Querying Private Databases (with Hidden Fields) via Delegated-Query Oblivious Transfer

Abstract—**To enhance privacy in databases we consider Oblivious Transfer (OT), which is an elegant cryptographic protocol. We observed that the existing research on this topic primarily focuses on theoretical cryptographic applications, neglecting several practical aspects: (i) OTs assume parties have direct access to databases. Our "1-out-of-2 Delegated-Query OT" allows parties to privately query a database (e.g., publishers) to which they do not have direct access. (ii) Due to the popularity of clouds, databases that were physically separated might no longer be. Our "1-out-of-2 Delegated-Query OT with Hidden Fields" protects privacy under such circumstances. (iii) Research ignored the limitations of thin clients, e.g., the Internet of Things (IoT) devices. We propose a compiler to transform any 1-out-of-n OT into a thin client version. (iv) OTs rely on (unproven) computational assumptions, except when using exotic approaches such as noisy channels or a fully trusted party. We find a very fast solution that avoids these assumptions and it has a wide range of applications.**

1. Introduction

Databases play a crucial role in e-commerce, advertising, intelligence analysis, combating crime, manufacturing, knowledge discovery, and conducting scientific research. Some databases (online libraries, aeroplane parts database, or Fortune 500 companies' databases about customers, e.g., purchase history) can be valued at millions of dollars and some of them (databases of a country's military and intelligence top secret projects/documents) are priceless.

Often the user of a database is not the one who created it. To simultaneously preserve the privacy of the user and the database itself from each other, the cryptographic-based technique, called Obvious Transfer (OT) has been proposed. It allows a user (called receiver) interested in the s-th element of the database (called sender) (m_0, m_1) to learn only m_s while preserving the privacy of: (i) index $s \in \{0,1\}$ from the sender (database) and (ii) the rest of the database's elements from the receiver. Numerous variants have been developed, since OT's introduction in 1981. There are (at least) a few research gaps in this research area.

Firstly, the state-of-the-art techniques have assumed that a receiver has *direct subscription/access to databases* and enough computation resources to generate queries that are sent to the sender. This assumption has to be relaxed when receivers are not subscribed to the database (e.g., they cannot afford it) or when receivers are thin clients, e.g., IoT devices with limited computational resources or battery lifespan.

Secondly, existing techniques are not suitable for the *real-world multi-receiver setting* where a sender (e.g., multi-tenant cloud) maintains multiple records[1] (containing elements of hidden/sensitive fields) each belonging to independent receivers. These techniques do not allow a receiver to privately query its related field elements without disclosing (i) the records the receiver accesses to the sender and (ii) the number of hidden field elements that other users of the database have to the receiver. The existing OTs reveal the entire database's size to receivers enabling them to acquire non-trivial information. The mere existence of private data itself can be considered sensitive information [1].

To tackle the size-leakage issue, one might consider two tempting approaches: (a) assigning a physically separated database to each receiver, resulting in more physical databases and increased maintenance costs compared to the single-database scenario, while also enabling the sender to learn which receiver can access specific records; or (b) allowing the receiver to disclose the record's index/ID of interest to the sender, which in itself presents a leakage.

Thirdly, the current techniques that allow a receiver to obtain data (response) with a *constant size* for its query necessitate the receiver to possess a storage space proportional to the size of the database (to locally store the database encryption). However, meeting this demanding requirement will become challenging for a thin client, if its available storage space is significantly smaller than the database size.

Fourthly, the state-of-the-art *unconditionally secure* techniques either (i) depend on the multi-sender setting, where each sender possesses a database replica, (ii) utilise a specific communication channel (i.e., noisy channel), or (iii) necessitate the presence of a fully trusted initialiser. Nevertheless, distributing the same database across multiple servers, establishing (or utilising) a highly specific communication channel, or involving a fully trusted party would result in an increase in the overall deployment cost of these schemes.

1.1. Our Contribution

In this paper, we propose solutions to the aforementioned limitations using new techniques as follows:

1) 1-out-of-2 Delegated-Query Oblivious Transfer ($\mathcal{DQ}$–$\mathcal{OT}^2_1$): a new notion of OT that (in addition to offering OT's basic features) lets a receiver *delegate*

1. A database table consists of records/rows and fields/columns. Each record in a table represents a set of related data, e.g., first name, last name, and address. A field contains the same data type, e.g., first names.

(i) the computation of the query and (ii) interacting with the sender to a couple of potentially semi-honest parties, P_1 and P_2, while ensuring that the sender and receiver privacy is also protected from P_1 and P_2. Section 4.2 presents $\mathcal{DQ}$–$\mathcal{OT}^2_1$.

   a) Delegated-Query OT (DQ-OT): a protocol that realises $\mathcal{DQ}$–$\mathcal{OT}^2_1$. Section 4.3 presents DQ-OT.
   b) Delegate-Unknown-Query OT (DUQ-OT): a variant of DQ-OT which lets the receiver extract the related message m_s even if it does not (and must not) know the related index s. Section 5.2 presents DUQ-OT.

2) 1-out-of-2 Delegated-Query OT with Hidden Fields ($\mathcal{DQ}^{\mathcal{HF}}$–$\mathcal{OT}^2_1$): a new notion of OT that (in addition to offering OT's primary features) ensures (i) a receiver learns nothing about the total number of records and their field elements and (ii) the sender who maintains z records $[(m_{0,0}, m_{1,0}), ..., (m_{0,z-1}, m_{1,z-1})]$ does not find out which query belongs to which record. Section 6.1.1 presents $\mathcal{DQ}^{\mathcal{HF}}$–$\mathcal{OT}^2_1$.
   a) Delegated-Query OT with Hidden Fields (DQ$^{\text{HF}}$–OT): an efficient protocol that realises $\mathcal{DQ}^{\mathcal{HF}}$–$\mathcal{OT}^2_1$. It is built upon DQ-OT and inherits its features. DQ$^{\text{HF}}$–OT achieves its goal by allowing P_1 to know which record is related to which receiver. Section 6.1.3 presents DQ$^{\text{HF}}$–OT.
   b) Delegate-Unknown-Query OT with Hidden Fields (DUQ$^{\text{HF}}$–OT): a variant of DQ$^{\text{HF}}$–OT which considers the case where P_1 and P_2 do not (and must not) know which record in the database belongs to which receiver. Section 6.2.2 presents DUQ$^{\text{HF}}$–OT.
3) A compiler: a generic wrapper that transforms any 1-out-of-n OT that requires the receiver to receive n messages (as a response) into a 1-out-of-n OT that allows a receiver to (i) receive only a *constant* number of messages and (ii) have constant storage space. Section 7 presents the compiler.
4) Supersonic OT: an unconditionally secure 1-out-of-2 OT that *does not* need to rely on (i) multiple senders, (ii) noisy channel, or (iii) the involvement of a trusted initialiser. Moreover, Supersonic OT:
   - is highly fast as it does not require any public-key-based cryptography.
   - lets the receiver obtain a response of size $O(1)$.

   Section 8 presents Supersonic OT.

2. Preliminaries

2.1. Notations

By ϵ we mean an empty string and by $|y|$ we mean a bit length of value y. We denote a sender by S and a receiver by R. We assume parties interact with each other through a regular secure channel. We define a parse function as $\texttt{parse}(\lambda, y) \rightarrow (a, b)$, which takes as input a value λ and a value y of length at least λ-bit. It parses y into two values a and b and returns (a, b) where the bit length of a is $|y| - \lambda$ and the bit length of b is λ. U denotes a universe of messages $m_1, ..., m_t$. We define σ as the maximum size of messages in U, i.e., $\sigma = Max(|m_1|, ..., |m_t|)$. We use two hash functions $\texttt{H} : \{0,1\}^* \rightarrow \{0,1\}^\sigma$ and $\texttt{G} : \{0,1\}^* \rightarrow \{0,1\}^{\sigma+\lambda}$ modelled as random oracles [2].

2.2. Security Model

In this paper, we use the simulation-based paradigm of secure multi-party computation [3] to define and prove the proposed protocol (see also [4]). Since we focus on the static passive (semi-honest) adversarial model, we will restate the security definition in this adversarial model.

2.2.1. Two-party Computation. A two-party protocol Γ problem is captured by specifying a random process that maps pairs of inputs to pairs of outputs, one for each party. Such 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 $f_2(x, y)$. When f is deterministic, then $f_1 = f_2$. In the setting where f is asymmetric and only one party (say the first one) receives the result, f is defined as $f := (f_1(x, y), \epsilon)$.

2.2.2. Security in the Presence of Passive Adversaries. In the passive adversarial model, the party corrupted by such an adversary correctly follows the protocol specification. Nonetheless, the adversary obtains the internal state of the corrupted party, including the transcript of all the messages received, and tries to use this to learn information that should remain private. Loosely speaking, a protocol is secure if whatever can be computed by a party in the protocol can be computed using its input and output only. 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. More formally, party i's view (during the execution of Γ) on input pair (x, y) is denoted by $\mathsf{View}^\Gamma_i(x, y)$ and equals $(w, r^i, m^i_1, ..., m^i_t)$, where $w \in \{x, y\}$ is the input of i^{th} party, r_i is the outcome of this party's internal random coin tosses, and m^i_j 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 $\mathsf{Output}^\Gamma_i(x, y)$ and can be generated from its own view of the execution.

Definition 1. Let f be the deterministic functionality defined above. Protocol Γ security computes f in the presence of a static passive adversary if there exist polynomial-time algorithms $(\mathsf{Sim}_1, \mathsf{Sim}_2)$ such that:

$$\{\mathsf{Sim}_1(x, f_1(x, y))\}_{x,y} \overset{c}{\equiv} \{\mathsf{View}^\Gamma_1(x, y)\}_{x,y}$$

$$\{\mathsf{Sim}_2(y, f_2(x, y))\}_{x,y} \overset{c}{\equiv} \{\mathsf{View}^\Gamma_2(x, y)\}_{x,y}$$

2.3. Random Permutation

A random permutation $\pi(e_0, ..., e_n) \rightarrow (e'_0, ..., e'_n)$ is a probabilistic function that takes a set of n elements $A = \{e_0, ..., e_n\}$ and returns the same set elements in a permuted order $B = \{e'_0, ..., e'_n\}$. The security of $\pi(.)$ requires that given set B the probability that one can find the original index of an element $e'_i \in B$ is $\frac{1}{n}$. In practice, the Fisher-Yates shuffle algorithm [5] can permute a set of n elements in time $O(n)$.

2.4. Diffie-Hellman Assumption

Let G be a group-generator scheme, which on input 1^λ outputs $(\mathbb{G}, p, g)$ where $\mathbb{G}$ is the description of a group, p is the order of the group which is always a prime number, $\log_2(p) = \lambda$ is a security parameter and g is a generator of the group.

2.4.1. Computational Diffie-Hellman (CDH) Assumption. We say that G is hard under CDH assumption, if for any probabilistic polynomial time (PPT) adversary $\mathcal{A}$, given (g^{a_1}, g^{a_2}) it has only negligible probability to correctly compute $g^{a_1 \cdot a_2}$. More formally, it holds that $Pr[\mathcal{A}(\mathbb{G}, p, g, g^{a_1}, g^{a_2}) \rightarrow g^{a_1 \cdot a_2}] \leq \mu(\lambda)$, where $(\mathbb{G}, p, g) \overset{\$}{\leftarrow} G(1^\lambda)$, $a_1, a_2 \overset{\$}{\leftarrow} \mathbb{Z}_p$, and μ is a negligible function [6].

2.5. Secret Sharing

A (threshold) secret sharing $\texttt{SS}^{(t,n)}$ scheme is a cryptographic protocol that enables a dealer to distribute a string s, known as the secret, among n parties in a way that the secret s can be recovered when at least a predefined number of shares, say t, are combined. However, if the number of shares in any subset is less than t, the secret remains unrecoverable and the shares divulge no information about s. This type of scheme is commonly referred to as (n, t)-secret sharing or $\texttt{SS}^{(t,n)}$ for brevity.

In the case where $t = n$, there exists a highly efficient XOR-based secret sharing [7]. In this case, to share the secret s, the dealer first picks $n - 1$ random bit strings $r_1, ..., r_{n-1}$ of the same length as the secret. Then, it computes $r_n = r_1 \oplus, ..., \oplus r_n \oplus s$. It considers each $r_i \in \{r_1, .., r_n\}$ as a share of the secret. To reconstruct the secret, one can easily compute $r_1 \oplus, ..., \oplus r_n$. Any subset of less than n shares reveals no information about the secret. We will use this scheme in this paper. Thus, a secret sharing scheme involves two main algorithms; namely, $\texttt{SS}(1^\lambda, s, n, t) \rightarrow (r_1, ..., r_n)$: to share a secret and $\texttt{RE}(r_1, ..., r_t, n, t) \rightarrow s$ to reconstruct the secret.

2.6. Additive Homomorphic Encryption

Additive homomorphic encryption involves three algorithms: (1) key generation: $\texttt{KGen}(1^\lambda) \rightarrow (sk, pk)$, which takes a security parameter as input and outputs a secret and public keys pair, (2) encryption: $\texttt{Enc}(pk, m) \rightarrow c$, that takes public key pk and a plaintext message m as input and returns a ciphertext c, and (3) decryption: $\texttt{Dec}(sk, c) \rightarrow m$, which takes secret key sk and ciphertext c as input and returns plaintext message m. It has the following properties:

- Given two ciphertexts $\texttt{Enc}(pk, m_1)$ and $\texttt{Enc}(pk, m_2)$, one can compute the encryption of the sum of related plaintexts: $\texttt{Enc}(pk, m_1) \overset{H}{+} \texttt{Enc}(pk, m_2) = \texttt{Enc}(pk, m_1 + m_2)$, where $\overset{H}{+}$ denotes homomorphic addition.
- Given a ciphertext $\texttt{Enc}(pk, m)$ and a plaintext message c, one can compute the encryption of the product of related plaintexts: $\texttt{Enc}(pk, m) \overset{H}{\times} c = \texttt{Enc}(pk, m \cdot c)$, where $\overset{H}{\times}$ denotes homomorphic multiplication.

We require that the encryption scheme satisfies indistinguishability against chosen-plaintext attacks (IND-CPA). We refer readers to [8] for a formal definition. One such scheme that meets all the aforementioned features is the Paillier public key cryptosystem, proposed in [9].

3. Related Work

Oblivious Transfer (OT) is one of the important building blocks of cryptographic protocols and has been used in various protocols, such as private set intersection, generic secure multi-party computation, and zero-knowledge proofs. A 1-out-of-2 OT ($\mathcal{OT}^2_1$) is a protocol that involves two parties, a sender S and a receiver R. S has a pair of input messages (m_0, m_1) and R has an index s. The aim of $\mathcal{OT}^2_1$ is to allow R to obtain m_s, without revealing anything about s to S, and without allowing R to learn anything about m_{1-s}. The $\mathcal{OT}^2_1$ functionality is defined as $\mathcal{F}_{\mathcal{OT}^2_1} : ((m_0, m_1), s) \rightarrow (\epsilon, m_s)$.

The notion of 1-out-of-2 OT was initially proposed by Rabin [10] which consequently was generalised by Even *et al.* [11]. Since then, numerous variants of OT have been proposed. For instance, (i) 1-out-of-n OT, e.g., in [12]–[14]: which allows R to pick one entry out of n entries held by S, (ii) k-out-of-n OT, e.g., in [15]–[17]: which allows R to pick k entries out of n entries held by S, (iii) OT extension, e.g., in [18]–[21]: that supports efficient executions of OT (that mainly relies on symmetric-key operations), in the case OT needs to be invoked many times, and (iv) distributed OT, e.g., in [22]–[24]: that allows the database to be distributed among m servers/senders.

3.1. Distributed OT

Naor and Pinkas [22] proposed several protocols for distributed OT where the role of sender S (in the original OT) is divided between several servers. In these schemes, a receiver must contact a threshold of the servers to run the OT protocol. The proposed protocols are in the semi-honest model. They use symmetric-key primitives and do not involve any modular exponentiation that can lead to efficient implementations. These protocols are based on various variants of polynomials (e.g., sparse and bivariate), polynomial evaluation, and pseudorandom function. In these distributed

OTs, the security against the servers holds as long as less than a predefined number of these servers collude. Later on, various distributed OTs have been proposed[2]. For instance, Corniaux and Ghodosi [23] proposed a verifiable 1-out-of-n distributed OT that considers the case where a threshold of the servers are potentially active adversaries. The scheme is based on a sparse n-variate polynomial, verifiable secret sharing, and error-correcting codes. Moreover, Zhao *et al.* [24] has proposed a distributed version of OT extension that aims to preserve the efficiency of OT extension while delegating the role of S to multiple servers a threshold of which can be potentially semi-honest. The scheme is based on a hash function and an oblivious pseudorandom function.

However, there exists no OT that supports the delegation of the query computation to a set of servers in a privacy-preserving manner.

3.2. Multi-receiver OT

Camenisch *et al.* [26] proposed a protocol for "OT with access control". This protocol involves a set of receivers and a sender which maintains records of the receivers. It offers a set of interesting features; namely, (i) only authorised receivers can access certain records; (ii) the sender does not learn which record a receiver accesses, and (iii) the sender does not learn which roles (or security clearance) the receiver has when it accesses the records. In this scheme, during the setup, the sender encrypts all records (along with their field elements) and publishes the entire encrypted database for the receivers to download. Subsequently, researchers proposed various variants of OT with access control, as seen in [27]–[31]. Nevertheless, in all of the aforementioned schemes, the size of the entire database is revealed to the receivers.

3.3. OT with Constant Response Size

Researchers have proposed several OTs, e.g., those proposed in [26], [32]–[35], that enable a receiver to obtain a constant-size response to its query. To achieve this level of communication efficiency, these protocols require the receiver to locally store the encryption of the entire database, in the initialisation phase. During the transfer phase, the sender assists the receiver with locally decrypting the message that the receiver is interested in. The main limitation of these protocols is that a thin client with limited available storage space cannot use them, as it cannot locally store the encryption of the entire database.

3.4. Unconditionally Secure OT

There have been efforts to design (both-sided) unconditionally secure OT protocols. Some schemes, e.g., the ones in [22], [23], [36], rely on multiple servers/senders that maintain an identical copy of the database. Other ones, e.g., those in [37]–[39], rely on a specific network structure, i.e., a noisy channel, to achieve unconditionally secure OT. There is also a scheme in [40] that achieves unconditionally secure OT using a fully trusted initialiser. Hence, there exists no (efficient) unconditionally secure OT that does not rely on noisy channels, multi-server, and fully trusted initialiser.

We refer readers to [41] for a recent survey of OT.

2. Distributed OT has also been called proxy OT in [25].

4. Delegated-Query OT

In this section, we present the notion of Delegated-Query 1-out-of-2 OT ($\mathcal{DQ}\text{–}\mathcal{OT}_1^2$) and a protocol that realises it. $\mathcal{DQ}\text{–}\mathcal{OT}_1^2$ involves four parties; namely, sender S, receiver R, and two helper servers P_1 and P_2 that assist R to compute the query.

Informally, $\mathcal{DQ}\text{–}\mathcal{OT}_1^2$ enables R to delegate (i) the computation of query and (ii) the interaction with S to P_1 and P_2, who jointly compute R's query and send it to S. $\mathcal{DQ}\text{–}\mathcal{OT}_1^2$ (in addition to offering the basic security of OT) ensures that R's privacy is preserved from P_1 and P_2, in the sense that P_1 and P_2 do not learn anything about the actual index (i.e., $s \in \{0, 1\}$) that R is interested in, as long as they do not collude with each other.

4.1. Functionality Definition

Informally, the functionality that $\mathcal{DQ}\text{–}\mathcal{OT}_1^2$ computes takes as input (i) a pair of messages (m_0, m_1) from S, (ii) empty string ϵ from P_1, (iii) empty string ϵ from P_2, and (iv) the index s (where $s \in \{0, 1\}$) from R. It outputs an empty string ϵ to S, P_1, and P_2, and outputs the message with index s, i.e., m_s, to R. Formally, we define the functionality as: $\mathcal{F}_{\mathcal{DQ}\text{-}\mathcal{OT}_1^2} : \big((m_0, m_1), \epsilon, \epsilon, s\big) \rightarrow (\epsilon, \epsilon, \epsilon, m_s)$.

4.2. Security Definition

Next, we present a formal definition of $\mathcal{DQ}\text{–}\mathcal{OT}_1^2$.

Definition 2 ($\mathcal{DQ}\text{–}\mathcal{OT}_1^2$)**.** Let $\mathcal{F}_{\mathcal{DQ}\text{-}\mathcal{OT}_1^2}$ be the delegated-query OT functionality defined above. We say protocol Γ realises $\mathcal{F}_{\mathcal{DQ}\text{-}\mathcal{OT}_1^2}$ in the presence of static passive adversary S, R, P_1, or P_2, if for every non-uniform probabilistic polynomial time (PPT) adversary $\mathcal{A}$ in the real model, there exists a non-uniform PPT adversary (or simulator) $\mathtt{Sim}$ in the ideal model, such that:

$$\Big\{\mathtt{Sim}_S\big((m_0, m_1), \epsilon\big)\Big\}_{m_0, m_1, s} \overset{c}{\equiv} \overset{c}{\equiv} \Big\{\mathtt{View}_S^{\Gamma}\big((m_0, m_1), \epsilon, \epsilon, s\big)\Big\}_{m_0, m_1, s} \quad (1)$$

$$\Big\{\mathtt{Sim}_{P_i}(\epsilon, \epsilon)\Big\}_{m_0, m_1, s} \overset{c}{\equiv} \overset{c}{\equiv} \Big\{\mathtt{View}_{P_i}^{\Gamma}\big((m_0, m_1), \epsilon, \epsilon, s\big)\Big\}_{m_0, m_1, s} \quad (2)$$

$$\Big\{\mathtt{Sim}_R\Big(s, \mathcal{F}_{\mathcal{DQ}\text{-}\mathcal{OT}_1^2}\big((m_0, m_1), \epsilon, \epsilon, s\big)\Big)\Big\}_{m_0, m_1, s} \overset{c}{\equiv} \overset{c}{\equiv} \Big\{\mathtt{View}_R^{\Gamma}\big((m_0, m_1), \epsilon, \epsilon, s\big)\Big\}_{m_0, m_1, s} \quad (3)$$

for all i, $i \in \{1, 2\}$.

Intuitively, Relation 1 states that the view of a corrupt S during the execution of protocol Γ (in the real model) can be simulated by a simulator $\mathtt{Sim}_S$ (in the ideal model) given only S's input and output, i.e., (m_0, m_1) and ϵ respectively.

Relation 2 states that the view of each corrupt server P_i during the execution of Γ can be simulated by a simulator $\mathtt{Sim}_{P_i}$ given only P_i's input and output, i.e., ϵ and ϵ respectively.

Relation 3 states that the view of a corrupt R during the execution of Γ can be simulated by a simulator $\mathtt{Sim}_R$ given only R's input and output, i.e., s and m_s respectively.

4.3. Protocol

Now, we present an efficient 1-out-of-2 OT protocol, called DQ-OT, that realises $\mathcal{DQ}\text{–}\mathcal{OT}_1^2$. We build DQ-OT upon the $\mathcal{OT}_1^2$ proposed by Naor and Pinkas [42, p. 451]. Our motivation for this choice is primarily didactic.

The high-level idea behind the design of DQ-OT is that R splits its index into two shares and sends each share to each P_i. Subsequently, each P_i computes a (partial) query and sends the result to S which generates the response for R in the same manner as the original OT in [42]. The primary challenge is to *ensure correctness* while preserving privacy. Below, we provide an explanation of how DQ-OT operates, followed by an explanation of how it achieves both correctness and security.

First, R splits the index that it is interested in into two binary shares, namely (s_1, s_2). Then, it picks two random values, (r_1, r_2), and then sends each pair (s_i, r_i) to each P_i.

Second, to compute a partial query, P_2 treats s_2 as the main index that R is interested in and computes a partial query, $\delta_{s_2} = g^{r_2}$. P_2 also generates another query, $\delta_{1-s_2} = \frac{C}{g^{r_2}}$, where C is a random public parameter (as defined in [42]). P_2 sorts the two queries in ascending order based on the value of s_2 and sends the resulting (δ_0, δ_1) to P_1.

Third, to compute its queries, P_1 treats δ_0 as the main index (that R is interested) and computes $\beta_{s_1} = \delta_0 \cdot g^{r_1}$. Additionally, it generates another query $\beta_{1-s_1} = \frac{\delta_1}{g^{r_1}}$. Subsequently, P_1 sorts the two queries in ascending order based on the value of s_1 and sends the resulting (β_0, β_1) to R.

Forth, given the queries, S computes the response in the same manner it does in the original OT in [42] and sends the result to R who extracts from it, the message that it asked for, with the help of s_i and r_i values. The detailed DQ-OT is presented in Figure 1.

Theorem 1. *Let $\mathcal{F}_{\mathcal{DQ}\text{-}\mathcal{OT}_1^2}$ be the functionality defined in Section 4.2. If Discrete Logarithm (DL), Computational Diffie-Hellman (CDH), and Random Oracle (RO) assumptions hold, then DQ-OT (presented in Figure 1) securely computes $\mathcal{F}_{\mathcal{DQ}\text{-}\mathcal{OT}_1^2}$ in the presence of (a) semi-honest receiver R, honest sender S, and honest servers P_1 and P_2, (b) semi-honest S, honest R, and honest P_1 and P_2, or (c) semi-honest P_i (where $i \in \{1, 2\}$), honest S, and honest S, w.r.t. Definition 2.*

1) <u>*R-side Delegation:*</u>
   a) split the private index s into two shares (s_1, s_2) by calling $\mathtt{SS}(1^\lambda, s, 2, 2) \rightarrow (s_1, s_2)$.
   b) pick two uniformly random values: $r_1, r_2 \overset{\$}{\leftarrow} \mathbb{Z}_{p-1}$.
   c) send (s_1, r_1) to P_1 and (s_2, r_2) to P_2.
2) <u>*P_2-side Query Generation:*</u>
   a) compute a pair of partial queries:
   $$\delta_{s_2} = g^{r_2}, \quad \delta_{1-s_2} = \frac{C}{g^{r_2}}$$
   b) send (δ_0, δ_1) to P_1.
3) <u>*P_1-side Query Generation:*</u>
   a) compute a pair of final queries as:
   $$\beta_{s_1} = \delta_0 \cdot g^{r_1}, \quad \beta_{1-s_1} = \frac{\delta_1}{g^{r_1}}$$
   b) send (β_0, β_1) to S.
4) <u>*S-side Response Generation:*</u>
   a) abort if $C \neq \beta_0 \cdot \beta_1$.
   b) pick two uniformly random values: $y_0, y_1 \overset{\$}{\leftarrow} \mathbb{Z}_{p-1}$
   c) compute a response pair (e_0, e_1) as follows:
   $$e_0 := (e_{0,0}, e_{0,1}) = (g^{y_0}, \mathtt{H}(\beta_0^{y_0}) \oplus m_0)$$
   $$e_1 := (e_{1,0}, e_{1,1}) = (g^{y_1}, \mathtt{H}(\beta_1^{y_1}) \oplus m_1)$$
   d) send (e_0, e_1) to R.
5) <u>*R-side Message Extraction:*</u>
   a) set $x = r_2 + r_1 \cdot (-1)^{s_2}$
   b) retrieve the related message: $m_s = \mathtt{H}((e_{s,0})^x) \oplus e_{s,1}$

Figure 1. DQ-OT: Our 1-out-of-2 OT that supports query delegation. In the protocol, $C = g^a$ is a random public value, $\mathtt{SS}(.)$ is the share-generation algorithm (of a secret sharing) that has been defined in Section 2.5, $\mathtt{H}(.)$ is a hash function, and \$ denotes picking a value uniformly at random. Note, R-side's initial values can be pre-computed before the protocol's execution.

4.4. Proof of Security

Below, we prove DQ-OT's security, i.e., Theorem 1.

Proof. We consider the case where each party is corrupt, at a time.

4.4.1. Corrupt Receiver R.

In the real execution, R's view is:

$\mathtt{View}_R^{DQ\text{-}OT}(m_0, m_1, \epsilon, \ \epsilon, s) \ = \ \{r_R, C, e_0, e_1, m_s\}$, where $C = g^a$ is a random value and public parameter, a is a random value, and r_R is the outcome of the internal random coin of R and is used to generate (r_1, r_2). Below, we construct an idea-model simulator $\mathtt{Sim}_R$ which receives $\{s, m_s\}$ from R.

1) constructs an empty view and appends uniformly random coin $r'_{\scriptscriptstyle R}$ to it, where $r'_{\scriptscriptstyle R}$ will be used to generate R-side randomness.
2) sets (e'_0, e'_1) as follows:
   - splits s into two shares: $\mathtt{SS}(1^\lambda, s, 2, 2) \rightarrow (s'_1, s'_2)$.
   - picks uniformly random values: $C', r'_1, r'_2, y'_0, y'_1 \xleftarrow{\$} \mathbb{Z}_{p-1}$.
   - sets $\beta'_s = g^x$, where x is set as follows:
     * $x = r'_2 + r'_1$, if $(s = s_1 = s_2 = 0)$ or $(s = s_1 = 1 \wedge s_2 = 0)$.
     * $x = r'_2 - r'_1$, if $(s = 0 \wedge s_1 = s_2 = 1)$ or $(s = s_2 = 1 \wedge s_1 = 0)$.
   - picks a uniformly random value $u \xleftarrow{\$} \mathbb{Z}_p$ and then sets $e'_s = (g^{y'_s}, \mathtt{H}(\beta_s'^{y'_s}) \oplus m_s)$ and $e'_{1-s} = (g^{y'_{1-s}}, u)$.
3) appends $(C', r'_1, r'_2, e'_0, e'_1, m_s)$ to the view and outputs the view.

Now we discuss why the two views in the ideal and real models are indistinguishable. Since we are in the semi-honest model, the adversary picks its randomness according to the protocol description; thus, $r_{\scriptscriptstyle R}$ and $r'_{\scriptscriptstyle R}$ model have identical distributions, so do values $(r_1, r_2,)$ in the real model and (r'_1, r'_2) in the ideal model. Also, C and C' have been picked uniformly at random and have identical distribution.

Next, we argue that e_{1-s} in the real model and e'_{1-s} in the ideal model are indistinguishable. In the real model, it holds that $e_{1-s} = (g^{y_{1-s}}, \mathtt{H}(\beta_{1-s}^{y_{1-s}}) \oplus m_{1-s})$, where $\beta_{1-s}^{y_{1-s}} = \frac{C}{g^x} = g^{a-x}$. Since y_{1-s} in the real model and y'_{1-s} in the ideal model have been picked uniformly at random and unknown to the adversary/distinguisher, $g^{y_{1-s}}$ and $g^{y'_{1-s}}$ have identical distributions.

Furthermore, in the real model, given $C = g^a$, due to DL problem, a cannot be computed by a PPT adversary. Also, due to CDH assumption, R cannot compute $\beta_{1-s}^{y_{1-s}}$ (i.e., the input of $\mathtt{H}(.)$), given $g^{y_{1-s}}$ and g^{a-x}. We also know that $\mathtt{H}(.)$ is modelled as a random oracle and its output is indistinguishable from a random value. Thus, $\mathtt{H}(\beta_{1-s}^{y_{1-s}}) \oplus m_{1-s}$ in the real model and u in the ideal model are indistinguishable. This means that e_{1-s} and e'_{1-s} are indistinguishable too, due to DL, CDH, and RO assumptions. Also, in the real and idea models, e_s and e'_s have been defined over $\mathbb{Z}_p$ and their decryption always result in the same value m_s. Thus, e_s and e'_s have identical distributions too. Also, m_s has identical distribution in both models.

We conclude that the two views are computationally indistinguishable, i.e., Relation 3 (in Section 4.2) holds.

4.4.2. Corrupt Sender S.

In the real model, S's view is: $\mathtt{View}_S^{\scriptscriptstyle DQ\text{-}OT}\big((m_0, m_1), \epsilon, \epsilon,\ s\big) = \{r_S, C, \beta_0, \beta_1\}$, where r_S is the outcome of the internal random coin of S. Next, we construct an idea-model simulator $\mathtt{Sim}_S$ which receives $\{m_0, m_1\}$ from S.

1) constructs an empty view and appends uniformly random coin r'_S to it, where r'_S will be used to generate random values for S.
2) picks random values $C', r' \xleftarrow{\$} \mathbb{Z}_{p-1}$.
3) sets $\beta'_0 = g^{r'}$ and $\beta'_1 = \frac{C'}{g^{r'}}$.
4) appends β'_0 and β'_1 to the view and outputs the view.

Next, we explain why the two views in the ideal and real models are indistinguishable. Recall, in the real model, (β_s, β_{1-s}) have the following form: $\beta_s = g^x$ and $\beta_{1-s} = g^{a-x}$, where $a = DL(C)$ and $C = g^a$. In this model, because a and x have been picked uniformly at random and unknown to the adversary, due to DL assumption, β_s and β_{1-s} have identical distributions and are indistinguishable. In the ideal model, r' has been picked uniformly at random and we know that a' in $C' = g^{a'}$ is a uniformly random value, unknown to the adversary; therefore, due to DL assumption, β'_0 and β'_1 have identical distributions too. Moreover, values $\beta_s, \beta_{1-s}, \beta'_0$, and β'_1 have been defined over the same field, $\mathbb{Z}_p$. Thus, they have identical distributions and are indistinguishable.

Therefore, the two views are computationally indistinguishable, i.e., Relation 1 (in Section 4.2) holds.

4.4.3. Corrupt Server P_2.

In the real execution, P_2's view is: $\mathtt{View}_{P_2}^{\scriptscriptstyle DQ\text{-}OT}\big((m_0, m_1), \epsilon,\ \epsilon, s\big) = \{C, s_2, r_2\}$. Below, we show how an ideal-model simulator $\mathtt{Sim}_{P_2}$ works.

1) constructs an empty view.
2) picks two uniformly random values $s'_2 \xleftarrow{\$} \mathbb{U}$ and $C', r'_2 \xleftarrow{\$} \mathbb{Z}_{p-1}$, where $\mathbb{U}$ is the output range of $\mathtt{SS}(.)$.
3) appends s'_2, C' and r'_2 to the view and outputs the view.

Next, we explain why the views in the ideal and real models are indistinguishable. Since r_2 and r'_2 have been picked uniformly at random from $\mathbb{Z}_{p-1}$, they have identical distributions. Also, due to the security of $\mathtt{SS}(.)$ each share s_2 is indistinguishable from a random value s'_2, where $s'_2 \in \mathbb{U}$. Also, both C and C' have been picked uniformly at random from $\mathbb{Z}_{p-1}$; therefore, they have identical distribution.

Thus, the two views are computationally indistinguishable, i.e., Relation 2 w.r.t. P_2 (in Section 4.2) holds.

4.4.4. Corrupt Server P_1.

In the real execution, P_1's view is: $\mathtt{View}_{P_1}^{\scriptscriptstyle DQ\text{-}OT}\big((m_0, m_1), \epsilon,\ \ \epsilon, s\big) = \{C, s_1, r_1, \delta_0, \delta_1\}$. Ideal-model $\mathtt{Sim}_{P_1}$ works as follows.

1) constructs an empty view.
2) picks two random values $\delta'_0, \delta'_1 \xleftarrow{\$} \mathbb{Z}_p$.
3) picks two uniformly random values $s'_1 \xleftarrow{\$} \mathbb{U}$ and $C', r'_1 \xleftarrow{\$} \mathbb{Z}_{p-1}$, where $\mathbb{U}$ is the output range of $\mathtt{SS}(.)$.
4) appends $s'_1, C', r'_1, \delta'_0, \delta'_1$ to the view and outputs the view.

Now, we explain why the views in the ideal and real models are indistinguishable. Recall, in the real model, P_1 receives $\delta_{s_2} = g^{r_2}$ and $\delta_{1-s_2} = g^{a-r_2}$ from P_2. Since a and r_2 have been picked uniformly at random and unknown to the adversary due to DL assumption, δ_{s_2} and δ_{1-s_2} (or δ_0 and δ_1) have identical distributions and are indistinguishable from random values (of the same field).

In the ideal model, δ'_0 and δ'_1 have been picked uniformly at random; therefore, they have identical distributions too.

Moreover, values $\delta_s, \delta_{1-s}, \delta'_0$, and δ'_1 have been defined over the same field, $\mathbb{Z}_p$. So, they have identical distributions and are indistinguishable. Due to the security of $\texttt{SS}(.)$ each share s_1 is indistinguishable from a random value s'_1, where $s'_1 \in \mathbb{U}$. Furthermore, (r_1, C) and (r'_1, C') have identical distributions, as they are picked uniformly at random from $\mathbb{Z}_{p-1}$.

Hence, the two views are computationally indistinguishable, i.e., Relation 2 w.r.t. P_1 (in Section 4.2) holds. $\square$

4.5. Proof of Correctness

In this section, we discuss why the correctness of DQ-OT always holds. Recall, in the original OT of Naor and Pinkas [42], the random value a (i.e., the discrete logarithm of random value C) is inserted by recipient R into the query β_{1-s} whose index (i.e., $1-s$) is not interesting to R while the other query β_s is free from value a. As we will explain below, in our DQ-OT, the same applies to the final queries that are sent to S. Briefly, in DQ-OT, when:

- $s = s_1 \oplus s_2 = 1$ (i.e., when $s_1 \neq s_2$), then a will always appear in $\beta_{1-s} = \beta_0$; however, a will not appear in β_1.
- $s = s_1 \oplus s_2 = 0$ (i.e., when $s_1 = s_2$), then a will always appear in $\beta_{1-s} = \beta_1$; but a will not appear in β_0.

This is evident in Table 1 which shows what δ_i and β_j are for the different values of s_1 and s_2. Therefore, the query pair (β_0, β_1) has the same structure as it has in [42].

	$s_2 = 0$	$s_2 = 1$
$s_1 = 0$	$\delta_0 = g^{r_2}$, $\delta_1 = g^{a-r_2}$ $\beta_0 = g^{r_2+r_1}$, $\beta_1 = g^{a-r_2-r_1}$	$\delta_0 = g^{a-r_2}$, $\delta_1 = g^{r_2}$ $\beta_0 = g^{a-r_2+r_1}$, $\beta_1 = g^{r_2-r_1}$
$s_1 = 1$	$\delta_0 = g^{r_2}$, $\delta_1 = g^{a-r_2}$ $\beta_0 = g^{a-r_2-r_1}$, $\beta_1 = g^{r_2+r_1}$	$\delta_0 = g^{a-r_2}$, $\delta_1 = g^{r_2}$ $\beta_0 = g^{r_2-r_1}$, $\beta_1 = g^{a-r_2+r_1}$

TABLE 1. δ_i AND β_j ARE FOR THE DIFFERENT VALUES OF s_1 AND s_2. WE EXPRESS EACH VALUE AS A POWER OF g.

Next, we show why, in DQ-OT, R can extract the correct message, i.e., m_s. Given S's reply pair (e_0, e_1) and its original index s, R knows which element to pick from the response pair, i.e., it picks e_s.

Moreover, given $g^{y_s} \in e_s$, R can always recompute $\texttt{H}(g^{y_s})^x$, as it knows the value of s, s_1, and s_2. Specifically, as Table 1 indicates, when:

- $(s = s_1 = s_2 = 0)$ or $(s = s_1 = 1 \wedge s_2 = 0)$, then R can set $x = r_2 + r_1$.
  - In the former case, it holds $\texttt{H}((g^{y_0})^x) = \texttt{H}((g^{y_0})^{r_2+r_1}) = q$. On the other hand, $e_0 = \texttt{H}(\beta_0^{y_0}) \oplus m_0 = \texttt{H}((g^{r_2+r_1})^{y_0}) \oplus m_0$. Thus, $q \oplus e_0 = m_0$.
  - In the latter case, it holds $\texttt{H}((g^{y_1})^x) = \texttt{H}((g^{y_1})^{r_2+r_1}) = q$. Moreover, $e_1 = \texttt{H}(\beta_1^{y_1}) \oplus m_1 = \texttt{H}((g^{r_2+r_1})^{y_1}) \oplus m_1$. Hence, $q \oplus e_1 = m_1$.
- $(s = 0 \wedge s_1 = s_2 = 1)$ or $(s = s_2 = 1 \wedge s_1 = 0)$, then R can set $x = r_2 - r_1$.
  - In the former case, it holds $\texttt{H}((g^{y_0})^x) = \texttt{H}((g^{y_0})^{r_2-r_1}) = q$. On the other hand, $e_0 = \texttt{H}(\beta_0^{y_0}) \oplus m_0 = \texttt{H}((g^{r_2-r_1})^{y_0}) \oplus m_0$. Therefore, $q \oplus e_0 = m_0$.
  - In the latter case, it holds $\texttt{H}((g^{y_1})^x) = \texttt{H}((g^{y_1})^{r_2-r_1}) = q$. Also, $e_1 = \texttt{H}(\beta_1^{y_1}) \oplus m_1 = \texttt{H}((g^{r_2-r_1})^{y_1}) \oplus m_1$. Hence, $q \oplus e_1 = m_1$.

We conclude that DQ-OT always allows honest R to recover the message of its interest, i.e., m_s.

5. Delegated-Unknown-Query OT

In certain circumstances, the receiver itself does not know the value of query s. Instead, the query is issued by a third-party query issuer (T). An example in the context of banking is that customers R do not know whether they are premium ones or not.

In this section, we present a new variant of $\mathcal{DQ}\text{–}\mathcal{OT}_1^2$, called Delegated-Unknown-Query 1-out-of-2 OT ($\mathcal{DUQ}\text{–}\mathcal{OT}_1^2$). It enables T to issue the query while (a) preserving the security of $\mathcal{DQ}\text{–}\mathcal{OT}_1^2$ and (b) preserving the privacy of query s from R.

5.1. Security Definition

The functionality that $\mathcal{DUQ}\text{–}\mathcal{OT}_1^2$ computes takes as input (a) a pair of messages (m_0, m_1) from S, (b) empty strings ϵ from P_1, (c) ϵ from P_2, (d) ϵ from R, and (e) the index s (where $s \in \{0, 1\}$) from T. It outputs an empty string ϵ to S, T, P_1, and P_2, and outputs the message with index s, i.e., m_s, to R. More formally, we define the functionality as: $\mathcal{F}_{\mathcal{DUQ}\text{-}\mathcal{OT}_1^2} : \big((m_0, m_1), \epsilon, \epsilon, \epsilon, s\big) \rightarrow (\epsilon, \epsilon, \epsilon, \epsilon, m_s)$. Next, we present a formal definition of $\mathcal{DUQ}\text{–}\mathcal{OT}_1^2$.

Definition 3 ($\mathcal{DUQ}\text{–}\mathcal{OT}_1^2$)**.** Let $\mathcal{F}_{\mathcal{DUQ}\text{-}\mathcal{OT}_1^2}$ be the delegated-unknown-query OT functionality defined above. We say protocol Γ realises $\mathcal{F}_{\mathcal{DUQ}\text{-}\mathcal{OT}_1^2}$ in the presence of static passive adversary S, R, P_1, or P_2, if for every PPT adversary $\mathcal{A}$ in the real model, there exists a non-uniform PPT simulator $\texttt{Sim}$ in the ideal model, such that:

$$\Big\{\texttt{Sim}_S\big((m_0, m_1), \epsilon\big)\Big\}_{m_0, m_1, s} \overset{c}{\equiv} \overset{c}{\equiv} \Big\{\texttt{View}_S^{\Gamma}\big((m_0, m_1), \epsilon, \epsilon, \epsilon, s\big)\Big\}_{m_0, m_1, s} \quad (4)$$

$$\Big\{\texttt{Sim}_{P_i}(\epsilon, \epsilon)\Big\}_{m_0, m_1, s} \overset{c}{\equiv} \overset{c}{\equiv} \Big\{\texttt{View}_{P_i}^{\Gamma}\big((m_0, m_1), \epsilon, \epsilon, \epsilon, s\big)\Big\}_{m_0, m_1, s} \quad (5)$$

$$\Big\{\texttt{Sim}_T(s, \epsilon)\Big\}_{m_0, m_1, s} \overset{c}{\equiv} \overset{c}{\equiv} \Big\{\texttt{View}_T^{\Gamma}\big((m_0, m_1), \epsilon, \epsilon, \epsilon, s\big)\Big\}_{m_0, m_1, s} \quad (6)$$

$$\Big\{\texttt{Sim}_R\Big(\epsilon, \mathcal{F}_{\mathcal{DUQ}\text{-}\mathcal{OT}_1^2}\big((m_0, m_1), \epsilon, \epsilon, \epsilon, s\big)\Big)\Big\}_{m_0, m_1, s} \overset{c}{\equiv} \overset{c}{\equiv} \Big\{\texttt{View}_R^{\Gamma}\big((m_0, m_1), \epsilon, \epsilon, \epsilon, s\big)\Big\}_{m_0, m_1, s} \quad (7)$$

for all i, $i \in \{1, 2\}$.

5.2. Protocol

In this section, we present DUQ-OT that realises $\mathcal{DUQ\text{–}OT}^2_1$.

5.2.1. Main Challenge to Overcome. One of the primary differences between DUQ-OT and previous OT protocols in the literature (and DQ-OT) is that in DUQ-OT, R does not know the secret index s. The knowledge of s would help R pick the suitable element from S's response; for instance, in the DQ-OT protocol, it picks e_s from (e_0, e_1). Then, it can extract the message from the chosen element. In DUQ-OT, to enable R to extract the desirable message from S's response without the knowledge of s, we rely on the following observation and technique.

We know that, in any OT, after decrypting e_{s-1} R would get a value indistinguishable from a random value (otherwise, it would learn extra information about m_{s-1}). Therefore, if S imposes a certain publicly known structure to messages (m_0, m_1), then after decrypting S's response, only m_s would preserve the same structure.

In DUQ-OT, S imposes a publicly known structure to (m_0, m_1) and then computes the response. Given the response, R tries to decrypt *every* message it received from S and accepts only the result that has the structure.

5.2.2. An Overview. Briefly, DUQ-OT operates as follows. First, R picks two random values and sends each to a P_i. Also, T splits the secret index s into two shares and sends each share to a P_i. Also, T picks a random value r_3 and sends it to R and S. Given the messages received from R and T, each P_i generates queries the same way they do in DQ-OT.

Given the final query pair and r_3, S first appends r_3 to m_0 and m_1 and then computes the response the same way it does in DQ-OT, with the difference that it also randomly permutes the elements of the response pair. Given the response pair and r_3, R decrypts each element in the pair and accepts the result that contains r_3. Figure 2 presents DUQ-OT in more detail.

Theorem 2. *Let $\mathcal{F}_{\mathcal{DUQ\text{-}OT}^2_1}$ be the functionality defined in Section 5.1. If DL, CDH, and RO assumptions hold and random permutation $\pi(.)$ is secure, then DUQ-OT (presented in Figure 2) securely computes $\mathcal{F}_{\mathcal{DUQ\text{-}OT}^2_1}$ in the presence of (a) semi-honest receiver R, honest S, T, P_1, and P_2, (b) semi-honest T and honest R S, P_1, and P_2, (c) semi-honest S, honest T, R, P_1, and P_2, or (d) semi-honest P_i (where $i \in \{1, 2\}$), honest S, T, and R, w.r.t. Definition 3.*

We refer readers to Appendix A for the proof of Theorem 2.

1) *R-side Delegation:*
   a) pick two uniformly random values: $r_1, r_2 \overset{\$}{\leftarrow} \mathbb{Z}_{p-1}$.
   b) send r_1 to P_1 and r_2 to P_2.
2) *T-side Query Generation:*
   a) split the private index s into two shares (s_1, s_2) by calling $\mathtt{SS}(1^\lambda, s, 2, 2) \rightarrow (s_1, s_2)$.
   b) pick a uniformly random value: $r_3 \overset{\$}{\leftarrow} \{0,1\}^\lambda$.
   c) send (s_2, r_3) to R, s_1 to P_1, s_2 to P_2, and r_3 to S.
3) *P_2-side Query Generation:*
   a) compute a pair of partial queries:
   $$\delta_{s_2} = g^{r_2}, \quad \delta_{1-s_2} = \frac{C}{g^{r_2}}$$
   b) send (δ_0, δ_1) to P_1.
4) *P_1-side Query Generation:*
   a) compute a pair of final queries as:
   $$\beta_{s_1} = \delta_0 \cdot g^{r_1}, \quad \beta_{1-s_1} = \frac{\delta_1}{g^{r_1}}$$
   b) send (β_0, β_1) to S.
5) *S-side Response Generation:*
   a) abort if $C \neq \beta_0 \cdot \beta_1$.
   b) pick two uniformly random values: $y_0, y_1 \overset{\$}{\leftarrow} \mathbb{Z}_{p-1}$
   c) compute a response pair (e_0, e_1) as follows:
   $$e_0 := (e_{0,0}, e_{0,1}) = (g^{y_0}, \mathtt{G}(\beta_0^{y_0}) \oplus (m_0 || r_3))$$
   $$e_1 := (e_{1,0}, e_{1,1}) = (g^{y_1}, \mathtt{G}(\beta_1^{y_1}) \oplus (m_1 || r_3))$$
   d) randomly permute the elements of the pair (e_0, e_1) as follows: $\pi(e_0, e_1) \rightarrow (e'_0, e'_1)$.
   e) send (e'_0, e'_1) to R.
6) *R-side Message Extraction:*
   a) set $x = r_2 + r_1 \cdot (-1)^{s_2}$
   b) retrieve message m_s as follows. $\forall i, 0 \leq i \leq 1$:
      i) set $y = \mathtt{G}((e'_{i,0})^x) \oplus e'_{i,1}$.
      ii) call $\mathtt{parse}(\gamma, y) \rightarrow (a, b)$.
      iii) set $m_s = a$, if $b = r_3$.

Figure 2. DUQ-OT: Our 1-out-of-2 OT that supports query delegation while preserving the privacy of query from R. In the protocol, $C = g^a$ is a random public value, $\mathtt{SS}(.)$ is the share-generation algorithm (of a secret sharing), $\mathtt{G}(.)$ is a hash function, $\pi(.)$ is a random permutation, and \$ denotes picking a value uniformly at random.

6. Delegated-Query Oblivious Transfers with Hidden Fields

In this section, we present two new variants of $\mathcal{DQ}\text{–}\mathcal{OT}^2_1$; namely, (1) Delegated-Query OT with Hidden Fields ($\mathcal{DQ}^{\mathcal{HF}}\text{–}\mathcal{OT}^2_1$) and (2) Delegated-Unknown-Query OT with Hidden Fields ($\mathcal{DUQ}^{\mathcal{HF}}\text{–}\mathcal{OT}^2_1$). They are suitable for the *multi-receiver* setting in which the sender maintains a (large) database containing z pairs of messages $\boldsymbol{m} = [(m_{0,0}, m_{1,0}), ..., (m_{0,z-1}, m_{1,z-1})]$.

In this setting, each pair, say v-th pair $(m_{0,v}, m_{1,v}) \in \boldsymbol{m}$ is related to a receiver, R_j, where $1 \leq v \leq z$. Both variants (in addition to offering the security guarantee of $\mathcal{DQ}\text{–}\mathcal{OT}^2_1$) ensure that (i) a receiver learns nothing about the total number of receivers/pairs (i.e., z) and (ii) the sender learns nothing about which receiver is sending the query, i.e., a message pair's index for which a query was generated. In the remainder of this section, we discuss these new variants.

6.1. Delegated-Query OT with Hidden Fields

The first variant $\mathcal{DQ}^{\mathcal{HF}}\text{–}\mathcal{OT}^2_1$ considers the setting where server P_1 or P_2 knows a client's related pair's index in the sender's database.

6.1.1. Security Definition. The functionality that $\mathcal{DQ}^{\mathcal{HF}}\text{–}\mathcal{OT}^2_1$ computes takes as input (i) a vector of messages $\boldsymbol{m} = [(m_{0,0}, m_{1,0}), ..., (m_{0,z-1}, m_{1,z-1})]$ from S, (ii) an index v of a pair in $\boldsymbol{m}$ from P_1, (iii) empty string ϵ from P_2, and (iv) the index s (where $s \in \{0,1\}$) from R. It outputs an empty string ϵ to S, P_1, and P_2, and outputs to R s-th message from v-th pair in the vector, i.e., $m_{s,v}$. Formally, we define the functionality as: $\mathcal{F}_{\mathcal{DQ}^{\mathcal{HF}}\text{-}\mathcal{OT}^2_1} : \big([(m_{0,0}, m_{1,0}), ..., (m_{0,z-1}, m_{1,z-1})], v, \epsilon, s\big) \rightarrow (\epsilon, \epsilon, \epsilon, m_{s,v})$, where $v \in \{0, ..., z-1\}$. Next, we present a formal definition of $\mathcal{DQ}^{\mathcal{HF}}\text{–}\mathcal{OT}^2_1$.

Definition 4 ($\mathcal{DQ}^{\mathcal{HF}}\text{–}\mathcal{OT}^2_1$)**.** Let $\mathcal{F}_{\mathcal{DQ}^{\mathcal{HF}}\text{-}\mathcal{OT}^2_1}$ be the functionality defined above. We say protocol Γ realises $\mathcal{F}_{\mathcal{DQ}^{\mathcal{HF}}\text{-}\mathcal{OT}^2_1}$ in the presence of static passive adversary S, R, P_1, or P_2, if for every non-uniform PPT adversary $\mathcal{A}$ in the real model, there exists a non-uniform PPT simulator $\texttt{Sim}$ in the ideal model, such that:

$$\Big\{\texttt{Sim}_S\big(\boldsymbol{m}, \epsilon\big)\Big\}_{\boldsymbol{m},v,s} \stackrel{c}{\equiv} \Big\{\texttt{View}^{\Gamma}_S\big(\boldsymbol{m}, v, \epsilon, s\big)\Big\}_{\boldsymbol{m},v,s} \quad (8)$$

$$\Big\{\texttt{Sim}_{P_1}\big(v, \epsilon\big)\Big\}_{\boldsymbol{m},v,s} \stackrel{c}{\equiv} \Big\{\texttt{View}^{\Gamma}_{P_1}\big(\boldsymbol{m}, v, \epsilon, s\big)\Big\}_{\boldsymbol{m},v,s} \quad (9)$$

$$\Big\{\texttt{Sim}_{P_2}\big(\epsilon, \epsilon\big)\Big\}_{\boldsymbol{m},v,s} \stackrel{c}{\equiv} \Big\{\texttt{View}^{\Gamma}_{P_2}\big(\boldsymbol{m}, v, \epsilon, s\big)\Big\}_{\boldsymbol{m},v,s} \quad (10)$$

$$\Big\{\texttt{Sim}_R\Big(s, \mathcal{F}_{\mathcal{DQ}^{\mathcal{HF}}\text{-}\mathcal{OT}^2_1}\big(\boldsymbol{m}, v, \epsilon, s\big)\Big)\Big\}_{\boldsymbol{m},v,s} \stackrel{c}{\equiv} \stackrel{c}{\equiv} \Big\{\texttt{View}^{\Gamma}_R\big(\boldsymbol{m}, v, \epsilon, s\big)\Big\}_{\boldsymbol{m},v,s} \quad (11)$$

where $\boldsymbol{m} = [(m_{0,0}, m_{1,0}), ..., (m_{0,z-1}, m_{1,z-1})]$.

6.1.2. Strwaman Approaches. One may consider using an existing single-receiver OT in the multi-receiver setting, employing one of the following approaches:

Approach 1: R_j sends a standard OT query to S which computes the response for all z pairs of messages. Subsequently, S sends z pair of responses to R_j which discards all pairs from the response except for v-th pair. R_j extracts its message m_v from the selected pair, similar to a regular 1-out-of-2 OT. However, it is important to note that Approach 1 results in the leakage of the entire database size to R_j.

Approach 2: R_j sends a standard OT query to S, along with the index v of its record. This can be perceived as if S holds a single record/pair. Accordingly, S generates a response in the same manner as it does in regular 1-out-of-2 OT. Nevertheless, Approach 2 also leaks the index of the record that R_j is interested to S.

6.1.3. Protocol. Next, we present protocol DQ$^{\text{HF}}$–OT that realises $\mathcal{DQ}^{\mathcal{HF}}\text{–}\mathcal{OT}^2_1$. We build DQ$^{\text{HF}}$–OT upon protocol DQ-OT (presented in Figure 1). DQ$^{\text{HF}}$–OT relies on our observation that in DQ-OT, given the response of S, P_1 cannot learn anything, e.g., about the plaintext messages m_i of S. Below, we formally state it.

Lemma 1. *Let g be a generator of a group $\mathbb{G}$ (defined in Section 2.4) whose order is a prime number p and $\log_2(p) = \lambda$ is a security parameter. Also, let (r_1, r_2, y_1, y_2) be elements of $\mathbb{G}$ picked uniformly at random, $C = g^a$ be a random public value whose discrete logarithm is unknown, (m_0, m_1) be two arbitrary messages, and $\texttt{H}$ be a hash function modelled as a RO (as defined in Section 2.1). Let $\gamma = \overset{+}{-} r_1 \overset{+}{-} r_2$, $\beta_0 = g^{a+\gamma}$, and $\beta_1 = g^{a-\gamma}$. If DL, RO, and CDH assumptions hold, then given r_1, C, g^{r_2}, and $\frac{C}{g^{r_2}}$, a PPT distinguisher cannot distinguish the elements of pairs $(g^{y_0}, \texttt{H}(\beta_0^{y_0}) \oplus m_0)$ and $(g^{y_1}, \texttt{H}(\beta_1^{y_1}) \oplus m_1)$ from random elements of $\mathbb{G}$, except for a negligible probability, $\mu(\lambda)$.*

Appendix B presents the proof of Lemma 1. The main idea behind the design of DQ$^{\text{HF}}$–OT is as follows. Given a message pair from P_1, S needs to compute the response for all of the receivers and sends the result to P_1, which picks and sends only one pair in the response to the specific receiver who sent the query and discards the rest of the pairs it received from S. Therefore, R receives a single pair (so it cannot learn the total number of receivers or the database size), and the server cannot know which receiver sent the query as it generates the response for all of them. As we will prove, P_1 itself cannot learn the actual query of R, too.

Specifically, consider the case where one of the receivers, say R_j wants to send a query. In this case, in DQ$^{\text{HF}}$–OT, messages (s_1, r_1), (s_2, r_2) and (β_0, β_1) are generated the same way as they are computed in DQ-OT. However, given (β_0, β_1), S generates z pairs and sends them to P_1 who forwards only v-th pair to R_j and discards the rest. Given the pair, R_j computes the result the same way a receiver does in DQ-OT. Figure 7 in Appendix C presents DQ$^{\text{HF}}$–OT in detail.

6.2. Delegated-Unknown-Query OT with Hidden Fields

The second variant $\mathcal{DUQ}^{\mathcal{HF}}\text{–}\mathcal{OT}_1^2$ can be considered as a variant of $\mathcal{DUQ}\text{–}\mathcal{OT}_1^2$. It is suitable for the setting where servers P_1 and P_2 do not (and must not) know a client's related index in the sender's database (as well as the index s of the message that the client is interested in).

6.2.1. Security Definition. The functionality that $\mathcal{DUQ}^{\mathcal{HF}}\text{–}\mathcal{OT}_1^2$ computes takes as input (i) a vector of messages $\boldsymbol{m} = [(m_{0,0}, m_{1,0}), ..., (m_{0,z-1}, m_{1,z-1})]$ from S, (ii) an index v of a pair in $\boldsymbol{m}$ from T, (iii) the index s of a message in a pair (where $s \in \{0,1\}$) from T, (iv) the total number z of message pairs from T, (v) empty string ϵ from P_1, (vi) ϵ from P_2, and (vii) ϵ from R. It outputs an empty string ϵ to S, P_1, P_2, and T, and outputs to R s-th message from v-th pair in $\boldsymbol{m}$, i.e., $m_{s,v}$. Formally, we define the functionality as: $\mathcal{F}_{\mathcal{DUQ}^{\mathcal{HF}}\text{–}\mathcal{OT}_1^2} : \big([(m_{0,0}, m_{1,0}), ..., (m_{0,z-1}, m_{1,z-1})], (v, s, z), \epsilon, \epsilon, \epsilon\big) \rightarrow (\epsilon, \epsilon, \epsilon, \epsilon, m_{s,v})$, where $v \in \{0, ..., z-1\}$. Next, we present a formal definition of $\mathcal{DUQ}^{\mathcal{HF}}\text{–}\mathcal{OT}_1^2$.

Definition 5 ($\mathcal{DUQ}^{\mathcal{HF}}\text{–}\mathcal{OT}_1^2$)**.** Let $\mathcal{F}_{\mathcal{DUQ}^{\mathcal{HF}}\text{–}\mathcal{OT}_1^2}$ be the functionality defined above. We say protocol Γ realises $\mathcal{F}_{\mathcal{DUQ}^{\mathcal{HF}}\text{–}\mathcal{OT}_1^2}$ in the presence of static passive adversary S, R, T, P_1, or P_2, if for every non-uniform PPT adversary $\mathcal{A}$ in the real model, there exists a non-uniform PPT simulator $\mathtt{Sim}$ in the ideal model, such that:

$$\Big\{\mathtt{Sim}_S\big(\boldsymbol{m}, \epsilon\big)\Big\}_{\boldsymbol{m},s} \overset{c}{\equiv} \overset{c}{\equiv} \Big\{\mathtt{View}_S^{\Gamma}\big(\boldsymbol{m}, (v,s,z), \epsilon, \epsilon, \epsilon\big)\Big\}_{\boldsymbol{m},s} \quad (12)$$

$$\Big\{\mathtt{Sim}_{P_i}\big(\epsilon, \epsilon\big)\Big\}_{\boldsymbol{m},s} \overset{c}{\equiv} \overset{c}{\equiv} \Big\{\mathtt{View}_{P_i}^{\Gamma}\big(\boldsymbol{m}, (v,s,z), \epsilon, \epsilon, \epsilon\big)\Big\}_{\boldsymbol{m},s} \quad (13)$$

$$\Big\{\mathtt{Sim}_T\big((v,s,z), \epsilon\big)\Big\}_{\boldsymbol{m},s} \overset{c}{\equiv} \overset{c}{\equiv} \Big\{\mathtt{View}_T^{\Gamma}\big(\boldsymbol{m}, (v,s,z), \epsilon, \epsilon, \epsilon\big)\Big\}_{\boldsymbol{m},s} \quad (14)$$

$$\Big\{\mathtt{Sim}_R\Big(\epsilon, \mathcal{F}_{\mathcal{DUQ}^{\mathcal{HF}}\text{–}\mathcal{OT}_1^2}\big(\boldsymbol{m}, (v,s,z), \epsilon, \epsilon, \epsilon\big)\Big)\Big\}_{\boldsymbol{m},s} \overset{c}{\equiv} \overset{c}{\equiv} \Big\{\mathtt{View}_R^{\Gamma}\big(\boldsymbol{m}, (v,s,z), \epsilon, \epsilon, \epsilon\big)\Big\}_{\boldsymbol{m},s} \quad (15)$$

where $\boldsymbol{m} = [(m_{0,0}, m_{1,0}), ..., (m_{0,z-1}, m_{1,z-1})]$ and $\forall i, i \in \{1,2\}$.

6.2.2. Protocol. Now, we present protocol DUQ$^{\text{HF}}$–OT that realises $\mathcal{DUQ}^{\mathcal{HF}}\text{–}\mathcal{OT}_1^2$. We build DQ$^{\text{HF}}$–OT upon protocol DUQ-OT (presented in Figure 2). DUQ$^{\text{HF}}$–OT mainly relies on Lemma 1 and the following technique.

To fetch a record m_v "securely" from a semi-honest S that holds a database of the form $\boldsymbol{a} = [m_0, m_1, ..., m_{z-1}]^T$ where T denotes transpose, without revealing which plaintext record we want to fetch, we can perform as follows:

i construct vector $\boldsymbol{b} = [b_0, ..., b_{z-1}]$, where all b_is are set to zero except for v-th element b_v which is set to 1.
ii encrypt each element of $\boldsymbol{b}$ using additively homomorphic encryption, e.g., Paillier encryption. Let $\boldsymbol{b}'$ be the vector of the encrypted elements.
iii send $\boldsymbol{b}'$ to the database holder which performs $\boldsymbol{b}' \times \boldsymbol{a}$ homomorphically, and sends us the single result res.
iv decrypt res to discover m_v.[3]

In our DUQ$^{\text{HF}}$–OT, $\boldsymbol{b}'$ is not sent for each query to S. Instead, $\boldsymbol{b}'$ is stored once in one of the servers, for example, P_1. Any time S computes a vector of responses, say $\boldsymbol{a}$, to an OT query, it sends $\boldsymbol{a}$ to P_1 which computes $\boldsymbol{b}' \times \boldsymbol{a}$ homomorphically and sends the result to R. Subsequently, R can decrypt it and find the message it was interested. Thus, P_1 *obliviously filters out* all other records of field elements that do not belong to R_j and sends to R_j only the messages that R_j is allowed to fetch. Figures 3 and 4 present DUQ$^{\text{HF}}$–OT in detail.

1) *R_j-side One-off Setup:*
   a) generate a key pair for additive homomorphic encryption, by calling $\mathtt{KGen}(1^\lambda) \rightarrow (sk, pk)$.
   b) send pk to T and S.
2) *T-side One-off Setup:*
   a) initialise an empty vector $\boldsymbol{w}_j = []$ of size z.
   b) create a compressing vector, by setting v-th position of $\boldsymbol{w}_j$ to encrypted 1 and setting the rest of $z-1$ positions to encrypted 0. $\forall t, 0 \leq t \leq z-1$:
      i) set $d = 1$, if $t = v$; set $d = 0$, otherwise.
      ii) append $\mathtt{Enc}(pk, d)$ to $\boldsymbol{w}_j$.
   c) send $\boldsymbol{w}_j$ to P_1.
3) *R_j-side Delegation:*
   a) pick random values: $r_1, r_2 \overset{\$}{\leftarrow} \mathbb{Z}_{p-1}$.
   b) send r_1 to P_1 and r_2 to P_2.
4) *T-side Query Generation:*
   a) split the private index s into two shares (s_1, s_2) by calling $\mathtt{SS}(1^\lambda, s, 2, 2) \rightarrow (s_1, s_2)$.
   b) pick a uniformly random value: $r_3 \overset{\$}{\leftarrow} \{0,1\}^\lambda$.
   c) send (s_2, r_3) to R_j, s_1 to P_1, s_2 to P_2, r_3 to S.
5) *P_2-side Query Generation:*
   a) compute queries: $\delta_{s_2} = g^{r_2}, \;\; \delta_{1-s_2} = \frac{C}{g^{r_2}}$.
   b) send (δ_0, δ_1) to P_1.
6) *P_1-side Query Generation:*
   a) compute final queries as: $\beta_{s_1} = \delta_0 \cdot g^{r_1}, \;\; \beta_{1-s_1} = \frac{\delta_1}{g^{r_1}}$.
   b) send (β_0, β_1) to S.

Figure 3. Phases 1–6 of DUQ$^{\text{HF}}$–OT.

3. Such a technique was previously used by Devet *et al.* [43] in the "private information retrieval" research line.

```
7) S-side Response Generation:
   a) abort if $C \neq \beta_0 \cdot \beta_1$.
   b) compute a response as follows. $\forall t, 0 \leq t \leq z-1$ :
      i) pick two random values $y_{0,t}, y_{1,t} \xleftarrow{\$} \mathbb{Z}_{p-1}$.
      ii) compute response:
          $e_{0,t} := (e_{0,0,t}, e_{0,1,t}) = (g^{y_{0,t}}, \mathtt{G}(\beta_0^{y_{0,t}}) \oplus (m_{0,t}||r_3))$
          $e_{1,t} := (e_{1,0,t}, e_{1,1,t}) = (g^{y_{1,t}}, \mathtt{G}(\beta_1^{y_{1,t}}) \oplus (m_{1,t}||r_3))$
      iii) randomly permute the elements of each pair $(e_{0,t}, e_{1,t})$ as $\pi(e_{0,t}, e_{1,t}) \rightarrow (e'_{0,t}, e'_{1,t})$.
   c) send $(e'_{0,0}, e'_{1,0}), ..., (e'_{0,z-1}, e'_{1,z-1})$ to $P_1$.
8) $P_1$-side Oblivious Filtering:
   a) compresses the $S$'s response using vector $\boldsymbol{w}_j$ as follows. $\forall i, i', 0 \leq i, i' \leq 1$ :
      $o_{i,i'} = (e'_{i,i',0} \overset{H}{\times} \boldsymbol{w}_j[0]) \overset{H}{+} ... \overset{H}{+} (e'_{i,i',z-1} \overset{H}{\times} \boldsymbol{w}_j[z-1])$
   b) send $(o_{0,0}, o_{0,1}), (o_{1,0}, o_{1,1})$ to $R_j$
9) R-side Message Extraction:
   a) decrypt the response from $P_1$ as follows:
      $\forall i, i', 0 \leq i, i' \leq 1 : \mathtt{Dec}(sk, o_{i,i'}) \rightarrow o'_{i,i'}$.
   b) set $x = r_2 + r_1 \cdot (-1)^{s_2}$.
   c) retrieve message $m_{s,v}$ as follows. $\forall i, 0 \leq i \leq 1$ :
      i) set $y = \mathtt{G}((o'_{i,0})^x) \oplus o'_{i,1}$.
      ii) call $\mathtt{parse}(\gamma, y) \rightarrow (a, b)$.
      iii) set $m_{s,v} = a$, if $b = r_3$.
```

Figure 4. Phases 7–9 of DUQ$^{\text{HF}}$–OT.

In both DQ$^{\text{HF}}$–OT and DUQ$^{\text{HF}}$–OT, S sends to P_1 a number of messages linear with the database size.

Theorem 3. *Let $\mathcal{F}_{\mathcal{DUQ}^{\mathcal{HF}}\text{-}\mathcal{OT}^2_1}$ be the functionality defined in Section 6.2.1. If DL, CDH, and RO assumptions hold and additive homomorphic encryption satisfies IND-CPA property, then DUQ$^{\text{HF}}$–OT (presented in Figures 3 and 4) securely computes $\mathcal{F}_{\mathcal{DUQ}^{\mathcal{HF}}\text{-}\mathcal{OT}^2_1}$ in the presence of (a) semi-honest receiver R, honest S, T, P_1, and P_2, (b) semi-honest S, honest R, T, P_1, and P_2, (c) semi-honest T and honest S, P_i, and R, or (d) semi-honest P_i (where $i \in \{1, 2\}$), honest S, T, and R, w.r.t. Definition 5.*

We refer readers to Appendix D for Theorem 3's proof.

7. A Compiler for Generic OT with Constant Size Response

In this section, we present a compiler that transforms *any* 1-out-of-n OT that requires R to receive n messages (as a response) into a 1-out-of-n OT that enables R to receive only a *constant* number of messages.

The main technique we rely on is the encrypted binary vector that we used in Section 6.2. The high-level idea is as follows. During query computation, R (along with its vector that encodes its index $s \in \{0, n-1\}$) computes a binary vector of size n, where all elements of the vector are set to 0 except for s-th element which is set to 1. R encrypts each element of the vector and sends the result as well as its query to S. Subsequently, S computes a response vector (the same manner it does in regular OT), homomorphically multiplies each element of the response by the element of the encrypted vector (component-wise), and then homomorphically sums all the product. It sends the result (which is now constant with regard to n) to R. Accordingly, R decrypts the response and retrieves the result m_s.

In the remainder of this section, we first present a syntax of generic OT (in Section 7.1). Then, we present the generic compiler (in Section 7.2) using this syntax.

7.1. Syntax of an OT

Since we would like to treat any OT in a block-box manner, we first present a syntax of an OT. Let a classical (or non-delegated) 1-out-of-n OT ($\mathcal{OT}^n_1$) have the following algorithms:

- $\mathtt{Setup}(1^\lambda) \rightarrow (sk, pk)$: a probabilistic algorithm run by R. It takes as input security parameter 1^λ and returns a pair of private and public keys (sk, pk).
- $\mathtt{GenQuery}(sk, pk, s) \rightarrow q$: a probabilistic algorithm run by R. It takes as input sk, pk, and an index s. It returns a query (vector) q.
- $\mathtt{GenRes}(pk, q) \rightarrow res$: a probabilistic algorithm run by S. It takes as input pk and q. It generates an encoded response (vector) res.
- $\mathtt{Retreive}(res, q, sk, pk, s) \rightarrow m_s$: a deterministic algorithm run by S. It takes as input res, q, sk, pk, and s. It returns message m_s.

The functionality that a 1-out-of-n OT computes can be defined as: $\mathcal{F}_{\mathcal{OT}^n_1} : \big((m_0, ..., m_{n-1}), s\big) \rightarrow (\epsilon, m_s)$. In formally, the security of 1-out-of-n OT states that (1) R's view can be simulated given its input query s and output message m_s and (2) S's view can be simulated given its input messages $(m_0, ..., m_{n-1})$. We refer readers to [4] for further discussion on 1-out-of-n OT.

7.2. The Compiler

We present the compiler in detail in Figure 5. We highlight that in the case where each $e_i \in res$ contains more than one value, e.g., $e_i = [e_{0,i}, ..., e_{w-1,i}]$ (due to a certain protocol design), then each element of e_i is separately multiplied and added by the element of vector $\boldsymbol{b}'$, e.g., the j-st element of the response is $e_{j,0} \overset{H}{\times} \boldsymbol{b}'[0] \overset{H}{+} ... \overset{H}{+} e_{j,n-1} \overset{H}{\times} \boldsymbol{b}'[n-1]$, for all j, $0 \leq j \leq w-1$. In this case, only w elements are sent to R.

Theorem 4. *Let $\mathcal{F}_{\mathcal{OT}^n_1}$ be the functionality defined above. If $\mathcal{OT}^n_1$ is secure and additive homomorphic encryption satisfies IND-CPA property, then generic OT with constant size response (presented in Figure 5) (i) securely computes $\mathcal{F}_{\mathcal{OT}^n_1}$ in the presence of semi-honest receiver R or semi-honest S and (ii) offers $O(1)$ response size, w.r.t. the total number of messages n.*

Appendix E presents the proof of Theorem 4.

1) <u>Setup.</u>
   This phase involves R.
   a) calls $\texttt{Setup}(1^{\lambda}) \rightarrow (sk, pk)$.
   b) publishes pk.
2) <u>Query Geenration.</u>
   This phase involves R.
   a) calls $\texttt{GenQuery}(sk, pk, s) \rightarrow q$.
   b) construct a vector $\boldsymbol{b} = [b_0, ..., b_{n-1}]$, as:
      i) sets every element b_i to zero except for s-th element b_s which is set to 1.
      ii) encrypts each element of $\boldsymbol{b}$ using additive homomorphic encryption. Let $\boldsymbol{b}'$ be the vector of the encrypted elements.
   c) sends q and $\boldsymbol{b}'$ to S.
3) <u>Generate Response.</u>
   This phase involves S.
   a) calls $\texttt{GenRes}(pk, q) \rightarrow res$. Let $res = [e_0, ..., e_{n-1}]$.
   b) compresses its response using vector $\boldsymbol{b}'$ as follows. $\forall i, 0 \leq i \leq n-1$:
   $$e = (e_0 \overset{H}{\times} \boldsymbol{b}'[0]) \overset{H}{+} ... \overset{H}{+} (e_{n-1} \overset{H}{\times} \boldsymbol{b}'[n-1])$$
   c) send e to R.
4) <u>Retreive.</u>
   This phase involves R.
   - call $\texttt{Retreive}(e, q, sk, pk, s) \rightarrow m_s$.

Figure 5. A compiler that turns an 1-out-of-n OT with response size $O(n)$ to a 1-out-of-n OT with response size $O(1)$.

8. Supersonic OT

In this section, we propose a 1-out-of-2 OT, called "Supersonic OT", that (i) is highly fast as it does not require any public-key-based cryptography, (ii) has $O(1)$ size response that R receives, and (iii) is information-theoretic secure; thus, it is post-quantum secure too.

8.1. Security Definition

Supersonic OT involves three types of entities, a sender S, a receiver R, and a server P. We assume each party can be corrupted by a static passive non-colluding adversary. The functionality $\mathcal{F}_{\mathcal{OT}_1^2}$ that Supersonic OT will compute is similar to classical OT with the difference that now an additional party P is introduced, having no input and receiving no output. Thus, we define the functionality as $\mathcal{F}_{\mathcal{OT}_1^2} : \big((m_0, m_1), \epsilon, s\big) \rightarrow (\epsilon, \epsilon, m_s)$. Next, we present a formal definition of $\mathcal{OT}_1^2$.

Definition 6 ($\mathcal{OT}_1^2$)**.** Let $\mathcal{F}_{\mathcal{OT}_1^2}$ be the OT functionality defined above. We say protocol Γ realises $\mathcal{F}_{\mathcal{OT}_1^2}$ in the presence of static passive adversary S, R, or P, if for every non-uniform PPT adversary $\mathcal{A}$ in the real model, there is a non-uniform PPT simulator $\texttt{Sim}$ in the ideal model, where:

$$\Big\{\texttt{Sim}_S\big((m_0, m_1), \epsilon\big)\Big\}_{m_0, m_1, s} \overset{c}{\equiv} \\ \overset{c}{\equiv} \Big\{\texttt{View}_S^{\Gamma}\big((m_0, m_1), \epsilon, s\big)\Big\}_{m_0, m_1, s} \tag{16}$$

$$\Big\{\texttt{Sim}_P(\epsilon, \epsilon)\Big\}_{m_0, m_1, s} \overset{c}{\equiv} \\ \overset{c}{\equiv} \Big\{\texttt{View}_P^{\Gamma}\big((m_0, m_1), \epsilon, s\big)\Big\}_{m_0, m_1, s} \tag{17}$$

$$\Big\{\texttt{Sim}_R\Big(s, \mathcal{F}_{\mathcal{OT}_1^2}\big((m_0, m_1), \epsilon, s\big)\Big)\Big\}_{m_0, m_1, s} \overset{c}{\equiv} \\ \overset{c}{\equiv} \Big\{\texttt{View}_R^{\Gamma}\big((m_0, m_1), \epsilon, s\big)\Big\}_{m_0, m_1, s} \tag{18}$$

8.2. Customised Random Swap

Before presenting Supersonic OT, we introduce the permutation $\overline{\pi}(.)$, which will be used in the protocol. $\overline{\pi}(.)$ takes two inputs: a binary value s and a pair (c_0, c_1). When $s = 0$, then it returns the input pair (c_0, c_1), i.e., it does not swap the elements. However, when $s = 1$, then it returns (c_1, c_0), effectively swapping the elements. It is evident that if s is uniformly chosen at random, then $\overline{\pi}(.)$ presents a random permutation, implying that the probability of swapping or not swapping is $\frac{1}{2}$.

8.3. The Protocol

Figure 6 presents Supersonic OT in detail.

1) <u>*Key Agreement:*</u>
   - R picks two random keys $(k_0, k_1) \overset{\$}{\leftarrow} \{0, 1\}^{\sigma}$ and sends them to S.
2) <u>*R-side Query Generation:*</u>
   a) splits the private index s into two shares (s_1, s_2) by calling $\texttt{SS}(1^{\lambda}, s, 2, 2) \rightarrow (s_1, s_2)$.
   b) sends s_1 to S and s_2 to P.
3) <u>*S-side Response Generation:*</u>
   a) encrypts each message as follows.
   $$\forall i, 0 \leq i \leq 1 : m'_i = m_i \oplus k_i$$
   Let $e = (m'_0, m'_1)$ contain the encrypted messages.
   b) permutes the elements of e as follows: $\overline{\pi}(s_1, e) \rightarrow e'$.
   c) sends e' to P.
4) <u>*P-side Oblivious Filtering:*</u>
   a) permutes the elements of e' as: $\overline{\pi}(s_2, e') \rightarrow e''$.
   b) sends (always) the first element in e'', say e''_0, to R and discards the second element in e''.
5) <u>*R-side Message Extraction:*</u>
   - retrieves the final related message m_s by decrypting e''_0 as: $m_s = e''_0 \oplus k_s$.

Figure 6. Supersonic OT.

At a high level, the protocol works as follows. Initially, R and S agree on a pair of keys. In the query generation phase, R splits its private index into two binary shares. It sends one share to S and the other to P. Given the share/query, S encrypts every message m_i (using a one-time pad) under one of the keys it initially agreed with R. Then, S permutes the encrypted messages using $\bar{\pi}$ and its share. It sends the resulting pair to P. Accordingly, P permutes the received pair using $\bar{\pi}$ and its share. P sends only the first element of the resulting pair (which is a ciphertext) to R and discards the second element of the pair. Subsequently, R decrypts the ciphertext and learns the message it was interested in.

Theorem 5. *Let $\mathcal{F}_{\mathcal{OT}_1^2}$ be the functionality defined in Section 8.1. Then, Supersonic OT (presented in Figure 6) securely computes $\mathcal{F}_{\mathcal{OT}_1^2}$ in the presence of (a) semi-honest receiver R, honest S, and P, (b) semi-honest S, honest R and P, or (c) semi-honest P and honest S and R, w.r.t. Definition 6.*

Appendix F presents the proof of Theorem 5.

8.4. Proof of Correctness

In this section, we demonstrate that R always receives the message m_s corresponding to its query s. To accomplish this, we will show that (in step 4b) the first element of pair e'' always equals the encryption of m_s. This outcome is guaranteed by the following two facts: (a) $s = s_1 \oplus s_2$ and (b) S and T permute their pairs based on the value of their share, i.e., s_1 and s_2 respectively.

s	s_1	s_2
0	1	1
	0	0
1	1	0
	0	1

TABLE 2. Relation between query s and behaviour of permutation $\bar{\pi}$ from the perspective of S and P. When $s_i = 1$, $\bar{\pi}$ swaps the elements of its input pairs and when $s_i = 0$, $\bar{\pi}$ does not swap the elements of the input pairs.

As table 2 indicates, when $s = 0$, then (i) either both S and P permute their pairs or (ii) neither does. In the former case, since both swap the elements of their pair, then the final permuted pair e'' will have the same order as the original pair e (before it was permuted). In the latter case, again e'' will have the same order as the original pair e because neither party has permuted it. Thus, in both of the above cases (when $s = 0$), the first element of e'' will be the encryption of m_0. Moreover, as Table 2 shows, when $s = 1$, then only one of the parties S and P will permute their input pair. This means that the first element of the final permuted pair e'' will always equal the encryption of m_1.

8.5. Comparison

To date, the most efficient OTs belong to the class of OT extension, such as the ones described in [18]–[21]. The protocols in this class are well-suited and efficient for cases where OTs need to be invoked multiple times. To achieve efficiency, these protocols require both the sender and receiver to execute a standard public-key-based OT several times (i.e., linearly with the security parameter k, where $k \geq 128$) to generate the system parameters. Subsequently, the sender and receiver can use these parameters to engage in an OT that involves only symmetrical key operations. For these protocols to be unconditionally secure, they must employ an unconditionally secure standard OT during the setup. However, as discussed in Section 3.4, the existing unconditionally secure OTs rely on multiple replicas of the database, a noisy channel, or the involvement of a trusted initialiser, all of which increase deployment costs.

However, Supersonic OT does not require parties to execute a public-key-based OT at any phase. As a result, it imposes significantly lower computation and communication costs compared to OT extensions. Unlike OT extensions, Supersonic OT maintains its efficiency regardless of the number of times it is executed. Furthermore, Supersonic OT achieves unconditional security without relying on database replications, noisy channels, or a fully trusted party. Nevertheless, unlike OT extensions that involve only the sender and receiver, Supersonic OT utilises the assistance of an additional party. However, it still maintains its security even if this party is semi-honest.

9. Conclusions

OT is a crucial protocol in the field of cryptography. OTs have found extensive applications in designing secure multi-party computation protocols and in accessing sensitive field elements of remote private databases while preserving privacy. In this work, we have identified several research gaps in the OT research line and proposed several novel OTs to address these gaps. Specifically, we have proposed the following:

i. Delegated-Query OT (and its variant), which enables the receiver to delegate query computation to potentially semi-honest parties while ensuring the privacy of both the sender and receiver.
ii. Delegated-Query OT with Hidden Field (and its variant), a multi-receiver OT that ensures each receiver remains unaware of the total number of records and their field elements while preventing the sender from learning which query corresponds to which record.
iii. a compiler that transforms any 1-out-of-n OT that requires the receiver to receive n messages into a 1-out-of-n OT enabling a receiver to receive only a constant number of messages and have constant storage space.
iv. Supersonic OT, a 1-out-of-2 OT that does not rely on database replications, noisy channels, or the involvement of a trusted initialiser. Supersonic OT is highly efficient. It enables the receiver to receive a constant-size response and offers post-quantum security.

As a future research direction, exploring how the exceptional efficiency of Supersonic OT can enhance the performance of protocols (such as generic MPC or Private Set Intersection) that heavily rely on OTs would be intriguing.

References

[1] C. P. Pfleeger, *Security in computing*. Prentice-Hall, Inc., 1988.

[2] R. Canetti, "Towards realizing random oracles: Hash functions that hide all partial information," *IACR Cryptol. ePrint Arch.*, 1997.

[3] S. Goldwasser, S. Micali, and C. Rackoff, "The knowledge complexity of interactive proof-systems (extended abstract)," in *Proceedings of the 17th Annual ACM Symposium on Theory of Computing, May 6-8, 1985, Providence, Rhode Island, USA*. ACM, 1985.

[4] O. Goldreich, *The Foundations of Cryptography - Volume 2, Basic Applications*. Cambridge University Press, 2004.

[5] D. E. Knuth, *The Art of Computer Programming, Volume II: Seminumerical Algorithms, 2nd Edition*. Addison-Wesley, 1981.

[6] W. Diffie and M. E. Hellman, "New directions in cryptography," *IEEE Trans. Inf. Theory*, 1976.

[7] G. R. Blakley, "One time pads are key safegaurding schemes, not cryptosystems. fast key safeguarding schemes (threshold schemes) exist." in *1980 IEEE Symposium on Security and Privacy*, 1980.

[8] J. Katz and Y. Lindell, *Introduction to Modern Cryptography, Second Edition*. CRC Press, 2014.

[9] P. Paillier, "Public-key cryptosystems based on composite degree residuosity classes," in *EUROCRYPT*, ser. Lecture Notes in Computer Science, 1999.

[10] Michael O. Rabin, "How to exchange secrets with oblivious transfer," 1981.

[11] S. Even, O. Goldreich, and A. Lempel, "A randomized protocol for signing contracts," *Commun. ACM*, 1985.

[12] M. Naor and B. Pinkas, "Oblivious transfer and polynomial evaluation," in *STOC*, 1999.

[13] W. Tzeng, "Efficient 1-out-n oblivious transfer schemes," in *Public Key Cryptography, 5th International Workshop on Practice and Theory in Public Key Cryptosystems, PKC 2002, Paris, France, February 12-14, 2002, Proceedings*, ser. Lecture Notes in Computer Science, D. Naccache and P. Paillier, Eds., 2002.

[14] M. Liu and Y. Hu, "Universally composable oblivious transfer from ideal lattice," *Frontiers Comput. Sci.*, 2019.

[15] C. Chu and W. Tzeng, "Efficient k-out-of-n oblivious transfer schemes with adaptive and non-adaptive queries," in *PKC*, 2005.

[16] S. Jarecki and X. Liu, "Efficient oblivious pseudorandom function with applications to adaptive OT and secure computation of set intersection," in *TCC*, 2009.

[17] Y. Chen, J. Chou, and X. Hou, "A novel k-out-of-n oblivious transfer protocols based on bilinear pairings," *IACR Cryptol. ePrint Arch.*, 2010.

[18] Y. Ishai, J. Kilian, K. Nissim, and E. Petrank, "Extending oblivious transfers efficiently," in *CRYPTO, 2003, Proceedings*, 2003.

[19] W. Henecka and T. Schneider, "Faster secure two-party computation with less memory," in *CCS*, 2013.

[20] J. B. Nielsen, "Extending oblivious transfers efficiently - how to get robustness almost for free," *IACR Cryptol. ePrint Arch.*, 2007.

[21] G. Asharov, Y. Lindell, T. Schneider, and M. Zohner, "More efficient oblivious transfer and extensions for faster secure computation," in *CCS'13*, 2013.

[22] M. Naor and B. Pinkas, "Distributed oblivious transfer," in *Advances in Cryptology - ASIACRYPT 2000, 6th International Conference on the Theory and Application of Cryptology and Information Security, Kyoto, Japan, December 3-7, 2000, Proceedings*. Springer, 2000.

[23] C. L. F. Corniaux and H. Ghodosi, "A verifiable 1-out-of-n distributed oblivious transfer protocol," *IACR Cryptol. ePrint Arch.*, 2013.

[24] S. Zhao, X. Song, H. Jiang, M. Ma, Z. Zheng, and Q. Xu, "An efficient outsourced oblivious transfer extension protocol and its applications," *Secur. Commun. Networks*, vol. 2020, 2020.

[25] G. Yao and D. Feng, "Proxy oblivious transfer protocol," in *International Conference on Availability, Reliability and Security, ARES*, 2006.

[26] J. Camenisch, M. Dubovitskaya, and G. Neven, "Oblivious transfer with access control," in *Proceedings of the 2009 ACM Conference on Computer and Communications Security, CCS*, 2009.

[27] J. Camenisch, M. Dubovitskaya, G. Neven, and G. M. Zaverucha, "Oblivious transfer with hidden access control policies," in *PKC*, 2011.

[28] J. Camenisch, M. Dubovitskaya, R. R. Enderlein, and G. Neven, "Oblivious transfer with hidden access control from attribute-based encryption," in *SCN*, 2012.

[29] J. Camenisch, M. Dubovitskaya, and G. Neven, "Unlinkable priced oblivious transfer with rechargeable wallets," in *FC*, 2010.

[30] W. Aiello, Y. Ishai, and O. Reingold, "Priced oblivious transfer: How to sell digital goods," in *EUROCRYPT*, 2001.

[31] S. Wang, Y. Tsai, and C. Shen, "Varied oblivious transfer protocols enabling multi-receiver and applications," in *BWCCA*, 2010.

[32] J. Camenisch, G. Neven, and A. Shelat, "Simulatable adaptive oblivious transfer," in *Advances in Cryptology - EUROCRYPT*, 2007.

[33] M. Green and S. Hohenberger, "Universally composable adaptive oblivious transfer," in *Advances in Cryptology - ASIACRYPT*, 2008.

[34] B. Zhang, H. Lipmaa, C. Wang, and K. Ren, "Practical fully simulatable oblivious transfer with sublinear communication," in *Financial Cryptography and Data Security - 17th International Conference, FC*, 2013.

[35] K. Kurosawa, R. Nojima, and L. T. Phong, "Efficiency-improved fully simulatable adaptive OT under the DDH assumption," in *Security and Cryptography for Networks, 7th International Conference, SCN*, 2010.

[36] C. Blundo, P. D'Arco, A. D. Santis, and D. R. Stinson, "On unconditionally secure distributed oblivious transfer," *J. Cryptol.*, 2007.

[37] C. Crépeau and J. Kilian, "Achieving oblivious transfer using weakened security assumptions (extended abstract)," in *29th Annual Symposium on Foundations of Computer Science*, 1988.

[38] C. Crépeau, K. Morozov, and S. Wolf, "Efficient unconditional oblivious transfer from almost any noisy channel," in *Security in Communication Networks, 4th International Conference, SCN*, 2004.

[39] Y. Ishai, E. Kushilevitz, R. Ostrovsky, M. Prabhakaran, A. Sahai, and J. Wullschleger, "Constant-rate oblivious transfer from noisy channels," in *Advances in Cryptology - CRYPTO*, 2011.

[40] R. Rivest, "Unconditionally secure commitment and oblivious transfer schemes using private channels and a trusted initializer," *technical report*, 1999, https://docplayer.net/185107526-Unconditionally-secure-commitment-and-oblivious-transfer-schemes-using-private-channels-and-a-trusted-initializer-ronald-l-rivest-laboratory-for-comp.html.

[41] V. K. Yadav, N. Andola, S. Verma, and S. Venkatesan, "A survey of oblivious transfer protocol," *ACM Comput. Surv.*, 2022.

[42] M. Naor and B. Pinkas, "Efficient oblivious transfer protocols," ser. SODA '01. Society for Industrial and Applied Mathematics, 2001.

[43] C. Devet, I. Goldberg, and N. Heninger, "Optimally robust private information retrieval," in *Proceedings of the 21th USENIX Security Symposium*, 2012.

Appendix A. DUQ-OT's Security Proof

Below, we prove DUQ-OT's security theorem, i.e., Theorem 2. Even though the proofs of DUQ-OT and DQ-OT have similarities, they have significant differences too; thus, for the sake of completeness we present a complete proof for DUQ-OT.

Proof. We consider the case where each party is corrupt, at a time.

A.0.1. Corrupt Receiver R. In the real execution, R's view is:

$\texttt{View}_{R}^{DUQ\text{-}OT}\big(m_0, m_1, \epsilon, \quad \epsilon, \epsilon, s\big) = \{r_{R}, C, r_3, s_2, e'_0, e'_1, m_s\}$, where r_{R} is the outcome of the internal random coin of R and is used to generate (r_1, r_2). Below, we construct an idea-model simulator $\texttt{Sim}_R$ which receives m_s from R.

1) constructs an empty view and appends uniformly random coin r'_{R} to it, where r'_{R} will be used to generate R-side randomness, i.e., (r'_1, r'_2).
2) sets response $(\bar{e}'_0, \bar{e}'_1)$ as follows:
   - picks random values: $C', r'_1, r'_2, y'_0, y'_1 \overset{\$}{\leftarrow} \mathbb{Z}_{p-1}$, $r'_3 \overset{\$}{\leftarrow} \{0,1\}^{\lambda}$, $s' \overset{\$}{\leftarrow} \{0,1\}$, and $u \overset{\$}{\leftarrow} \{0,1\}^{\sigma+\lambda}$.
   - sets $x = r'_2 + r'_1 \cdot (-1)^{s'}$ and $\beta'_0 = g^x$.
   - sets $\bar{e}_0 = (g^{y'_0}, \texttt{G}(\beta'^{y'_0}_0) \oplus (m_s || r'_3))$ and $\bar{e}_1 = (g^{y'_1}, u)$.
   - randomly permutes the element of pair $(\bar{e}_0, \bar{e}_1)$. Let $(\bar{e}'_0, \bar{e}'_1)$ be the result.
3) appends $(C', r'_3, s', \bar{e}'_0, \bar{e}'_1, m_s)$ to the view and outputs the view.

Next, we argue that the views in the ideal and real models are indistinguishable. As we are in the semi-honest model, the adversary picks its randomness according to the protocol description; therefore, r_{R} and r'_{R} model have identical distributions, the same hold for values (r_3, s_2) in the real model and (r'_3, s') in the ideal model, componentwise.

For the sake of simplicity, in the ideal mode let $\bar{e}'_j = \bar{e}_1 = (g^{y'_1}, u)$ and in the real model let $e'_i = e_{1-s} = (g^{y_{1-s}}, \texttt{G}(\beta^{y_{1-s}}_{1-s}) \oplus (m_{1-s} || r_3))$, where $i, j \in \{0,1\}$. We will explain that e'_i in the real model and $\bar{e}'_j$ in the ideal model are indistinguishable. In the real model, it holds that $e_{1-s} = (g^{y_{1-s}}, \texttt{G}(\beta^{y_{1-s}}_{1-s}) \oplus (m_{1-s} || r_3))$, where $\beta^{y_{1-s}}_{1-s} = \frac{C}{g^x} = g^{a-x}$. Since y_{1-s} in the real model and y'_1 in the ideal model have been picked uniformly at random and unknown to the adversary, $g^{y_{1-s}}$ and $g^{y'_1}$ have identical distributions.

Moreover, in the real model, given $C = g^a$, because of DL problem, a cannot be computed by a PPT adversary. Furthermore, due to CDH assumption, R cannot compute $\beta^{y_{1-s}}_{1-s}$ (i.e., the input of $\texttt{G}(.)$), given $g^{y_{1-s}}$ and g^{a-x}. We know that $\texttt{G}(.)$ is considered as a random oracle and its output is indistinguishable from a random value. Therefore, $\texttt{G}(\beta^{y_{1-s}}_{1-s}) \oplus (m_{1-s} || r_3)$ in the real model and u in the ideal model are indistinguishable. This means that e_{1-s} and $\bar{e}'_j$ are indistinguishable too, due to DL, CDH, and RO assumptions.

Moreover, since (i) y_s in the real model and y'_0 in the ideal model have picked uniformly at random and (ii) the decryption of both e'_{1-i} and $\bar{e}'_{1-j}$ contain m_s, e'_{1-i} and $\bar{e}'_{1-j}$ have identical distributions. m_s also has identical distribution in both models. Both C and C' have also been picked uniformly at random from $\mathbb{Z}_{p-1}$; therefore, they have identical distribution.

In the ideal model, $\bar{e}_0$ always contains an encryption of actual message m_s while $\bar{e}_1$ always contains a dummy value u. However, in the ideal model the elements of pair $(\bar{e}_0, \bar{e}_1)$ and in the real model the elements of pair (e_0, e_1) have been randomly permuted, which result in $(\bar{e}'_0, \bar{e}'_1)$ and (e'_0, e'_1) respectively. Therefore, the permuted pairs have identical distributions too.

We conclude that the two views are computationally indistinguishable, i.e., Relation 7 (in Section 5.1) holds.

A.0.2. Corrupt Sender S. In the real model, S's view is: $\texttt{View}_{S}^{DUQ\text{-}OT}\big((m_0, m_1), \epsilon, \epsilon, \epsilon, \ s\big) = \{r_{S}, C, r_3, \beta_0, \beta_1\}$, where r_{S} is the outcome of the internal random coin of S. Next, we construct an idea-model simulator $\texttt{Sim}_S$ which receives $\{m_0, m_1\}$ from S.

1) constructs an empty view and appends uniformly random coin r'_{S} to it, where r'_{S} will be used to generate random values for S.
2) picks random values $C', r' \overset{\$}{\leftarrow} \mathbb{Z}_{p-1}, r'_3 \overset{\$}{\leftarrow} \{0,1\}^{\lambda}$.
3) sets $\beta'_0 = g^{r'}$ and $\beta'_1 = \frac{C'}{g^{r'}}$.
4) appends C', r'_3, β'_0, and β'_1 to the view and outputs the view.

Next, we explain why the two views in the ideal and real models are indistinguishable. Recall, in the real model, (β_s, β_{1-s}) have the following form: $\beta_s = g^x$ and $\beta_{1-s} = g^{a-x}$, where $a = DL(C)$ and $C = g^a$. In this ideal model, as a and x have been picked uniformly at random and unknown to the adversary, due to DL assumption, β_s and β_{1-s} have identical distributions and are indistinguishable. In the ideal model, r' and C' have been picked uniformly at random and we know that a' in $C' = g^{a'}$ is a uniformly random value, unknown to the adversary; thus, due to DL assumption, β'_0 and β'_1 have identical distributions too. The same holds for values C and C'. Moreover, values $\beta_s, \beta_{1-s}, \beta'_0$, and β'_1 have been defined over the same field, $\mathbb{Z}_p$. Thus, they have identical distributions and are indistinguishable. The same holds for values r_3 in the real model and r'_3 in the ideal model.

Therefore, the two views are computationally indistinguishable, i.e., Relation 4 (in Section 5.1) holds.

A.0.3. Corrupt Server P_2. In the real execution, P_2's view is:

$\texttt{View}_{P_2}^{DUQ\text{-}OT}\big((m_0, m_1), \epsilon, \ \epsilon, \epsilon, s\big) = \{C, s_2, r_2\}$. Below, we show how an ideal-model simulator $\texttt{Sim}_{P_2}$ works.

1) constructs an empty view.
2) picks two uniformly random values $s'_2 \overset{\$}{\leftarrow} \mathbb{U}$ and $C', r'_2 \overset{\$}{\leftarrow} \mathbb{Z}_{p-1}$, where $\mathbb{U}$ is the output range of $\texttt{SS}(.)$.
3) appends s'_2, C' and r'_2 to the view and outputs the view.

Next, we explain why the views in the ideal and real models are indistinguishable. Since r_2 and r'_2 have been picked uniformly at random from $\mathbb{Z}_{p-1}$, they have identical distributions. Also, due to the security of $\texttt{SS}(.)$ each share s_2 is indistinguishable from a random value s'_2, where $s'_2 \in \mathbb{U}$. Also, both C and C' have been picked uniformly at random from $\mathbb{Z}_{p-1}$; therefore, they have identical distribution.

Thus, the two views are computationally indistinguishable, i.e., Relation 5 w.r.t. P_2 (in Section 5.1) holds.

A.0.4. Corrupt Server P_1. In the real execution, P_1's view is:

$\texttt{View}_{P_1}^{DUQ\text{-}OT}\big((m_0, m_1), \epsilon,\ \epsilon, \epsilon, s\big) = \{C, s_1, r_1, \delta_0, \delta_1\}$. Ideal-model $\texttt{Sim}_{P_1}$ works as follows.

1) constructs an empty view.
2) picks two random values $\delta'_0, \delta'_1 \overset{\$}{\leftarrow} \mathbb{Z}_p$.
3) picks two uniformly random values $s'_1 \overset{\$}{\leftarrow} \mathbb{U}$ and $C', r'_1 \overset{\$}{\leftarrow} \mathbb{Z}_{p-1}$, where $\mathbb{U}$ is the output range of $\texttt{SS}(.)$.
4) appends $s'_1, C', r'_1, \delta'_0, \delta'_1$ to the view and outputs the view.

Now, we explain why the views in the ideal and real models are indistinguishable. Recall, in the real model, P_1 receives $\delta_{s_2} = g^{r_2}$ and $\delta_{1-s_2} = g^{a-r_2}$ from P_2. Since a and r_2 have been picked uniformly at random and unknown to the adversary due to DL assumption, δ_{s_2} and δ_{1-s_2} (or δ_0 and δ_1) have identical distributions and are indistinguishable from random values (of the same field).

In the ideal model, δ'_0 and δ'_1 have been picked uniformly at random; therefore, they have identical distributions too. Moreover, values $\delta_s, \delta_{1-s}, \delta'_0$, and δ'_1 have been defined over the same field, $\mathbb{Z}_p$. So, they have identical distributions and are indistinguishable. Due to the security of $\texttt{SS}(.)$ each share s_1 is indistinguishable from a random value s'_1, where $s'_1 \in \mathbb{U}$. Furthermore, (r_1, C) and (r'_1, C') have identical distributions, as they are picked uniformly at random from $\mathbb{Z}_{p-1}$.

Hence, the two views are computationally indistinguishable, i.e., Relation 5 w.r.t. P_2 (in Section 5.1) holds.

A.0.5. Corrupt T. T's view can be easily simulated. It has input s, but it receives no messages from its counterparties and receives no output from the protocol. Thus, its real world view is defined as $\texttt{View}_{T}^{DUQ\text{-}OT}\big((m_0, m_1), \epsilon, \epsilon, \epsilon, s\big) = \{r_T\}$, where r_T is the outcome of the internal random coin of T and is used to generate random values.

Ideal-model $\texttt{Sim}_T$ constructs an empty view, picks r'_T uniformly at random, and adds it to the view. Since, in the real model, the adversary is passive, then it picks its randomness according to the protocol's description; thus, r_T and r'_T have identical distributions.

Thus, the two views are computationally indistinguishable, i.e., Relation 6 (in Section 5.1) holds.

□

Appendix B. Proof of Lemma 1

Below, we prove Lemma 1 presented in Section 6.1.1.

Proof. First, we focus on the first element of pairs $(g^{y_0}, \texttt{H}(\beta_0^{y_0}) \oplus m_0)$ and $(g^{y_1}, \texttt{H}(\beta_1^{y_1}) \oplus m_1)$. Since y_0 and y_1 have been picked uniformly at random and unknown to the adversary, g^{y_0} and g^{y_1} are indistinguishable from random elements of group $\mathbb{G}$.

Next, we turn our attention to the second elements of the pairs. Given $C = g^a$, due to DL problem, value a cannot be extracted by a PPT adversary, except for a probability at most $\mu(\lambda)$. We also know that, due to CDH assumption, a PPT adversary cannot compute $\beta_i^{y_i}$ (i.e., the input of $\texttt{H}(.)$), given g^{y_i}, r_1, C, g^{r_2}, and $\frac{C}{g^{r_2}}$, where $i \in \{0, 1\}$, except for a probability at most $\mu(\lambda)$. We know that $\texttt{H}(.)$ has been considered as a random oracle and its output is indistinguishable from a random value. Therefore, $\texttt{H}(\beta_0^{y_0}) \oplus m_0$ and $\texttt{H}(\beta_1^{y_1}) \oplus m_1$ are indistinguishable from random elements of $\mathbb{G}$, except for a negligible probability, $\mu(\lambda)$. □

Appendix C. DQ$^{\text{HF}}$–OT in more Detail

Figure 7 presents the DQ$^{\text{HF}}$–OT that realises $\mathcal{DQ}^{\mathcal{HF}}\text{–}\mathcal{OT}_1^2$.

Theorem 6. *Let $\mathcal{F}_{\mathcal{DQ}^{\mathcal{HF}}\text{-}\mathcal{OT}_1^2}$ be the functionality defined in Section 6.1.1. If DL, CDH, and RO assumptions hold, then DQ$^{\text{HF}}$–OT (presented in Figure 7) securely computes $\mathcal{F}_{\mathcal{DQ}^{\mathcal{HF}}\text{-}\mathcal{OT}_1^2}$ in the presence of (a) semi-honest receiver R, honest S, P_1, and P_2, (b) semi-honest S, honest R, P_1, and P_2, or (c) semi-honest P_i (where $i \in \{1, 2\}$), honest S and R, w.r.t. Definition 4.*

C.1. Proof of Theorem 6

Next, we prove the security of DQ$^{\text{HF}}$–OT, i.e., Theorem 6.

Proof. To prove the above theorem, we consider the cases where each party is corrupt at a time.

C.1.1. Corrupt R. Recall that in DQ$^{\text{HF}}$–OT, sender S holds a vector $\boldsymbol{m}$ of z pairs of messages (as apposed to DQ-OT where S holds only a single pair of messages). In the real execution, R's view is:

$\texttt{View}_R^{\text{DQ}^{\text{HF}}\text{-OT}}\big(m_0, m_1, v,\ \epsilon, s\big) = \{r_R, C, e_{0,v}, e_{1,v}, m_{s,v}\}$, where $C = g^a$ is a random value and public parameter, a is a random value, and r_R is the outcome of the internal random coin of R and is used to generate (r_1, r_2).

We will construct a simulator $\texttt{Sim}_R$ that creates a view for R such that (i) R will see only a pair of messages (rather than z pairs), and (ii) the view is indistinguishable from the view of corrupt R in the real model. $\texttt{Sim}_R$ which receives $\{s, m_s\}$ from R operates as follows.

1) constructs an empty view and appends uniformly random coin r'_R to it, where r'_R will be used to generate R-side randomness.
2) sets (e'_0, e'_1) as follows:
   - splits s into two shares: $\texttt{SS}(1^\lambda, s, 2, 2) \rightarrow (s'_1, s'_2)$.
   - picks uniformly random values: $C', r'_1, r'_2, y'_0, y'_1 \overset{\$}{\leftarrow} \mathbb{Z}_{p-1}$.
   - sets $\beta'_s = g^x$, where x is set as follows:
     - $*$ $x = r'_2 + r'_1$, if $(s = s_1 = s_2 = 0)$ or $(s = s_1 = 1 \wedge s_2 = 0)$.
     - $*$ $x = r'_2 - r'_1$, if $(s = 0 \wedge s_1 = s_2 = 1)$ or $(s = s_2 = 1 \wedge s_1 = 0)$.

1) *R_j-side Delegation:*
   a) split the private index s into two shares (s_1, s_2) by calling $\mathtt{SS}(1^\lambda, s, 2, 2) \rightarrow (s_1, s_2)$.
   b) pick two uniformly random values: $r_1, r_2 \xleftarrow{\$} \mathbb{Z}_{p-1}$.
   c) send (s_1, r_1) to P_1 and (s_2, r_2) to P_2.
2) *P_2-side Query Generation:*
   a) compute a pair of partial queries:
$$\delta_{s_2} = g^{r_2}, \quad \delta_{1-s_2} = \frac{C}{g^{r_2}}$$
   b) send (δ_0, δ_1) to P_1.
3) *P_1-side Query Generation:*
   a) compute a pair of final queries as:
$$\beta_{s_1} = \delta_0 \cdot g^{r_1}, \quad \beta_{1-s_1} = \frac{\delta_1}{g^{r_1}}$$
   b) send (β_0, β_1) to S.
4) *S-side Response Generation:*
   a) abort if $C \neq \beta_0 \cdot \beta_1$.
   b) compute a response as follows. $\forall t, 0 \leq t \leq z-1$:
      i) pick two random values $y_{0,t}, y_{1,t} \xleftarrow{\$} \mathbb{Z}_{p-1}$
      ii) compute response:
$$e_{0,t} := (e_{0,0,t}, e_{0,1,t}) = (g^{y_{0,t}}, \mathtt{H}(\beta_0^{y_{0,t}}) \oplus m_{0,t})$$
$$e_{1,t} := (e_{1,0,t}, e_{1,1,t}) = (g^{y_{1,t}}, \mathtt{H}(\beta_1^{y_{1,t}}) \oplus m_{1,t})$$
   c) send $(e_{0,0}, e_{1,0}), ..., (e_{0,z-1}, e_{1,z-1})$ to P_1.
5) *P_1-side Oblivious Filtering:*
   - forward $(e_{0,v}, e_{1,v})$ to R_j and discard the rest of the messages received from S.
6) *R-side Message Extraction:*
   a) set $x = r_2 + r_1 \cdot (-1)^{s_2}$.
   b) retrieve message $m_{s,v}$ by setting:
$$m_{s,v} = \mathtt{H}((e_{s,0,v})^x) \oplus e_{s,1v}$$

Figure 7. DQ$^{\text{HF}}$–OT: Our protocol that realises $\mathcal{DQ}^{\mathcal{HF}}\text{-}\mathcal{OT}_1^2$. In the protocol, S maintains a vector of pairs $\boldsymbol{m} = [(m_{0,0}, m_{1,0}), ..., (m_{0,z-1}, m_{1,z-1})]$. For simplicity, we assume v-th pair $(m_{0,v},\ m_{1,v}) \in \boldsymbol{m}$ is related to j-th receiver R_j. Also, $C = g^a$ is a random public value, $\mathtt{SS}(.)$ is the share-generation algorithm (of a secret sharing) that has been defined in Section 2.5, $\mathtt{H}(.)$ is a hash function, and \$ denotes picking a value uniformly at random.

- picks a uniformly random value $u \xleftarrow{\$} \mathbb{Z}_p$ and then sets $e'_s = (g^{y'_s}, \mathtt{H}(\beta_s'^{y'_s}) \oplus m_s)$ and $e'_{1-s} = (g^{y'_{1-s}}, u)$.

3) appends $(C', r'_1, r'_2, e'_0, e'_1, m_s)$ to the view and outputs the view.

The above simulator is identical to the simulator we constructed for DQ-OT. Thus, the same argument that we used (in the corrupt R case in Section 4.4) to argue why real model and ideal model views are indisindrguishable, can be used in this case as well. That means, even though S holds z pairs of messages and generates response for all of them, R's view is still identical to the case where S holds only two pairs of messages. Hence, Relation 11 (in Section 6.1.1) holds.

C.1.2. Corrupt S. This case is identical to the corrupt S in the proof of DQ-OT (in Section 4.4) with a minor difference. Specifically, the real-model view of S in this case is identical to the real-model view of S in DQ-OT; however, now $\mathtt{Sim}_S$ receives a vector $\boldsymbol{m} = [(m_{0,0}, m_{1,0}), ..., (m_{0,z-1},\ m_{1,z-1})]$ from S, instead of only a single pair that $\mathtt{Sim}_S$ receives in the proof of DQ-OT. $\mathtt{Sim}_S$ still operates the same way it does in the corrupt S case in the proof of DQ-OT. Therefore, the same argument that we used (in Section 4.4) to argue why real model and ideal model views are indistinguishable (when S is corrupt), can be used in this case as well.

Therefore, Relation 8 (in Section 6.1.1) holds.

C.1.3. Corrupt P_2. This case is identical to the corrupt P_2 case in the proof of DQ-OT. So, Relation 10 (in Section 6.1.1) holds.

C.1.4. Corrupt P_1. In the real execution, P_1's view is: $\mathtt{View}_{P_1}^{\text{DQ}^{\text{HF}}\text{-OT}}\big((m_0, m_1), v, \quad \epsilon, s\big) = \{C, s_1, r_1, \delta_0, \delta_1, (e_{0,0}, e_{1,0}), ..., (e_{0,z-1}, \quad e_{1,z-1})\}$. Ideal-model $\mathtt{Sim}_{P_1}$ that receives v from P_1 operates as follows.

1) constructs an empty view.
2) picks two random values $\delta'_0, \delta'_1 \xleftarrow{\$} \mathbb{Z}_p$.
3) picks two uniformly random values $s'_1 \xleftarrow{\$} \mathbb{U}$ and $C', r'_1 \xleftarrow{\$} \mathbb{Z}_{p-1}$, where $\mathbb{U}$ is the output range of $\mathtt{SS}(.)$.
4) picks z pairs of random values as follows $(a_{0,0}, a_{1,0}), ..., (a_{0,z-1},\ a_{1,z-1}) \xleftarrow{\$} \mathbb{Z}_{p-1}$.
5) appends $s'_1, C', r'_1, \delta'_0, \delta'_1$ and pairs $(a_{0,0}, a_{1,0}), ..., (a_{0,z-1},\ a_{1,z-1})$ to the view and outputs the view.

Now, we explain why the views in the ideal and real models are indistinguishable. The main difference between this case and the corrupt P_1 case in the proof of DQ-OT (in Section 4.4) is that now P_1 has z additional pairs $(e_{0,0}, a_{1,0}), ..., (e_{0,z-1},\ a_{1,z-1})$. Therefore, regarding the views in real and ideal models excluding the additional z pairs, we can use the same argument we provided for the corrupt P_1 case in the proof of DQ-OT to show that the two views are indistinguishable. Moreover, due to Lemma 1, the elements of each pair $(e_{0,i}, e_{1,i})$ in the real model is indistinguishable from the elements of each pair $(a_{0,i}, a_{1,i})$ in the ideal model, for all i, $0 \leq i \leq z-1$. Hence, Relation 9 (in Section 6.1.1) holds. $\square$

Appendix D. Proof of Theorem 3

Below, we prove the security of DUQ$^{\text{HF}}$–OT, i.e., Theorem 3.

Proof. To prove the theorem, we consider the cases where each party is corrupt at a time.

D.0.1. Corrupt R. In the real execution, R's view is:

$\texttt{View}_R^{\text{DUQ}^{\text{HF}}\text{–OT}}\big(\boldsymbol{m},(v,s,z)\ \epsilon,\epsilon,\epsilon\big)=\{r_R,C,r_3,s_2,o_0,o_1,m_{s,v}\}$, where

$o_0:=(o_{0,0},o_{0,1})$, $o_1:=(o_{1,0},o_{1,1})$, $C=g^a$ is a random value and public parameter, a is a random value, and r_R is the outcome of the internal random coin of R that is used to (i) generate (r_1,r_2) and (ii) its public and private keys pair for additive homomorphic encryption.

We will construct a simulator $\texttt{Sim}_R$ that creates a view for R such that (i) R will see only a pair of messages rather than z pairs, and (ii) the view is indistinguishable from the view of corrupt R in the real model. $\texttt{Sim}_R$ which receives $m_{s,v}$ from R performs as follows.

1) constructs an empty view and appends uniformly random coin r'_R to it, where r'_R will be used to generate R-side randomness.
2) sets response as follows:
   - picks random values: $C',r'_1,r'_2,y'_0,y'_1\overset{\$}{\leftarrow}\mathbb{Z}_{p-1}$, $r'_3\overset{\$}{\leftarrow}\{0,1\}^{\lambda}$, $s'\overset{\$}{\leftarrow}\{0,1\}$, and $u\overset{\$}{\leftarrow}\{0,1\}^{\sigma+\lambda}$.
   - sets $x=r'_2+r'_1\cdot(-1)^{s'}$ and $\beta'_0=g^x$.
   - sets $\bar{e}_0:=\big(\bar{e}_{0,0}=g^{y'_0},\bar{e}_{0,1}=\texttt{G}(\beta_0'^{y'_0})\oplus(m_{s,v}||r'_3)\big)$ and $\bar{e}_1:=(\bar{e}_{1,0}=g^{y'_1},\bar{e}_{1,1}=u)$.
   - encrypts the elements of the pair under pk as follows. $\forall i,i',0\leq i,i'\leq 1:\bar{o}_{i,i'}=\texttt{Enc}(pk,\bar{e}_{i,i'})$. Let $\bar{o}_0:=(\bar{o}_{0,0},\bar{o}_{0,1})$ and $\bar{o}_1:=(\bar{o}_{1,0},\bar{o}_{1,1})$.
   - randomly permutes the element of pair $(\bar{o}_0,\bar{o}_1)$. Let $(\bar{o}'_0,\bar{o}'_1)$ be the result.
3) appends $(C',r'_3,s',\bar{o}'_0,\bar{o}'_1,m_{s,v})$ to the view and outputs the view.

Now, we argue that the views in the ideal and real models are indistinguishable. As we are in the semi-honest model, the adversary picks its randomness according to the protocol description; so, r_R and r'_R model have identical distributions, so do values (r_3,s_2) in the real model and (r'_3,s') in the ideal model, component-wise.

For the sake of simplicity, in the ideal model let $\bar{e}'_j=\bar{e}_1=(g^{y'_1},u)$ and in the real model let $e'_i=e_{1-s}=(g^{y_{1-s,v}},\texttt{G}(\beta_{1-s}^{y_{1-s,v}})\oplus(m_{1-s,v}||r_3))$, where $i,j\in\{0,1\}$. Note that $\bar{e}'_j$ and e'_i contain the elements that the adversary gets after decrypting the messages it receives from P_1 in the real model and from $\texttt{Sim}_R$ in the ideal model.

We will explain that e'_i in the real model and $\bar{e}'_j$ in the ideal model are indistinguishable. In the real model, it holds that $e_{1-s}=(g^{y_{1-s,v}},\texttt{G}(\beta_{1-s}^{y_{1-s,v}})\oplus(m_{1-s,v}||r_3))$, where $\beta_{1-s}^{y_{1-s,v}}=\frac{C}{g^x}=g^{a-x}$. Since $y_{1-s,v}$ in the real model and y'_1 in the ideal model have been picked uniformly at random and unknown to the adversary, $g^{y_{1-s,v}}$ and $g^{y'_1}$ have identical distributions. Moreover, in the real model, given $C=g^a$, due to DL problem, a cannot be computed by a PPT adversary. Also, due to CDH assumption, R cannot compute $\beta_{1-s}^{y_{1-s,v}}$, given $g^{y_{1-s}}$ and g^{a-x}. We know that $\texttt{G}(.)$ is considered a random oracle and its output is indistinguishable from a random value. Therefore, $\texttt{G}(\beta_{1-s}^{y_{1-s,v}})\oplus(m_{1-s,v}||r_3)$ in the real model and u in the ideal model are indistinguishable. This means that e'_i and $\bar{e}'_j$ are indistinguishable too, due to DL, CDH, and RO assumptions.

Also, ciphertexts $\bar{o}_{1,0}=\texttt{Enc}(pk,g^{y'_1})$ and $\bar{o}_{1,1}=\texttt{Enc}(pk,u)$ in the ideal model and ciphertexts $o_{1-s,0}=\texttt{Enc}(pk,g^{y_{1-s,v}})$ and $o_{1-s,1}=\texttt{Enc}(pk,\texttt{G}(\beta_{1-s}^{y_{1-s,v}})\oplus(m_{1-s,v}||r_3))$ in the real model have identical distributions due to IND-CPA property of the additive homomorphic encryption.

Further, (i) $y_{s,v}$ in the real model and y'_0 in the ideal model have been picked uniformly at random and (ii) the decryption of both e'_{1-i} and $\bar{e}'_{1-j}$ contain $m_{s,v}$; therefore, e'_{1-i} and $\bar{e}'_{1-j}$ have identical distributions. Also, $m_{s,v}$ has identical distribution in both models. Both C and C' have also been picked uniformly at random from $\mathbb{Z}_{p-1}$; therefore, they have identical distributions.

In the ideal model, $\bar{e}_0$ always contains encryption of actual message $m_{s,v}$ while $\bar{e}_1$ always contains a dummy value u. However, in the ideal model the encryption of the elements of pair $(\bar{e}_0,\bar{e}_1)$ and in the real model the encryption of the elements of pair $(e_{0,v},e_{1,v})$ have been randomly permuted, which result in $(\bar{o}'_0,\bar{o}'_1)$ and (o_0,o_1) respectively.

Moreover, ciphertexts $\bar{o}_{0,0}=\texttt{Enc}(pk,g^{y'_0})$ and $\bar{o}_{0,1}=\texttt{Enc}(pk,\ \texttt{G}(\beta_0'^{y'_0})\oplus(m_{s,v}||r'_3)))$ in the ideal model and ciphertexts $o_{s,0}=\texttt{Enc}(pk,g^{y_{s,v}})$ and $o_{s,1}=\texttt{Enc}(pk,\texttt{G}(\beta_s^{y_{s,v}})\oplus(m_{s,v}||r_3))$ have identical distributions due to IND-CPA property of the additive homomorphic encryption. Thus, the permuted pairs have identical distributions too.

We conclude that the two views are computationally indistinguishable, i.e., Relation 15 (in Section 6.2) holds. That means, even though S holds z pairs of messages and generates a response for all of them, R's view is still identical to the case where S holds only two pairs of messages.

D.0.2. Corrupt S. This case is identical to the corrupt S in the proof of DUQ-OT (in Appendix A) with a minor difference. Specifically, the real-model view of S in this case is identical to the real-model view of S in DUQ-OT. Nevertheless, now $\texttt{Sim}_S$ receives a vector $\boldsymbol{m}=[(m_{0,0},m_{1,0}),...,(m_{0,z-1},m_{1,z-1})]$ from S, instead of only a single pair that $\texttt{Sim}_S$ receives in the proof of DUQ-OT. $\texttt{Sim}_S$ still carries out the same way it does in the corrupt S case in the proof of DUQ-OT. Therefore, the same argument that we used (in Appendix A) to argue why real model and ideal model views are indistinguishable (when S is corrupt), can be used in this case as well.

Therefore, Relation 12 (in Section 6.2) holds.

D.0.3. Corrupt P_2. This case is identical to the corrupt P_2 case in the proof of DUQ-OT. Thus, Relation 13 (in Section 6.2) holds.

D.0.4. Corrupt P_1. In the real execution, P_1's view is: $\texttt{View}_{P_1}^{\text{DUQ}^{\text{HF}}\text{-OT}}\big(\boldsymbol{m}, (v, s, z), \quad \epsilon, \epsilon, \epsilon\big) = \{C, s_1, \boldsymbol{w}_j, r_1, \delta_0, \delta_1, (e'_{0,0}, e'_{1,0}), \ \ldots, \ (e'_{0,z-1}, \ e'_{1,z-1})\}$. Ideal-model $\texttt{Sim}_{P_1}$ operates as follows.

1) constructs an empty view.
2) picks two random values $\delta'_0, \delta'_1 \overset{\$}{\leftarrow} \mathbb{Z}_p$.
3) constructs an empty vector $\boldsymbol{w}'$. It picks z uniformly at random elements $w'_0, ..., w'_z$ from the encryption (ciphertext) range and inserts the elements into $\boldsymbol{w}'$.
4) picks two uniformly random values $s'_1 \overset{\$}{\leftarrow} \mathbb{U}$ and $C', r'_1 \overset{\$}{\leftarrow} \mathbb{Z}_{p-1}$, where $\mathbb{U}$ is the output range of $\texttt{SS}(.)$.
5) picks z pairs of random values as follows $(a_{0,0}, a_{1,0}), ..., (a_{0,z-1}, \ a_{1,z-1}) \overset{\$}{\leftarrow} \mathbb{Z}_{p-1}$.
6) appends $s'_1, C', r'_1, \delta'_0, \delta'_1$ and pairs $(a_{0,0}, a_{1,0}), ..., (a_{0,z-1}, \ a_{1,z-1})$ to the view and outputs the view.

Next, we argue that the views in the ideal and real models are indistinguishable. The main difference between this case and the corrupt P_1 case in the proof of DUQ-OT (in Appendix A) is that now, in the real model, P_1 has: (i) a vector $\boldsymbol{w}_j$ of ciphertexts and (ii) z pairs $(e'_{0,0}, e'_{1,0}), ..., (e'_{0,z-1}, \ e'_{1,z-1})$. Therefore, we can reuse the same argument we provided for the corrupt P_1 case in the proof of DUQ-OT to argue that the views (excluding $\boldsymbol{w}_j$ and $(e'_{0,0}, e'_{1,0}), ..., (e'_{0,z-1}, \ e'_{1,z-1})$) have identical distributions.

Due to Lemma 1, the elements of each pair $(e'_{0,i}, e'_{1,i})$ in the real model are indistinguishable from the elements of each pair $(a_{0,i}, a_{1,i})$ in the ideal model, for all i, $0 \leq i \leq z-1$. Also, due to the IND-CPA property of the additive homomorphic encryption scheme, the elements of $\boldsymbol{w}_j$ in the real model are indistinguishable from the elements of $\boldsymbol{w}'$ in the ideal model.

Hence, Relation 13 (in Section 6.2) holds.

D.0.5. Corrupt T. This case is identical to the corrupt T in the proof of DUQ-OT, with a minor difference; namely, in this case, T also has input z which is the total number of message pairs that S holds. Thus, we can reuse the same argument provided for the corrupt T in the proof of DUQ-OT to show that the real and ideal models are indistinguishable. Thus, Relation 14 (in Section 6.2) holds. □

Appendix E. Proof of Theorem 4

Proof sketch. Compared to an original 1-out-of-n OT, the only extra information that S learns in the real model is a vector of n encrypted binary elements. Since the elements have been encrypted and the encryption satisfies IND-CPA, each ciphertext in the vector is indistinguishable from an element picked uniformly at random from the ciphertext (or encryption) range. Therefore, it would suffice for a simulator to pick n random values and add them to the view. As long as the view of S in the original 1-out-of-n OT can be simulated, the view of S in the new 1-out-of-n OT can be simulated too (given the above changes).

Interestingly, in the real model, R learns less information than it learns in the original 1-out-of-n OT because it only learns the encryption of the final message m_s. The simulator (given m_s and s) encrypts m_s the same way as it does in the ideal model in the 1-out-of-n OT. After that, it encrypts the result again (using the additive homomorphic encryption) and sends the ciphertext to R. Since in both models, R receives the same number of values in response, the values have been encrypted twice, and R can decrypt them using the same approaches, the two models have identical distributions.

Moreover, the response size is $O(1)$, because the response is the result of (1) multiplying two vectors of size n component-wise and (2) then summing up the products which results in a single value in the case where each element of the response contains a single value (or w values if each element of the response contains w values). □

Appendix F. Proof of Theorem 5

Proof. We consider the case where each party is corrupt at a time.

F.0.1. Corrupt Receiver R. In the real execution, R's view is: $\texttt{View}_R^{\textit{Supersonic-OT}}\big((m_0, m_1), \epsilon, s\big) = \{r_{\scriptscriptstyle R}, e''_0, m_s\}$, where $r_{\scriptscriptstyle R}$ is the outcome of the internal random coin of R and is used to generate (s_1, s_2, k_0, k_1). Below, we construct an idea-model simulator $\texttt{Sim}_R$ which receives (s, m_s) from R.

1) constructs an empty view and appends a uniformly random coin $r'_{\scriptscriptstyle R}$ to the view.
2) picks a random key $k \overset{\$}{\leftarrow} \{0,1\}^{\sigma}$, using $r'_{\scriptscriptstyle R}$.
3) encrypts message m_s as follows $e = m_s \oplus k$.
4) appends e to the view and outputs the view.

Since we are in the passive adversarial model, the adversary picks its random coin $r_{\scriptscriptstyle R}$ (in the real models) according to the protocol. Therefore, $r_{\scriptscriptstyle R}$ and $r'_{\scriptscriptstyle R}$ have identical distributions. Moreover, e''_0 in the real model and e in the ideal model have identical distributions as both are the result of XORing message m_s with a fresh uniformly random value. Also, m_s is the same in both models so it has identical distribution in the real and ideal models. We conclude that Relation 18 (in Section 8.1) holds.

F.0.2. Corrupt Sender S. In the real execution, S's view is: $\texttt{View}_S^{\textit{Supersonic-OT}}\big((m_0, m_1), \epsilon, s\big) = \{r_{\scriptscriptstyle S}, s_1, k_0, k_1\}$, where $r_{\scriptscriptstyle S}$ is the outcome of the internal random coin of S and is used to generate its random values. Next, we construct an idea-model simulator $\texttt{Sim}_S$ which receives (m_0, m_1) from S.

1) constructs an empty view and appends a uniformly random coin $r'_{\scriptscriptstyle S}$ to the view.
2) picks a binary random value $s' \overset{\$}{\leftarrow} \{0,1\}$.

3) picks two uniformly random keys $(k'_0, k'_1) \xleftarrow{\$} \{0,1\}^{\sigma}$.
4) appends s', k'_0, k'_1 to the view and outputs the view.

Next, we explain why the two views are indistinguishable. The random coins r_S and r'_S in the real and ideal models have identical distribution as they have been picked according to the protocol's description (as we consider the passive adversarial model). Moreover, s_1 in the real model and s' in the ideal model are indistinguishable, as due to the security of the secret sharing scheme, binary share s_1 is indistinguishable from a random binary value s'. Also, the elements of pair (k_0, k_1) in the real model and the elements of pair (k'_0, k'_1) in the ideal model have identical distributions as they have been picked uniformly at random from the same domain. Hence, Relation 16 (in Section 8.1) holds.

F.0.3. Corrupt Server P. In the real execution, P's view is:

$\texttt{View}_P^{\textit{Supersonic-OT}}\big((m_0, m_1), \epsilon, s\big) = \{r_P, s_2, e'\}$, where r_P is the outcome of the internal random coin of P and is used to generate its random values and e' is a pair (e'_0, e'_1) and is an output of $\bar{\pi}$. Next, we construct an idea-model simulator $\texttt{Sim}_P$.

1) constructs an empty view and appends a uniformly random coin r'_P to the view.
2) picks a binary random value $s' \xleftarrow{\$} \{0,1\}$.
3) constructs a pair v of two uniformly random values $(v_0, v_1) \xleftarrow{\$} \{0,1\}^{\sigma}$.
4) appends s', v to the view and outputs the view.

Since we consider the passive adversarial model, the adversary picks its random coins r_P and r'_P (in the real and ideal models respectively) according to the protocol. So, they have identical distributions. Also, s_2 in the real model and s' in the ideal model are indistinguishable, as due to the security of the secret sharing scheme, binary share s_2 is indistinguishable from a random binary value s'.

In the real model, the elements of e' which are e'_0 and e'_1 have been encrypted/padded with two fresh uniformly random values. In the ideal model, the elements of v which are v_0 and v_1 have been picked uniformly at random. Due to the security of a one-time pad, e'_i $(\forall i, 0 \leq i \leq 1)$ is indistinguishable from a uniformly random value, including v_0 and v_1.

Also, in the real model, the pair e' that is given to P is always permuted based on the value of S's share (i.e., $s_1 \in \{0,1\}$) which is not known to P; whereas, in the ideal model, the pair v is not permuted. However, given the permuted pair e' and not permuted pair v, a distinguisher cannot tell where each pair has been permuted with probability greater than $\frac{1}{2}$. Therefore, Relation 17 (in Section 8.1) holds. $\square$