLE.

2.5 Trusted Execution Environments

Trusted Execution Environment ($\mathcal{TEE}$), also known as a secure enclave, constitutes a secure processing environment comprising processing, memory, and storage hardware units [42, 61]. An ideal $\mathcal{TEE}$ guarantees the preservation of data integrity and confidentiality. Side-channel attacks on different deployments of $\mathcal{TEE}$s have been demonstrated in the literature [52]. These attacks pose a threat as they could enable attackers to extract secrets from $\mathcal{TEE}$s. In Section 5, we use $\mathcal{TEE}$ to construct efficient post-quantum three-party FOT schemes. Our security and trust assumptions regarding $\mathcal{TEE}$s are conservative. Specifically, our solutions avoid disclosing any plaintext messages or private keys to $\mathcal{TEE}$. Hence, a weak $\mathcal{TEE}$ that might be corrupted by a semi-honest adversary would be sufficient for our schemes. We assume that the $\mathcal{TEE}$ is unconditionally secure. We formally show that $\mathcal{TEE}$ at the most only learns the size of the encrypted computation result. In our work, $\mathcal{TEE}$ can be substituted with any semi-honest server that does not collude with other entities.

3 Security Model

Functional Oblivious Transfer (FOT) involves a sender $\mathcal{S}$ and a receiver $\mathcal{R}$. The sender has a vector of n messages $\vec{m} = [m_0, \dots, m_{n-1}]$ and the receiver has a vector of t indices $\vec{e} = [b, \dots, z]$, where $1 \leq t \leq n$ and $\forall e \in \vec{e} : 0 \leq e \leq n-1$. Informally, the functionality G^{F} that FOT computes, for a function F, takes the parties' input and returns (i) nothing to $\mathcal{S}$ and (ii) $y = \mathsf{F}(m_b, \dots, m_z)$ to $\mathcal{R}$. Below, we formally state it, by initially defining F followed by the definition of G^{F}.

Definition 4 (Functionality F). *A functionality F is defined over the plaintext space U as a function $\mathsf{F} : \underbrace{U \times \dots \times U}_{t \text{ times}} \rightarrow \{0,1\}^*$, where F is defined as a deterministic Turing Machine.*

Definition 5 (Functionality G^{F}). *The functionality G^{F}, which FOT for a function F will compute, is defined as:*

$$\mathsf{G}^{\mathsf{F}} : \Big([m_0, \dots, m_{n-1}], [b, \dots, z]\Big) \rightarrow y$$

that takes as input a vector of messages $\vec{m} = [m_0, \dots, m_{n-1}]$ and a vector of indices $\vec{e} = [b, \dots, z]$, belonging to an index space I, and returns $y = \mathsf{F}(m_b, \dots, m_z)$, where $m_b, \dots, m_z$ are some elements of $\vec{m}$ indexed by $\vec{e}$.

Informally, the security of FOT requires that $\mathcal{S}$ gains no knowledge about $\mathcal{R}$'s private input or the output y. Similarly, $\mathcal{R}$ learns nothing beyond the output y, including any information about $\mathcal{S}$'s input. A formal definition is provided below.

Definition 6 (Security of FOT). *Let G^{F} be the functional OT's functionality defined above. We assert that protocol Γ securely realizes G^{F} in the presence of passive adversaries if for every probabilistic polynomial time (PPT) adversary $\mathcal{A}$ in the real model, there is a PPT simulator Sim in the ideal model, where:*

$$\Big\{\mathsf{Sim}^{\Gamma}_{\mathcal{S},aux}\big(\vec{m}, \epsilon\big)\Big\}_{\vec{m},\vec{e}} \stackrel{c}{\equiv} \Big\{\mathsf{View}^{\Gamma}_{\mathcal{S},\mathcal{A}(aux)}\big(\vec{m}, \vec{e}\big)\Big\}_{\vec{m},\vec{e}}$$

$$\Big\{\mathsf{Sim}^{\Gamma}_{\mathcal{R},aux}\Big(\vec{e}, \mathsf{G}^{\mathsf{F}}\big(\vec{m}, \vec{e}\big)\Big)\Big\}_{\vec{m},\vec{e}} \stackrel{c}{\equiv} \Big\{\mathsf{View}^{\Gamma}_{\mathcal{R},\mathcal{A}(aux)}\big(\vec{m}, \vec{e}\big)\Big\}_{\vec{m},\vec{e}}$$

where aux is an auxiliary (public) input known to the adversary prior to the protocol execution.

4 Two-Party FOT Protocols

In this section, we present two variants of two-party generic FOT protocols. The first variant is simpler but less efficient while the second variant is more involved but more efficient. In both variants, we will use standard oblivious filtering previously used to achieve communication efficiency [17, 18]. We will use this technique to allow a sender $\mathcal{S}$ to obliviously filter out $n-t$ messages and then evaluate a certain function on the remaining t messages. Before presenting the FOT protocols, we briefly discuss this technique. To retrieve a single record m_v securely from $\mathcal{S}$ that holds messages $[m_0, \dots, m_{n-1}]^T$ where T denotes transpose, without revealing to $\mathcal{S}$ which one is fetched, $\mathcal{R}$ can take the following steps:

1. set $\vec{b} = [b_0, \dots, b_{n-1}]$, where every element of $\vec{b}$ is set to zero except for v-th element b_v which is set to 1.
2. encrypt each element of $\vec{b}$ using a (fully) homomorphic encryption scheme. Let $\vec{b'}$ contain the encrypted elements.
3. send $\vec{b'}$ to $\mathcal{S}$ which homomorphically computes the dot product of $\vec{b'}$ and $\vec{m}$. $\mathcal{S}$ sends to $\mathcal{R}$ the result res.
4. decrypt res to discover m_v.

4.1 Basic Approach with $O(t \cdot n) + Comp(\mathsf{F}, t)$ FHE Calls

The first variant of FOT mainly uses FHE and the filtering technique. Recall that in the context of FOT, the receiver $\mathcal{R}$ has a vector $\vec{e}$ of t indices. For each element e_j in $\vec{e}$, $\mathcal{R}$ generates a vector b_j of n bits where all bits are set to 0 except the e_j-th bit that is set to 1. It encrypts each bit of the vector using a FHE and sends these t encrypted vectors to $\mathcal{S}$. Next, $\mathcal{S}$ homomorphically computes the dot product of each encrypted vector and its message vector. The result is t encrypted values that $\mathcal{R}$ is interested. Given these t encrypted values, $\mathcal{S}$ homomorphically evaluates F on them and sends the encrypted result to $\mathcal{R}$, which decrypts and finds the result. The computation cost of $\mathcal{R}$ is $O(t \cdot n)$, while the computation cost of $\mathcal{S}$ is $O(t \cdot n) + Comp(\mathsf{F}, t)$ mainly involving FHE. Shortly, in Section 4.2, we present the second variant of this protocol that reduces the number of FHE invocations. Since the second variant overlaps with the first one, we will present the second variant in detail, in the next section. For readers interested in the first variant, we provide its details in Figure 8 of Appendix B.

4.2 Enhanced Approach with $O(t) + Comp(\mathsf{F}, t)$ FHE Calls

The second variant, denoted as Γ_{FOT_2}, involves a much fewer number of FHE invocations. To be precise, the number of times FHE will be called is $O(t) + Comp(\mathsf{F}, t)$, which is linear with the number of indices that $\mathcal{R}$ is interested and the complexity of F, but it is *constant* with respect to n. Our primary observation is that oblivious filtering (used in the first variant) that involves the dot product of two vectors: (1) is the only procedure in the protocol that requires $2 \cdot n$ (and t) invocations of FHE, and (2) involves only modular addition and multiplication. We could achieve efficiency if we replace FHE with a more efficient method to compute the dot product. To achieve the above objective, we replace FHE with a careful combination of several tools and techniques, including OLE, one-time pad, and zero-sum values. We provide an overview of Γ_{FOT_2}. Since there are t indices, for each t, $\mathcal{R}$ constructs a vector $\vec{b_j}$ as before and securely computes its dot product with $\vec{m}$ that $\mathcal{S}$ holds. The dot product of $\vec{b_j}$ and $\vec{m}$ will be computed through two main phases:

- In the first phase, $\mathcal{S}$ and $\mathcal{R}$ obliviously multiply their vectors $\vec{b_j}$ and $\vec{m}$ component-wise, by invoking n instances of OLE. This allows $\mathcal{S}$ to learn each output of OLE which is a *masked* version of the product $b_{j,i} \cdot m_i$, for every i, where $0 \leq i \leq n-1$. Note that $\mathcal{S}$ cannot learn whether $b_{j,i} = 0$ due to the way the output is encoded.
- In the second phase, $\mathcal{S}$ sums the outputs of OLE. This results in the dot product of $\vec{b_j}$ and $\vec{m}$ that is still masked by a blinding factor, say g_j.

To let $\mathcal{S}$ unmask every j-th masked dot product securely so that later it can perform oblivious computation on all t dot products, $\mathcal{R}$ sends to $\mathcal{S}$ the multiplicative inverse of each g_j that is encrypted using FHE, i.e., $\mathcal{R}$ sends $\mathsf{HE.Encrypt}(pk_{\mathsf{HE}}, (g_j)^{-1})$ to $\mathcal{S}$. Now, $\mathcal{S}$ can homomorphically multiply each j-th dot product with $\mathsf{HE.Encrypt}(pk_{\mathsf{HE}}, (g_j)^{-1})$ to obtain an encrypted *unmasked* dot product which is exactly one of the messages in $\vec{m}$ that $\mathcal{R}$ is interested. Give all t encrypted unmasked dot products, $\mathcal{S}$ can invoke $\mathsf{HE.Evaluate}$ to obliviously run function F on the ciphertexts. It sends the result to $\mathcal{R}$ which decrypts it to obtain $\mathsf{F}(m_b, \ldots, m_z)$. As it is evident, during the oblivious filtering (imposing computation complexity of $O(t \cdot n)$) FHE is not involved anymore. Instead, it is involved during removing t blinding factors $g_0, \ldots, g_{t-1}$ and the oblivious evaluation of F on t selected messages; however, these operations' cost is (independent of n and) linear with t and the complexity of F. Thus, the novelty of Γ_{FOT_2}'s design lies in its ability to support arbitrary functions with minimal cost, achieved through careful integration of standard tools and techniques. These include oblivious filtering [17,18], OLE, FHE, (zero-sum) one-time pads, and oblivious one-time pad removal.

Detailed Description of $\boldsymbol{\Gamma_{\text{FOT}_2}}$. Below, we explain the protocol Γ_{FOT_2} in detail. In this protocol, the sender $\mathcal{S}$ has a vector of n plaintext messages $\vec{m} = [m_0, \ldots, m_{n-1}]$ while the receiver $\mathcal{R}$ has a vector of t indices $\vec{e} = [b, \ldots, z]$.

1. Setup: $\texttt{Setup}(1^{\lambda}) \rightarrow (sk_{\mathsf{HE}}, pk_{\mathsf{HE}})$
   This phase involves $\mathcal{R}$.
   (a) calls $\mathsf{HE.KeyGeneration}(1^{\lambda}) \rightarrow (sk_{\mathsf{HE}}, pk_{\mathsf{HE}})$.
   (b) publishes pk_{HE}.
2. Query Generation: $\texttt{GenQuery}(pk_{\mathsf{HE}}, n, \vec{e}) \rightarrow qry := (qry_1, qry_2)$
   This phase involves $\mathcal{S}$ and $\mathcal{R}$. They take the following steps. $\forall e_j \in \vec{e}$:
   (a) $\mathcal{R}$ selects a uniformly random value $g_j \overset{\$}{\leftarrow} \{0,1\}^{\lambda}$ and encrypts its multiplicative inverse over the message space of FHE as: $\mathsf{HE.Encrypt}(pk_{\mathsf{HE}}, (g_j)^{-1})$, where $(g_j)^{-1} \in \mathbb{F}_q$.
   (b) $\mathcal{R}$ generates a vector of n uniformly random values $[r_{j,0}, \ldots, r_{j,n-1}]$, such that their sum is zero, as follows:
      i. selects $n-1$ random values $r_{j,0}, \ldots, r_{j,n-2} \overset{\$}{\leftarrow} \{0,1\}^{\lambda}$.
      ii. sets $r_{n-1} = -\sum\limits_{l=1}^{n-2} r_l \bmod p$.
   (c) $\mathcal{R}$ constructs $\vec{b}_j = [b_{j,0}, \ldots, b_{j,n-1}]$, by setting every element $b_{j,i}$ to 0 except for the e_j-th element, set to 1.
   (d) $\mathcal{R}$ and $\mathcal{S}$ for every i (where $0 \leq i \leq n-1$) run an instance of OLE. The input of $\mathcal{R}$ is $g_j \cdot b_{j,i}$ and $r_{j,i}$. The input of $\mathcal{S}$ is m_i. The same instance of OLE returns:
   $$b'_{j,i} = g_j \cdot b_{j,i} \cdot m_j + r_{j,i} \bmod p$$
   to $\mathcal{S}$. Let $\vec{b'_j} = [b'_{j,0}, \ldots, b'_{j,n-1}]$. Vector $\vec{b'} = [\vec{b'}_0, \ldots, \vec{b'}_{t-1}]$ is the first part of $\mathcal{R}$'s query, qry_1, which is already given to $\mathcal{S}$.
   Subsequently, $\mathcal{R}$ sets the second part of its query, denoted by qry_2, to $\big[\mathsf{HE.Encrypt}(pk_{\mathsf{HE}}, (g_0)^{-1}), \ldots, \mathsf{HE.Encrypt}(pk_{\mathsf{HE}}, (g_{t-1})^{-1})\big]$ and sends qry_2 to $\mathcal{S}$.
3. Response Generation: $\texttt{GenRes}(\vec{m}, pk_{\mathsf{HE}}, qry) \rightarrow res$
   This phase involves $\mathcal{S}$.
   (a) obliviously identifies t messages that $\mathcal{R}$ is interested, using each $\vec{b'_j}$, as follows. $\forall j, 0 \leq j \leq t-1$:
   $$a_j = \sum_{i=0}^{n-1} b'_{j,i} \bmod p = g_j \cdot \sum_{i=0}^{n-1} b_{j,i} \cdot m_j \bmod p$$
   (b) obliviously removes each blinding factor g_j from each a_j using qry_2: $\forall j, 0 \leq j \leq t-1$:
   $$\begin{aligned} e_j &= \mathsf{HE.Encrypt}(pk_{\mathsf{HE}}, (g_j)^{-1}) \overset{H}{\times} a_j \\ &= \mathsf{HE.Encrypt}(pk_{\mathsf{HE}}, \sum_{i=0}^{n-1} b_{j,i} \cdot m_j) \end{aligned}$$
   (c) obliviously evaluates the function F on the t ciphertexts generated in the previous step.
   $$\mathsf{HE.Evaluate}(\mathsf{F}, pk_{\mathsf{HE}}, e_0, \ldots, e_{t-1}) \rightarrow \theta$$
   (d) sets $res_{\mathcal{R}} = \theta$ and sends $res_{\mathcal{R}}$ to $\mathcal{R}$.
4. Message Extraction. $\texttt{Retreive}(res_{\mathcal{R}}, sk_{\mathsf{HE}}) \rightarrow y$
   - $\mathcal{R}$ decrypts $res_{\mathcal{R}}$ as $\mathsf{HE.Decrypt}(sk_{\mathsf{HE}}, res_{\mathcal{R}}) \rightarrow y$.

Theorem 1. *Let G^{F} be the functionality formalized in Definition 5. If FHE is IND-CPA, the protocol Γ_{FOT_2} securely computes G^{F}, w.r.t. Definition 6, in the $\mathcal{F}_{\mathsf{OLE}}$-hybrid model.*

4.3 Security Proof of $\boldsymbol{\Gamma_{\text{FOT}_2}}$

In this section, we prove the security of Γ_{FOT_2}, i.e., Theorem 1.

Proof. We prove the theorem in the case where each party is corrupt. The proof consists of a series of hybrid experiments, gradually transitioning from the real execution to the ideal execution.

Corrupt $\mathcal{R}$. Initially, we consider the case where the receiver $\mathcal{R}$ is corrupt. We will involve a simulator $\mathsf{Sim}_{\mathcal{R}}^{\Gamma_{\mathrm{FOT}_2}}$ (which receives $\mathcal{R}$'s input $\vec{e}$ and output y) to eventually generate a view in an ideal execution, i.e., $\mathsf{Sim}_{\mathcal{R}}^{\Gamma_{\mathrm{FOT}_2}}(\vec{e}, y) \rightarrow \mathsf{View}_{\mathsf{Sim}_{\mathcal{R}}}^{\Gamma_{\mathrm{FOT}_2}}$.

- Hybrid 0 ($\mathsf{View}_{\mathcal{A}}^{\text{Hybrid 0}}$): Real execution. The view of the adversary in the real execution of Γ_{FOT_2} is:
$$\mathsf{View}_{\mathcal{A}}^{\text{Hybrid 0}} = \{r_{\mathcal{R}}, \mathsf{View}_{\mathcal{R}}^{\mathsf{OLE}}, res_{\mathcal{R}}\}$$
where $r_{\mathcal{R}}$ is the random coin sampled uniformly at random by $\mathcal{R}$, $\mathsf{View}_{\mathcal{R}}^{\mathsf{OLE}}$ refers to the receiver's real-model view during the execution of OLE, and $res_{\mathcal{R}}$ is the response computed by $\mathcal{S}$ using $\mathsf{HE.Evaluate}$.
- Hybrid 1 ($\mathsf{View}_{\mathcal{A}}^{\text{Hybrid 1}}$): Replace $r_{\mathcal{R}}$ with $r'_{\mathcal{R}}$. In this hybrid, $\mathsf{Sim}_{\mathcal{R}}^{\Gamma_{\mathrm{FOT}_2}}$ replaces the coin $r_{\mathcal{R}}$ with a freshly generated random coin $r'_{\mathcal{R}}$. The adversary's view becomes:
$$\mathsf{View}_{\mathcal{A}}^{\text{Hybrid 1}} = \{r'_{\mathcal{R}}, \mathsf{View}_{\mathcal{R}}^{\mathsf{OLE}}, res_{\mathcal{R}}\}$$
Since $r_{\mathcal{R}}$ and $r'_{\mathcal{R}}$ are both uniformly random, the distributions of $\mathsf{View}_{\mathcal{A}}^{\text{Hybrid 0}}$ and $\mathsf{View}_{\mathcal{A}}^{\text{Hybrid 1}}$ are identical:
$$\mathsf{View}_{\mathcal{A}}^{\text{Hybrid 0}} \equiv \mathsf{View}_{\mathcal{A}}^{\text{Hybrid 1}}$$
- Hybrid 2 ($\mathsf{View}_{\mathcal{A}}^{\text{Hybrid 2}}$): Replace $\mathsf{View}_{\mathcal{R}}^{\mathsf{OLE}}$ with $\mathsf{SimView}_{\mathcal{R}}^{\mathsf{OLE}}$. In this hybrid, $\mathsf{Sim}_{\mathcal{R}}^{\Gamma_{\mathrm{FOT}_2}}$ replaces $\mathsf{View}_{\mathcal{R}}^{\mathsf{OLE}}$ with a simulated view $\mathsf{SimView}_{\mathcal{R}}^{\mathsf{OLE}}$ generated as follows. $\mathsf{Sim}_{\mathcal{R}}^{\Gamma_{\mathrm{FOT}_2}}$ constructs an empty set $\mathsf{SimView}_{\mathcal{R}}^{\mathsf{OLE}}$. Then, it takes the following steps for every e_j in $\vec{e}$.
  1. using $r'_{\mathcal{R}}$, picks a uniformly random value $\hat{g}_j \stackrel{\$}{\leftarrow} \{0,1\}^{\lambda}$.
  2. generates a vector $\vec{o_j} = [o_{j,0}, \ldots, o_{j,n-1}]$, by setting each element $o_{j,i}$ to 0 except the e_j-th element set to 1.
  3. using $r'_{\mathcal{R}}$, constructs a vector of n uniformly random values $[\hat{r}_{j,0}, \ldots, \hat{r}_{j,n-1}]$, such that their sum is zero. To compute the vector's elements, it takes the following steps.
     (a) selects $n-1$ uniformly random values, $\hat{r}_{j,0}, \ldots, \hat{r}_{j,n-2} \stackrel{\$}{\leftarrow} \{0,1\}^{\lambda}$.
     (b) sets $\hat{r}_{n-1} = -\sum\limits_{l=1}^{n-2} \hat{r}_l \bmod p$.
  4. for every i (where $0 \le i \le n-1$) invokes OLE's ideal functionality $\mathcal{F}_{\mathsf{OLE}}$, where its inputs are $\hat{g}_j \cdot o_{j,i}$ and $\hat{r}_{j,i}$. Let $\mathsf{SimView}_{\mathcal{R},i}^{\mathsf{OLE}}$ be the corresponding simulated view of the simulator (and accordingly $\mathcal{R}$) when interacting with $\mathcal{F}_{\mathsf{OLE}}$.
  5. appends $\mathsf{SimView}_{\mathcal{R},i}^{\mathsf{OLE}}$ to $\mathsf{SimView}_{\mathcal{R}}^{\mathsf{OLE}}$.

  The adversary's view becomes:
$$\mathsf{View}_{\mathcal{A}}^{\text{Hybrid 2}} = \{r'_{\mathcal{R}}, \mathsf{SimView}_{\mathcal{R}}^{\mathsf{OLE}}, res_{\mathcal{R}}\}$$
Since we are in the $\mathcal{F}_{\mathsf{OLE}}$-hybrid model, the distributions of $\mathsf{View}_{\mathcal{R}}^{\mathsf{OLE}}$ and $\mathsf{SimView}_{\mathcal{R}}^{\mathsf{OLE}}$ are identical; thus, it holds that:
$$\mathsf{View}_{\mathcal{A}}^{\text{Hybrid 1}} \equiv \mathsf{View}_{\mathcal{A}}^{\text{Hybrid 2}}$$
- Hybrid 3 ($\mathsf{View}_{\mathcal{A}}^{\text{Hybrid 3}}$): Replace $res_{\mathcal{R}}$ with $res'_{\mathcal{R}}$. In this hybrid, $\mathsf{Sim}_{\mathcal{R}}^{\Gamma_{\mathrm{FOT}_2}}$ replaces $res_{\mathcal{R}}$ with a simulated response $res'_{\mathcal{R}}$, generated by encrypting the plaintext output y of $\mathcal{R}$ as: $\mathsf{HE.Encrypt}(pk_{\mathsf{HE}}, y) \rightarrow res'_{\mathcal{R}}$. The adversary's view becomes:
$$\mathsf{View}_{\mathcal{A}}^{\text{Hybrid 3}} = \{r'_{\mathcal{R}}, \mathsf{SimView}_{\mathcal{R}}^{\mathsf{OLE}}, res'_{\mathcal{R}}\}$$
Since FHE is IND-CPA, ciphertexts $res_{\mathcal{R}}$ and $res'_{\mathcal{R}}$ are computationally indistinguishable. Moreover, both ciphertexts result in y after they are decrypted; specifically, $\mathsf{HE.Decrypt}(sk_{\mathsf{HE}}, res_{\mathcal{R}}) \rightarrow y$ and $\mathsf{HE.Decrypt}(sk_{\mathsf{HE}}, res'_{\mathcal{R}}) \rightarrow y$. Hence, it holds that:
$$\mathsf{View}_{\mathcal{A}}^{\text{Hybrid 2}} \stackrel{c}{\equiv} \mathsf{View}_{\mathcal{A}}^{\text{Hybrid 3}}$$
Conclusion. At the final step (Hybrid 3), the simulator has constructed the full ideal view:
$$\mathsf{View}_{\mathsf{Sim}_{\mathcal{R}}}^{\Gamma_{\mathrm{FOT}_2}} = \{r'_{\mathcal{R}}, \mathsf{SimView}_{\mathcal{R}}^{\mathsf{OLE}}, res'_{\mathcal{R}}\}$$
By the transitivity of indistinguishability across the hybrids, we conclude:
$$\mathsf{View}_{\mathcal{A}}^{\text{Hybrid 0}} \stackrel{c}{\equiv} \mathsf{View}_{\mathcal{A}}^{\text{Hybrid 3}} \quad \Rightarrow \quad \mathsf{View}_{\mathcal{R},\mathcal{A}}^{\Gamma_{\mathrm{FOT}_2}} \stackrel{c}{\equiv} \mathsf{View}_{\mathsf{Sim}_{\mathcal{R}}}^{\Gamma_{\mathrm{FOT}_2}}$$
where $\mathsf{View}_{\mathcal{R},\mathcal{A}}^{\Gamma_{\mathrm{FOT}_2}}$ and $\mathsf{View}_{\mathsf{Sim}_{\mathcal{R}}}^{\Gamma_{\mathrm{FOT}_2}}$ are the real and ideal model views respectively.

Corrupt $\mathcal{S}$. In this case, we will involve a simulator $\mathsf{Sim}_{\mathcal{S}}^{\Gamma_{\mathrm{FOT}_2}}$ (which receives $\mathcal{S}$'s input $\vec{m}$) to eventually generate a view in an ideal execution, i.e., $\mathsf{Sim}_{\mathcal{S}}^{\Gamma_{\mathrm{FOT}_2}}(\vec{m}) \rightarrow \mathsf{View}_{\mathsf{Sim}_{\mathcal{S}}}^{\Gamma_{\mathrm{FOT}_2}}$.

- Hybrid 0 ($\mathsf{View}_{\mathcal{A}}^{\text{Hybrid } 0}$): Real execution. The view of the adversary in the real execution of Γ_{FOT_2} is:

$$\mathsf{View}_{\mathcal{A}}^{\text{Hybrid } 0} = \{r_{\mathcal{S}}, \mathsf{View}_{\mathcal{S}}^{\mathsf{OLE}}, qry_1, qry_2\}$$

where $r_{\mathcal{S}}$ is the coin sampled uniformly at random by $\mathcal{S}$, $\mathsf{View}_{\mathcal{S}}^{\mathsf{OLE}}$ is the sender's real-model view during the execution of OLE, and qry_1 contains the outputs of OLE, i.e., $qry_1 = \vec{b'} = [\vec{b'}_0, \ldots, \vec{b'}_{t-1}]$ and $\vec{b'}_j = [b'_{j,0}, \ldots, b'_{j,n-1}]$. Moreover, it holds that $qry_2 = \big[\mathsf{HE.Encrypt}(pk_{\mathsf{HE}}, (g_0)^{-1}), \ldots, \mathsf{HE.Encrypt}(pk_{\mathsf{HE}}, (g_{t-1})^{-1})\big]$.

- Hybrid 1 ($\mathsf{View}_{\mathcal{A}}^{\text{Hybrid } 1}$): Replace $r_{\mathcal{S}}$ with $r'_{\mathcal{S}}$. In this hybrid, $\mathsf{Sim}_{\mathcal{S}}^{\Gamma_{\mathrm{FOT}_2}}$ replaces the coin $r_{\mathcal{S}}$ with a freshly random coin $r'_{\mathcal{S}}$. Consequently, the adversary's view becomes:

$$\mathsf{View}_{\mathcal{A}}^{\text{Hybrid } 1} = \{r'_{\mathcal{S}}, \mathsf{View}_{\mathcal{S}}^{\mathsf{OLE}}, qry_1, qry_2\}$$

Since $r_{\mathcal{R}}$ and $r'_{\mathcal{R}}$ are both uniformly random, the distributions of $\mathsf{View}_{\mathcal{A}}^{\text{Hybrid } 0}$ and $\mathsf{View}_{\mathcal{A}}^{\text{Hybrid } 1}$ are identical:

$$\mathsf{View}_{\mathcal{A}}^{\text{Hybrid } 0} \equiv \mathsf{View}_{\mathcal{A}}^{\text{Hybrid } 1}$$

- Hybrid 2 ($\mathsf{View}_{\mathcal{A}}^{\text{Hybrid } 2}$): Replace $\mathsf{View}_{\mathcal{S}}^{\mathsf{OLE}}$ with $\mathsf{SimView}_{\mathcal{S}}^{\mathsf{OLE}}$. In this hybrid, $\mathsf{Sim}_{\mathcal{S}}^{\Gamma_{\mathrm{FOT}_2}}$ replaces $\mathsf{View}_{\mathcal{S}}^{\mathsf{OLE}}$ with a simulated view $\mathsf{SimView}_{\mathcal{S}}^{\mathsf{OLE}}$ generated as follows. $\mathsf{Sim}_{\mathcal{S}}^{\Gamma_{\mathrm{FOT}_2}}$ constructs an empty vector $\mathsf{SimView}_{\mathcal{S}}^{\mathsf{OLE}}$. Then, it takes the following steps for every j, where $0 \leq j \leq t-1$.
  1. For every i (where $0 \leq i \leq n-1$) invokes OLE's ideal functionality $\mathcal{F}_{\mathsf{OLE}}$, where its input is m_i. Let $\mathsf{SimView}_{\mathcal{S},i}^{\mathsf{OLE}}$ be the related simulated view of the simulator (and accordingly $\mathcal{S}$) when interacting with $\mathcal{F}_{\mathsf{OLE}}$.
  2. Appends $\mathsf{SimView}_{\mathcal{S},i}^{\mathsf{OLE}}$ to $\mathsf{SimView}_{\mathcal{S}}^{\mathsf{OLE}}$.

  As a result, the adversary's view becomes:

$$\mathsf{View}_{\mathcal{A}}^{\text{Hybrid } 2} = \{r'_{\mathcal{S}}, \mathsf{SimView}_{\mathcal{S}}^{\mathsf{OLE}}, qry_1, qry_2\}$$

Since we are operating in the $\mathcal{F}_{\mathsf{OLE}}$-hybrid model, the distributions of $\mathsf{View}_{\mathcal{S}}^{\mathsf{OLE}}$ and $\mathsf{SimView}_{\mathcal{S}}^{\mathsf{OLE}}$ are identical. Therefore, we have that:

$$\mathsf{View}_{\mathcal{A}}^{\text{Hybrid } 1} \equiv \mathsf{View}_{\mathcal{A}}^{\text{Hybrid } 2}$$

- Hybrid 3 ($\mathsf{View}_{\mathcal{A}}^{\text{Hybrid } 3}$): Replace qry_1 with qry'_1. In this hybrid, $\mathsf{Sim}_{\mathcal{S}}^{\Gamma_{\mathrm{FOT}_2}}$ substitutes OLE's real-model output qry_1 that $\mathcal{S}$ receives with a simulated output qry'_1 constructed as follows. It sets qry'_1 to t vectors $\vec{o} = [\vec{o}_0, \ldots, \vec{o}_{t-1}]$, where each vector $\vec{o}_j$ in $\vec{o}$ contains n fresh uniformly random values, $o_{j,0}, \ldots, o_{j,n-1} \overset{\$}{\leftarrow} \{0,1\}^{\lambda}$. Hence, the adversary's view becomes:

$$\mathsf{View}_{\mathcal{A}}^{\text{Hybrid } 3} = \{r'_{\mathcal{S}}, \mathsf{SimView}_{\mathcal{S}}^{\mathsf{OLE}}, qry'_1, qry_2\}$$

In the real model, each element of each vector $\vec{b'_j}$ in qry_1 is masked with a fresh one-time pad (of size λ-bit). Also, when all elements of each vector $\vec{b'_j}$ are summed, the result is a value masked with a fresh one-time pad. In the ideal model, each element of each vector $\vec{o}_j$ in qry_2 is a fresh value (of size λ-bit) selected uniformly at random. Due to the perfect security of the one-time pad, the masked elements of each vector $\vec{b'_j}$ and the sum of these elements have identical distributions to random values (of the same size). Thus, it holds that:

$$\mathsf{View}_{\mathcal{A}}^{\text{Hybrid } 2} \equiv \mathsf{View}_{\mathcal{A}}^{\text{Hybrid } 3}$$

- Hybrid 4 ($\mathsf{View}_{\mathcal{A}}^{\text{Hybrid } 4}$): Replace qry_2 with qry'_2. In this hybrid, $\mathsf{Sim}_{\mathcal{S}}^{\Gamma_{\mathrm{FOT}_2}}$ replaces qry_2 with a simulated query qry'_2 by (1) constructing an empty vector qry'_2 and (2) appending to it t values that are picked uniformly at random from the space of the FHE's ciphertext. Therefore, the adversary's view becomes:

$$\mathsf{View}_{\mathcal{A}}^{\text{Hybrid } 4} = \{r'_{\mathcal{S}}, \mathsf{SimView}_{\mathcal{S}}^{\mathsf{OLE}}, qry'_1, qry'_2\}$$

Due to the IND-CPA of FHE, the t ciphertexts produced by FHE (i.e., the elements of qry_2) are computationally indistinguishable from t uniformly random elements (i.e., the elements of qry'_2) sampled independently from the ciphertext space. Therefore, it holds that:

$$\mathsf{View}_{\mathcal{A}}^{\text{Hybrid } 3} \overset{c}{\equiv} \mathsf{View}_{\mathcal{A}}^{\text{Hybrid } 4}$$

Conclusion. At the Hybrid 4, $\mathsf{Sim}_{\mathcal{S}}^{\varGamma_{\text{FOT}_2}}$ has fully constructed the ideal view:

$$\mathsf{View}_{\mathsf{Sim}_{\mathcal{S}}}^{\varGamma_{\text{FOT}_2}} = \{r'_{\mathcal{S}}, \mathsf{SimView}_{\mathcal{S}}^{\mathsf{OLE}}, qry'_1, qry'_2\}$$

By the transitivity of indistinguishability across the hybrids, we conclude:

$$\mathsf{View}_{\mathcal{A}}^{\text{Hybrid 0}} \overset{c}{\equiv} \mathsf{View}_{\mathcal{A}}^{\text{Hybrid 4}} \quad \Rightarrow \quad \mathsf{View}_{\mathcal{S},\mathcal{A}}^{\varGamma_{\text{FOT}_2}} \overset{c}{\equiv} \mathsf{View}_{\mathsf{Sim}_{\mathcal{S}}}^{\varGamma_{\text{FOT}_2}}$$

where $\mathsf{View}_{\mathcal{S},\mathcal{A}}^{\varGamma_{\text{FOT}_2}}$ and $\mathsf{View}_{\mathsf{Sim}_{\mathcal{S}}}^{\varGamma_{\text{FOT}_2}}$ are the real and ideal model views respectively. □

5 Unconditionally Secure Three-Party FOT

The main objective of this section is to develop *efficient* and *unconditionally secure* (hence post-quantum) concrete FOT schemes, without involving any public-key primitive. To attain our goal, we rely on a new combination of various techniques, such as permutation maps, one-time-pad, and padding, along with the assistance of a third-party $\mathcal{H}$ in the protocols. The third party's role is to aid in generating the result. However, we assume this party might be corrupted by a semi-honest adversary who does not collude with the protocols' participants. In practice, this party's role can be played by a weak (or semi-honest) $\mathcal{TEE}$ or server. As we will discuss, no secret key is stored in the $\mathcal{TEE}$ beyond what is used during the manufacturing of a $\mathcal{TEE}$ for remote attestation.

Since $\mathcal{H}$ helps $\mathcal{S}$ and $\mathcal{R}$ perform computation on their encrypted sets, there is a possibility of leakage to $\mathcal{H}$. Depending on the protocol that realizes F, this leakage could contain different types of information; for instance, it could contain (i) the number of identical inputs (e.g., identical labels in the context of machine learning) in $\mathcal{S}$'s input data or (ii) nothing at all. Often such leakage is defined as a leakage function $\varOmega$ that takes as input all parties' (encoded) inputs and returns the amount of leakage. We will use $\varOmega$ shortly, to define the security of three-party functional oblivious transfer.

5.1 Security Model

In this section, we present the formal definition of three-party functional OT which is an extension of the definition for two-party functional OT presented in Section 3. Three-Party Functional Oblivious Transfer (FOT_3) involves a sender $\mathcal{S}$, a receiver $\mathcal{R}$, and a third party $\mathcal{H}$. Each party might be corrupted by a semi-honest adversary. As in two-party FOT, in FOT_3, $\mathcal{S}$ has a vector of messages $\vec{m} = [m_0, \ldots, m_{n-1}]$ and $\mathcal{R}$ has a vector of t indices $\vec{e} = [b, \ldots, z]$. $\mathcal{H}$ has no input. The functionality that FOT_3 computes for a function F is also G^{F}, presented in Definition 5. Because $\mathcal{H}$ has no input and receives no output, beyond the scheme's parameters (t, n) known prior to all parties. Informally, the security of FOT_3 states that $\mathcal{S}$ learns nothing about $\mathcal{R}$'s input or the output y, and $\mathcal{R}$ learns nothing beyond the output y. $\mathcal{H}$ learns nothing beyond the output of predefined leakage $\varOmega$. Formally, $\mathcal{H}$'s view can be simulated given the leakage. Below, we formally state the security of FOT_3.

Definition 7 (Security of FOT_3). *Let G^{F} be the functional OT's functionality formalized in Definition 5. We assert that a three-party protocol $\varGamma$ securely realizes G^{F} in the presence of passive adversaries if for every adversary $\mathcal{A}$ in the real model, there is a simulator Sim in the ideal model, where:*

$$\Big\{\mathsf{Sim}_{\mathcal{S},aux}^{\varGamma}\big(\vec{m}, \epsilon\big)\Big\}_{\vec{m},\vec{e}} \equiv \Big\{\mathsf{View}_{\mathcal{S},\mathcal{A}(aux)}^{\varGamma}\big(\vec{m}, \vec{e}\big)\Big\}_{\vec{m},\vec{e}}$$

$$\Big\{\mathsf{Sim}_{\mathcal{R},aux}^{\varGamma}\Big(\vec{e}, \mathsf{G}^{\mathsf{F}}(\vec{m}, \vec{e})\Big)\Big\}_{\vec{m},\vec{e}} \equiv \Big\{\mathsf{View}_{\mathcal{R},\mathcal{A}(aux)}^{\varGamma}\big(\vec{m}, \vec{e}\big)\Big\}_{\vec{m},\vec{e}}$$

$$\Big\{\mathsf{Sim}_{\mathcal{H},\varOmega,aux}^{\varGamma}(\epsilon, \epsilon)\Big\}_{\vec{m},\vec{e}} \equiv \Big\{\mathsf{View}_{\mathcal{H},\mathcal{A}(aux)}^{\varGamma}\big(\vec{m}, \vec{e}\big)\Big\}_{\vec{m},\vec{e}}$$

The above definition excludes "PPT" adversary and includes "$\equiv$" (instead of "$\overset{c}{\equiv}$") because it formulates unconditional security. Next, we define the core security of the abstract algorithm Encode, introduced in Section 2. We require that Encode satisfies the simulation-based security definition, as outlined in Definition 1. More precisely, the view of an adversary $\mathcal{A}$, who (i) has selected the input plaintext messages $\vec{m} = [m_0, \ldots, m_{n-1}]$ and (ii) receives the output $\vec{h} \leftarrow \mathsf{Encode}(\vec{m}, \vec{pp}, \vec{sk})$, which may reveal certain structural information about $\vec{m}$, is simulated using: (a) $\mathcal{A}$'s input, (b) a leakage function Ψ that defines the structural information exposed by Encode, and (c) the output of Encode. This is formally stated below.

Definition 8 (Security of Encode). *An encoding scheme Encode is secure, if for every adversary $\mathcal{A}$, there is a simulator Sim such that:* $\left\{\mathsf{Sim}^{\mathsf{Encode}}_{\Psi, aux}(\vec{m}, \mathsf{Encode}(\vec{m}, \vec{pp}, \vec{sk}))\right\}_{\vec{m}, \vec{pp}, \vec{sk}} \equiv \left\{\mathsf{View}^{\mathsf{Encode}}_{\mathcal{A}(aux)}(\vec{m}, (\vec{pp}, \vec{sk}))\right\}_{\vec{m}, \vec{pp}, \vec{sk}}$, *where aux includes $\vec{pp}$.*

In the above definition, $\vec{m}$ represents $\mathcal{A}$'s input in the real-model view, whereas $(\vec{pp}, \vec{sk})$ are not part of its input. In particular, the secret keys in $\vec{sk}$ are not provided to $\mathcal{A}$.

5.2 An Overview of the Protocol

In this section, we introduce a protocol, denoted as Γ_{FOT_3}, which implements FOT_3. We treat Γ_{FOT_3} as a "skeleton" protocol because it will rely on three *abstract* algorithms, ($\mathsf{Encode}, \mathsf{Evaluate}, \mathsf{Decode}$), defined in Section 2. We assume that Encode is secure. Broadly speaking, Encode is an encryption algorithm, $\mathsf{Evaluate}$ is an algorithm for evaluating ciphertexts, and Decode is a decryption algorithm. Later, we will provide concrete instantiations of these algorithms for specific functions, such as Mean and Mode.

Γ_{FOT_3} mainly uses random permutation and a tool called a *permutation map* [5]. A permutation map is a vector indicating the new position of each element of a vector $\vec{v}$ of n elements after $\vec{v}$ is randomly permuted. In our protocol, the permutation map lets $\mathcal{R}$ instruct $\mathcal{H}$ on how to select t out of n permuted messages sent by $\mathcal{S}$, while ensuring that neither $\mathcal{H}$ nor $\mathcal{S}$ can deduce the original indices of these t messages. At a high level, Γ_{FOT_3} operates as follows. Initially, $\mathcal{R}$ selects n secret keys and sends them to $\mathcal{S}$. Furthermore, $\mathcal{R}$ generates two permutation maps: (1) $\vec{w}$ for $\mathcal{S}$, which allows $\mathcal{S}$ to *randomly* but *deterministically* permute the vector of n messages $\vec{m} = [m_0, \ldots, m_{n-1}]$ that it holds, and (2) $\vec{c}$ for $\mathcal{H}$, which enables $\mathcal{H}$ to obliviously find t (encrypted) messages out of n messages that have already been permuted by $\mathcal{S}$, without being able to identify their original indices.

$\mathcal{R}$ sends the permutation maps, $\vec{w}$ and $\vec{c}$, to $\mathcal{S}$ and $\mathcal{H}$. Next, $\mathcal{S}$ encodes the n messages using Encode and the secret key provided by $\mathcal{R}$. It then permutes the encoded messages using $\vec{w}$ and sends the result to $\mathcal{H}$. With the n permuted encoded messages and $\vec{c}$, $\mathcal{H}$ retrieves the t encoded messages and obliviously evaluates the function F on them using $\mathsf{Evaluate}$. This results in an encoded evaluated value. Finally, $\mathcal{H}$ sends the result to $\mathcal{R}$, which decodes it using the corresponding secret keys and Decode, yielding the plaintext evaluated value.

5.3 Detailed Description of Γ_{FOT_3}

In this section, we provide a detailed description of the protocol Γ_{FOT_3}. We assume that the system's public parameters, denoted as $\vec{pp}$, are known to all participants. These parameters mainly depend on the type of function F evaluated on the OT response, as well as the specific encryption and decryption methods (Encode and Decode) used by Γ_{FOT_3}. For instance, consider the case where: (i) F is defined as Mean, and (ii) Encode and Decode use a one-time pad. In this scenario, $\vec{pp}$ consists of: (a) a description of F, (b) a sufficiently large prime number p, which can be generated by anyone, (c) n, the number of messages to be encoded, and (d) t, the number of inputs to F, where $n \geq t$. Therefore, in this case, generating the parameters in $\vec{pp}$ does not require a trusted setup.

1. *Setup:* $\mathsf{Setup}(1^\lambda) \rightarrow \vec{r}$
   It is run by $\mathcal{R}$.
   (a) picks n random values $(r_0, \ldots, r_{n-1}) \overset{\$}{\leftarrow} \{0,1\}^\lambda$. Let $\vec{r}$ be $\vec{r} = [r_0, \ldots, r_{n-1}]$. These elements is used as a one-time pad by $\mathcal{S}$ to encrypt each message that $\mathcal{S}$ sends.
   (b) sends $\vec{r}$ to $\mathcal{S}$.
2. *Query Generation:* $\mathsf{GenQuery}(1^\lambda, \vec{e}) \rightarrow qry := (qry_{\mathcal{S}}, qry_{\mathcal{H}})$
   It is run by $\mathcal{R}$.
   (a) determines to which position, each index in a vector $\vec{v}$ of size n is moved, if $\vec{v}$ is randomly permuted once. To do that, it takes the following steps.
      i. initiates a vector $\vec{v}$, such that its i-th element is set to i as:
$$\forall i, 0 \leq i \leq n-1: \quad \vec{v}[i] \leftarrow i$$
      ii. randomly permutes $\vec{v}$ as: $\pi(\vec{v}) \rightarrow \vec{w}$.
   (b) finds the index of each element of its index vector $\vec{e}$ in $\vec{w}$. To do that, it initiates an empty vector $\vec{c}$ of size t and takes the following steps.
$$\forall j, 0 \leq j \leq t-1: \quad \mathsf{Find}(\vec{w}, \vec{e}[j]) \rightarrow c_j, \quad \vec{c}[j] \leftarrow c_j$$
   Recall that $\vec{e}$ contains the indices of $\mathcal{R}$'s t preferred elements in $[1, \ldots, n]$, while $\vec{c}$ determines the position of these indices in $\vec{e}$ after they are permuted based on the permutation map $\vec{w}$.
   (c) sets $qry_{\mathcal{S}} \leftarrow \vec{w}$ and $qry_{\mathcal{H}} \leftarrow \vec{c}$. It sends $qry_{\mathcal{S}}$ to $\mathcal{S}$ and $qry_{\mathcal{H}}$ to $\mathcal{H}$.
3. *Response Generation:* $\mathsf{GenRes}(m_0, \ldots, m_{n-1}, \vec{r}, qry_{\mathcal{S}}, \vec{pp}) \rightarrow res_{\mathcal{H}}$
   It is run by $\mathcal{S}$.
   (a) encrypts each message in $\vec{m} = [m_0, \ldots, m_{n-1}]$ using the elements of $\vec{r} = [r_0, \ldots, r_{n-1}]$ as:
$$\mathsf{Encode}(\vec{m}, \vec{pp}, \vec{r}) \rightarrow \vec{h}$$
   (b) permutes vector $\vec{h}$ according the permutation map $\vec{w} \in qry_{\mathcal{S}}$. To do that, it initiates an empty vector $\vec{x}$ of size n. It finds the position of each value i in the permuted vector $\vec{w}$, let i' denote that position. It inserts the i-th element from $\vec{h}$ into the i'-th position in $\vec{x}$. Specifically,
$$\forall i, 0 \leq i \leq n-1: \quad \mathsf{Find}(\vec{w}, i) \rightarrow i', \quad \vec{x}[i'] \leftarrow \vec{h}[i]$$
   (c) sets $res_{\mathcal{H}} \leftarrow \vec{x}$ and sends $res_{\mathcal{H}}$ to $\mathcal{H}$.
4. *Oblivious Evaluation:* $\mathsf{OblEvaluate}(res_{\mathcal{H}}, qry_{\mathcal{H}}) \rightarrow res_{\mathcal{R}}$
   It is run by $\mathcal{H}$.
   (a) uses elements of $\vec{c} \in qry_{\mathcal{H}}$ to retrieve $\mathcal{R}$'s preferred encrypted messages in the permuted vector $\vec{x} \in res_{\mathcal{H}}$ and append them to an empty vector $\vec{u}$. Specifically, it takes the following steps.
$$\forall j, 0 \leq j \leq t-1: \quad \vec{u}[j] \leftarrow \vec{x}\Big[\vec{c}[j]\Big]$$
   (b) obliviously evaluates the function F (specified in $\vec{pp}$) on the plaintext encoded in $\vec{u}$, by executing:
$$\mathsf{Evaluate}(\vec{u}, \vec{pp}) \rightarrow \theta$$
   (c) sets $res_{\mathcal{R}}$ to θ and sends $res_{\mathcal{R}}$ to $\mathcal{R}$.
5. *Message Extraction:* $\mathsf{Retrieve}(res_{\mathcal{R}}, \vec{r}, \vec{e}, \vec{pp}) \rightarrow y$
   It is run by $\mathcal{R}$.
   – retrieves the related secret keys for decoding. To do that, it initiates an empty vector $\vec{g}$ and appends to $\vec{g}$ the secret keys in $\vec{r}$ whose indices are specified in $\vec{e}$:
$$\forall j, 0 \leq j \leq t-1: \quad \vec{g}[j] \leftarrow r_{\vec{e}[j]}$$
   – decodes θ by invoking $\mathsf{Decode}(\theta, \vec{pp}, \vec{g}) \rightarrow y$.

Theorem 2. *Let G^{F} denote the functionality defined in Definition 5, and let* Encode *be an encoding scheme that is secure according to Definition 8. Then, the protocol Γ_{FOT_3} securely realizes G^{F} according to Definition 7.*

5.4 Security Proof of $\varGamma_{\text{FOT}_3}$

In this section, we prove the security of $\varGamma_{\text{FOT}_3}$, i.e., Theorem 2.

Proof. We prove Theorem 2 in the case where each party is corrupt. Since the public parameters $\overrightarrow{pp}$ are known to all participants of the protocol in the real model, the simulator will be provided with the $\overrightarrow{pp}$ as well. As before, the proof involves a sequence of hybrid experiments, transitioning from the real execution to the ideal execution.

Corrupt $\mathcal{R}$. We begin by considering the case where the receiver $\mathcal{R}$ is corrupt. We will employ a simulator $\mathsf{Sim}_{\mathcal{R}}^{\varGamma_{\text{FOT}_3}}$ (which receives $\mathcal{R}$'s input $\vec{e}$ and output y) to generate an ideal execution view incrementally through a sequence of hybrids, i.e., $\mathsf{Sim}_{\mathcal{R}}^{\varGamma_{\text{FOT}_3}}(\vec{e}, y) \rightarrow \mathsf{View}_{\mathsf{Sim}_{\mathcal{R}}}^{\varGamma_{\text{FOT}_3}}$.

- Hybrid 0 ($\mathsf{View}_{\mathcal{A}}^{\text{Hybrid 0}}$): Real execution. The adversary's view in the real execution of $\varGamma_{\text{FOT}_3}$ is:

$$\mathsf{View}_{\mathcal{A}}^{\text{Hybrid 0}} = \{r_{\mathcal{R}}, res_{\mathcal{R}}\}$$

  where $r_{\mathcal{R}}$ is the random coin sampled uniformly at random by $\mathcal{R}$ and $res_{\mathcal{R}}$ is the response computed by the protocol using $\mathsf{Evaluate}$.
- Hybrid 1 ($\mathsf{View}_{\mathcal{A}}^{\text{Hybrid 1}}$): Replace $r_{\mathcal{R}}$ with $r'_{\mathcal{R}}$. In this hybrid, the simulator replaces the random coin $r_{\mathcal{R}}$ with a freshly generated random coin $r'_{\mathcal{R}}$. The adversary's view becomes:

$$\mathsf{View}_{\mathcal{A}}^{\text{Hybrid 1}} = \{r'_{\mathcal{R}}, res_{\mathcal{R}}\}$$

  Since $r_{\mathcal{R}}$ and $r'_{\mathcal{R}}$ are both uniformly random, the distributions of $\mathsf{View}_{\mathcal{A}}^{\text{Hybrid 0}}$ and $\mathsf{View}_{\mathcal{A}}^{\text{Hybrid 1}}$ are identical:

$$\mathsf{View}_{\mathcal{A}}^{\text{Hybrid 0}} \equiv \mathsf{View}_{\mathcal{A}}^{\text{Hybrid 1}}$$

- Hybrid 2 ($\mathsf{View}_{\mathcal{A}}^{\text{Hybrid 2}}$): Replace $res_{\mathcal{R}}$ with $res'_{\mathcal{R}}$. In this hybrid, the simulator replaces $res_{\mathcal{R}}$ with a simulated response $res'_{\mathcal{R}}$, generated as follows:
  - The simulator computes a random value z, using $r'_{\mathcal{R}}$.
  - The simulator uses z to encode the output y via the encoding algorithm: $\mathsf{Encode}(\vec{y}, \vec{pp}, z) \rightarrow res'_{\mathcal{R}}$. Here, $res'_{\mathcal{R}}$ is constructed such that it can be decoded to y, satisfying: $\mathsf{Decode}(res'_{\mathcal{R}}, \vec{pp}, z) \rightarrow y$.

  Since $res_{\mathcal{R}}$ in the real protocol and $res'_{\mathcal{R}}$ in this hybrid are encoded using fresh randomness and both yield the same decoded output y, their distributions are identical. Thus, the adversary cannot distinguish between Hybrid 1 and Hybrid 2:

$$\mathsf{View}_{\mathcal{A}}^{\text{Hybrid 1}} \equiv \mathsf{View}_{\mathcal{A}}^{\text{Hybrid 2}}$$

- Hybrid 3 ($\mathsf{View}_{\mathcal{A}}^{\text{Hybrid 3}}$): Ideal model. In this final hybrid, the simulator replaces the real execution entirely with the ideal execution. The adversary's view becomes:

$$\mathsf{View}_{\mathcal{A}}^{\text{Hybrid 3}} = \{r'_{\mathcal{R}}, res'_{\mathcal{R}}\}.$$

  By the simulator's construction, the distributions of $\mathsf{View}_{\mathcal{A}}^{\text{Hybrid 2}}$ and $\mathsf{View}_{\mathcal{A}}^{\text{Hybrid 3}}$ are identical:

$$\mathsf{View}_{\mathcal{A}}^{\text{Hybrid 2}} \equiv \mathsf{View}_{\mathcal{A}}^{\text{Hybrid 3}}$$

Conclusion. In Hybrid 3, the simulator has fully constructed the ideal view:

$$\mathsf{View}_{\mathsf{Sim}_{\mathcal{S}}}^{\varGamma_{\text{FOT}_3}} = \{r'_{\mathcal{R}}, res'_{\mathcal{R}}\}$$

By the transitivity of indistinguishability across the hybrids, we conclude:

$$\mathsf{View}_{\mathcal{A}}^{\text{Hybrid 0}} \equiv \mathsf{View}_{\mathcal{A}}^{\text{Hybrid 3}} \quad \Rightarrow \quad \mathsf{View}_{\mathcal{R},\mathcal{A}}^{\varGamma_{\text{FOT}_3}} \equiv \mathsf{View}_{\mathsf{Sim}_{\mathcal{R}}}^{\varGamma_{\text{FOT}_3}}$$

where $\mathsf{View}_{\mathcal{R}}^{\varGamma_{\text{FOT}_3}}$ and $\mathsf{View}_{\mathsf{Sim}_{\mathcal{R}}}^{\varGamma_{\text{FOT}_3}}$ are the real and ideal model views respectively.

Corrupt $\mathcal{S}$. We now consider the case where the sender $\mathcal{S}$ is corrupt. We will use a simulator $\mathsf{Sim}_{\mathcal{S}}^{\Gamma_{\text{FOT}_3}}$ (which receives $\mathcal{S}$'s input $\vec{m}$) to generate an ideal execution view, i.e., $\mathsf{Sim}_{\mathcal{S}}^{\Gamma_{\text{FOT}_3}}(\vec{m}) \rightarrow \mathsf{View}_{\mathsf{Sim}_{\mathcal{S}}}^{\Gamma_{\text{FOT}_3}}$.

- Hybrid 0 ($\mathsf{View}_{\mathcal{A}}^{\text{Hybrid 0}}$): Real execution. The adversary's view in the real execution of Γ_{FOT_3} is:

$$\mathsf{View}_{\mathcal{A}}^{\text{Hybrid 0}} = \{r_{\mathcal{S}}, \vec{r}, \vec{w}\}$$

  where $r_{\mathcal{S}}$ is the random coin used by $\mathcal{S}$, $\vec{r}$ is the vector of random values exchanged during the protocol, and $\vec{w}$ is a permutation map received from $\mathcal{R}$.
- Hybrid 1 ($\mathsf{View}_{\mathcal{A}}^{\text{Hybrid 1}}$): Replace $r_{\mathcal{S}}$ with $r'_{\mathcal{S}}$. In this hybrid, the simulator replaces the random coin $r_{\mathcal{S}}$ with a freshly generated random coin $r'_{\mathcal{S}}$. The adversary's view becomes:

$$\mathsf{View}_{\mathcal{A}}^{\text{Hybrid 1}} = \{r'_{\mathcal{S}}, \vec{r}, \vec{w}\}$$

  Since $r_{\mathcal{S}}$ and $r'_{\mathcal{S}}$ are both uniformly random, the distributions of $\mathsf{View}_{\mathcal{A}}^{\text{Hybrid 0}}$ and $\mathsf{View}_{\mathcal{A}}^{\text{Hybrid 1}}$ are identical:

$$\mathsf{View}_{\mathcal{A}}^{\text{Hybrid 0}} \equiv \mathsf{View}_{\mathcal{A}}^{\text{Hybrid 1}}$$

- Hybrid 2 ($\mathsf{View}_{\mathcal{A}}^{\text{Hybrid 2}}$): Replace $\vec{r}$ with $\vec{z}$. In this hybrid, the simulator replaces the vector of random values $\vec{r}$ with a simulated vector $\vec{z} = [z_0, \ldots, z_{n-1}]$, where $z_i \xleftarrow{\$} \{0,1\}^{\lambda}$. The adversary's view becomes:

$$\mathsf{View}_{\mathcal{A}}^{\text{Hybrid 2}} = \{r'_{\mathcal{S}}, \vec{z}, \vec{w}\}$$

  Since $\vec{r}$ and $\vec{z}$ are both vectors of independently sampled uniform random values, the distributions of $\mathsf{View}_{\mathcal{A}}^{\text{Hybrid 1}}$ and $\mathsf{View}_{\mathcal{A}}^{\text{Hybrid 2}}$ are identical:

$$\mathsf{View}_{\mathcal{A}}^{\text{Hybrid 1}} \equiv \mathsf{View}_{\mathcal{A}}^{\text{Hybrid 2}}$$

- Hybrid 3 ($\mathsf{View}_{\mathcal{A}}^{\text{Hybrid 3}}$): Replace $\vec{w}$ with $\vec{w'}$. In this final hybrid, the simulator replaces the permutation map $\vec{w}$ with simulated map $\vec{w'}$, where $\vec{w'}$ is generated by applying a random permutation π to $[0, n-1]$. The adversary's view becomes:

$$\mathsf{View}_{\mathcal{A}}^{\text{Hybrid 3}} = \{r'_{\mathcal{S}}, \vec{z}, \vec{w'}\}$$

  Since $\vec{w}$ and $\vec{w'}$ are both random permutations of $[0, n-1]$, their distributions are identical. Thus, the distributions of $\mathsf{View}_{\mathcal{A}}^{\text{Hybrid 2}}$ and $\mathsf{View}_{\mathcal{A}}^{\text{Hybrid 3}}$ are identical:

$$\mathsf{View}_{\mathcal{A}}^{\text{Hybrid 2}} \equiv \mathsf{View}_{\mathcal{A}}^{\text{Hybrid 3}}$$

Conclusion. By the end of Hybrid 3, the simulator has constructed the full ideal view:

$$\mathsf{View}_{\mathsf{Sim}_{\mathcal{S}}}^{\Gamma_{\text{FOT}_3}} = \{r'_{\mathcal{S}}, \vec{z}, \vec{w'}\}$$

Therefore, it holds that:

$$\mathsf{View}_{\mathcal{A}}^{\text{Hybrid 0}} \equiv \mathsf{View}_{\mathcal{A}}^{\text{Hybrid 3}} \quad \Rightarrow \quad \mathsf{View}_{\mathcal{S},\mathcal{A}}^{\Gamma_{\text{FOT}_3}} \equiv \mathsf{View}_{\mathsf{Sim}_{\mathcal{S}}}^{\Gamma_{\text{FOT}_3}}$$

where $\mathsf{View}_{\mathcal{S},\mathcal{A}}^{\Gamma_{\text{FOT}_3}}$ and $\mathsf{View}_{\mathsf{Sim}_{\mathcal{S}}}^{\Gamma_{\text{FOT}_3}}$ are the real and ideal model views respectively, as formalized in Section 5.1.

Corrupt $\mathcal{H}$. We now consider the case where the third party $\mathcal{H}$ is corrupt. We denote $\mathcal{F}^{\Psi}_{\mathsf{Encode}}$ as an ideal functionality of Encode. The functionality is parameterized with the leakage function Ψ, it takes a vector of messages and outputs encoding of the messages according to the leakage function.

We will use a simulator $\mathsf{Sim}^{\Gamma_{\mathrm{FOT}_3}}_{\mathcal{H}}$ to generate an ideal execution view using the auxiliary data (public parameters $\vec{pp}$), i.e., $\mathsf{Sim}^{\Gamma_{\mathrm{FOT}_3}}_{\mathcal{H}}(\vec{pp}) \rightarrow \mathsf{View}^{\Gamma_{\mathrm{FOT}_3}}_{\mathsf{Sim}_{\mathcal{H}}}$.

- Hybrid 0 ($\mathsf{View}^{\text{Hybrid 0}}_{\mathcal{A}}$): Real execution. The adversary's view in the real execution of Γ_{FOT_3} is:

  $$\mathsf{View}^{\text{Hybrid 0}}_{\mathcal{A}} = \{r_{\mathcal{H}}, \vec{c}, \mathsf{View}^{\mathsf{Encode}}_{\mathcal{H}}\}$$

  where:
  - $r_{\mathcal{H}}$ is the random coin generated internally by $\mathcal{H}$.
  - $\vec{c}$ is a t-element vector of indices, which is a subset of a randomly permuted vector $\vec{w}$.
  - $\mathsf{View}^{\mathsf{Encode}}_{\mathcal{H}}$ refers to $\mathcal{H}$'s real-model view during the execution of Encode. Note that, as defined in Section 2.1, a view like $\mathsf{View}^{\mathsf{Encode}}_{\mathcal{H}}$ also includes the output of Encode that a party may receive (if the protocol allows to). However, in the real execution of Γ_{FOT_3}, $\mathcal{H}$ receives the output of Encode that has been permuted randomly based on a permutation map. Thus, we allow $\mathsf{View}^{\mathsf{Encode}}_{\mathcal{H}}$ to contain the *permuted* output of Encode rather than its direct output.
- Hybrid 1 ($\mathsf{View}^{\text{Hybrid 1}}_{\mathcal{A}}$): Replace $r_{\mathcal{H}}$ with $r'_{\mathcal{H}}$. In this hybrid, the simulator replaces $r_{\mathcal{H}}$ with a freshly generated random coin $r'_{\mathcal{H}}$. The adversary's view becomes:

  $$\mathsf{View}^{\text{Hybrid 1}}_{\mathcal{A}} = \{r'_{\mathcal{H}}, \vec{c}, \mathsf{View}^{\mathsf{Encode}}_{\mathcal{H}}\}$$

  Since $r_{\mathcal{H}}$ and $r'_{\mathcal{H}}$ are both uniformly random, the distributions of $\mathsf{View}^{\text{Hybrid 0}}_{\mathcal{A}}$ and $\mathsf{View}^{\text{Hybrid 1}}_{\mathcal{A}}$ are identical:

  $$\mathsf{View}^{\text{Hybrid 0}}_{\mathcal{A}} \equiv \mathsf{View}^{\text{Hybrid 1}}_{\mathcal{A}}$$

- Hybrid 2 ($\mathsf{View}^{\text{Hybrid 2}}_{\mathcal{A}}$): Replace $\vec{c}$ with $\vec{c'}$. In this hybrid, the simulator replaces $\vec{c}$ with a simulated vector $\vec{c'} = [c'_0, \ldots, c'_{t-1}]$, where each c'_j is selected uniformly at random without replacement from $[0, n-1]$, ensuring all elements are unique. The adversary's view becomes:

  $$\mathsf{View}^{\text{Hybrid 2}}_{\mathcal{A}} = \{r'_{\mathcal{H}}, \vec{c'}, \mathsf{View}^{\mathsf{Encode}}_{\mathcal{H}}\}$$

  We now argue that the distributions of $\vec{c}$ in Hybrid 1 and $\vec{c'}$ in Hybrid 2 are identical. In the real model (and accordingly Hybrid 1), $\vec{c}$ is a subset of $\vec{w}$, which is a randomly permuted vector of $[0, n-1]$. Since $\vec{w}$ is uniformly permuted, each element of $\vec{c}$ is chosen uniformly at random from $[0, n-1]$, and the probability of any specific value appearing in a given position of $\vec{c}$ is $\frac{1}{n}$.

  In Hybrid 2, $\vec{c'}$ is generated by selecting t unique elements uniformly at random without replacement from $[0, n-1]$. Despite this difference in construction, the probability of any specific value appearing in a given position of $\vec{c'}$ is also $\frac{1}{n}$.

  Thus, the distributions of $\vec{c}$ and $\vec{c'}$ are identical, and the adversary cannot distinguish between Hybrid 1 and Hybrid 2:

  $$\mathsf{View}^{\text{Hybrid 1}}_{\mathcal{A}} \equiv \mathsf{View}^{\text{Hybrid 2}}_{\mathcal{A}}$$

- Hybrid 3 ($\mathsf{View}^{\text{Hybrid 3}}_{\mathcal{A}}$): Replace $\mathsf{View}^{\mathsf{Encode}}_{\mathcal{H}}$ with $\vec{x'}$. In this final hybrid, the simulator replaces $\mathsf{View}^{\mathsf{Encode}}_{\mathcal{H}}$ with $\vec{x'}$, the randomly permuted output of $\mathcal{F}^{\Psi}_{\mathsf{Encode}}$, generated as follows:
  - The simulator selects n messages $\vec{\bar{m}} = [\bar{m}_0, \ldots, \bar{m}_{n-1}]$ uniformly at random from the message domain U.
  - The simulator sends $\vec{\bar{m}}$ to $\mathcal{F}^{\Psi}_{\mathsf{Encode}}$ and receives the output $\vec{\bar{m}'}$
  - The simulator applies a random permutation to $\vec{\bar{m}'}$ to obtain $\vec{x'}$.

The adversary's view becomes:

$$\mathsf{View}_{\mathcal{A}}^{\text{Hybrid 3}} = \{r'_{\mathcal{H}}, \vec{c'}, \vec{x'}\}$$

Since the permuted output of $\mathcal{F}_{\mathsf{Encode}}^{\Psi}$ in Hybrid 3 is indistinguishable from $\mathsf{View}_{\mathcal{H}}^{\mathsf{Encode}}$ in the real model (and accordingly Hybrid 2) due to the $\mathcal{F}_{\mathsf{Encode}}^{\Psi}$-hybrid model assumption, we have:

$$\mathsf{View}_{\mathcal{A}}^{\text{Hybrid 2}} \equiv \mathsf{View}_{\mathcal{A}}^{\text{Hybrid 3}}$$

Conclusion. In Hybrid 3, the simulator has fully constructed the ideal view:

$$\mathsf{View}_{\mathsf{Sim}_{\mathcal{H}}}^{\Gamma_{\text{FOT}_3}} = \{r'_{\mathcal{H}}, \vec{c'}, \vec{x'}\}$$

By the transitivity of indistinguishability across the hybrids, we conclude:

$$\mathsf{View}_{\mathcal{A}}^{\text{Hybrid 0}} \equiv \mathsf{View}_{\mathcal{A}}^{\text{Hybrid 3}} \quad \Rightarrow \quad \mathsf{View}_{\mathcal{H},\mathcal{A}}^{\Gamma_{\text{FOT}_3}} \equiv \mathsf{View}_{\mathsf{Sim}_{\mathcal{H}}}^{\Gamma_{\text{FOT}_3}}$$

where $\mathsf{View}_{\mathcal{H},\mathcal{A}}^{\Gamma_{\text{FOT}_3}}$ and $\mathsf{View}_{\mathsf{Sim}_{\mathcal{H}}}^{\Gamma_{\text{FOT}_3}}$ are the real and ideal model views respectively. □

5.5 Subroutines for Mean

Given a vector of plaintext messages $m = [m_0, \dots, m_{n-1}]$ and vectors of secret keys $[r_1, \dots, r_n]$, the encoding algorithm $\mathsf{Encode}_{\mathsf{Mean}}(\vec{m}, \vec{pp}, \vec{sk})$ mainly involves modular addition of each secret key r_i and the related message m_i. It returns n encoded values. This simple encoding (or encryption) method offers several benefits. It ensures unconditional security while being highly efficient. As we will see shortly, using modular addition lets us perform oblivious evaluation and ultimately decode the evaluated value to obtain a correct result. Figure 3a explains how $\mathsf{Encode}_{\mathsf{Mean}}$ works. Next, we explain how $\mathsf{Evaluate}_{\mathsf{Mean}}$ works.

Recall that a plain $\mathsf{Mean}(a_0, \dots, a_{t-1})$ with t plaintext inputs is defined as $\frac{1}{t} \cdot \sum_{i=0}^{t-1} a_i$. Hence, for $\mathsf{Evaluate}_{\mathsf{Mean}}$, aiming at simulating Mean's functionality, to work over $\mathbb{F}_p$ and values $\vec{h} = [h_0, \dots, h_{t-1}]$ that are encoded using the approach explained above, we require $\mathsf{Evaluate}_{\mathsf{Mean}}(\vec{h}, \vec{pp})$ to compute $\theta = t^{-1} \cdot \sum_{i=0}^{t-1} h_i \bmod p$ as the evaluated encoded value. Figure 3b shows how $\mathsf{Evaluate}_{\mathsf{Mean}}$ works. Given the encoded evaluated value θ, using the secret keys, one can decode (or decrypt) it by subtracting $t^{-1} \cdot \sum_{i=0}^{t-1} r_i$ from θ. This decoding is possible and easy due to the way the plaintext messages were encoded and evaluated. Figure 3c presents $\mathsf{Evaluate}_{\mathsf{Mean}}$.

Theorem 3. *Let* Encode *be the functionality defined in Section 2 and* Ψ *be a leakage function that always returns empty, i.e., no leakage. Then,* $\mathsf{Encode}_{\mathsf{Mean}}$ *(presented in Figure 3a) securely realizes* Encode*, w.r.t. Definition 8.*

Note that given that Γ_{FOT_3} and $\mathsf{Encode}_{\mathsf{Mean}}$ are secure, the hybrid argument ensures that the protocol Γ_{FOT_3} using $\mathsf{Encode}_{\mathsf{Mean}}$ satisfies the security requirements outlined in Definition 7. Moreover, as $\mathsf{Evaluate}_{\mathsf{Mean}}$ operates solely on the output of $\mathsf{Encode}_{\mathsf{Mean}}$ without using any secret keys, its security is guaranteed by the security of $\mathsf{Encode}_{\mathsf{Mean}}$. Specifically, the security of $\mathsf{Encode}_{\mathsf{Mean}}$ ensures that executing any function or algorithm—including $\mathsf{Evaluate}_{\mathsf{Mean}}$—does not reveal any information about the plaintext messages. Furthermore, $\mathsf{Decode}_{\mathsf{Mean}}$ reveals no information beyond the output of $\mathsf{Evaluate}_{\mathsf{Mean}}$, as $\mathsf{Evaluate}_{\mathsf{Mean}}$ is executed honestly on the ciphertexts by a semi-honest adversary.

$\mathsf{Encode}_{\mathsf{Mean}}(\vec{m}, \vec{pp}, \vec{sk}) \to \vec{h}$

- *Input.* $\vec{m} = [m_0, \ldots, m_{n-1}]$: a vector of plaintext messages, $\vec{pp}$: public parameter containing (i) a security parameter λ, (ii) a sufficiently large prime number p, where $\log_2(p) = \lambda$, (iii) t: the number of inputs to Mean, and (iv) n: the size of $\vec{m}$, where $n \geq t$, and $\vec{sk} = [r_0, \ldots, r_{n-1}]$: a vector of secret keys selected uniformly at random, where $\log_2(r_i) = \lambda$.
- *Output.* $\vec{h} = [h_0, \ldots, h_{n-1}]$: a vector of encoded messages.

1. Encodes each element using each r_i as a one-time pad:

$$\forall i, 0 \leq i \leq n-1: \quad h_i = m_i + r_i \bmod p$$

2. Returns $\vec{h} = [h_0, \ldots, h_{n-1}]$.

(a) Encoding algorithm.

$\mathsf{Evaluate}_{\mathsf{Mean}}(\vec{h}, \vec{pp}) \to \theta$

- *Input.* $\vec{h} = [h_0, \ldots, h_{t-1}]$: a vector of encoded messages, and $\vec{pp}$: the public parameters, containing $[\lambda, p, n, t]$.
- *Output.* θ: an encoded evaluated value.

1. Computes an encoded output of Mean, by multiplying the multiplicative inverse of t by the sum of the elements in $\vec{h}$ as:

$$\begin{aligned}\theta &= t^{-1} \cdot \sum_{i=0}^{t-1} h_i \bmod p \\ &= t^{-1} \cdot \sum_{i=0}^{t-1} m_i + r_i \bmod p\end{aligned}$$

2. Returns θ.

(b) Evaluation algorithm.

$\mathsf{Decode}_{\mathsf{Mean}}(\theta, \vec{pp}, \vec{sk}) \to y$

- *Input.* θ: an encoded evaluated value, $\vec{pp}$: the public parameters containing $[\lambda, p, n, t]$, and $\vec{sk} = [r_0, \ldots, r_{t-1}]$: the vector of secret keys.
- *Output.* y: a decoded evaluated value.

1. Decodes θ in a way that the result is the evaluated value. Specifically:

$$\begin{aligned}y &= \theta - t^{-1} \cdot \sum_{i=0}^{t-1} r_i \bmod p \\ &= t^{-1} \cdot \sum_{i=0}^{t-1} m_i \bmod p\end{aligned}$$

2. Returns y.

(c) Decoding algorithm.

Fig. 3: Encoding, evaluation, and decoding algorithms for secure evaluation of Mean.

5.6 Proof of Theorem 3

Proof. We will show that the adversary's view in the real model and the ideal model are indistinguishable. To do so, we use a hybrid argument.

- Hybrid 0: Real model. The first hybrid corresponds to the real model. The adversary's view during the execution of $\mathsf{Encode}_{\mathsf{Mean}}$ includes:

$$\mathsf{View}_{\mathcal{A}}^{\mathsf{Encode}_{\mathsf{Mean}}} = \{r_{\mathcal{A}}, \vec{h}\}$$

where $r_{\mathcal{A}}$ is the random output of the adversary's internal coin tosses, and $\vec{h}$ is the vector of encoded messages, with each element masked using fresh one-time pads.

- Hybrid 1: Replace randomness. In the first hybrid, we replace $r_{\mathcal{A}}$ with $r'_{\mathcal{A}}$, which is sampled uniformly at random as in the ideal model. Since $r_{\mathcal{A}}$ and $r'_{\mathcal{A}}$ are both sampled independently and uniformly from the same domain, their distributions are identical:

$$\mathsf{View}_{\mathcal{A}}^{\mathsf{Encode}_{\mathsf{Mean}}} \equiv \mathsf{View}_{\mathcal{A}}^{\text{Hybrid 1}}$$

- Hybrid 2: Replace encoded messages. In the second hybrid, we replace the vector of encoded messages $\vec{h}$ with a vector $\vec{z}$, where each $z_i \in \vec{z}$ is sampled uniformly at random from $\{0,1\}^{\lambda}$.

  By the perfect secrecy of the one-time pad, each element of $\vec{h}$ (masked with a fresh one-time pad) is indistinguishable from a uniformly random value. Thus, replacing $\vec{h}$ with $\vec{z}$ does not change the adversary's view:

$$\mathsf{View}_{\mathcal{A}}^{\text{Hybrid 1}} \equiv \mathsf{View}_{\mathcal{A}}^{\text{Hybrid 2}}$$

- Hybrid 3: Ideal model. In the final hybrid, we arrive at the ideal model. Here, the simulator constructs the adversary's view as:

$$\mathsf{View}_{\mathsf{Sim}_{\mathcal{A}}}^{\mathsf{Encode}_{\mathsf{Mean}}} = \{r'_{\mathcal{A}}, \vec{z}\}$$

  where $r'_{\mathcal{A}}$ is a uniformly random coin toss and $\vec{z}$ is a vector of n uniformly random values. This hybrid matches the ideal model by definition. Thus:

$$\mathsf{View}_{\mathcal{A}}^{\text{Hybrid 3}} = \mathsf{View}_{\mathsf{Sim}_{\mathcal{A}}}^{\mathsf{Encode}_{\mathsf{Mean}}}$$

Conclusion. By the transitivity of indistinguishability, we have:

$$\mathsf{View}_{\mathcal{A}}^{\mathsf{Encode}_{\mathsf{Mean}}} \equiv \mathsf{View}_{\mathsf{Sim}_{\mathcal{A}}}^{\mathsf{Encode}_{\mathsf{Mean}}}$$

Thus, $\mathsf{Encode}_{\mathsf{Mean}}$ securely realizes Encode, completing the proof. □

5.7 Subroutines for Mode

The design of encoding, evaluation, and decoding algorithms for Mode is more involved than for Mean as in the former case we need to address two primary challenges, outlined below.

– *Encoding private inputs that supports post-quantum secure evaluation.* As defined in Section 2, the plain Mode involves finding the most frequent element among the input set. The main challenge is to develop an encoding algorithm $\mathsf{Encode}_{\mathsf{Mode}}(\vec{m}, \vec{pp}, \vec{sk})$ that encodes each plaintext message in $\vec{m}$ such that, later on, the execution of Mode on these encoded messages will result in a correct output while ensuring the privacy of the plaintext messages and output against quantum adversaries that perform the evaluation. Briefly, to address these challenges, we require $\mathsf{Encode}_{\mathsf{Mode}}$ to encrypt the input plaintext messages *deterministically using a one-time pad.* Specifically, $\mathsf{Encode}_{\mathsf{Mode}}$ assigns a secret key from $\vec{sk}$ to each *unique* plaintext message in $\vec{m}$. Thus, two identical plaintext messages will have identical ciphertext. This allows one to execute Mode on these encoded messages without needing to have the knowledge of the plaintext messages. Also, since one-time pads are unconditionally secure, the plaintext messages remain secure against quantum adversaries.

– *Finding a correct key.* The next challenge is to enable $\mathsf{Decode}_{\mathsf{Mode}}(\theta, \vec{pp}, \vec{sk})$ to identify a correct secret key to decode and retrieve a correct plaintext message. Using an incorrect key to decode a message will result in output that does not match the original plaintext message. This challenge arises for two main reasons: (1) by definition of Mode, the algorithm $\mathsf{Evaluate}_{\mathsf{Mean}}(\vec{h}, \vec{pp}) \rightarrow \theta$ outputs only a single (or a small set of) output(s) and (2) the party who executes $\mathsf{Decode}_{\mathsf{Mode}}(\theta, \vec{pp}, \vec{sk})$, which is the receiver in the context of OT, does not know which key was used to encrypt the output θ. To address this challenge and help $\mathsf{Decode}_{\mathsf{Mode}}(\theta, \vec{pp}, \vec{sk})$ output a correct plaintext message, we use a padding technique, discussed in Section 2. This technique adds a pad to each plaintext message, accordingly imposing a certain structure to it before the message is encoded. Within the decoding process, every input secret key in $\overrightarrow{sk}$ is used to decode θ. Then, the result is checked to ensure whether it has the predefined structure. If it still possesses the structure, then the decoding process stops and that message will be considered as the decoded message (after the pad is removed).

```
$\mathsf{Encode}_{\mathsf{Mode}}(\vec{m}, \vec{pp}, \overrightarrow{sk}) \rightarrow \vec{h}$

• Input. $\vec{m} = [m_0, \dots, m_{n-1}]$: plaintext messages, $\vec{pp} = [\lambda, p, \gamma, n, t]$: public parameters, and $\overrightarrow{sk} = [r_0, \dots, r_{n-1}]$: a vector of secret keys selected uniformly at random, where $\log_2(r_i) = \lambda$.
• Output. $\vec{h} = [h_0, \dots, h_{n-1}]$: encoded messages.
1. Pads each element with $\gamma$ zeros:
       $\forall i, 0 \le i \le n-1 : \quad \mathsf{Pad}(m_i, \gamma) \rightarrow \bar{m}_i$
2. Encodes deterministically each unique $\bar{m}_i$ using a fresh $r_i$, by creating an empty vector $\vec{v}$ and taking the following steps. $\forall i, 0 \le i \le n-1 :$
   – if $\nexists\ \bar{m}_j \in \vec{v},\ s.t.\ \bar{m}_i = \bar{m}_j$, then:
      (a) $h_i = \bar{m}_i + r_i \bmod p$
      (b) $\vec{v} \leftarrow (\bar{m}_i, i)$
   – else if $\exists\ \bar{m}_j \in \vec{v},\ s.t.\ \bar{m}_i = \bar{m}_j$, then:
      ◇ $h_i = \bar{m}_i + r_j \bmod p$
3. Returns $\vec{h} = [h_0, \dots, h_{n-1}]$.
```

(a) Encoding algorithm.

```
$\mathsf{Evaluate}_{\mathsf{Mode}}(\vec{h}, \vec{pp}) \rightarrow \theta$

• Input. $\vec{h} = [h_0, \dots, h_{t-1}]$: a vector of encoded padded messages, and $\vec{pp} = [\lambda, p, \gamma, n, t]$: public parameters.
• Output. $\theta$: an encoded evaluated value, i.e., encoded padded mean.
1. Finds the most frequent (encoded padded) message among the $t$ encoded padded messages by calling:
       $\mathsf{Mode}(h_0, \dots, h_{t-1}) \rightarrow h_t$
2. Sets $\theta$ to $h_t$ and returns $\theta$.
```

(b) Evaluation algorithm.

```
$\mathsf{Decode}_{\mathsf{Mode}}(\theta, \vec{pp}, \overrightarrow{sk}) \rightarrow y$

• Input. $\theta$: an encoded evaluated value, $\vec{pp} = [\lambda, p, \gamma, n, t]$, and $\overrightarrow{sk} = [r_0, \dots, r_{t-1}]$.
• Output. $y$: a decoded evaluated value.
1. Retrieves the plaintext. $\forall i, 0 \le i \le t-1 :$
   (a) Decrypts the value using each key: $d'_j = \theta - r_j \bmod p$.
   (b) Checks the plaintext's correctness: $\mathsf{Check}_{\mathsf{Pad}}(d'_j, \gamma) \rightarrow res$.
       – If $res = True$:
           i. removes the pad by calling: $\mathsf{Unpad}(d'_j, \gamma, p) \rightarrow d_j$.
           ii. sets $y \leftarrow d_j$.
           iii. proceeds to step 2.
       – Otherwise ($res = False$), halts.
2. Returns $y$.
```

(c) Decoding algorithm.

Fig. 4: Encoding, Evaluation, and Decoding algorithms for secure evaluation of Mode.

Figures 4a, 4b, and 4c provide descriptions of $\mathsf{Encode}_{\mathsf{Mode}}$, $\mathsf{Evaluate}_{\mathsf{Mode}}$, and $\mathsf{Decode}_{\mathsf{Mode}}$, respectively. The novelty of Γ_{FOT_3}'s subroutine design for Mode lies in its ability to achieve both efficiency and unconditional security by effectively integrating Γ_{FOT_3} with one-time pads, the padding technique, and the deterministic masking method for identical values. Informally, the only information that the output of $\mathsf{Encode}_{\mathsf{Mode}}$ reveals is the number of times that some (encrypted) elements are repeated. Below, we formally define it.

Definition 9 (Function Ψ). *Let $\vec{s} = [s_0, \ldots, s_{n-1}]$ be a vector of n (encrypted) elements. Let* $\mathsf{Frequency}$ *be the frequency function formalized in Definition 2. The leakage function Ψ is defined as:*

$$\Psi(\vec{s}) \rightarrow \Big\{ \big(s_i, \mathsf{Frequency}(\vec{s}, s_i), \mathsf{Position}(\vec{s}, s_i)\big) \Big\}_{\forall s_i \in \mathsf{Unique}(\vec{s})}$$

where $\mathsf{Position}(\vec{s}, s_i)$ *returns all indices in $\vec{s}$ where s_i occurs and* $\mathsf{Unique}(\vec{s})$ *returns a vector of all distinct elements in $\vec{s}$.*

With the leakage function defined, we now present the security theorem for $\mathsf{Encode}_{\mathsf{Mode}}$.

Theorem 4. *Let Ψ denote the leakage function formalized in Definition 9. Let* Encode *represent the functionality defined in Section 2. Then,* $\mathsf{Encode}_{\mathsf{Mode}}$ *(presented in Figure 4a) securely realizes* Encode*, with respect to Definition 8.*

5.8 Proof of Theorem 4

Proof. We prove Theorem 4 by showing that the adversary's view in the real model and the ideal model are indistinguishable.

- Hybrid 0: Real model. This hybrid corresponds to the real model. The adversary's view during the execution of $\mathsf{Encode}_{\mathsf{Mode}}$ includes:

$$\mathsf{View}_{\mathcal{A}}^{\mathsf{Encode}_{\mathsf{Mode}}} = \{r_{\mathcal{A}}, \vec{h}\}$$

  where $r_{\mathcal{A}}$ is the random output of the adversary's internal coin tosses, and $\vec{h}$ is the vector of encoded messages, where each element of $\vec{h}$ is masked using fresh one-time pads.
- Hybrid 1: Replace randomness. In this hybrid, we replace $r_{\mathcal{A}}$ with $r'_{\mathcal{A}}$, which is sampled uniformly at random as in the ideal model. Since $r_{\mathcal{A}}$ and $r'_{\mathcal{A}}$ are both sampled independently and uniformly from the same domain, their distributions are identical:

$$\mathsf{View}_{\mathcal{A}}^{\mathsf{Encode}_{\mathsf{Mode}}} \equiv \mathsf{View}_{\mathcal{A}}^{\text{Hybrid 1}}$$

- Hybrid 2: Replace encoded messages with simulated messages. In this hybrid, we replace the vector of encoded messages $\vec{h}$ with a simulated vector $\vec{z}$. Each $z_i \in \vec{z}$ is generated based on the leakage function Ψ, i.e., $\Psi(\vec{h}) \rightarrow \big\{\big(h_i, \mathsf{Frequency}(\vec{h}, h_i), \mathsf{Position}(\vec{h}, h_i)\big)\big\}_{\forall h_i \in \mathsf{Unique}(\vec{h})}$. Specifically, we initiate an empty vector $\vec{z}$ of size n. Then, we take the following steps, $\forall i, 0 \leq i \leq n-1$:
  - if $\vec{z}[i]$ is empty:
    1. select a fresh random value $z_i \xleftarrow{\$} \{0,1\}^{\lambda}$.
    2. assign z_i to all positions in $\vec{z}$ determined by $\mathsf{Position}(\vec{h}, h_i)$, ensuring that $\mathsf{Frequency}(\vec{z}, z_i) = \mathsf{Frequency}(\vec{h}, h_i)$.
  - if $\vec{z}[i]$ is not empty, then take no action.

  By the perfect secrecy of the one-time pad, each element of $\vec{h}$ is indistinguishable from a random value. Thus, replacing $\vec{h}$ with $\vec{z}$ does not change the adversary's view:

$$\mathsf{View}_{\mathcal{A}}^{\text{Hybrid 1}} \equiv \mathsf{View}_{\mathcal{A}}^{\text{Hybrid 2}}$$

- Hybrid 3: Ideal model. In the final hybrid, we arrive at the ideal model. Here, the simulator constructs the adversary's view as:

  $$\mathsf{View}^{\mathsf{Encode}_{\mathsf{Mode}}}_{\mathsf{Sim}_{\mathcal{A}}} = \{r'_{\mathcal{A}}, \vec{z}\}$$

  where $r'_{\mathcal{A}}$ is a uniformly random coin toss and $\vec{z}$ is a simulated vector generated using Ψ. This matches the ideal model by definition:

  $$\mathsf{View}^{\text{Hybrid 3}}_{\mathcal{A}} = \mathsf{View}^{\mathsf{Encode}_{\mathsf{Mode}}}_{\mathsf{Sim}_{\mathcal{A}}}$$

Conclusion. By the transitivity of indistinguishability, we have:

$$\mathsf{View}^{\mathsf{Encode}_{\mathsf{Mode}}}_{\mathcal{A}} \equiv \mathsf{View}^{\mathsf{Encode}_{\mathsf{Mode}}}_{\mathsf{Sim}_{\mathcal{A}}}$$

Thus, $\mathsf{Encode}_{\mathsf{Mode}}$ securely realizes Encode, completing the proof. □

5.9 Subroutines for Additions and Multiplications

Simplified versions of the subroutines for Mean (in Figure 3) support (i) the addition operation, defined as $\mathsf{Add}(a_0, \ldots, a_{t-1}) \rightarrow \sum_{i=0}^{t-1} a_i$ or (ii) the multiplication operation, defined as $\mathsf{Multiply}(a_0, \ldots, a_{t-1}) \rightarrow \prod_{i=0}^{t-1} a_i$. Briefly, the simplification involves excluding t^{-1} from these subroutines. Specifically, for Add the following amendments are needed: (a) in step 1 of Figure 3b, θ is computed as: $\theta = \sum_{i=0}^{t-1} h_i \bmod p$, and (b) in step 1 of Figure 3c, y is computed as: $y = \theta - \sum_{i=0}^{t-1} r_i \bmod p$. For $\mathsf{Multiply}$ the following minor adjustments are needed: (a) in step 1 of Figure 3a, each h_i is computed as $h_i = m_i \cdot r_i \bmod p$, (b) in step 1 of Figure 3b, θ is computed as $\theta = \prod_{i=0}^{t-1} h_i \bmod p$, and (c) in step 1 of Figure 3c, y is computed as: $y = \theta \cdot \prod_{i=0}^{t-1} r_i^{-1} \bmod p$.

5.10 Extension: Converged OT

In this section, we discuss an extension of the protocol Γ_{FOT_3} presented in Section 5.3 to scenarios involving multiple senders $\mathcal{S}_1, \ldots, \mathcal{S}_k$, where each sender holds a vector of n messages. In this setting, the receiver $\mathcal{R}$ has a vector of t indices for each sender and ultimately it obtains a function of the messages corresponding to its selection. We refer to a protocol with these features as *Converged OT* to highlight its unification of key aspects, including oblivious transfer, multi-sender support, and the computation of functions over selected messages. Thus, the functionality G^{F} in Definition 5 will change, now it takes as input a vector of messages from each sender, and k vectors of indices $(\vec{e}_1, \ldots, \vec{e}_k)$ from $\mathcal{R}$. More specifically, now the functionality is defined as:

$$\mathsf{G}^{\mathsf{F}} : \Big(\vec{m}_1, \ldots, \vec{m}_k, (\vec{e}_1, \ldots, \vec{e}_k)\Big) \rightarrow y$$

where each $\vec{m}_1$ is defined as $\vec{m}_l = [m_{l,0}, \ldots, m_{l,n-1}]$, and $1 \leq l \leq k$. At a high level, Γ_{FOT_3} should be adjusted as follows. The receiver for each sender independently generates a similar pair of queries to the one it generates in a single sender setting and sends the related query to each sender and $\mathcal{H}$. Each sender independently generates their response and sends it to $\mathcal{H}$ which collects the responses, runs the evaluation function of them, and sends the result to $\mathcal{R}$. Given $\mathcal{H}$'s response, $\mathcal{R}$ decodes it to obtain the evaluated value. Appendix C presents the Converged OT in detail. In the Converged OT, the senders do not learn anything about the sender's indices, even if all senders collude with each other. On the other hand, the receiver will not learn anything beyond the evaluated value, i.e., the output of the function evaluated on the selected messages. The subroutines presented in Sections 5.5 and 5.7 remain applicable to the Converged OT without requiring fundamental modifications, provided the public parameters in $\vec{pp}$ are adjusted accordingly. For instance, in Figure 3, t is now defined as $|\vec{e}_1| + \ldots + |\vec{e}_k|$.

6 Evaluation

6.1 Asymptotic Cost Analysis

In this section, we analyze the complexity of the proposed schemes. Table 1 summarizes the result.

Table 1: Asymptotic costs of the proposed three protocols.

Protocol	Party	Computation	Communication	Storage	$\mathcal{R}$'s Download
$\mathit{\Gamma}_{\text{FOT}_2}$	$\mathcal{R}$	$O(t \cdot n)$	$O(t \cdot n)$	$O(n)$	$O(1)$
	$\mathcal{S}$	$O(t \cdot n) + Comp(\mathsf{F}, t)$	$O(t \cdot n)$	$O(n)$	–
$\mathit{\Gamma}_{\text{FOT}_3}$– Mean	$\mathcal{R}$	$O(n)$	$O(n)$	$O(n)$	$O(1)$
	$\mathcal{S}$	$O(n)$	$O(n)$	$O(n)$	–
	$\mathcal{H}$	$O(t)$	$O(1)$	$O(n)$	–
$\mathit{\Gamma}_{\text{FOT}_3}$– Mode	$\mathcal{R}$	$O(n)$	$O(n)$	$O(n)$	$O(1)$
	$\mathcal{S}$	$O(n)$	$O(n)$	$O(n)$	–
	$\mathcal{H}$	$O(t)$	$O(1)$	$O(n)$	–

$\mathit{\Gamma}_{\text{FOT}_2}$. The parties' computation complexity is as follows. $\mathcal{R}$ invokes t instances of FHE in step 2a, performs $n-1$ modular addition in step 2(b)ii, invokes $t \cdot n$ instances of OLE in step 2d, and invokes a single instance of FHE in Phase 4. Thus, its total computation complexity is $O(t \cdot n)$. $\mathcal{S}$ invokes $t \cdot n$ instances of OLE in step 2d, performs n modular addition in step 3a, invokes t instances of FHE in step 3b, and obliviously evaluates the function F on the t ciphertexts using FHE with the complexity of $Comp(\mathsf{F}, t)$, in step 3c. Therefore, the total computation complexity of $\mathcal{S}$ is $O(t \cdot n) + Comp(\mathsf{F}, t)$. The parties' communication complexity is as follows. $\mathcal{R}$ publishes a single public key in step 1b, invokes $t \cdot n$ instances of OLE in step 2d with the communication complexity of $O(t \cdot n)$, and also transmits t ciphertexts at the end of Phase 2. Thus, $\mathcal{R}$'s total communication complexity is $O(t \cdot n)$. On the other hand, $\mathcal{S}$ invokes $t \cdot n$ instances of OLE in step 2d with the communication complexity of $O(t \cdot n)$, and sends the encrypted evaluated value to $\mathcal{R}$ in step 3d. Hence, $\mathcal{S}$'s total communication cost is also $O(t \cdot n)$. Next, we estimate the parties' storage costs. The storage space required by $\mathcal{R}$ is $O(n)$, because it needs to generate n random messages in step 2b and use them as input to OLE in step 2d. The storage space required by $\mathcal{S}$ is also $O(n)$ as it maintains a vector of n messages. The download complexity of $\mathcal{R}$ is $O(1)$ because it only downloads an encrypted evaluated value in step 3d.

$\mathit{\Gamma}_{\text{FOT}_3}$–Mean. We proceed to analyze the parties' overheads in the case where they participate in $\mathit{\Gamma}_{\text{FOT}_3}$ to compute Mean. We initially focus of the parties' computation complexity. $\mathcal{R}$ randomly permutes a vector of n elements with a complexity of $O(n)$ in step 2(a)ii, finds t elements in a vector of n elements with a complexity of $O(n)$ in step 2b, and performs $t+1$ modular addition and one multiplication to decode the result in step 5. Therefore, $\mathcal{R}$'s total computation complexity is $O(n)$. $\mathcal{S}$ performs n modular addition to encode the massages in step 3a, and permutes a vector of n elements with the complexity of $O(n)$ in step 3b. Hence, the total computation complexity of $\mathcal{S}$ is $O(n)$. Moreover, $\mathcal{H}$ evaluates the function on t ciphertexts that involve t modular addition and a single modular multiplication in step 4b; hence, its complexity is $O(t)$. Now, we turn our attention to the parties' communication costs. $\mathcal{R}$ sends a vector of n elements to $\mathcal{S}$ in steps 1b and 2c and a vector of t messages to $\mathcal{H}$ in step 2c; thus, its total communication complexity is $O(n)$. Also, $\mathcal{S}$'s complexity is $O(n)$ as it only sends a vector of n elements to $\mathcal{H}$ in step 3c. The communication complexity of $\mathcal{H}$ is $O(1)$ because it only sends a single value to $\mathcal{R}$ in step 4c.

Next, we estimate the parties' storage costs. The storage space needed by $\mathcal{R}$ is $O(n)$, as it needs to randomly permute a vector of n messages in step 2a; however, the size of each element of the vector is very small. The storage space required by $\mathcal{S}$ is also $O(n)$, as it needs to store a vector containing n messages. The storage cost of $\mathcal{H}$ is $O(n)$ because it receives n messages from $\mathcal{S}$ in step 3c. The download complexity of $\mathcal{R}$ is $O(1)$ as it only downloads an encoded evaluated value in step 4c. The size of the message that $\mathcal{R}$ downloads corresponds to the security parameter λ, e.g., 128-bit.

$\mathit{\Gamma}_{\text{FOT}_3}$–Mode. Next, we evaluate the parties' costs in the scenarios where they engage in $\mathit{\Gamma}_{\text{FOT}_3}$ to compute Mode. $\mathcal{R}$'s complexity is $O(n)$, because it takes the same steps as it takes in $\mathit{\Gamma}_{\text{FOT}_3}$ for computing Mean. The computation complexity of $\mathcal{S}$ is $O(n)$ because it pads n elements and performs at most n modular addition to encode n messages in step 3a and permutes a vector of n elements with the complexity of $O(n)$ in step 3b. Also, $\mathcal{H}$'s complexity is $O(t)$ because it searches through the t encrypted elements and finds the one with the highest frequency in step 4b. The parties' communication and computation complexities remain the same as their related complexities when they participate in $\mathit{\Gamma}_{\text{FOT}_3}$ for computing Mean. The download complexity of $\mathcal{R}$ is also $O(1)$ as it only downloads an encoded evaluated value in step 4c, where the size of the downloaded message is about the same as the security parameter λ, e.g., 256-bit. The size of a message downloaded in this setting is larger than in the setting used for computing Mode via $\mathit{\Gamma}_{\text{FOT}_3}$. This increase is due to the use of padding, which expands the size of each message and, consequently, the size of the finite field.

6.2 Concrete Cost Analysis

We implemented $\mathit{\Gamma}_{\text{FOT}_3}$–Mean and $\mathit{\Gamma}_{\text{FOT}_3}$–Mode in C++ and evaluated their concrete runtime. The source code for the implementation is publicly available [2, 3]. We used a MacBook Pro with an Apple M3 Pro CPU and 36 GB of RAM for the experiment. No parallelization was applied. The experiment was repeated an average of 10 times. In $\mathit{\Gamma}_{\text{FOT}_3}$–Mean, we set the message size σ and the security parameter λ to 128. In $\mathit{\Gamma}_{\text{FOT}_3}$–Mode, we set σ and pad size γ to 128, and λ to 257.

Table 2: Runtimes (in ms) of $\mathit{\Gamma}_{\text{FOT}_3}$-Mean and $\mathit{\Gamma}_{\text{FOT}_3}$-Mode across different values of n and t.

Protocol	t	Number of messages: n					
		2^4	2^8	2^{12}	2^{16}	2^{20}	2^{24}
$\mathit{\Gamma}_{\text{FOT}_3}$-Mean	2	0.33	0.58	4.35	65.46	1301.15	30966
	2^4	–	0.58	4.37	65.77	1389.56	31148
	2^8	–	–	4.43	65.94	1398.13	31144.1
	2^{12}	–	–	–	66.14	1408.9	31192.9
	2^{16}	–	–	–	–	1463.95	31302.6
	2^{20}		–	–	–	–	32096.6
$\mathit{\Gamma}_{\text{FOT}_3}$-Mode	2	0.35	0.93	9.21	168.27	3125.88	65754.7
	2^4	–	0.93	9.87	167.33	3271.86	66514.9
	2^8	–	–	10.24	168.15	3281.13	66877.3
	2^{12}	–	–	–	168.56	3211.47	66930.9
	2^{16}	–	–	–	–	3265.62	67802.4
	2^{20}		–	–	–	–	68425.8

We analyzed the schemes' runtime for various values of t (from 2 to about 16 million) and n (from 2 to 268 million). Table 2 shows the results. As the table indicates, the schemes scale well even for large values of n and t. For instance, when $n = 2^{24}$ and $t = 2^{20}$, it takes about (i) 32 seconds for $\mathit{\Gamma}_{\text{FOT}_3}$-Mean and (ii) 68 seconds for $\mathit{\Gamma}_{\text{FOT}_3}$–Mode to complete. On average, $\mathit{\Gamma}_{\text{FOT}_3}$–Mode is 2.2 times slower than $\mathit{\Gamma}_{\text{FOT}_3}$–Mean.

Comparing Scalable OTs. We also compared the runtime of $\mathit{\Gamma}_{\text{FOT}_3}$–Mean and $\mathit{\Gamma}_{\text{FOT}_3}$–Mode with the runtime of the scalable t-out-of-n OTs [5,32], for different numbers of invocations (from 125,000 to 1,250,000) and values of t (from 2 to 12) when $n = 16$. Figure 5 summarizes the result for 1,250,000 OT invocations. For the runtime of [32], we derived the figures from [43]. The message size in the scheme proposed by the reference [32] is only 4 bits. The OT proposed in [5,32] has the highest runtime growth as t increases. In contrast, the OT from [5] remains the most efficient, although neither supports computation on selected messages. $\mathit{\Gamma}_{\text{FOT}_3}$–Mean has a runtime closer to the OT in [5], whereas $\mathit{\Gamma}_{\text{FOT}_3}$–Mode has a higher runtime than $\mathit{\Gamma}_{\text{FOT}_3}$–Mean. Table 3 provides further details.

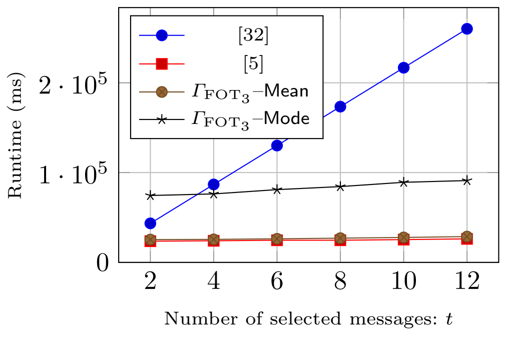

Fig. 5: Performance comparison for 1.25×10^6 invocations.

7 Applications

In this section, we examine four key applications where functional OT provides enhancements: (1) privacy-preserving K-Nearest Neighbors (KNN), (2) privacy-preserving client selection in federated learning, (3) privacy-preserving recommender systems, and (4) secure multi-party dataset aggregation.

7.1 Privacy-Preserving K-Nearest Neighbors

Context. The K-Nearest Neighbors (K-NN) algorithm is a simple yet powerful technique widely used in supervised machine learning for both classification and regression tasks [20]. At a high level, K-NN works by comparing a given data point to its closest points (neighbors) in a dataset and using their properties to make a prediction. Unlike parametric models, K-NN does not assume a specific functional form for the underlying data distribution. Instead, it relies on a distance-based approach to find similar instances in the dataset.

The dataset $\mathcal{D}$ is often defined as a set of labeled data points: $\mathcal{D} = \{(x_i, y_i) \mid x_i \in \mathbb{R}^{n'}, y_i \in \mathcal{Y}, i = 0, \ldots, n-1\}$, where n is the total number of data points, $x_i \in \mathbb{R}^{n'}$ represents the i-th data point as a feature vector of dimension n', $y_i \in \mathcal{Y}$ is the corresponding label for x_i. For classification, $\mathcal{Y} = \{0, \ldots, C\}$, where C is the number of distinct classes, and for regression, $\mathcal{Y} \subseteq \mathbb{R}$ represents continuous values. In K-NN, a query x_q is an unlabeled data point for which the algorithm predicts a label. At an abstract level, K-NN involves four phases:

1. Setup: select a positive integer K, according to some rules.
2. Calculate the distance: for a given input data point, K-NN calculates the distance to all the points in the training set.

Table 3: Comparing the runtime (in ms) of $\mathit{\Gamma}_{\text{FOT}_3}$–Mean and $\mathit{\Gamma}_{\text{FOT}_3}$–Mode with the scalable OTs proposed in [5, 32], for different values of t and various numbers of OT invocations, when $n = 16$. For the OT in [32] the message size is only 4 bits.

Protocol	t	Number of invocations			
		1.25×10^5	2.5×10^5	5×10^5	1.25×10^6
[32]	2	4320	8460	17000	43360
	4	8640	16920	34000	86720
	6	12960	25380	51000	130080
	8	17280	33840	68000	173440
	10	21600	42300	85000	216800
	12	25920	50760	102000	260160
[5]	2	2228.56	4529.27	9290.04	23495.6
	4	2346.3	4713.24	9554.61	23963.7
	6	2400.08	4897.03	9823.35	24541.6
	8	2408.6	4985.46	9934.83	24569
	10	2448.86	4993.86	9999.01	25249.2
	12	2478.03	5136.36	10323.4	26076.2
$\mathit{\Gamma}_{\text{FOT}_3}$–Mean	2	2387.55	4903.5	9913.72	25307.6
	4	2487.36	5047.76	9956.19	25502.4
	6	2490.75	5232.62	10499	26159.8
	8	2574.68	5319.06	10905.4	26867.4
	10	2689.82	5599.08	11059.2	27671.1
	12	2793.81	5626.98	11403.2	28640.9
$\mathit{\Gamma}_{\text{FOT}_3}$–Mode	2	7262.59	14408.4	28973.9	74395.6
	4	7495.64	14991.7	29810.7	76181.5
	6	7887.44	15405.6	32639.9	80955.1
	8	8375.55	16059.4	34414.1	84351
	10	8493.99	17167.5	34796	89130.2
	12	8869.04	17914.3	34818.2	91053.1

3. Select the K nearest neighbors, closest to the data point.
4. Make a prediction: (i) for classification, it assigns the class label that appears most frequently among the K neighbors, and (ii) for regression: it computes the average of the labels related to the K neighbors.

State-of-the-Art. There have been works on Privacy-Preserving K-NN Search (PPK-NNS) [10, 13, 50, 63]. A PPK-NNS is a protocol that enables the identification of the K data points closest to a given query within a dataset, while ensuring the privacy of both the data and the query. It ensures that the dataset holder (a.k.a the sender) will not learn anything about the query while the query sender (a.k.a. the receiver) will not learn anything beyond the K associate data, e.g., labels. PPK-NNS has various applications in scenarios where sensitive data is involved, including targeted advertising, biometric data, DNA data analysis, and face recognition; see [13, 50] for a comprehensive list of applications. Some of the PPK-NNSs are highly scalable and efficient [13, 50]. Nevertheless, they are somehow limited in functionality. For instance, in the context of machine learning, in particular, the K-NN algorithm, PPK-NNSs do not provide the final prediction produced by a conventional K-NN algorithm. In these schemes, the receiver must take an additional step, to perform a local computation on these K labels to derive the final prediction. In these PPK-NNSs, the set of K labels is revealed to the receiver, rather than the ultimate output of K-NN, the predicted value.

Our Solution: Privacy-Preserving K-NN Based on PPK-NNS and Functional OT. Companies like Apple are offering privacy-preserving AI where clients' queries can remain oblivious to the server, at inference time [6]. Our functional OT protocols can help achieve this goal. Specifically, our protocols can complement these PPK-NNSs by further minimizing this information disclosed to the receiver. Consider a scenario where each data point in the sender's dataset is assigned a public index (in addition to its original label). The dataset is sorted according to these indices, ranging from 0 to $n-1$. At the end of PPK-NNS [10,13,63], the receiver obtains the indices of the top K nearest data points. To compute the final prediction, the receiver and the sender invoke our functional OT protocol. The receiver's input consists of the K indices, while the sender's input comprises n labels. The functional OT protocol returns a function of the labels corresponding to these indices to the receiver. As a result, the receiver will not learn the K labels (except for the K public indices). Simultaneously, the sender learns nothing about the receiver's query and the output of the function. Figure 6 depicts this process that combines PPK-NNS and the functional OT.

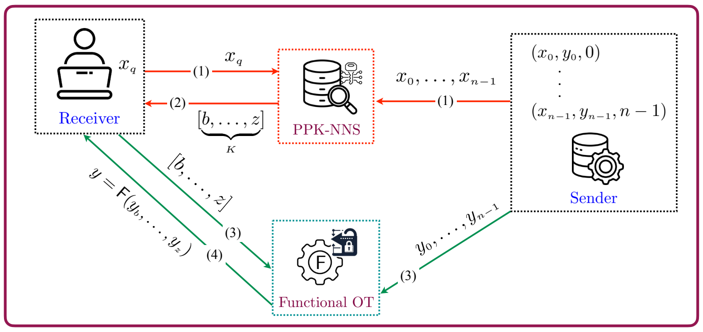

Fig. 6: The Parties' interaction in the proposed privacy-preserving K-NN protocol, minimizing information leakage to the receiver via PPK-NNS and functional OT. In this figure, $b,\ldots,z \in [0,n-1]$.

7.2 Privacy-Preserving Client Selection in Federated Learning

Context. Client selection is a critical component in enhancing the performance of Federated Learning (FL) [35] by determining the optimal subset of clients participating in each training round [21,34,37,45,55]. The heterogeneity among clients—including variations in local data distributions, computational resources, and network latency—introduces challenges to effective selection. Robust client selection mechanisms are vital for ensuring model accuracy, accelerating convergence, and maintaining overall system efficiency. Client selection can be guided by various criteria, as highlighted in [21,34,37,45,55]. These criteria include:

- *Computation capability*: The processing power available on the client device.
- *Communication capability*: The bandwidth and network stability required for effective participation. This also includes the overall availability of a device at a specific time within a defined time window.
- *Battery level*: The availability of sufficient battery power to complete the required tasks.
- *Device workload*: The presence of resource-intensive applications running in the background.
- *Heterogeneous data distributions*: Variability in the data characteristics across clients.
- *Data utility*: The value of a client's data, assessed by factors such as the number of data samples or the cumulative loss associated with their data.
- *Data relevance*: A relevance score that measures how well a client's data aligns with the objectives of the task.
- *Device location*: The geographical location of the client device, which may impact its relevance to the task, fairness, availability, or compliance with specific requirements.

State-of-the-Art. A few studies have explored enabling servers to select clients in a privacy-preserving manner [33, 53, 62], focusing on a limited set of characteristics, such as the size of the clients' label sets and the relevance of their data samples. Among these approaches, [33, 62] require each client to disclose certain characteristics of their dataset (e.g., relevant samples) in plaintext to the server. These methods also utilize a private set intersection (PSI) scheme to enable the server to identify matching metadata (e.g., labels) and accept a client if the cardinality of the intersection exceeds a predefined threshold. However, PSI inherently reveals individual elements common to both parties' sets, resulting in information leakage that functional OT seeks to prevent. Moreover, these schemes inadvertently expose non-trivial information about each client's dataset, such as its statistical homogeneity.

A mechanism proposed in [53] facilitates client selection in FL by involving a third party to assist the server and clients in identifying irrelevant data (or clients). This scheme assumes the third party possesses a small set of benchmark data. Initially, the third party trains a benchmark model using only the benchmark data. The trained model is then sent to each client who collaborates with the server to compute a set of parameters enabling the client to identify their relevant data. This approach is efficient and avoids the use of cryptographic tools. However, it relies on several assumptions: only the server is untrusted while the clients are honest, benchmark data is readily available, and the scheme focuses solely on a single aspect—data relevance.

Our Solution: Privacy-Preserving Client Selection Based on Functional OT. The functional OT protocols we proposed can enhance privacy in the client selection process in the FL setting. We will elaborate on that with a concrete example. For the sake of simplicity, let us assume the existence of a public checklist, as shown in Table 4. Each client computes and populates the output in the last column of each row. Depending on the criterion, this output may be computed independently by the client or collaboratively with the server in a secure manner, ensuring that only the client learns the result. From the client's perspective, these outputs are considered private.

Index	Criterion	Specific Check	Output (1/0)
0	Computation Capability–CPU	The device has at least 4 CPU cores	$o_0 \in \{1, 0\}$
1	Computation Capability–RAM	The available RAM is greater than 4 GB	$o_1 \in \{1, 0\}$
2	Communication Capability–Bandwidth	The network bandwidth is greater than 5 Mbps	$o_2 \in \{1, 0\}$
3	Communication Capability–Availability	The device will be online from 1–5 AM on 5th June 2025	$o_3 \in \{1, 0\}$
4	Battery Level	The device will have a battery level above 50%	$o_4 \in \{1, 0\}$
5	Device Workload	The CPU usage will be below 50% (excluding FL workload)	$o_5 \in \{1, 0\}$
6	Statistical Homogeneity	The device's data statistical homogeneity exceeds threshold t	$o_6 \in \{1, 0\}$
7	Data Utility	The device has more than 1,000 data samples available	$o_7 \in \{1, 0\}$
8	Data Relevance	The data relevance score is above threshold t'	$o_8 \in \{1, 0\}$
9	Device Location	The device will be in the USA	$o_9 \in \{1, 0\}$

Table 4: An example of a public checklist for selecting a client in federated learning.

In this case, in the context of functional OT, the server is the receiver with a vector of indices $\vec{e} = [b, \dots, z]$ (where each element of $\vec{e}$ is in the range $[0, n-1]$) and the client is the sender with binary messages $[o_0, \dots, o_{n-1}]$, where each o_j is the output of the check for the j-th criterion, e.g., $o_0 = 0$, if "the device has at least 4 CPU cores". Let the functionality G^{F} (w.r.t. Definition 5) that the functional OT computes be defined as:

$$\mathsf{G}^{\mathsf{F}} : \Big(\vec{m} = [o_0, \dots, o_{n-1}], \vec{e} = [b, \dots, z]\Big) \rightarrow y = \sum_{\forall e_j \in \vec{e}} o_{e_j}$$

The server wants to check if the client meets a set of requirements considered vital. Accordingly, the server sets the elements of $\vec{e}$ to the indices of the criteria that it should be met by the client and accepts the client if functional OT's output y is at least a predefined threshold w, where $w \leq |\vec{e}|$. Figure 7 depicts this process. In this setting, due to the security of functional OT, the server only learns whether the client meets certain criteria without being able to learn if the client meets each individual criterion. On the other hand, the client cannot figure out which criteria the server is interested in.

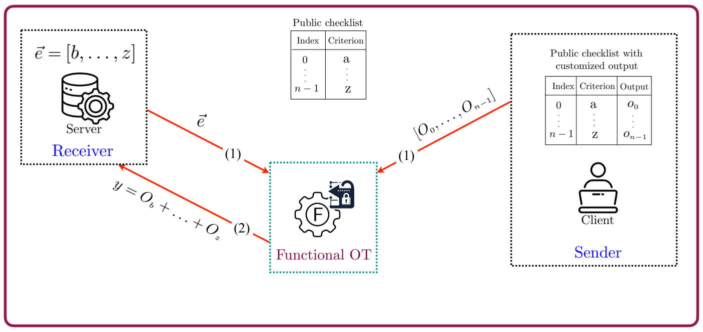

Fig. 7: Depicting privacy-preserving client selection process, using functional OT, a public checklist, and the client's checklist with related output. Here, $b, \ldots, z \in [0, n-1]$.

7.3 Privacy-Preserving Recommender Systems

Context. Recommender systems are widely implemented in various digital platforms to provide personalized suggestions based on user interactions and historical data [47]. Despite their effectiveness, these systems often require access to detailed user information, raising significant privacy concerns, particularly in contexts where sensitive user behavior and preferences are involved [51].

State-of-the-Art. Current privacy-preserving recommender systems rely on techniques such as differential privacy or homomorphic encryption. While effective, these methods either degrade recommendation accuracy or involve significant computational overhead [36].

Our Solution: Privacy-Preserving Recommendations Using Functional OT. Our efficient functional OTs (presented in Section 5) can improve privacy-preserving recommendations by allowing:

- **User-side privacy:** Users provide indices corresponding to their preferred categories without exposing their exact choices.
- **Server-side privacy:** The server computes personalized recommendations without revealing the entire dataset to the user.

Concretely, a functional OT instance is executed where the user inputs a set of indices representing their interests. The server holds a database of potential recommendations. The functional OT protocol returns only a function of the selected recommendations (e.g., top-rated items for the user's interests) without leaking additional dataset details. This approach ensures that the server remains unaware of the user's precise preferences. The user gains access only to a personalized subset of recommendations, preserving dataset privacy. By leveraging functional OT, recommender systems can achieve strong privacy guarantees while maintaining high recommendation accuracy, making them viable for real-world deployment.

7.4 Secure Multi-Party Dataset Aggregation

Context. In many multi-party computation scenarios, each participant may possess multiple datasets and need to compute an aggregate function over a subset of these datasets without revealing their individual inputs or which subset was used. Such scenarios frequently arise in applications like collaborative healthcare research, joint financial analysis, shared IoT data analytics, and FL, where clients often hold multiple datasets. In these settings, secure aggregation is critical for preserving privacy and confidentiality without exposing sensitive data or selection patterns.

State-of-the-Art. Secure multi-party computation (MPC) and homomorphic encryption (HE) are the prevailing methods for secure data aggregation [1,8,11]. While these methods ensure security and verifiability, they suffer from significant computational and communication overheads due to the reliance on public key-based operations. Additionally, existing solutions typically assume that each client has only a single dataset, thereby failing to address the practical scenario where clients possess multiple datasets, and only a subset of these datasets may be used in each aggregation round.

Our Solution: Secure Data Aggregation Using Functional OT. Our FOT protocols, proposed in Section 5, offer efficient, symmetric-key-based solutions for secure data aggregation. These protocols allow parties to compute aggregate statistics (e.g., sum, mean, mode, or product) on a selected subset of datasets without disclosing individual data points or which datasets were selected. Unlike traditional MPC and HE methods, these FOT protocols achieve this with significantly lower computational overhead due to their reliance on symmetric-key primitives. Moreover, they retain the core OT properties: the sender remains oblivious to the receiver's selection, and the receiver only learns the computed aggregate without individual dataset details. These protocols can be used for scenarios where clients have multiple datasets, ensuring that the clients cannot determine which subset was used for aggregation. In federated learning settings involving *multiple rounds of secure aggregation*, they can be combined with other techniques to prevent clients from inferring the subset selection from model updates shared by the server.

8 Related Work

The notion of 1-out-of-n OT was introduced by Rabin [44]. To generalize the notion of 1-out-of-2 OT, t-out-of-n OTs were proposed. They are suitable for scenarios where $n > 2$ and $t \geq 1$. Naor and Pinkas [38] proposed two variants of OT. They use a pseudorandom function and any standard 1-out-of-2 OT. Tzeng [54] proposed a 1-out-of-n OT, based on the Decisional Diffie-Hellman (DDH) assumption and involves public key operations. Another t-out-of-n OT was proposed in [31], which relies on the Discrete logarithm problem (DLP), involves modular exponentiation linear with n. Wei *et al.* [56] proposed server-aided t-out-of-n OT, based on the DDH assumption, which involves modular exponentiation with computational complexity that scales linearly with t and n.

The efficient OT extensions [7, 28, 29, 40] have initially been designed for the 1-out-of-2 OT setting; however, they can be invoked multiple times to meet the requirements of t-out-of-n OTs. To date, the fastest 1-out-of-n semi-honest and malicious secure OTs are the OT extensions proposed in [32] and [43] respectively, with a caveat. They have been designed to work efficiently when the input secret messages are *very short*, $\log(n)$. There have been efforts to design unconditionally secure OTs. Some schemes use multiple senders that maintain an identical copy of the database [9, 39]. Other ones use a specific network structure, i.e., a noisy channel, to achieve unconditionally secure OT [15, 16, 30]. There is a scheme that achieves unconditionally secure OT using a fully trusted initializer [48]. There are also unconditionally secure OTs that aim to avoid using a noisy channel or trusted party [4, 5]. However, all the above schemes provide the receiver with each individual message that it selected.

9 Conclusion and Future Work

In this work, we introduced the concept of Functional Oblivious Transfer (FOT), a variant of Oblivious Transfer (OT) designed for scenarios where the receiver must compute a function on selected messages. FOT enhances the security of conventional OT by ensuring that the receiver learns only the function's output, while the sender remains oblivious to both the receiver's selection and the computed result. We presented several instantiations of FOT, including unconditionally secure protocols to compute Mean and Mode. We formally proved their security within the simulation-based paradigm. Our implementations and concrete cost analysis demonstrated that these schemes are efficient and scalable. We have shown that our schemes can enhance privacy-preserving machine learning, particularly K-NN algorithms, and secure client selection in federated learning to ensure data privacy while enabling efficient computation.

It would be interesting to explore how the concept of FOT can be extended to other privacy-preserving information retrieval schemes, such as *private information retrieval* [14] and *oblivious RAM* [25]. By incorporating FOT, these schemes can be enhanced to ensure that the receiver learns only a function of the queried messages while the sender remains oblivious to the receiver's query and the function's output.

Acknowledgments

We appreciate Lorenzo Martinico's general feedback.

References

1. Abadi, A.: Tempora-fusion: Time-lock puzzle with efficient verifiable homomorphic linear combination. IACR Cryptol. ePrint Arch. (2024)
2. Abadi, A.: Source code of functional oblivious transfer computing mean (2025), https://github.com/AydinAbadi/FOT/blob/main/main--mean.cpp
3. Abadi, A.: Source code of functional oblivious transfer computing mode (2025), https://github.com/AydinAbadi/FOT/blob/main/main--mode.cpp
4. Abadi, A., Desmedt, Y.: Supersonic OT: fast unconditionally secure oblivious transfer. IACR Cryptol. ePrint Arch. (2024)
5. Abadi, A., Desmedt, Y.: Scalable post-quantum oblivious transfers for resource-constrained receivers. Cryptology ePrint Archive (2025), https://eprint.iacr.org/2025/036
6. Apple Inc.: Private cloud compute: A new frontier for ai privacy in the cloud (2024), https://security.apple.com/blog/private-cloud-compute/
7. Asharov, G., Lindell, Y., Schneider, T., Zohner, M.: More efficient oblivious transfer and extensions for faster secure computation. In: CCS'13 (2013)
8. Bell, J., Gascón, A., Lepoint, T., Li, B., Meiklejohn, S., Raykova, M., Yun, C.: ACORN: input validation for secure aggregation. In: USENIX (2023)
9. Blundo, C., D'Arco, P., Santis, A.D., Stinson, D.R.: On unconditionally secure distributed oblivious transfer. J. Cryptol. (2007)
10. Boldyreva, A., Tang, T.: Privacy-preserving approximate k-nearest-neighbors search that hides access, query and volume patterns. Proc. Priv. Enhancing Technol. (2021)
11. Bonawitz, K.A., Ivanov, V., Kreuter, B., Marcedone, A., McMahan, H.B., Patel, S., Ramage, D., Segal, A., Seth, K.: Practical secure aggregation for privacy-preserving machine learning. In: CCS (2017)
12. Boneh, D., Sahai, A., Waters, B.: Functional encryption: Definitions and challenges. In: TCC (2011)
13. Chen, H., Chillotti, I., Dong, Y., Poburinnaya, O., Razenshteyn, I.P., Riazi, M.S.: SANNS: scaling up secure approximate k-nearest neighbors search. In: USENIX. USENIX Association (2020)
14. Chor, B., Kushilevitz, E., Goldreich, O., Sudan, M.: Private information retrieval. Journal of the ACM (JACM) (1998)
15. Crépeau, C., Kilian, J.: Achieving oblivious transfer using weakened security assumptions (extended abstract). In: FoCS (1988)
16. Crépeau, C., Morozov, K., Wolf, S.: Efficient unconditional oblivious transfer from almost any noisy channel. In: Security in Communication Networks, 4th International Conference, SCN (2004)

17. Desmedt, Y., Abadi, A.: Delegated-query oblivious transfer and its practical applications. CoRR (2024)
18. Devet, C., Goldberg, I., Heninger, N.: Optimally robust private information retrieval. In: USENIX Security (2012)
19. Even, S., Goldreich, O., Lempel, A.: A randomized protocol for signing contracts. Commun. ACM (1985)
20. Fix, E., Hodges, J.L.: Discriminatory analysis, nonparametric discrimination (1951)
21. Fu, L., Zhang, H., Gao, G., Zhang, M., Liu, X.: Client selection in federated learning: Principles, challenges, and opportunities. IEEE Internet Things J. (2023)
22. Gentry, C.: Fully homomorphic encryption using ideal lattices. In: STOC (2009)
23. Ghosh, S., Nilges, T.: An algebraic approach to maliciously secure private set intersection. In: EUROCRYPT (2019)
24. Goldreich, O.: The Foundations of Cryptography - Volume 2, Basic Applications. Cambridge University Press (2004)
25. Goldreich, O., Ostrovsky, R.: Software protection and simulation on oblivious rams. J. ACM (1996)
26. Gunupudi, V., Tate, S.R.: Generalized non-interactive oblivious transfer using count-limited objects with applications to secure mobile agents. In: FC (2008)
27. Harnik, D., Ishai, Y., Kushilevitz, E.: How many oblivious transfers are needed for secure multiparty computation? In: CRYPTO (2007)
28. Henecka, W., Schneider, T.: Faster secure two-party computation with less memory. In: CCS (2013)
29. Ishai, Y., Kilian, J., Nissim, K., Petrank, E.: Extending oblivious transfers efficiently. In: CRYPTO, (2003)
30. Ishai, Y., Kushilevitz, E., Ostrovsky, R., Prabhakaran, M., Sahai, A., Wullschleger, J.: Constant-rate oblivious transfer from noisy channels. In: CRYPTO (2011)
31. Jain, A., Hari, C.: A new efficient protocol for k-out-of-n oblivious transfer. Cryptologia (2010)
32. Kolesnikov, V., Kumaresan, R.: Improved OT extension for transferring short secrets. In: CRYPTO (2013)
33. Li, A., Zhang, L., Tan, J., Qin, Y., Wang, J., Li, X.: Sample-level data selection for federated learning. In: INFOCOM (2021)
34. Li, J., Chen, T., Teng, S.: A comprehensive survey on client selection strategies in federated learning. Comput. Networks (2024)
35. McMahan, H.B., Moore, E., Ramage, D., y Arcas, B.A.: Federated learning of deep networks using model averaging. CoRR (2016)
36. McSherry, F., Mironov, I.: Differentially private recommender systems: Building privacy into the netflix prize contenders. In: Proceedings of the 15th ACM SIGKDD international conference on Knowledge discovery and data mining. pp. 627–636 (2009)
37. Nagalapatti, L., Mittal, R.S., Narayanam, R.: Is your data relevant?: Dynamic selection of relevant data for federated learning. In: AAAI. AAAI Press (2022)
38. Naor, M., Pinkas, B.: Oblivious transfer and polynomial evaluation. In: STOC (1999)
39. Naor, M., Pinkas, B.: Distributed oblivious transfer. In: ASIACRYPT. Springer (2000)
40. Nielsen, J.B.: Extending oblivious transfers efficiently - how to get robustness almost for free. IACR Cryptol. ePrint Arch. (2007)
41. Nuida, K., Kurosawa, K.: (batch) fully homomorphic encryption over integers for non-binary message spaces. In: EUROCRYPT (2015)
42. Pass, R., Shi, E., Tramèr, F.: Formal abstractions for attested execution secure processors. In: EUROCRYPT (2017)
43. Patra, A., Sarkar, P., Suresh, A.: Fast actively secure OT extension for short secrets. In: NDSS (2017)
44. Rabin, M.O.: How to exchange secrets with oblivious transfer (1981)
45. Rahman, S.A., Tout, H., Mourad, A., Talhi, C.: Fedmccs: Multicriteria client selection model for optimal iot federated learning. IEEE Internet Things J. (2021)
46. Ren, Z., Yang, L., Chen, K.: Improving availability of vertical federated learning: Relaxing inference on non-overlapping data. ACM Trans. Intell. Syst. (2022)
47. Ricci, F., Rokach, L., Shapira, B.: Recommender systems: Techniques, applications, and challenges. Recommender systems handbook pp. 1–35 (2021)
48. Rivest, R.: Unconditionally secure commitment and oblivious transfer schemes using private channels and a trusted initializer. technical report (1999)
49. Scarani, V., Bechmann-Pasquinucci, H., Cerf, N.J., Dušek, M., Lütkenhaus, N., Peev, M.: The security of practical quantum key distribution. Reviews of modern physics (2009)
50. Servan-Schreiber, S., Langowski, S., Devadas, S.: Private approximate nearest neighbor search with sublinear communication. In: SP. IEEE (2022)
51. Shokri, R., Shmatikov, V.: Privacy-preserving deep learning. In: Proceedings of the 22nd ACM SIGSAC conference on computer and communications security. pp. 1310–1321 (2015)

52. Tramèr, F., Zhang, F., Lin, H., Hubaux, J., Juels, A., Shi, E.: Sealed-glass proofs: Using transparent enclaves to prove and sell knowledge. In: EuroS&P (2017)
53. Tuor, T., Wang, S., Ko, B.J., Liu, C., Leung, K.K.: Overcoming noisy and irrelevant data in federated learning. In: ICPR. IEEE (2020)
54. Tzeng, W.: Efficient 1-out-n oblivious transfer schemes. In: Naccache, D., Paillier, P. (eds.) PKC (2002)
55. Wahab, O.A., Mourad, A., Otrok, H., Taleb, T.: Federated machine learning: Survey, multi-level classification, desirable criteria and future directions in communication and networking systems. IEEE Commun. Surv. Tutorials (2021)
56. Wei, X., Zhao, C., Jiang, H., Xu, Q., Wang, H.: Practical server-aided k-out-of-n oblivious transfer protocol. In: GPC. Lecture Notes in Computer Science (2016)
57. Wiesner, S.: Conjugate coding. SIGACT News **15**(1), 78–88 (1983)
58. Xu, G., Li, H., Zhang, Y., Xu, S., Ning, J., Deng, R.H.: Privacy-preserving federated deep learning with irregular users. IEEE Trans. Dependable Secur. Comput. (2022)
59. Yang, Q., Liu, Y., Chen, T., Tong, Y.: Federated machine learning: Concept and applications. ACM Trans. Intell. Syst. Technol. (2019)
60. Yao, A.C.: Protocols for secure computations (extended abstract). In: 23rd Annual Symposium on Foundations of Computer Science (1982)
61. Zhang, F., Zhang, H.: Sok: A study of using hardware-assisted isolated execution environments for security. In: HASP (2016)
62. Zhang, L., Li, A., Peng, H., Han, F., Huang, F., Li, X.: Privacy-preserving data selection for horizontal and vertical federated learning. IEEE Trans. Parallel Distributed Syst. (2024)
63. Zhang, X., Wang, Q., Xu, C., Peng, Y., Xu, J.: Fedknn: Secure federated k-nearest neighbor search. Proc. ACM Manag. Data (2024)

A IND-CPA for Asymmetric Key Encryption Schemes

Let $\mathcal{E} = (\mathsf{KeyGeneration}, \mathsf{Encrypt}, \mathsf{Decrypt})$ be a public-key encryption scheme with the following components:

- **Key generation**: $\mathsf{KeyGeneration}(1^\lambda) \to (pk, sk)$, where λ is the security parameter, pk is the public key, and sk is the secret key.
- **Encryption**: $\mathsf{Encrypt}(pk, m) \to c$, where $m \in \mathcal{M}$ is a message from the message space and c is a ciphertext.
- **Decryption**: $\mathsf{Decrypt}(sk, c) \to m$ or $\perp$ if decryption fails.

The IND-CPA (Indistinguishability under Chosen Plaintext Attack) security of an encryption scheme $\mathcal{E}$ is defined via the following game between a challenger and an adversary $\mathcal{A}$:

IND-CPA Game

1. **Setup**: The challenger runs $\mathsf{KeyGeneration}(1^\lambda)$ to obtain a key pair (pk, sk). It gives the public key pk to the adversary $\mathcal{A}$.
2. **Query Phase**: The adversary $\mathcal{A}$ may adaptively query the encryption oracle $\mathsf{Encrypt}(pk, \cdot)$ on any message $m \in \mathcal{M}$ of its choice and receive the corresponding ciphertext $c = \mathsf{Encrypt}(pk, m)$.
3. **Challenge Phase**: The adversary $\mathcal{A}$ submits two messages $m_0, m_1 \in \mathcal{M}$ of equal length ($|m_0| = |m_1|$) to the challenger. The challenger selects a uniform random bit $b \in \{0, 1\}$ and computes the challenge ciphertext $c^* = \mathsf{Encrypt}(pk, m_b)$. The challenger sends c^* to $\mathcal{A}$.
4. **Guess**: The adversary $\mathcal{A}$ outputs a guess $b' \in \{0, 1\}$.

Winning Condition

The adversary $\mathcal{A}$ wins the game if $b' = b$. The advantage of the adversary $\mathcal{A}$ in the IND-CPA game is defined as:

$$\mathrm{Adv}_{\mathcal{E},\mathcal{A}}^{\text{IND-CPA}} = \left| \Pr[b' = b] - \frac{1}{2} \right|$$

An encryption scheme $\mathcal{E}$ is IND-CPA secure if, for every probabilistic polynomial-time (PPT) adversary $\mathcal{A}$, the advantage $\mathrm{Adv}_{\mathcal{E},\mathcal{A}}^{\text{IND-CPA}}$ is negligible in the security parameter λ: $\mathrm{Adv}_{\mathcal{E},\mathcal{A}}^{\text{IND-CPA}} \leq \mathsf{negl}(\lambda)$.

B Two-Party FOT Using Only FHE

Figure 8 presents the detailed protocol that we described in Section 4.1.

- *Input.* $\vec{m} = [m_0, \dots, m_{n-1}]$: sender $\mathcal{S}$'s input that includes n plaintext messages and $\vec{e} = [b, \dots, z]$: receiver $\mathcal{R}$'s input that contains a vector of t indices.
- *Output.* y: the result of evaluating F over $[m_b, \dots, m_z]$.

1. Setup: $\mathtt{Setup}(1^{\lambda}) \rightarrow (sk_{\mathsf{HE}}, pk_{\mathsf{HE}})$
   This phase involves $\mathcal{R}$.
   (a) calls $\mathsf{HE.KeyGeneration}(1^{\lambda}) \rightarrow (sk_{\mathsf{HE}}, pk_{\mathsf{HE}})$.
   (b) publishes pk_{HE}.
2. Query Generation: $\mathtt{GenQuery}(pk_{\mathsf{HE}}, n, \vec{e}) \rightarrow qry$
   This phase involves $\mathcal{R}$. It takes the following steps. $\forall e_j \in \vec{e}$:
   (a) constructs a vector $\vec{b_j} = [b_{j,0}, \dots, b_{j,n-1}]$, as follows:
      i. sets every element $b_{j,i}$ to 0 except for e_j-th element which is set to 1.
      ii. encrypts each element of $\vec{b_j}$ using FHE, $\forall i, 0 \leq i \leq n-1 : b'_{j,i} = \mathsf{HE.Encrypt}(pk_{\mathsf{HE}}, b_{j,i})$. Let $\vec{b'_j}$ be the vector of the encrypted elements.
   (b) sets query qry to $[\vec{b'_0}, \dots, \vec{b'_{t-1}}]$ and sends qry to $\mathcal{S}$.
3. Response Generation: $\mathtt{GenRes}(\vec{m}, pk_{\mathsf{HE}}, qry) \rightarrow res$
   This phase involves $\mathcal{S}$.
   (a) obliviously identifies the t messages that $\mathcal{R}$ is interested, using each vector $\vec{b'_j}$, as follows. $\forall j, 0 \leq j \leq t-1$:
   $$e_j = (m_0 \overset{H}{\times} b'_{j,0}) \overset{H}{+} \dots \overset{H}{+} (m_{n-1} \overset{H}{\times} b'_{j,n-1})$$
   (b) obliviously evaluate the function F on the t ciphertexts generated in the previous step.
   $$\mathsf{HE.Evaluate}(\mathsf{F}, pk_{\mathsf{HE}}, e_0, \dots, e_{t-1}) \rightarrow \theta$$
   (c) sets $res = \theta$ and sends θ to $\mathcal{R}$.
4. Message Extraction. $\mathtt{Retreive}(res, sk_{\mathsf{HE}}) \rightarrow y$
   - $\mathcal{R}$ decrypts the response as $\mathsf{HE.Decrypt}(sk_{\mathsf{HE}}, res) \rightarrow y$.

Fig. 8: Two-Party FOT Involving $O(t \cdot n)$ FHE.

C Converged OT: Multi-Sender Functional Oblivious Transfer

In this section, we present the Converged OT, a multi-sender extension of the protocol presented in Section 5.3. In this protocol, there are a receiver $\mathcal{R}$, a third party $\mathcal{H}$, and k senders $\mathcal{S}_1, \dots, \mathcal{S}_k$.

1. *Setup:* $\mathsf{Setup}(1^{\lambda}) \rightarrow \vec{r} = [\vec{r_1}, \dots, \vec{r_k}]$
   It is run by $\mathcal{R}$. For each sender $\mathcal{S}_l$ (where $1 \leq l \leq k$), it takes the following steps.
   (a) selects n random values $(r_{l,0}, \dots, r_{l,n-1}) \overset{\$}{\leftarrow} \{0,1\}^{\lambda}$. Let vector $\vec{r_l}$ be defined as $\vec{r_l} = [r_{l,0}, \dots, r_{l,n-1}]$. These elements will be used as a one-time pad by $\mathcal{S}$ to encrypt each message that $\mathcal{S}$ sends.
   (b) sends $\vec{r_l}$ to $\mathcal{S}_l$.

2. *Query Generation:* $\mathsf{GenQuery}(1^\lambda, \vec{e}_1, \dots, \vec{e}_k) \rightarrow qry := (qry_{\mathcal{S}_1}, \dots, qry_{\mathcal{S}_k}, qry_{\mathcal{H}_1}, \dots, qry_{\mathcal{H}_k})$
   It is run by $\mathcal{R}$. For each $\mathcal{S}_l$, it takes the following steps.
   (a) determines to which position, each index in a vector $\vec{v}_l$ of size n is moved, if $\vec{v}_l$ is independently and randomly permuted once. To do that, it takes the following steps.
      i. initiates a vector $\vec{v}_l$, such that its i-th element is set to i:
      $$\forall i, 0 \leq i \leq n-1: \quad \vec{v}_l[i] \leftarrow i$$
      ii. randomly permutes $\vec{v}_l$:
      $$\pi(\vec{v}_l) \rightarrow \vec{w}_l$$
   (b) finds the index of each element of its index vector $\vec{e}_l$ in $\vec{w}_l$. To do that, it initiates an empty vector $\vec{c}_l$ of size t and takes the following steps.
   $$\forall j, 0 \leq j \leq t-1: \quad \mathsf{Find}(\vec{w}_l, \vec{e}_l[j]) \rightarrow c_{l,j}, \quad \vec{c}_l[j] \leftarrow c_{l,j}$$
   (c) sets $qry_{\mathcal{S}_l} \leftarrow \vec{w}_l$ and $qry_{\mathcal{H}_l} \leftarrow \vec{c}_l$. It sends $qry_{\mathcal{S}_l}$ to $\mathcal{S}_l$ and $qry_{\mathcal{H}_l}$ to $\mathcal{H}$.
3. *Response Generation:* $\mathsf{GenRes}(m_{l,0}, \dots, m_{l,n-1}, \vec{r}_l, qry_{\mathcal{S}_l}, \vec{pp}) \rightarrow res_{\mathcal{H}_l}$
   It is run by each $\mathcal{S}_l$.
   (a) encrypts each message in $\vec{m}_l = [m_{l,0}, \dots, m_{l,n-1}]$ using the elements of $\vec{r}_l = [r_{l,0}, \dots, r_{l,n-1}]$ as:
   $$\mathsf{Encode}(\vec{m}_l, \vec{pp}, \vec{r}_l) \rightarrow \vec{h}_l$$
   (b) permutes vector $\vec{h}_l$ according the permutation map $\vec{w}_l \in qry_{\mathcal{S}_l}$. To do that, it initiates an empty vector $\vec{x}_l$ of size n. It finds the position of each value i in the permuted vector $\vec{w}_l$, let i' denote that position.
   It inserts i-th element from $\vec{h}_l$ into i'-th position in $\vec{x}_l$. More specifically,
   $$\forall i, 0 \leq i \leq n-1: \quad \mathsf{Find}(\vec{w}_l, i) \rightarrow i', \quad \vec{x}_l[i'] \leftarrow \vec{h}_l[i]$$
   (c) sets $res_{\mathcal{H}_l} \leftarrow \vec{x}_l$ and sends $res_{\mathcal{H}_l}$ to $\mathcal{H}$.
4. *Oblivious Evaluation:* $\mathsf{OblEvaluate}(res_{\mathcal{H}_1}, \dots, res_{\mathcal{H}_k}, qry_{\mathcal{H}_1}, \dots, qry_{\mathcal{H}_k}) \rightarrow res_{\mathcal{R}}$
   It is run by $\mathcal{H}$.
   (a) uses elements of $\vec{c}_l \in qry_{\mathcal{H}_l}$ to retrieve $\mathcal{R}$'s preferred encrypted messages in the permuted vector $\vec{x}_l \in res_{\mathcal{H}_l}$ and appends them to an empty vector $\vec{u}$. Specifically, it takes the following steps.
   $$\forall l, j, \quad 1 \leq l \leq k, \quad 0 \leq j \leq t-1: \quad \vec{u} \leftarrow \vec{x}_l\Big[\vec{c}_l[j]\Big]$$
   (b) obliviously evaluates the function F (specified in $\vec{pp}$) on the plaintext encoded in $\vec{u}$, by executing:
   $$\mathsf{Evaluate}(\vec{u}, \vec{pp}) \rightarrow \theta$$
   (c) sets $res_{\mathcal{R}}$ to θ and sends $res_{\mathcal{R}}$ to $\mathcal{R}$.
5. *Message Extraction:* $\mathsf{Retrieve}(res_{\mathcal{R}}, \vec{r}, \vec{e}_1, \dots, \vec{e}_k, \vec{pp}) \rightarrow y$
   It is run by $\mathcal{R}$.
   - retrieves the related secret keys for decoding. To do that, it initiates an empty vector $\vec{g}$ and appends to $\vec{g}$ the secret keys in $\vec{r}_l$ whose indices are specified in $\vec{e}_l$:
   $$\forall l, j, \quad 1 \leq l \leq k, \quad 0 \leq j \leq t-1: \quad \vec{g} \leftarrow r_{\vec{e}_l[j]}$$
   - decodes θ by invoking $\mathsf{Decode}(\theta, \vec{pp}, \vec{g}) \rightarrow y$.