Delegated-Query Oblivious Transfer and its Practical Applications

Yvo Desmedt*[1] and Aydin Abadi**[2]

[1] The University of Texas at Dallas
[2] Newcastle University

Abstract. Databases play a pivotal role in the contemporary World Wide Web and the world of cloud computing. Unfortunately, numerous privacy violations have recently garnered attention in the news. To enhance database privacy, we consider Oblivious Transfer (OT), an elegant cryptographic technology. Our observation reveals that existing research in this domain primarily concentrates on theoretical cryptographic applications, overlooking various practical aspects:

– OTs assume parties have direct access to databases. Our "1-out-of-2 Delegated-Query OT" enables parties to privately query a database, without direct access.
– With the rise of cloud computing, physically separated databases may no longer remain so. Our "1-out-of-2 Delegated-Query Multi-Receiver OT" protects privacy in such evolving scenarios.
– Research often ignores the limitations of thin clients, e.g., Internet of Things devices. To address this, we propose a compiler that transforms any 1-out-of-n OT into a thin client version.

1 Introduction

Databases play a vital role in e-commerce, advertising, intelligence analysis, combating crime, knowledge discovery, and conducting scientific research. Privacy breaches involving databases, impacting both organizations and individuals, have become headline news. Some databases (e.g., Fortune 500 companies' databases about customers' purchase history) can be valued at millions of dollars.

Privacy issues arise when users seek access to databases they do not own or create. Furthermore, many databases are now hosted in the cloud, adding another layer of complexity to the privacy landscape. To simultaneously protect the privacy of the user and the database itself from each other, the cryptographic-based technology, called Oblivious Transfer (OT) has been proposed. It allows a user (called a receiver) interested in the s-th element of a database (m_0, m_1) (held by a sender) to learn only m_s while preserving the privacy of (i) index $s \in \{0, 1\}$ from the sender and (ii) the rest of the database's elements from the receiver. Numerous variants have been developed since OT's introduction in 1981 [42].

OT is an important cryptographic protocol that has found applications within various domains, such as generic secure Multi-Party Computation (MPC) [51,3,24], Private Set Intersection (PSI) [18], contract signing [20], Federated Learning (FL) [50,43,48], and accessing sensitive field elements of remote private databases while preserving privacy [6,1,31]. As evidenced by this work, numerous research gaps persist in this domain. Many real-world applications have been overlooked. These oversights align with gaps in the research on OT, as expounded upon in the next section.

* y.desmedt@cs.ucl.ac.uk
** aydin.abadi@newcastle.ac.uk

2 Motivations and Survey

In this section, we motivate the paper through real-world scenarios, discuss gaps in the OT research, and outline our contributions.

2.1 Motivations

Dealing with Insiders, e.g., in Financial Institutions. Insider attacks pose imminent threats to various organizations and their clients, such as financial institutions and their customers. Insiders may collaborate with external fraudsters, obtaining highly valuable data. There have been real-world incidents where bank employees have leaked or misused customers' information.

The "Swiss Leaks" [30] is a good example to illustrate the problem of insider leaks in the banking world. In the Swiss Leaks case, an insider attempted to sell information about accounts held by HSBC in Geneva. Later, when he failed, he leaked the information to the public. As another example, in the case of "JPMorgan Chase Insider Thief" [15], a former JPMorgan Chase personal banker has been arrested by the FBI on charges that he stole customers' account information and sold it to an undercover informant. Another notable case involved a Citibank employee who accessed sensitive customer information and used it to commit fraud [47].

Additionally, a former financial advisor at Morgan Stanley, was discovered to have illicitly accessed and leaked sensitive information from approximately 730,000 accounts. This data breach compromised customers' personal details such as their names, addresses, account numbers, and investment information. This employee, who worked within Morgan Stanley's private wealth management division, entered a guilty plea in federal court in Manhattan [46].

In this context, an insider can exclusively target high-profile wealthy individuals and sell the victims' information to their rivals, who might make strategic investments, often remaining stealthy from the victims' perspective. For an insider, a data breach in private banking or private financial advising can be more alluring than leaking hundreds of bank accounts. Indeed, the former could yield a higher payoff while concurrently posing a lower risk of exposure. Additionally, outsiders who infiltrate the computers of an individual advisor or the third-party database can compromise the privacy of customers' queries.

Furthermore, in this setting, financial advisors, within a financial institution, frequently maintain paid subscriptions to a valuable database (e.g., containing real estate market information, market trends, and capital flows) offered by third-party providers such as CoreLogic[3], Multiple Listing Service[4], or Real Capital Analytics[5]. In contrast, clients of these advisors do not necessarily need to subscribe to the database themselves. Instead, they interact with the advisors and direct their queries to them.

Hence, there is a pressing need to (i) protect customer query privacy from advisors and databases, (ii) ensure the privacy of the database from both customers and advisors, and (iii) secure the privacy of customers in the event of a data breach on the advisor's or database's side. As explained in Section 2.2, current OTs fall short of providing these features simultaneously.

Multi-Receiver OT. The adoption of cloud computing has been accelerating. The "PwC's 2023 Cloud Business Survey" suggests that 78% of executives participating in the survey have mentioned that their companies had adopted cloud in most or all parts of the business [41]. Moreover, multiple (sensitive) databases belonging to different parties have been merged and hosted by a single cloud provider. Indeed, the recent cyber-attack revealed that data belonging to British Airways, Boots, BBC, and Aer Lingus was kept by the same cloud [33]. Another example is Salesforce data exposure, where a Salesforce software bug allowed users from different organizations to access each other's data within the Salesforce Marketing Cloud [38]. This incident has potentially impacted various customers, including organizations like Aldo, Dunkin Donuts, Nestle Waters, and Sony [44]. The current OTs do not allow us to deal with this scenario, as we will elaborate in Section 2.2.

[3] https://www.corelogic.com/data-solutions/property-data-solutions/discovery-platform
[4] https://www.mls.com
[5] https://www.msci.com/our-solutions/real-assets/real-capital-analytics

Querying Databases with Hidden Fields. In specific applications, such as finance or healthcare, sensitive details about customers or patients must be withheld from them (at least for a certain time period). In the financial sector, this may include (a) a binary flag that determines whether a certain customer is deemed suspicious [2,19], or (b) proprietary banking strategies tailored to individual clients. In the medical sector, such information may involve a binary flag indicating whether a patient has a certain condition. In certain cases, revealing specific details about an illness or test result might endanger the patient [14,21]. Hence, the result that a client/receiver obtains for its request (e.g., seeking investment advice) depends on the private flag/query s, provided by a third party to its advisor who is directly dealing with the client, while the client itself is not aware of the value of s. We will further discuss it in Section 2.2.

2.2 Research Gaps

Support for Delegated-Query OT. Current OT technologies assume that a receiver that generates the query *always* 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. We introduce *Delegated-Query Oblivious Transfer* to address these limitations and deal with insider attacks (see Section 5).

Querying Merged Databases. Existing techniques do not support querying *merged databases* in a privacy-preserving manner. Specifically, they are not suitable for the *real-world multi-receiver setting* where a sender maintains multiple records[6] each belonging to a different receiver. The existing techniques do not allow a receiver to privately query such records without disclosing (i) the records, that the receiver accesses, to the sender and (ii) the number of records, that other database users have, to each receiver. Receivers with different levels of security form a natural example. The existing OTs reveal the entire database's size to receivers enabling them to acquire non-trivial information. The mere existence of private data can be considered sensitive information [40]. We propose several *Multi-Receiver OTs* to support querying merged databases in a privacy-preserving way (see Section 7).

Databases with Hidden Fields. The current OT concept assumes the receiver knows the full query, which may not always be desired, as discussed in Section 2.1. We will propose OT variants supporting a (partially) *unknown query* (see Sections 6.2 and 7.2).

Constant Size Response. The current techniques that allow a receiver to obtain a 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 (e.g., in IoT settings), if its available storage space is significantly smaller than the database size. We will introduce a generic compiler that transforms any OT with a non-constant response size to one with a constant response size (see Section 8).

2.3 Our Contributions

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

1. 1-out-of-2 Delegated-Query Oblivious Transfer ($\mathcal{DQ}\text{–}\mathcal{OT}^{2}_{1}$): a new notion of OT that extends the basic features of OT by allowing the receiver to *delegate* two tasks: (i) computing the query and (ii) interacting with the sender. These tasks can be assigned to a pair 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 5.2 presents $\mathcal{DQ}\text{–}\mathcal{OT}^{2}_{1}$.

[6] A database table consists of records/rows and fields/columns. Each record in a table represents a set of related data, e.g., last name and address.

   (a) Delegated-Query OT (DQ-OT): a protocol that realizes $\mathcal{DQ}$–$\mathcal{OT}^{2}_{1}$. Section 5.3 presents DQ-OT.
   (b) Delegate-Unknown-Query OT (DUQ-OT): a variant of DQ-OT which allows the receiver to extract the related message m_s even if it does not (and must not) know the related index s. Section 6.2 presents DUQ-OT.
2. 1-out-of-2 Delegated-Query Multi-Receiver OT ($\mathcal{DQ}^{\mathcal{MR}}$–$\mathcal{OT}^{2}_{1}$): 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 $[(m_{0,0}, m_{1,0}), \ldots, (m_{0,z-1}, m_{1,z-1})]$ does not find out which query belongs to which record. Section 7.1 presents $\mathcal{DQ}^{\mathcal{MR}}$–$\mathcal{OT}^{2}_{1}$.
   (a) Delegated-Query Multi-Receiver OT (DQ$^{\text{MR}}$–OT): an efficient protocol that realizes $\mathcal{DQ}^{\mathcal{MR}}$–$\mathcal{OT}^{2}_{1}$. It is built upon DQ-OT and inherits its features. DQ$^{\text{MR}}$–OT achieves its goal by allowing P_1 to know which record is related to which receiver. Section 7.1 presents DQ$^{\text{MR}}$–OT.
   (b) Delegate-Unknown-Query Multi-Receiver OT (DUQ$^{\text{MR}}$–OT): a variant of DQ$^{\text{MR}}$–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 7.2 presents DUQ$^{\text{MR}}$–OT.
3. A compiler: a generic compiler 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 lets a receiver (i) receive only a *constant* number of messages and (ii) have constant storage space. Section 8 presents the compiler.

3 Preliminaries

3.1 Notations

By ϵ we mean an empty string. When y represents a single value, $|y|$ refers to the bit length of y. However, when y is a tuple, $|y|$ denotes the number of elements contained within 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 $\texttt{parse}(\lambda, y) \rightarrow (u_1, u_2)$, which takes as input a value λ and a value y of length at least λ-bit. It parses y into two values u_1 and u_2 and returns (u_1, u_2) where the bit length of u_1 is $|y| - \lambda$ and the bit length of u_2 is λ. Also, U denotes a universe of messages $m_1, \ldots, m_t$. We define σ as the maximum size of messages in U, i.e., $\sigma = Max(|m_1|, \ldots, |m_t|)$. 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 [10].

3.2 Security Model

In this paper, we rely on the simulation-based model of secure multi-party computation [22] to define and prove the proposed protocols. Below, we restate the formal security definition within this model.

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 $f_2(x, y)$. In the setting where f is asymmetric and only one party (say the first one) receives the result, f is defined as $f := (f_1(x, y), \epsilon)$.

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 $\mathsf{View}_i^{\Gamma}(x, y)$ and equals $(w, r_i, m_1^i, \ldots, m_t^i)$, 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_j^i represents the j^{th} message this party receives. The output of the i^{th} party during the execution of Γ on (x, y) is denoted by $\mathsf{Output}_i^{\Gamma}(x, y)$ and can be generated from its own view of the execution.

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

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

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

3.3 Random Permutation

A random permutation $\pi(e_0, \ldots, e_n) \rightarrow (e'_0, \ldots, e'_n)$ is a probabilistic function that takes a set $A = \{e_0, \ldots, e_n\}$ and returns the same set of elements in a permuted order $B = \{e'_0, \ldots, e'_n\}$. 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 [29] can permute a set of n elements in time $O(n)$. We will use $\pi(.)$ in the protocols presented in Figures 2 and 3.

3.4 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. In this paper, g and p can be selected by sender S (in the context of OT).

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 $(\mathbb{G}, p, g) \xleftarrow{\$} G(1^\lambda)$, $a_1, a_2 \xleftarrow{\$} \mathbb{Z}_p$, and μ is a negligible function [17].

3.5 Secret Sharing

A (threshold) secret sharing $\mathtt{SS}^{(t,n)}$ 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. 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 referred to as (n, t)-secret sharing or $\mathtt{SS}^{(t,n)}$ for brevity.

In the case where $t = n$, there exists a highly efficient XOR-based secret sharing [4]. In this case, to share the secret s, the dealer first picks $n-1$ random bit strings $r_1, \ldots, r_{n-1}$ of the same length as the secret. Then, it computes $r_n = r_1 \oplus \ldots \oplus r_n \oplus s$. It considers each $r_i \in \{r_1, \ldots, r_n\}$ as a share of the secret. To reconstruct the secret, one can easily compute $r_1 \oplus \ldots \oplus r_n$. Any subset of less than n shares reveals no information about the secret. We will use this scheme in this paper. A secret sharing scheme involves two main algorithms; namely, $\mathtt{SS}(1^\lambda, s, n, t) \rightarrow (r_1, \ldots, r_n)$: to share a secret and $\mathtt{RE}(r_1, \ldots, r_t, n, t) \rightarrow s$ to reconstruct the secret.

3.6 Additive Homomorphic Encryption

Additive homomorphic encryption involves three algorithms: (1) key generation: $\mathtt{KGen}(1^\lambda) \rightarrow (sk, pk)$, which takes a security parameter as input and outputs a secret and public keys pair, (2) encryption: $\mathtt{Enc}(pk, m) \rightarrow c$, that takes public key pk and a plaintext message m as input and returns a ciphertext c, and (3) decryption: $\mathtt{Dec}(sk, c) \rightarrow m$, 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: $\texttt{Dec}(sk, \texttt{Enc}(pk, m_1) \overset{H}{+} \texttt{Enc}(pk, m_2)) = m_1 + m_2$, 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: $\texttt{Dec}(sk, \texttt{Enc}(pk, m) \overset{H}{\times} c) = m \cdot c$, 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 [28] for a formal definition. One such scheme that meets the above features is the Paillier public key cryptosystem, proposed in [39].

4 Related Work

Oblivious Transfer (OT) is one of the vital building blocks of cryptographic protocols and has been used in various mechanisms, such as PSI, generic MPC, and zero-knowledge proofs. The traditional 1-out-of-2 OT ($\mathcal{OT}_1^2$) 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 $\mathcal{OT}_1^2$ 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 traditional $\mathcal{OT}_1^2$ functionality is defined as $\mathcal{F}_{\mathcal{OT}_1^2} : ((m_0, m_1), s) \rightarrow (\epsilon, m_s)$.

The notion of 1-out-of-2 OT was initially proposed by Rabin [42] which consequently was generalized by Even *et al.* [20]. Since then, numerous variants of OT have been proposed. For instance, (i) 1-out-of-n OT, e.g., in [34,45,32]: which allows R to pick one entry out of n entries held by S, (ii) k-out-of-n OT, e.g., in [12,27,11]: which allows R to pick k entries out of n entries held by S, (iii) OT extension, e.g., in [26,25,37,3]: that supports efficient executions of OT (which mainly relies on symmetric-key operations), in the case OT needs to be invoked many times, and (iv) distributed OT, e.g., in [35,13,54]: that allows the database to be distributed among m servers/senders.

In the remainder of this section, we discuss several variants of OT that have extended and enhanced the original OT in [42].

4.1 Distributed OT

Naor and Pinkas [35] 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.

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, various distributed OTs have been proposed[7]. For instance, Corniaux and Ghodosi [13] 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.* [54] 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 third-party servers in a privacy-preserving manner.

[7] Distributed OT has also been called proxy OT in [52].

4.2 Multi-Receiver OT

Camenisch *et al.* [6] proposed a protocol for "OT with access control". It involves a set of receivers and a sender which maintains records of the receivers. It offers a set of interesting features; namely, (i) only authorized 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 encrypted database for the receivers to download. Subsequently, researchers proposed various variants of OT with access control, as seen in [8,5,7]. Nevertheless, in all the aforementioned schemes, the size of the entire database is revealed to the receivers.

4.3 OT with Constant Response Size

Researchers have proposed several OTs, e.g., those proposed in [9,23,53], 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 initialization 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. We refer readers to [49] for a recent survey of OT.

5 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 realizes 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 in computing the query. $\mathcal{DQ}\text{–}\mathcal{OT}_1^2$ enables R to delegate (i) the computation of the 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, if they do not collude with each other.

5.1 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: $\mathcal{F}_{\mathcal{DQ}\text{–}\mathcal{OT}_1^2} : \big((m_0, m_1), \epsilon, \epsilon, s\big) \rightarrow (\epsilon, \epsilon, \epsilon, m_s)$.

5.2 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 realizes $\mathcal{F}_{\mathcal{DQ}\text{–}\mathcal{OT}_1^2}$ in the presence of 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 adversary (or simulator)* `Sim` *in the ideal model, such that:*

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

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

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

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.

A $\mathcal{DQ}\text{–}\mathcal{OT}_1^2$ scheme must meet two (new) properties, *efficiency* and *sender-push communication* (SPC). Efficiency states that the query generation of the receiver is faster compared to traditional (non-delegated) OT. SPC, on the other hand, stipulates that the sender transmits responses to the receiver without requiring the receiver to directly initiate a query to the sender. Below, we formally state these properties.

Definition 3 (Efficiency). *A $\mathcal{DQ}\text{–}\mathcal{OT}_1^2$ scheme is considered efficient if the running time of the receiver-side request (or query) generation algorithm, denoted as $\mathtt{Request}(1^\lambda, \mathtt{s}, \mathtt{pk})$, satisfies two conditions:*

- *The running time is upper-bounded by poly$(|m|)$, where poly is a fixed polynomial, m is a tuple of messages that the sender holds, and $|m|$ represents the number of elements/messages in tuple m.*
- *The running time is asymptotically constant with respect to the security parameter λ, i.e., it is $O(1)$.*

Definition 4 (Sender-push communication). *Let (a) $\mathtt{Action}_R(t)$ represent the set of actions available to R at time t (these actions may include sending requests, receiving messages, or any other interactions R can perform within the scheme's execution), (b) $\mathtt{Action}_S(t)$ be the set of actions available to S at time t, (c) $\mathtt{SendRequest}(R, S)$ be the action of R sending a request to S, and (d) $\mathtt{SendMessage}(S, R)$ represents the action of S sending a message to R. Then, we say that a $\mathcal{DQ}\text{–}\mathcal{OT}_1^2$ scheme supports sender-push communication, if it meets the following conditions:*

- *Receiver-side restricted interaction: For all t in the communication timeline (i.e., within the execution of the scheme), the set of actions $\mathtt{Action}_R(t)$ available to the receiver R is restricted such that it does not include direct requests to the sender S. Formally,*

$$\forall t : \mathtt{Action}_R(t) \cap \{\mathtt{SendRequest}(R, S)\} = \emptyset$$

- *Sender-side non-restricted interaction: For all t in the communication timeline, the sender S has the capability to push messages to the receiver R without receiving explicit request directly from R. Formally,*

$$\forall t : \mathtt{Action}_S(t) \subseteq \{\mathtt{SendMessage}(S, R)\}$$

Looking ahead, the above two properties are also valid for the variants of $\mathcal{DQ}\text{–}\mathcal{OT}_1^2$; namely, $\mathcal{DUQ}\text{–}\mathcal{OT}_1^2$, $\mathcal{DQ}^{\mathcal{MR}}\text{–}\mathcal{OT}_1^2$, and $\mathcal{DUQ}^{\mathcal{MR}}\text{–}\mathcal{OT}_1^2$, with a minor difference being that the receiver-side request generation algorithm in these three variants is denoted as $\mathtt{R.Request}()$.

5.3 Protocol

Now, we present an efficient 1-out-of-2 OT protocol, called DQ-OT, that realizes $\mathcal{DQ}\text{–}\mathcal{OT}_1^2$. We build DQ-OT upon the $\mathcal{OT}_1^2$ proposed by Naor and Pinkas [36, pp. 450, 451]. Our motivation for this choice is primarily didactic. Appendix A restates this OT.

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. 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 [36]. Below, we explain how DQ-OT operates, followed by an explanation of how it achieves correctness.

First, R splits the index that it is interested in into two binary shares, (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, $\delta_{s_2} = g^{r_2}$. Also, P_2 generates another query, $\delta_{1-s_2} = \frac{C}{g^{r_2}}$, where C is a random public parameter (as defined in [36]). 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.

Fourth, given the queries, S computes the response in the same manner it does in the original OT in [36] 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 $\mathcal{F}_{\mathcal{DQ}\text{-}\mathcal{OT}^2_1}$ be the functionality defined in Section 5.2. If Discrete Logarithm (DL), Computational Diffie-Hellman (CDH), and Random Oracle (RO) assumptions hold, then DQ-OT (presented in Figure 1) securely computes $\mathcal{F}_{\mathcal{DQ}\text{-}\mathcal{OT}^2_1}$ in the presence of semi-honest adversaries, w.r.t. Definition 2.*

5.4 DQ-OT's Security Proof

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

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

Corrupt Receiver *R*. In the real execution, R's view is: $\mathtt{View}_R^{DQ\text{-}OT}(m_0, m_1, \epsilon, \epsilon, s) = \{r_R, g, C, p, e_0, e_1, m_s\}$, where g is a random generator, $C = g^a$ is a random public parameter, a is a random value, p is a large random prime number, 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. initiates an empty view and appends uniformly random coin r'_R to it, where r'_R will be used to generate R-side randomness. It chooses a large random prime number p and a random generator g.
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: $C', r'_1, r'_2, y'_0, y'_1 \xleftarrow{\$} \mathbb{Z}_p$.
   - 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 $e'_s = (g^{y'_s}, \mathtt{H}(\beta'^{y'_s}_s) \oplus m_s)$ and $e'_{1-s} = (g^{y'_{1-s}}, u)$.
3. appends $(g, C', p, r'_1, r'_2, e'_0, e'_1, m_s)$ 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_R and r'_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 distributions. The same applies to values g and p in the real and ideal models.

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 $e_{1-s} = (g^{y_{1-s}}, \mathtt{H}(\beta^{y_{1-s}}_{1-s}) \oplus m_{1-s})$, 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-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 the DL problem, a cannot be computed by a PPT adversary. Also, due to CDH assumption, R cannot compute $\beta^{y_{1-s}}_{1-s}$ (i.e., the input of $\mathtt{H}(.)$), given $g^{y_{1-s}}$ and

1. *S-side Initialization:* $\texttt{Init}(1^\lambda) \rightarrow pk$
   (a) chooses a sufficiently large prime number p.
   (b) selects random element $C \overset{\$}{\leftarrow} \mathbb{Z}_p$ and generator g.
   (c) publishes $pk = (C, p, g)$.
2. *R-side Delegation:* $\texttt{Request}(1^\lambda, s, pk) \rightarrow req = (req_1, req_2)$
   (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) picks two uniformly random values: $r_1, r_2 \overset{\$}{\leftarrow} \mathbb{Z}_p$.
   (c) sends $req_1 = (s_1, r_1)$ to P_1 and $req_2 = (s_2, r_2)$ to P_2.
3. *P_2-side Query Generation:* $\texttt{P}_2.\texttt{GenQuery}(req_2, , pk) \rightarrow q_2$
   (a) computes a pair of partial queries:

$$\delta_{s_2} = g^{r_2}, \quad \delta_{1-s_2} = \frac{C}{g^{r_2}}$$

   (b) sends $q_2 = (\delta_0, \delta_1)$ to P_1.
4. *P_1-side Query Generation:* $\texttt{P}_1.\texttt{GenQuery}(req_1, q_2, pk) \rightarrow q_1$
   (a) computes 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) sends $q_1 = (\beta_0, \beta_1)$ to S.
5. *S-side Response Generation:* $\texttt{GenRes}(m_0, m_1, pk, q_1) \rightarrow res$
   (a) aborts if $C \neq \beta_0 \cdot \beta_1$.
   (b) picks two uniformly random values: $y_0, y_1 \overset{\$}{\leftarrow} \mathbb{Z}_p$.
   (c) computes a response pair (e_0, e_1) as follows:

$$e_0 := (e_{0,0}, e_{0,1}) = (g^{y_0}, \texttt{H}(\beta_0^{y_0}) \oplus m_0)$$
$$e_1 := (e_{1,0}, e_{1,1}) = (g^{y_1}, \texttt{H}(\beta_1^{y_1}) \oplus m_1)$$

   (d) sends $res = (e_0, e_1)$ to R.
6. *R-side Message Extraction:* $\texttt{Retreive}(res, req, pk) \rightarrow m_s$
   (a) sets $x = r_2 + r_1 \cdot (-1)^{s_2}$
   (b) retrieves the related message: $m_s = \texttt{H}((e_{s,0})^x) \oplus e_{s,1}$

Fig. 1: DQ-OT: Our 1-out-of-2 OT that supports query delegation. The input of R is a private binary index s and the input of S is a pair of messages (m_0, m_1). Note, $\texttt{SS}(.)$ is the share-generation algorithm, $\texttt{H}(.)$ is a hash function, and \$ denotes picking a value uniformly at random.

g^{a-x}. We also know that $\texttt{H}(.)$ is modeled as a random oracle and its output is indistinguishable from a random value. Thus, $\texttt{H}(\beta_{1-s}^{y_{1-s}}) \oplus m_{1-s}$ 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 5.2) holds.

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

1. initiates 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 $C', r' \xleftarrow{\$} \mathbb{Z}_p$.
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 5.2) holds.

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

1. initiates an empty view. It selects a random generator g and a large random prime number p.
2. picks two uniformly random values $s'_2 \xleftarrow{\$} \mathbb{U}$ and $C', r'_2 \xleftarrow{\$} \mathbb{Z}_p$, 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 values g and p have been picked uniformly at random in both models, they have identical distributions in the real and ideal models. Since r_2 and r'_2 have been picked uniformly at random from $\mathbb{Z}_p$, 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 $\mathbb{Z}_p$; therefore, they have identical distribution. Thus, the two views are computationally indistinguishable, i.e., Relation 2 w.r.t. P_2 (in Section 5.2) holds.

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

1. initiates an empty view. It chooses a random generator g and a large random prime number p.
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
   $C', r'_1 \xleftarrow{\$} \mathbb{Z}_p$, 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. Values g and p have been picked uniformly at random in both models. Hence, g and p in the real and ideal models have identical distributions (pair-wise).

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, $\delta_s, \delta_{1-s}, \delta'_0$, 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 $\mathtt{SS}(.)$ each share s_1 is indistinguishable from a random value s'_1, where $s'_1 \in \mathbb{U}$. Also, (r_1, C) and (r'_1, C') have identical distributions, as they are picked uniformly at random from $\mathbb{Z}_p$.

Hence, the two views are computationally indistinguishable, i.e., Relation 2 w.r.t. P_1 (in Section 5.2) holds. $\square$

5.5 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 [36], the random value a (i.e., the discrete logarithm of random value C) is inserted by receiver 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 [36].

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

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 recompute $\mathtt{H}(g^{y_s})^x$, as it knows the value of s, s_1, and s_2. Specifically, as Table 1 indicates, when:

- $\overbrace{(s = s_1 = s_2 = 0)}^{\text{Case 1}}$ or $\overbrace{(s = s_1 = 1 \wedge s_2 = 0)}^{\text{Case 2}}$, then R can set $x = r_2 + r_1$.
  - In Case 1, it holds $\mathtt{H}((g^{y_0})^x) = \mathtt{H}((g^{y_0})^{r_2+r_1}) = q$. Also, $e_0 = \mathtt{H}(\beta_0^{y_0}) \oplus m_0 = \mathtt{H}((g^{r_2+r_1})^{y_0}) \oplus m_0$. Thus, $q \oplus e_0 = m_0$.
  - In Case 2, it holds $\mathtt{H}((g^{y_1})^x) = \mathtt{H}((g^{y_1})^{r_2+r_1}) = q$. Moreover, $e_1 = \mathtt{H}(\beta_1^{y_1}) \oplus m_1 = \mathtt{H}((g^{r_2+r_1})^{y_1}) \oplus m_1$. Hence, $q \oplus e_1 = m_1$.

- $\overbrace{(s=0 \wedge s_1=s_2=1)}^{\text{Case 3}}$ or $\overbrace{(s=s_2=1 \wedge s_1=0)}^{\text{Case 4}}$, then R can set $x = r_2 - r_1$.
  - In Case 3, it holds $\mathtt{H}((g^{y_0})^x) = \mathtt{H}((g^{y_0})^{r_2-r_1}) = q$. On the other hand, $e_0 = \mathtt{H}(\beta_0^{y_0}) \oplus m_0 = \mathtt{H}((g^{r_2-r_1})^{y_0}) \oplus m_0$. Therefore, $q \oplus e_0 = m_0$.
  - In Case 4, it holds $\mathtt{H}((g^{y_1})^x) = \mathtt{H}((g^{y_1})^{r_2-r_1}) = q$. Also, $e_1 = \mathtt{H}(\beta_1^{y_1}) \oplus m_1 = \mathtt{H}((g^{r_2-r_1})^{y_1}) \oplus m_1$. 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.

6 Delegated-Unknown-Query OT

In certain cases, the receiver itself may not know the value of query s. Instead, the query is issued by a third-party query issuer (T). 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.

6.1 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: $\mathcal{F}_{\mathcal{DUQ}\text{–}\mathcal{OT}_1^2} : \big((m_0, m_1), \epsilon, \epsilon, \epsilon, s\big) \rightarrow (\epsilon, \epsilon, \epsilon, m_s, \epsilon)$. Next, we present a formal definition of $\mathcal{DUQ}\text{–}\mathcal{OT}_1^2$.

Definition 5 ($\mathcal{DUQ}\text{–}\mathcal{OT}_1^2$). *Let $\mathcal{F}_{\mathcal{DUQ}\text{–}\mathcal{OT}_1^2}$ be the functionality defined above. We assert that protocol Γ realizes $\mathcal{F}_{\mathcal{DUQ}\text{–}\mathcal{OT}_1^2}$ in the presence of 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 $\mathtt{Sim}$ in the ideal model, such that:*

$$\Big\{\mathtt{Sim}_S\big((m_0, m_1), \epsilon\big)\Big\}_{m_0,m_1,s} \stackrel{c}{\equiv} \Big\{\mathtt{View}_S^{\Gamma}\big((m_0, m_1), \epsilon, \epsilon, \epsilon, s\big)\Big\}_{m_0,m_1,s} \tag{4}$$

$$\Big\{\mathtt{Sim}_{P_i}(\epsilon, \epsilon)\Big\}_{m_0,m_1,s} \stackrel{c}{\equiv} \Big\{\mathtt{View}_{P_i}^{\Gamma}\big((m_0, m_1), \epsilon, \epsilon, \epsilon, s\big)\Big\}_{m_0,m_1,s} \tag{5}$$

$$\Big\{\mathtt{Sim}_T(s, \epsilon)\Big\}_{m_0,m_1,s} \stackrel{c}{\equiv} \Big\{\mathtt{View}_T^{\Gamma}\big((m_0, m_1), \epsilon, \epsilon, \epsilon, s\big)\Big\}_{m_0,m_1,s} \tag{6}$$

$$\Big\{\mathtt{Sim}_R\Big(\epsilon, \mathcal{F}_{\mathcal{DUQ}\text{–}\mathcal{OT}_1^2}\big((m_0, m_1), \epsilon, \epsilon, \epsilon, s\big)\Big)\Big\}_{m_0,m_1,s} \stackrel{c}{\equiv} \Big\{\mathtt{View}_R^{\Gamma}\big((m_0, m_1), \epsilon, \epsilon, \epsilon, s\big)\Big\}_{m_0,m_1,s} \tag{7}$$

for all i, $i \in \{1,2\}$. Since $\mathcal{DUQ}\text{–}\mathcal{OT}_1^2$ is a variant of $\mathcal{DQ}\text{–}\mathcal{OT}_1^2$, it also supports efficiency and SPC, as discussed in Section 5.2.

6.2 Protocol

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

Main Challenge to Overcome. One of the primary differences between DUQ-OT and previous OTs 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, 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 obtain 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.

An Overview. 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. Moreover, T selects a random value r_3 and sends it to R and S. Given the messages receives 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 $\mathcal{F}_{\mathcal{DUQ}\text{-}\mathcal{OT}^2_1}$ be the functionality defined in Section 6.1. If DL, CDH, and RO assumptions hold and random permutation $\pi(.)$ is secure, then DUQ-OT (presented in Figure 2) securely computes $\mathcal{F}_{\mathcal{DUQ}\text{-}\mathcal{OT}^2_1}$ in the presence of semi-honest adversaries, w.r.t. Definition 5.*

6.3 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.

Corrupt Receiver *R*. In the real execution, R's view is: $\mathtt{View}_R^{DUQ\text{-}OT}\big(m_0, m_1, \epsilon, \epsilon, \epsilon, s\big) = \{r_R, g, C, p, r_3, s_2, e'_0, e'_1, m_s\}$, 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 $\mathtt{Sim}_R$ which receives m_s from R.

1. initiates 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. selects a random generator g and a large random prime number p.
3. sets response $(\bar{e}'_0, \bar{e}'_1)$ as follows:
   - picks random values: $C', r'_1, r'_2, y'_0, y'_1 \xleftarrow{\$} \mathbb{Z}_p$, $r'_3 \xleftarrow{\$} \{0,1\}^\lambda$, $s' \xleftarrow{\$} \{0,1\}$, and $u \xleftarrow{\$} \{0,1\}^{\sigma+\lambda}$.
   - sets $x = r'_2 + r'_1 \cdot (-1)^{s'}$ and $\beta'_0 = g^x$.
   - sets $\bar{e}_0 = (g^{y'_0}, \mathtt{G}(\beta_0'^{y'_0}) \oplus (m_s || r'_3))$ 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.
4. appends $(g, C', p, r'_3, s', \bar{e}'_0, \bar{e}'_1, m_s)$ 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 holds for values (r_3, s_2) in the real model and (r'_3, s') in the ideal model, component-wise. Furthermore, because values g and p have been selected uniformly at random in both models, they have identical distributions in the real and ideal models.

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 $e'_i = e_{1-s} = (g^{y_{1-s}}, \mathtt{G}(\beta_{1-s}^{y_{1-s}}) \oplus (m_{1-s}||r_3))$, 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 $e_{1-s} = (g^{y_{1-s}}, \mathtt{G}(\beta_{1-s}^{y_{1-s}}) \oplus (m_{1-s}||r_3))$, 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 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 $\beta_{1-s}^{y_{1-s}}$ (i.e., the input of $\mathtt{G}(.)$), given $g^{y_{1-s}}$ and g^{a-x}. We know that $\mathtt{G}(.)$ is considered as a random oracle and its output is indistinguishable from a random value. Therefore, $\mathtt{G}(\beta_{1-s}^{y_{1-s}}) \oplus (m_{1-s}||r_3)$ 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

1. *S-side Initialization:* $\texttt{Init}(1^\lambda) \rightarrow pk$
   (a) chooses a sufficiently large prime number p.
   (b) selects random element $C \xleftarrow{\$} \mathbb{Z}_p$ and generator g.
   (c) publishes $pk = (C, p, g)$.
2. *R-side Delegation:* $\texttt{R.Request}(pk) \rightarrow req = (req_1, req_2)$
   (a) picks two uniformly random values: $r_1, r_2 \xleftarrow{\$} \mathbb{Z}_p$.
   (b) sends $req_1 = r_1$ to P_1 and $req_2 = r_2$ to P_2.
3. *T-side Query Generation:* $\texttt{T.Request}(1^\lambda, s, pk) \rightarrow (req'_1, req'_2, sp_S)$
   (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) picks a uniformly random value: $r_3 \xleftarrow{\$} \{0,1\}^\lambda$.
   (c) sends $req'_1 = s_1$ to P_1, $req'_2 = s_2$ to P_2. It also sends secret parameter $sp_S = r_3$ to S and $sp_R = (req'_2, sp_S)$ to R.
4. *P_2-side Query Generation:* $\texttt{P}_2\texttt{.GenQuery}(req_2, req'_2, pk) \rightarrow q_2$
   (a) computes a pair of partial queries:

$$\delta_{s_2} = g^{r_2}, \quad \delta_{1-s_2} = \frac{C}{g^{r_2}}$$

   (b) sends $q_2 = (\delta_0, \delta_1)$ to P_1.
5. *P_1-side Query Generation:* $\texttt{P}_1\texttt{.GenQuery}(req_1, req'_1, q_2, pk) \rightarrow q_1$
   (a) computes 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) sends $q_1 = (\beta_0, \beta_1)$ to S.
6. *S-side Response Generation:* $\texttt{GenRes}(m_0, m_1, pk, q_1, sp_S) \rightarrow res$
   (a) aborts if $C \neq \beta_0 \cdot \beta_1$.
   (b) picks two uniformly random values: $y_0, y_1 \xleftarrow{\$} \mathbb{Z}_p$.
   (c) computes a response pair (e_0, e_1) as follows:

$$e_0 := (e_{0,0}, e_{0,1}) = (g^{y_0}, \texttt{G}(\beta_0^{y_0}) \oplus (m_0 || r_3))$$
$$e_1 := (e_{1,0}, e_{1,1}) = (g^{y_1}, \texttt{G}(\beta_1^{y_1}) \oplus (m_1 || r_3))$$

   (d) randomly permutes the elements of the pair (e_0, e_1) as follows: $\pi(e_0, e_1) \rightarrow (e'_0, e'_1)$.
   (e) sends $res = (e'_0, e'_1)$ to R.
7. *R-side Message Extraction:* $\texttt{Retreive}(res, req, pk, sp_R) \rightarrow m_s$
   (a) sets $x = r_2 + r_1 \cdot (-1)^{s_2}$.
   (b) retrieves message m_s as follows. $\forall i, 0 \leq i \leq 1$:
      i. sets $y = \texttt{G}((e'_{i,0})^x) \oplus e'_{i,1}$.
      ii. calls $\texttt{parse}(\gamma, y) \rightarrow (u_1, u_2)$.
      iii. sets $m_s = u_1$, if $u_2 = r_3$.

Fig. 2: DUQ-OT: Our 1-out-of-2 OT that supports query delegation while preserving the privacy of query from R.

identical distribution in both models. Both C and C' have also been picked uniformly at random from $\mathbb{Z}_p$; therefore, they have identical distributions.

In the ideal model, $\bar{e}_0$ always contains 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 results 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 6.1) holds.

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. initiates 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 $C', r' \xleftarrow{\$} \mathbb{Z}_p, r'_3 \xleftarrow{\$} \{0,1\}^\lambda$.
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 6.1) holds.

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

1. initiates an empty view. It chooses a random generator g and a large random prime number p.
2. picks two uniformly random values $s'_2 \xleftarrow{\$} \mathbb{U}$ and $C', r'_2 \xleftarrow{\$} \mathbb{Z}_p$, 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. Values g and p have been selected uniformly at random in both models. Thus, they have identical distributions in the real and ideal models. 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 $\mathbb{Z}_p$. Therefore, they have identical distributions.

Thus, the two views are computationally indistinguishable, i.e., Relation 5 w.r.t. P_2 (in Section 6.1) holds.

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

1. initiates an empty view. It chooses a large random prime number p and a random generator g.
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 $C', r'_1 \xleftarrow{\$} \mathbb{Z}_p$, where $\mathbb{U}$ is the output range of $\texttt{SS}(.)$.

4. appends $s'_1, g, C', p, r'_1, \delta'_0, \delta'_1$ 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 $\mathbb{Z}_p$. Values g and p in the real and ideal models have identical distributions as they have been picked uniformly at random.

Hence, the two views are computationally indistinguishable, i.e., Relation 5 w.r.t. P_2 (in Section 6.1) holds.

Corrupt *T*. T's view can be easily simulated. It has input s, but it receives no messages from its counterparts and receives no output from the protocol. Thus, its real-world view is defined as $\texttt{View}_T^{\mathcal{DUQ}\text{-}\mathcal{OT}}\big((m_0, m_1), \epsilon, \epsilon, \epsilon, s\big) = \{r_T, g, C, p\}$, where r_T is the outcome of the internal random coin of T and is used to generate random values. Ideal-model $\texttt{Sim}_T$ initiates an empty view, picks r'_T, g, C, and p uniformly at random, and adds them 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, g, C, p and r'_T, g, C, p have identical distributions.

Thus, the two views are computationally indistinguishable, i.e., Relation 6 (in Section 6.1) holds. □

7 Delegated-Query Multi-Receiver Oblivious Transfers

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

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 $0 \leq v \leq z-1$. Both variants (in addition to offering the efficiency, SPC, and 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.

7.1 Delegated-Query Multi-Receiver OT

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

Security Definition. The functionality that $\mathcal{DQ}^{\mathcal{MR}}\text{–}\mathcal{OT}^2_1$ computes takes as input (i) a vector of messages $\boldsymbol{m} = [(m_{0,0}, m_{1,0}), \dots, (m_{0,z-1}, m_{1,z-1})]$ 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, z to P_1, ϵ to 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: $\mathcal{F}_{\mathcal{DQ}^{\mathcal{MR}}\text{–}\mathcal{OT}^2_1} : \big([(m_{0,0}, m_{1,0}), \dots, (m_{0,z-1}, m_{1,z-1})], v, \epsilon, s\big) \rightarrow (\epsilon, z, \epsilon, m_{s,v})$, where $v \in \{0, \dots, z-1\}$. Next, we present a formal definition of $\mathcal{DQ}^{\mathcal{MR}}\text{–}\mathcal{OT}^2_1$.

Definition 6 **($\mathcal{DQ}^{\mathcal{MR}}\text{–}\mathcal{OT}^2_1$).** *Let $\mathcal{F}_{\mathcal{DQ}^{\mathcal{MR}}\text{–}\mathcal{OT}^2_1}$ be the functionality defined above. We say that protocol Γ realizes $\mathcal{F}_{\mathcal{DQ}^{\mathcal{MR}}\text{–}\mathcal{OT}^2_1}$ in the presence of 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:*

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

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

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

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

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

Strawman Approaches. One may consider using one of the following ideas in the multi-receiver setting:

1. *Using an existing single-receiver OT, e.g., in [26]*, employing one of the following approaches:
   - *Approach 1*: receiver 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 receiver 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, this approach 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 leaks to S the index v of the record that R_j is interested.
2. *Using an existing multi-receiver OT, e.g., in [6]*. This will also come with a privacy cost. The existing multi-receiver OTs reveal the entire database's size to each receiver (as discussed in Section 4.2). In this scenario, a receiver can learn the number of private records other companies have in the same database. This type of leakage is particularly significant, especially when coupled with specific auxiliary information.

Hence, a fully private multi-receiver OT is necessary to ensure user privacy in real-world cloud settings.

Protocol. We present DQ$^{\text{MR}}$–OT that realizes $\mathcal{DQ}^{\mathcal{MR}}$–$\mathcal{OT}_1^2$. We build DQ$^{\text{MR}}$–OT upon DQ-OT (presented in Figure 1). DQ$^{\text{MR}}$–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 3.4) 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 $\mathtt{H}$ be a hash function modelled as a RO (as defined in Section 3.1), where its output size is δ-bit. 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 (i) g^{y_0} and g^{y_1} form random elements of $\mathbb{G}$ and (ii) $\mathtt{H}(\beta_0^{y_0}) \oplus m_0$ and $\mathtt{H}(\beta_1^{y_1}) \oplus m_1$ from random elements from $\{0,1\}^{\sigma}$, except for a negligible probability $\mu(\lambda)$.*

Proof. First, we focus on the first element of pairs $(g^{y_0}, \mathtt{H}(\beta_0^{y_0}) \oplus m_0)$ and $(g^{y_1}, \mathtt{H}(\beta_1^{y_1}) \oplus m_1)$. 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 element 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 $\mathtt{H}(.)$), 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 $\mathtt{H}(.)$ has been considered as a random oracle and its output is indistinguishable from a random value. Therefore, $\mathtt{H}(\beta_0^{y_0}) \oplus m_0$ and $\mathtt{H}(\beta_1^{y_1}) \oplus m_1$ are indistinguishable from random elements of $\{0,1\}^{\delta}$, except for a negligible probability, $\mu(\lambda)$. □

The main idea behind the design of DQ$^{\text{MR}}$–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.

Consider the case where one of the receivers, say R_j, wants to send a query. In this case, within DQ$^{\text{MR}}$–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 6 in Appendix B presents DQ$^{\text{MR}}$–OT in detail.

7.2 Delegated-Unknown-Query Multi-Receiver OT

The second variant $\mathcal{DUQ}^{\mathcal{MR}}$–$\mathcal{OT}^2_1$ can be considered as a variant of $\mathcal{DUQ}$–$\mathcal{OT}^2_1$. 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).

Security Definition. The functionality that $\mathcal{DUQ}^{\mathcal{MR}}$–$\mathcal{OT}^2_1$ computes takes as input (i) a vector of messages $\boldsymbol{m} = [(m_{0,0}, m_{1,0}), \ldots, (m_{0,z-1}, m_{1,z-1})]$ 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 and T, z to P_1, ϵ to P_2, 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: $\mathcal{F}_{\mathcal{DUQ}^{\mathcal{MR}}\text{–}\mathcal{OT}^2_1} : \big([(m_{0,0}, m_{1,0}), \ldots, (m_{0,z-1}, m_{1,z-1})], (v, s, z), \epsilon, \epsilon, \epsilon\big) \rightarrow (\epsilon, \epsilon, z, \epsilon, m_{s,v})$, where $v \in \{0, \ldots, z-1\}$. Next, we present a formal definition of $\mathcal{DUQ}^{\mathcal{MR}}$–$\mathcal{OT}^2_1$.

Definition 7 **($\mathcal{DUQ}^{\mathcal{MR}}$–$\mathcal{OT}^2_1$).** *Let $\mathcal{F}_{\mathcal{DUQ}^{\mathcal{MR}}\text{–}\mathcal{OT}^2_1}$ be the functionality defined above. We assert that protocol Γ realizes $\mathcal{F}_{\mathcal{DUQ}^{\mathcal{MR}}\text{–}\mathcal{OT}^2_1}$ in the presence of 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* `Sim` *in the ideal model, such that:*

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

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

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

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

where $\boldsymbol{m} = [(m_{0,0}, m_{1,0}), \ldots, (m_{0,z-1}, m_{1,z-1})]$, $out_1 = z$, $out_2 = \epsilon$, and $\forall i$, $i \in \{1, 2\}$.

Protocol. We proceed to present DUQ$^{\text{MR}}$–OT that realizes $\mathcal{DUQ}^{\mathcal{MR}}\text{–}\mathcal{OT}^2_1$. We build DQ$^{\text{MR}}$–OT upon protocol DUQ-OT (presented in Figure 2). DUQ$^{\text{MR}}$–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 $\boldsymbol{a} = [m_0, m_1, \ldots, m_{z-1}]^T$ where T denotes transpose, without revealing which plaintext record we want to fetch, we can perform as follows:

1. set vector $\boldsymbol{b} = [b_0, \ldots, b_{z-1}]$, where all b_is are set to zero except for v-th element b_v which is set to 1.
2. encrypt each element of $\boldsymbol{b}$ using additively homomorphic encryption, e.g., Paillier encryption. Let $\boldsymbol{b'}$ be the vector of the encrypted elements.
3. send $\boldsymbol{b'}$ to the database holder which performs $\boldsymbol{b'} \times \boldsymbol{a}$ homomorphically, and sends us the single result res.
4. decrypt res to discover m_v.[8]

In our DUQ$^{\text{MR}}$–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. Figure 3 presents DUQ$^{\text{MR}}$–OT in detail.

Theorem 3. *Let $\mathcal{F}_{\mathcal{DUQ}^{\mathcal{MR}}\text{–}\mathcal{OT}^2_1}$ be the functionality defined in Section 7.2. If DL, CDH, and RO assumptions hold and additive homomorphic encryption satisfies IND-CPA, then DUQ^{MR}–OT (presented in Figure 3) securely computes $\mathcal{F}_{\mathcal{DUQ}^{\mathcal{MR}}\text{–}\mathcal{OT}^2_1}$ in the presence of semi-honest adversaries, w.r.t. Definition 7.*

7.3 DUQ$^{\text{MR}}$–OT's Security Proof

We prove the security of DUQ$^{\text{MR}}$–OT, i.e., Theorem 3.

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

Corrupt R. In the real execution, R's view is: $\texttt{View}_R^{\text{DUQ}^{\text{MR}}\text{–OT}}\big(\boldsymbol{m}, (v, s, z)\ \epsilon, \epsilon, \epsilon\big) = \{r_R, g, C, p, r_3, s_2, o_0, o_1, m_{s,v}\}$, where g is a random generator, p is a large random prime number, $o_0 := (o_{0,0}, o_{0,1})$, $o_1 := (o_{1,0}, o_{1,1})$, $C = g^a$ 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. initiates an empty view and appends uniformly random coin r'_R to it, where r'_R will be used to generate R-side randomness. It selects a large random prime number p and a random generator g.
2. sets response as follows:
   - picks random values: $C', r'_1, r'_2, y'_0, y'_1 \xleftarrow{\$} \mathbb{Z}_p$, $r'_3 \xleftarrow{\$} \{0,1\}^\lambda$, $s' \xleftarrow{\$} \{0,1\}$, and $u \xleftarrow{\$} \{0,1\}^{\sigma+\lambda}$.
   - 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 $(g, C', p, r'_3, s', \bar{o}'_0, \bar{o}'_1, m_{s,v})$ to the view and outputs the view.

[8] Such a technique was previously used by Devet *et al.* [16] in the "private information retrieval" research line.

1. *S-side Initialization:* $\mathtt{Init}(1^{\lambda}) \rightarrow pk$

   Chooses a large random prime number p, random element $C \stackrel{\$}{\leftarrow} \mathbb{Z}_p$, and generator g. Publishes $pk = (C, p, g)$.
2. *R_j-side One-off Setup:* $\mathtt{R.Setup}(1^{\lambda}) \rightarrow (pk_j, sk_j)$

   Generates a key pair for the homomorphic encryption, by calling $\mathtt{KGen}(1^{\lambda}) \rightarrow (sk_j, pk_j)$. Sends pk_j to T and S.
3. *T-side One-off Setup:* $\mathtt{T.Setup}(z, pk_j) \rightarrow \boldsymbol{w}_j$
   (a) initializes an empty vector $\boldsymbol{w}_j = []$ of size z.
   (b) creates 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. sets $d = 1$, if $t = v$; sets $d = 0$, otherwise.
      ii. appends $\mathtt{Enc}(pk_j, d)$ to $\boldsymbol{w}_j$.
   (c) sends $\boldsymbol{w}_j$ to P_1.
4. *R_j-side Delegation:* $\mathtt{R.Request}(pk) \rightarrow req = (req_1, req_2)$
   (a) picks random values: $r_1, r_2 \stackrel{\$}{\leftarrow} \mathbb{Z}_p$.
   (b) sends $req_1 = r_1$ to P_1 and $req_2 = r_2$ to P_2.
5. *T-side Query Generation:* $\mathtt{T.Request}(1^{\lambda}, s, pk) \rightarrow (req'_1, req'_2, sp_S)$
   (a) splits 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) picks a uniformly random value: $r_3 \stackrel{\$}{\leftarrow} \{0,1\}^{\lambda}$.
   (c) sends $req'_1 = s_1$ to P_1, $req'_2 = s_2$ to P_2. It sends secret parameter $sp_S = r_3$ to S and $sp_R = (req'_2, sp_S)$ to R.
6. *P_2-side Query Generation:* $\mathtt{P_2.GenQuery}(req_2, req'_2, pk) \rightarrow q_2$
   (a) computes queries: $\delta_{s_2} = g^{r_2}, \quad \delta_{1-s_2} = \frac{C}{g^{r_2}}$.
   (b) sends $q_2 = (\delta_0, \delta_1)$ to P_1.
7. *P_1-side Query Generation:* $\mathtt{P_1.GenQuery}(req_1, req'_1, q_2, pk) \rightarrow q_1$
   (a) computes queries as: $\beta_{s_1} = \delta_0 \cdot g^{r_1}, \beta_{1-s_1} = \frac{\delta_1}{g^{r_1}}$
   (b) sends $q_1 = (\beta_0, \beta_1)$ to S.
8. *S-side Response Generation:* $\mathtt{GenRes}(m_{0,0}, m_{1,0}, \ldots, m_{0,z-1}, m_{1,z-1}, pk, q_1, sp_S) \rightarrow res$
   (a) aborts if $C \neq \beta_0 \cdot \beta_1$.
   (b) computes a response as follows. $\forall t, 0 \leq t \leq z-1:$
      i. picks two random values $y_{0,t}, y_{1,t} \stackrel{\$}{\leftarrow} \mathbb{Z}_p$.
      ii. computes 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 permutes 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) sends $res = (e'_{0,0}, e'_{1,0}), \ldots, (e'_{0,z-1}, e'_{1,z-1})$ to P_1.
9. *P_1-side Oblivious Filtering:* $\mathtt{OblFilter}(res, pk_j, \boldsymbol{w}_j) \rightarrow res'$
   (a) compresses S's response using vector $\boldsymbol{w}_j$ as follows. $\forall i, i', 0 \leq i, i' \leq 1:$
      $o_{i,i'} = (e'_{i,i',0} \overset{H}{\times} \boldsymbol{w}_j[0]) \overset{H}{+} \ldots \overset{H}{+} (e'_{i,i',z-1} \overset{H}{\times} \boldsymbol{w}_j[z-1])$.
   (b) sends $res' = (o_{0,0}, o_{0,1}), (o_{1,0}, o_{1,1})$ to R_j.
10. *R-side Message Extraction:* $\mathtt{Retreive}(res', req, sk_j, pk, sp_R) \rightarrow m_s$
   (a) decrypts the response from P_1 as follows. $\forall i, i', 0 \leq i, i' \leq 1: \mathtt{Dec}(sk_j, o_{i,i'}) \rightarrow o'_{i,i'}$.
   (b) sets $x = r_2 + r_1 \cdot (-1)^{s_2}$.
   (c) retrieves message $m_{s,v}$ as follows. $\forall i, 0 \leq i \leq 1:$
      i. sets $y = \mathtt{G}((o'_{i,0})^x) \oplus o'_{i,1}$.
      ii. calls $\mathtt{parse}(\gamma, y) \rightarrow (u_1, u_2)$.
      iii. sets $m_{s,v} = u_1$, if $u_2 = r_3$.

Fig. 3: Phases 1–8 of DUQ$^{\text{MR}}$–OT.

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. Moreover, values g and p in the real and ideal models, as they have been picked uniformly at random. 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 $e'_i = e_{1-s} = (g^{y_{1-s,v}}, \mathtt{G}(\beta_{1-s}^{y_{1-s,v}}) \oplus (m_{1-s,v}||r_3))$, 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 $\mathtt{Sim}_R$ 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 $e_{1-s} = (g^{y_{1-s,v}}, \mathtt{G}(\beta_{1-s}^{y_{1-s,v}}) \oplus (m_{1-s,v}||r_3))$, 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 $\mathtt{G}(.)$ is considered a random oracle and its output is indistinguishable from a random value. Therefore, $\mathtt{G}(\beta_{1-s}^{y_{1-s,v}}) \oplus (m_{1-s,v}||r_3)$ 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 $\bar{o}_{1,0} = \mathtt{Enc}(pk, g^{y'_1})$ and $\bar{o}_{1,1} = \mathtt{Enc}(pk, u)$ in the ideal model and ciphertexts $o_{1-s,0} = \mathtt{Enc}(pk, g^{y_{1-s,v}})$ and $o_{1-s,1} = \mathtt{Enc}(pk, \mathtt{G}(\beta_{1-s}^{y_{1-s,v}}) \oplus (m_{1-s,v}||r_3))$ in the real model have identical distributions due to IND-CPA property of the additive homomorphic encryption. Furthermore, (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 results in $(\bar{o}'_0, \bar{o}'_1)$ and (o_0, o_1) respectively. Moreover, ciphertexts $\bar{o}_{0,0} = \mathtt{Enc}(pk, g^{y'_0})$ and $\bar{o}_{0,1} = \mathtt{Enc}(pk,\ \mathtt{G}(\beta_0'^{y'_0}) \oplus (m_{s,v}||r'_3)))$ in the ideal model and ciphertexts $o_{s,0} = \mathtt{Enc}(pk, g^{y_{s,v}})$ and $o_{s,1} = \mathtt{Enc}(pk, \mathtt{G}(\beta_s^{y_{s,v}}) \oplus (m_{s,v}||r_3))$ 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 7.2) 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.

Corrupt S. This case is identical to the corrupt S in the proof of DUQ-OT (in Section 6.3) 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 $\mathtt{Sim}_S$ 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 DUQ-OT. $\mathtt{Sim}_S$ 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 Section 6.3) 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 7.2) holds.

Corrupt P_2. This case is identical to the corrupt P_2 case in the proof of DUQ-OT. Thus, Relation 13 (in Section 7.2) holds.

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

1. initiates an empty view. It selects a large random prime number p and a random generator g.

2. picks two random values $\delta'_0, \delta'_1 \xleftarrow{\$} \mathbb{Z}_p$.
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 $s'_1 \xleftarrow{\$} \mathbb{U}$ and $C', r'_1 \xleftarrow{\$} \mathbb{Z}_p$, where $\mathbb{U}$ is the output range of $\texttt{SS}(.)$.
5. picks z pairs of random values as follows $(a_{0,0}, a_{1,0}), ..., (a_{0,z-1},\ a_{1,z-1}) \xleftarrow{\$} \mathbb{Z}_p$.
6. appends $s'_1, g, C', p, r'_1, \delta'_0, \delta'_1$ 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 Section 6.3) 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 7.2) holds.

Corrupt $\boldsymbol{T}$. 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 7.2) holds. □

8 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 7.2. 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 products. It sends the result (which is now constant with regard to n) to R, which decrypts the response and retrieves the result m_s.

Next, we will present a generic OT's syntax, and introduce the generic compiler using the syntax.

8.1 Syntax of a Conventional OT

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

- $\texttt{S.Init}(1^\lambda) \rightarrow pk$: a probabilistic algorithm run by S. It takes as input security parameter 1^λ and returns a public key pk.
- $\texttt{R.GenQuery}(pk, n, s) \rightarrow (q, sp)$: a probabilistic algorithm run by R. It takes as input pk, the total number of messages n, and a secret index s. It returns a query (vector) q and a secret parameter sp.
- $\texttt{S.GenRes}(m_0, \ldots, m_{n-1}, pk, q) \rightarrow res$: a probabilistic algorithm run by S. It takes as input pk and q. It generates an encoded response (vector) res.
- $\texttt{R.Retreive}(res, q, sp, pk, s) \rightarrow m_s$: a deterministic algorithm run by S. It takes as input res, q, sp, pk, and s. It returns message m_s.

The functionality that a 1-out-of-n OT computes can be defined as: $\mathcal{F}_{\mathcal{OT}_1^n} : \big((m_0, \ldots, m_{n-1}), s\big) \rightarrow (\epsilon, m_s)$. Informally, 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 $(m_0, \ldots, m_{n-1})$. We refer readers to [22] for further discussion on 1-out-of-n OT.

8.2 The Compiler

We present the compiler in detail in Figure 4. We highlight that in the case where each $e_i \in res$ contains more than one value, e.g., $e_i = [e_{0,i}, \ldots, e_{w-1,i}]$ (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 $e_{j,0} \overset{H}{\times} \boldsymbol{b}'[0] \overset{H}{+} \ldots \overset{H}{+} e_{j,n-1} \overset{H}{\times} \boldsymbol{b}'[n-1]$, for all j, $0 \leq j \leq w-1$. In this case, only w elements are sent to R.

1. *S-side Initialization:* $\texttt{Init}(1^\lambda) \rightarrow pk$
   This phase involves S.
   (a) calls $\texttt{S.Init}(1^\lambda) \rightarrow pk$.
   (b) publishes pk.
2. R-side Setup: $\texttt{Setup}(1^\lambda) \rightarrow (sk_R, pk_R)$
   This phase involves R.
   (a) calls $\texttt{KGen}(1^\lambda) \rightarrow (sk_R, pk_R)$.
   (b) publishes pk_R.
3. R-side Query Generation: $\texttt{GenQuery}(pk_R, n, s) \rightarrow (q, sp, \boldsymbol{b}')$
   This phase involves R.
   (a) calls $\texttt{R.GenQuery}(pk, n, s) \rightarrow (q, sp)$.
   (b) constructs a vector $\boldsymbol{b} = [b_0, \ldots, b_{n-1}]$, 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, $\forall 0 \leq i \leq n-1 : b'_i = \texttt{Enc}(pk_R, b_i)$. Let $\boldsymbol{b}'$ be the vector of the encrypted elements.
   (c) sends q and $\boldsymbol{b}'$ to S and locally stores sp.
4. S-side Response Generation: $\texttt{GenRes}(m_0, \ldots, m_{n-1}, pk, pk_R, q, \boldsymbol{b}') \rightarrow res$
   This phase involves S.
   (a) calls $\texttt{S.GenRes}(m_0, \ldots, m_{n-1}, pk, q) \rightarrow res$. Let $res = [e_0, \ldots, e_{n-1}]$.
   (b) compresses the response using vector $\boldsymbol{b}'$ as follows. $\forall i, 0 \leq i \leq n-1$:
   $$e = (e_0 \overset{H}{\times} \boldsymbol{b}'[0]) \overset{H}{+} \ldots \overset{H}{+} (e_{n-1} \overset{H}{\times} \boldsymbol{b}'[n-1])$$
   (c) sends $res = e$ to R.
5. R-side Message Extraction. $\texttt{Retreive}(res, q, sp, pk, sk_R, s) \rightarrow m_s$
   This phase involves R.
   (a) calls $\texttt{Dec}(sk_R, res) \rightarrow res'$.
   (b) calls $\texttt{R.Retreive}(res', q, sp, pk, s) \rightarrow m_s$.

Fig. 4: A compiler that turns a 1-out-of-n OT with response size $O(n)$ to a 1-out-of-n OT with response size $O(1)$.

Theorem 4. *Let $\mathcal{F}_{\mathcal{OT}_1^n}$ be the functionality defined above. If $\mathcal{OT}_1^n$ is secure and additive homomorphic encryption meets IND-CPA, then generic OT with constant size response (presented in Figure 4) (i) securely computes $\mathcal{F}_{\mathcal{OT}_1^n}$ in the presence of semi-honest adversaries and (ii) offers $O(1)$ response size, w.r.t. the total number of messages n.*

8.3 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). $\square$

9 Conclusion

OT is a crucial privacy-preserving technology. OTs have found extensive applications in designing secure Multi-Party Computation (MPC) protocols [51], Federated Learning (FL) [50], and in accessing sensitive field elements of remote private databases while preserving privacy [6]. In this work, we have identified various gaps both in the privacy of databases and in the OT research. We proposed several novel OTs to address these gaps. We have presented a few real-world applications of the proposed OTs, while also formally defining and proving their security within the simulation-based model.

References

1. Aiello, W., Ishai, Y., Reingold, O.: Priced oblivious transfer: How to sell digital goods. In: EUROCRYPT (2001)
2. Arora, S., Beams, A., Chatzigiannis, P., Meiser, S., Patel, K., Raghuraman, S., Rindal, P., Shah, H., Wang, Y., Wu, Y., et al.: Privacy-preserving financial anomaly detection via federated learning & multi-party computation. arXiv preprint (2023)
3. Asharov, G., Lindell, Y., Schneider, T., Zohner, M.: More efficient oblivious transfer and extensions for faster secure computation. In: CCS'13 (2013)
4. Blakley, G.R.: One time pads are key safegaurding schemes, not cryptosystems. fast key safeguarding schemes (threshold schemes) exist. In: IEEE S&P (1980)
5. Camenisch, J., Dubovitskaya, M., Enderlein, R.R., Neven, G.: Oblivious transfer with hidden access control from attribute-based encryption. In: SCN (2012)
6. Camenisch, J., Dubovitskaya, M., Neven, G.: Oblivious transfer with access control. In: CCS (2009)
7. Camenisch, J., Dubovitskaya, M., Neven, G.: Unlinkable priced oblivious transfer with rechargeable wallets. In: FC (2010)
8. Camenisch, J., Dubovitskaya, M., Neven, G., Zaverucha, G.M.: Oblivious transfer with hidden access control policies. In: PKC (2011)
9. Camenisch, J., Neven, G., Shelat, A.: Simulatable adaptive oblivious transfer. In: Advances in Cryptology - EUROCRYPT (2007)
10. Canetti, R.: Towards realizing random oracles: Hash functions that hide all partial information. IACR Cryptol. ePrint Arch. (1997)
11. Chen, Y., Chou, J., Hou, X.: A novel k-out-of-n oblivious transfer protocols based on bilinear pairings. IACR Cryptol. ePrint Arch. (2010)
12. Chu, C., Tzeng, W.: Efficient k-out-of-n oblivious transfer schemes with adaptive and non-adaptive queries. In: PKC (2005)

13. Corniaux, C.L.F., Ghodosi, H.: A verifiable 1-out-of-n distributed oblivious transfer protocol. IACR Cryptol. ePrint Arch. (2013)
14. Council, G.M.: The dialogue leading to a decision (2020), t.ly/UA_Yn
15. Department of Justice–U.S. Attorney's Office: Former jp morgan chase bank employee sentenced to four years in prison for selling customer account information (2018)
16. Devet, C., Goldberg, I., Heninger, N.: Optimally robust private information retrieval. In: USENIX Security (2012)
17. Diffie, W., Hellman, M.E.: New directions in cryptography. IEEE Trans. Inf. Theory (1976)
18. Dong, C., Chen, L., Wen, Z.: When private set intersection meets big data: an efficient and scalable protocol. In: CCS (2013)
19. DrivenData: U.S. PETs prize challenge–transforming financial crime prevention (2023), https://www.drivendata.org/competitions/98/nist-federated-learning-1/page/524/
20. Even, S., Goldreich, O., Lempel, A.: A randomized protocol for signing contracts. Commun. ACM (1985)
21. General Medical Council: Withholding information from patients (2022)
22. Goldreich, O.: The Foundations of Cryptography - Volume 2, Basic Applications. Cambridge University Press (2004)
23. Green, M., Hohenberger, S.: Universally composable adaptive oblivious transfer. In: ASIACRYPT (2008)
24. Harnik, D., Ishai, Y., Kushilevitz, E.: How many oblivious transfers are needed for secure multiparty computation? In: CRYPTO (2007)
25. Henecka, W., Schneider, T.: Faster secure two-party computation with less memory. In: CCS (2013)
26. Ishai, Y., Kilian, J., Nissim, K., Petrank, E.: Extending oblivious transfers efficiently. In: CRYPTO, (2003)
27. Jarecki, S., Liu, X.: Efficient oblivious pseudorandom function with applications to adaptive OT and secure computation of set intersection. In: TCC (2009)
28. Katz, J., Lindell, Y.: Introduction to Modern Cryptography, Second Edition. CRC Press (2014)
29. Knuth, D.E.: The Art of Computer Programming, Volume II: Seminumerical Algorithms, 2nd Edition. Addison-Wesley (1981)
30. Leigh, D., Ball, J., Garside, J., Pegg, D.: Hsbc files timeline: From swiss bank leak to fallout. The Guardian (2015)
31. Libert, B., Ling, S., Mouhartem, F., Nguyen, K., Wang, H.: Adaptive oblivious transfer with access control from lattice assumptions. Theoretical Computer Science (2021)
32. Liu, M., Hu, Y.: Universally composable oblivious transfer from ideal lattice. Frontiers Comput. Sci. (2019)
33. Milmo, D., Hern, A.: BA, Boots and BBC cyber-attack: who is behind it and what happens next? (2023)
34. Naor, M., Pinkas, B.: Oblivious transfer and polynomial evaluation. In: STOC (1999)
35. Naor, M., Pinkas, B.: Distributed oblivious transfer. In: ASIACRYPT. Springer (2000)
36. Naor, M., Pinkas, B.: Efficient oblivious transfer protocols. In: SODA (2001)
37. Nielsen, J.B.: Extending oblivious transfers efficiently - how to get robustness almost for free. IACR Cryptol. ePrint Arch. (2007)
38. Osborne, C.: Salesforce warns customers of data leak caused by api error (2018), https://www.zdnet.com/article/salesforce-warns-customers-of-data-leak-caused-by-api-error/
39. Paillier, P.: Public-key cryptosystems based on composite degree residuosity classes. In: EUROCRYPT (1999)
40. Pfleeger, C.P.: Security in computing. Prentice-Hall, Inc. (1988)
41. PwC: Accelerating transformation with industry cloud (2023), https://www.pwc.com/gx/en/news-room/analyst-citations/2023/idc-infobrief-pwc-industry-cloud-2023.html
42. Rabin, M.: How to exchange secrets with oblivious transfer (1981)
43. Ren, Z., Yang, L., Chen, K.: Improving availability of vertical federated learning: Relaxing inference on non-overlapping data. ACM Trans. Intell. Syst. (2022)
44. Tara Seals: Salesforce.com warns marketing customers of data leakage snafu (2018), https://threatpost.com/salesforce-com-warns-marketing-customers-of-data-leakage-snafu/134703
45. Tzeng, W.: Efficient 1-out-n oblivious transfer schemes. In: Naccache, D., Paillier, P. (eds.) PKC (2002)
46. United States Attorney's Office: Ex-morgan stanley adviser pleads guilty in connection with data breach (2015), https://www.reuters.com/article/idUSKCN0RL229
47. United States Attorney's Office: Former citibank employee pleads guilty to credit card fraud (2017), https://www.justice.gov/usao-mdfl/pr/former-citibank-employee-pleads-guilty-credit-card-fraud
48. Xu, G., Li, H., Zhang, Y., Xu, S., Ning, J., Deng, R.H.: Privacy-preserving federated deep learning with irregular users. IEEE Trans. Dependable Secur. Comput. (2022)
49. Yadav, V.K., Andola, N., Verma, S., Venkatesan, S.: A survey of oblivious transfer protocol. ACM Comput. Surv. (2022)

50. Yang, Q., Liu, Y., Chen, T., Tong, Y.: Federated machine learning: Concept and applications. ACM Trans. Intell. Syst. Technol. (2019)
51. Yao, A.C.: Protocols for secure computations (extended abstract). In: 23rd Annual Symposium on Foundations of Computer Science (1982)
52. Yao, G., Feng, D.: Proxy oblivious transfer protocol. In: International Conference on Availability, Reliability and Security, ARES (2006)
53. Zhang, B., Lipmaa, H., Wang, C., Ren, K.: Practical fully simulatable oblivious transfer with sublinear communication. In: FC (2013)
54. Zhao, S., Song, X., Jiang, H., Ma, M., Zheng, Z., Xu, Q.: An efficient outsourced oblivious transfer extension protocol and its applications. Secur. Commun. Networks (2020)

A The Original OT of Naor and Pinkas

Figure 5 restates the original OT of Naor and Pinkas [36, pp. 450, 451].

1. *S-side Initialization:* $\texttt{S.Init}(1^{\lambda}) \rightarrow pk$
   (a) chooses a random large prime number p (where $\log_2 p = \lambda$), a random element $C \overset{\$}{\leftarrow} \mathbb{Z}_p$ and random generator g.
   (b) publishes $pk = (C, g, p)$.
2. *R-side Query Generation:* $\texttt{R.GenQuery}(pk, s) \rightarrow (q, sp)$
   (a) picks a random value $r \overset{\$}{\leftarrow} \mathbb{Z}_p \setminus \{0\}$ and sets $sp = r$.
   (b) sets $\beta_s = g^r$ and $\beta_{1-s} = \frac{C}{\beta_s}$.
   (c) sends $q = \beta_0$ to S and locally stores sp.
3. *S-side Response Generation:* $\texttt{S.GenRes}(m_0, m_1, pk, q) \rightarrow res$
   (a) computes $\beta_1 = \frac{C}{\beta_0}$.
   (b) chooses two random values, $y_0, y_1 \overset{\$}{\leftarrow} \mathbb{Z}_p$.
   (c) encrypts the elements of the pair (m_0, m_1) as follows:
   $$e_0 := (e_{0,0}, e_{0,1}) = (g^{y_0}, \texttt{H}(\beta_0^{y_0}) \oplus m_0)$$
   $$e_1 := (e_{1,0}, e_{1,1}) = (g^{y_1}, \texttt{H}(\beta_1^{y_1}) \oplus m_1)$$
   (d) sends $res = (e_0, e_1)$ to R.
4. *R-side Message Extraction:* $\texttt{R.Retreive}(res, sp, pk, s) \rightarrow m_s$
   – retrieves the related message m_s by computing: $m_s = \texttt{H}((e_{s,0})^r) \oplus e_{s,1}$

Fig. 5: Original OT proposed by Naor and Pinkas [36, pp. 450, 451]. In this protocol, the input of R is a private binary index s and the input of S is a pair of private messages (m_0, m_1).

B DQ$^{\text{MR}}$–OT in more Detail

Figure 6 presents the DQ$^{\text{MR}}$–OT that realizes $\mathcal{DQ}^{\mathcal{MR}}\text{–}\mathcal{OT}_1^2$.

Theorem 5. *Let $\mathcal{F}_{\mathcal{DQ}^{\mathcal{MR}}\text{-}\mathcal{OT}_1^2}$ be the functionality defined in Section 7.1. If DL, CDH, and RO assumptions hold, then DQ^{MR}–OT (presented in Figure 6) securely computes $\mathcal{F}_{\mathcal{DQ}^{\mathcal{MR}}\text{-}\mathcal{OT}_1^2}$ in the presence of semi-honest adversaries, w.r.t. Definition 6.*

1. *S-side Initialization:* $\texttt{Init}(1^\lambda) \rightarrow pk$
   (a) chooses a sufficiently large prime number p.
   (b) selects random element $C \xleftarrow{\$} \mathbb{Z}_p$ and generator g.
   (c) publishes $pk = (C, p, g)$.
2. *R_j-side Delegation:* $\texttt{R.Request}(1^\lambda, s, pk) \rightarrow req = (req_1, req_2)$
   (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) picks two uniformly random values: $r_1, r_2 \xleftarrow{\$} \mathbb{Z}_p$.
   (c) sends $req_1 = (s_1, r_1)$ to P_1 and $req_2 = (s_2, r_2)$ to P_2.
3. *P_2-side Query Generation:* $\texttt{P}_2\texttt{.GenQuery}(req_2, , pk) \rightarrow q_2$
   (a) computes a pair of partial queries:

$$\delta_{s_2} = g^{r_2}, \quad \delta_{1-s_2} = \frac{C}{g^{r_2}}$$

   (b) sends $q_2 = (\delta_0, \delta_1)$ to P_1.
4. *P_1-side Query Generation:* $\texttt{P}_1\texttt{.GenQuery}(req_1, q_2, pk) \rightarrow q_1$
   (a) computes 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) sends $q_1 = (\beta_0, \beta_1)$ to S.
5. *S-side Response Generation:* $\texttt{GenRes}(m_{0,0}, m_{1,0}, \ldots, m_{0,z-1}, m_{1,z-1}, pk, q_1) \rightarrow res$
   (a) aborts if $C \neq \beta_0 \cdot \beta_1$.
   (b) computes a response as follows. $\forall t, 0 \leq t \leq z-1$:
      i. picks two random values $y_{0,t}, y_{1,t} \xleftarrow{\$} \mathbb{Z}_p$.
      ii. computes response:

$$e_{0,t} := (e_{0,0,t}, e_{0,1,t}) = (g^{y_{0,t}}, \texttt{H}(\beta_0^{y_{0,t}}) \oplus m_{0,t})$$

$$e_{1,t} := (e_{1,0,t}, e_{1,1,t}) = (g^{y_{1,t}}, \texttt{H}(\beta_1^{y_{1,t}}) \oplus m_{1,t})$$

   (c) sends $res = (e_{0,0}, e_{1,0}), ..., (e_{0,z-1}, e_{1,z-1})$ to P_1.
6. *P_1-side Oblivious Filtering:* $\texttt{OblFilter}(res) \rightarrow res'$
   - forwards $res' = (e_{0,v}, e_{1,v})$ to R_j and discards the rest of the messages received from S.
7. *R-side Message Extraction:* $\texttt{Retreive}(res', req, pk) \rightarrow m_s$
   (a) sets $x = r_2 + r_1 \cdot (-1)^{s_2}$.
   (b) retrieves message $m_{s,v}$ by setting: $m_{s,v} = \texttt{H}((e_{s,0,v})^x) \oplus e_{s,1v}$

Fig. 6: DQ$^{\text{MR}}$–OT: Our protocol that realizes $\mathcal{DQ}^{\mathcal{MR}}$–$\mathcal{OT}^2_1$.

B.1 Proof of Theorem 5

Below, we prove the security of DQ^{MR}–OT, i.e., Theorem 5.

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

Corrupt *R*. Recall that in DQ^{MR}–OT, sender S holds a vector $\boldsymbol{m}$ of z pairs of messages (as opposed to DQ-OT where S holds only a single pair of messages). In the real execution, R's view is: $\texttt{View}_R^{\text{DQ}^{\text{MR}}-\text{OT}}\big(m_0, m_1, v, \epsilon, s\big) = \{r_R, g, C, p, e_{0,v}, e_{1,v}, m_{s,v}\}$, where $C = g^a$ is a random value and public parameter, where g is a random generator, a is a random value, p is a large random prime number, 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. initiates an empty view and appends uniformly random coin r'_R to it, where r'_R will be used to generate R-side randomness. It chooses a large random prime number p and a random generator g.
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: $C', r'_1, r'_2, y'_0, y'_1 \overset{\$}{\leftarrow} \mathbb{Z}_p$.
   - 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 $u \overset{\$}{\leftarrow} \mathbb{Z}_p$ and then sets $e'_s = (g^{y'_s}, \texttt{H}(\beta'^{y'_s}_s) \oplus m_s)$ and $e'_{1-s} = (g^{y'_{1-s}}, u)$.
3. appends $(g, C', p, r'_1, r'_2, e'_0, e'_1, m_s)$ 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 5.4) to argue why real model and ideal model views are indistinguishable, can be used in this case as well. 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. Hence, Relation 11 (in Section 7.1) holds.

Corrupt *S*. This case is identical to the corrupt S in the proof of DQ-OT (in Section 5.4) 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 $\texttt{Sim}_S$ 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 DQ-OT. $\texttt{Sim}_S$ 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 5.4) 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 7.1) holds.

Corrupt *P*$_2$. This case is identical to the corrupt P_2 case in the proof of DQ-OT. So, Relation 10 (in Section 7.1) holds.

Corrupt *P*$_1$. In the real execution, P_1's view is: $\texttt{View}_{P_1}^{\text{DQ}^{\text{MR}}-\text{OT}}\big((m_0, m_1), v, \epsilon, s\big) = \{g, C, p, s_1, r_1, \delta_0, \delta_1, (e_{0,0}, e_{1,0}), \dots, (e_{0,z-1}, e_{1,z-1})\}$. Ideal-model $\texttt{Sim}_{P_1}$ that receives v from P_1 operates as follows.

1. initiates an empty view. It selects a large random prime number p and a random generator g.
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. picks z pairs of random values as follows $(a_{0,0}, a_{1,0}), ..., (a_{0,z-1}, a_{1,z-1}) \overset{\$}{\leftarrow} \mathbb{Z}_p$.

5. appends $s'_1, g, C', p, r'_1, \delta'_0, \delta'_1$ 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 5.4) 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 are indistinguishable from the elements of each pair $(a_{0,i}, a_{1,i})$ in the ideal model, for all i, $0 \leq i \leq z-1$. Hence, Relation 9 (in Section 7.1) holds. $\square$