Oblivis: A Framework for Delegated and Efficient Oblivious Transfer

Aydin Abadi
Newcastle University
aydin.abadi@ncl.ac.uk

Yvo Desmedt
The University of Texas at Dallas
y.desmedt@cs.ucl.ac.uk

Abstract

As database deployments shift toward cloud platforms and edge devices, thin clients need to securely retrieve sensitive records without leaking their query intent or metadata to the proxies that mediate access. Oblivious Transfer (OT) is a core tool for private retrieval, yet existing OTs assume direct client–database interaction and lack support for delegated querying or lightweight clients.

We present Oblivis, a modular framework of new OT protocols that enable delegated, privacy-preserving query execution. Oblivis allows clients to retrieve database records without direct access, protects against leakage to both databases and proxies, and is designed with practical efficiency in mind. Its components include: (1) Delegated-Query OT, which permits secure outsourcing of query generation; (2) Multi-Receiver OT for merged, cloud-hosted databases; (3) a compiler producing constant-size responses suitable for thin clients; and (4) Supersonic OT, a proxy-based, information-theoretic, and highly efficient 1-out-of-2 OT. The protocols are formally defined and proven secure in the simulation-based paradigm, under non-colluding assumption. We implement and empirically evaluate Supersonic OT. It achieves at least a 92× speedup over a highly efficient 1-out-of-2 OT, and a 2.6×–106× speedup over a standard OT extension across 200–100,000 invocations. Our implementation further shows that Supersonic OT remains efficient even on constrained hardware, e.g., it completes an end-to-end transfer in 1.36 ms on a Raspberry Pi 4.

Keywords

Oblivious Transfer, Privacy-Preserving Query Processing.

1 Introduction

Modern data-driven services increasingly operate in cloud-hosted environments, where many clients are thin and lack substantial local resources. In these settings, clients often rely on intermediaries (both technical and organisational) to reach the underlying service or database. At the technical level, platforms such as Cloudflare [20] sit between users and origin servers, handling client traffic and therefore observing request structure and metadata; recent compromises of such proxies highlight the associated risks [19]. At the organisational level, financial advisors routinely issue queries to proprietary data providers on behalf of clients, creating opportunities for misuse or leakage of sensitive query intent, as shown in prior incidents [27, 46].

Proceedings on Privacy Enhancing Technologies 2026(3), 743–764

https://doi.org/10.56553/popets-2026-0104

In these settings, a recurring challenge emerges: clients, often resource-constrained, cannot communicate directly with the database, yet must retrieve information without revealing their query intent or exposing metadata to the proxies that mediate access.

Oblivious Transfer (OT) is a foundational primitive for private data retrieval. Classical OT ensures that a client retrieves exactly one database item without exposing its index and without learning anything about the remaining items. Existing OT constructions fall short in several fundamental ways, which we analyse below.

1.1 Motivating Scenarios

To illustrate these challenges, we highlight representative scenarios.

1.1.1 Proxy-Assisted Access in Financial Systems. In financial settings, advisors often act on behalf of clients, querying proprietary databases (e.g., containing real estate market information, market trends, and capital flows) offered by third-party providers such as Bloomberg Terminal [8], CoStar [22], or Multiple Listing Service [49], without granting clients direct access. This proxy model introduces privacy risks, especially when insider threats arise, e.g., "Swiss Leaks" and JPMorgan Chase incidents [27, 46]. In such cases, neither the advisor nor the database should learn the content or intent of a client's query. Existing OTs do not account for this kind of secure delegation. Encrypting a classical OT query would hide the client's index from the proxy advisor, but it does not support delegation: the receiver must still compute the full OT query locally, leaving the proxy unable to assist with the query-generation task. Delegation is valuable in these settings because it allows clients to remain lightweight and improves scalability by shifting the expensive steps away from the client.

1.1.2 Merged Cloud-Hosted Databases. Cloud providers often merge datasets belonging to different organisations into a single logical database to reduce operational cost and simplify indexing. A recent incident, for example, showed that data belonging to British Airways, Boots, and the BBC was stored within the same cloud [26]. When staff from different organizations query only their own records, direct privacy-preserving retrieval reveals sensitive metadata about the merged dataset. A similar situation arises even within a single organisation (e.g., RMIT University) when the cloud provider merges datasets from multiple internal departments, to enable organisation-wide analytics [58].

1.1.3 Hidden Query Components. In domains such as finance and healthcare, clients cannot always be informed about the exact conditions used to compute their outputs. An investment recommendation may depend on a proprietary internal flag associated with an account [3, 4]; a patient may be shielded from certain diagnoses [23, 33]; and some systems assign each client a private access policy [10, 42]. Secure retrieval must therefore succeed without revealing the underlying index to the receiver.

1.1.4 Performance Constraints in Lightweight Clients. Many systems rely on lightweight or long-lived clients (e.g., mobile applications or autonomous devices) that cannot run heavy machinery, download/store large encrypted datasets. These clients constantly interact with cloud services, including in settings where long-term security is important or where hardness assumptions are undesirable, since quantum algorithms show that quantum computers can break certain hardness assumptions. Sectors such as financial services and healthcare may operate under such requirements.

1.2 Limitations of Existing Techniques

We now discuss key research gaps our work addresses.

1.2.1 Lack of Support for Proxy-Based Delegation. Existing OT schemes assume that the receiver can compute the query index locally and interact directly with the sender, which is incompatible with settings that rely on proxy-based query generation. This restriction is inherent to classical OTs, since the receiver's choice bit is bound to the protocol execution to prevent any untrusted intermediary from inferring the selection.

1.2.2 Absence of Multi-Receiver Privacy and Hidden-Query Support. OT variants designed for multi-user or access-controlled databases reveal non-trivial metadata in multi-tenant cloud environments (e.g., the total database size or receiver access patterns). These constructions further assume that the receiver knows the query index, which conflicts with hidden-query settings.

1.2.3 Constant-Size Responses Without Large Local Storage. Protocols achieving constant-size responses usually require substantial receiver-side storage. Existing OTs offer no generic way to obtain constant-size responses without imposing heavy local storage costs.

1.2.4 Information-Theoretic OT. Information-theoretic OT avoids hardness assumptions, but existing constructions depend on multiple senders, noisy channels, or a fully trusted setup. These requirements limit their practicality and prevent deployment in lightweight or latency-sensitive environments.

1.3 Our Contributions

We introduce Oblivis[1], a framework organized into two families. The first family is setting-oriented and provides three key OT primitives. The second one is performance-oriented. We formally define and prove the proposed solutions in the simulation-based paradigm. Our main contributions are as follows.

- Setting-Oriented Contributions:
  (1) **Delegated-Query OT (DQ-OT):** Our construction introduces a new technique in which a receiver's choice index is secret-shared, jointly processed, and ordered by two proxies whose sequential computations yield a query structurally identical to that of the classical Naor–Pinkas OT. This technique enables secure, fully delegated query generation without revealing the index to either proxy.
  (2) **Delegate-Unknown-Query OT (DUQ-OT):** An extension of DQ-OT that supports hidden query indices, where the receiver securely obtains the correct record without learning the index used to retrieve it. This enables access policies and private conditioning that conventional OTs cannot capture.
  (3) **Delegated-Query Multi-Receiver OT (DQ^{MR}–OT):** An extension of DQ-OT that supports privately querying merged cloud datasets, where receivers learn nothing about the total number of records or unrelated fields, and the sender learns nothing about which receiver queried which record.
- Performance-Oriented Contributions:
  (1) **A Generic Compiler for Constant-Size Responses**: It transforms any 1-out-of-n OT into one with constant-size receiver size download complexity.
  (2) **Supersonic OT**: We introduce Supersonic OT, a highly efficient proxy-based, information-theoretic 1-out-of-2 OT that requires neither multiple senders, nor noisy channels, nor a trusted setup. The construction employs a controlled-swap mechanism, which, to our knowledge, has not been previously reported in the OT literature. Supersonic OT enables constant-size responses without relying on pre-stored database encryptions or heavy preprocessing. It achieves:
     – constant-size responses (i.e., receiver download: $O(1)$);
     – 0.53 ms per transfer on a modern laptop;
     – 92× speedup over the efficient base OT of [17], and a 2.6×–106× speedup over the OT extension of [39];
     – end-to-end execution on a Raspberry Pi 4, completing each transfer in 1.36 ms, showing smooth scaling (e.g., 3.26 ms at 1,000 invocations).

1.4 Context and Applications

Oblivis targets proxy-mediated access where intermediaries are unavoidable. Consider the following use cases. (a) Financial advisory platform: A client wants to retrieve a proprietary report via a brokerage platform, which today learns the client's intent; DQ-OT enables mediation without revealing the requested record. (b) Multi-tenant SaaS behind a gateway: SaaS APIs often sit behind an edge gateway (e.g., Cloudflare) for routing/access control. Oblivis uses the gateway for delegation, such that the tenant sends a request, proxies generate OT queries and contact the backend, and only the intended encrypted response reaches the tenant. (c) Travel booking via a portal: Corporate travel portals mediate access to proprietary offer databases. Oblivis retrieves the chosen offer without exposing the selection to the portal.

2 Preliminaries

2.1 Notations and Assumptions

By ϵ we mean an empty string. When y represents a single value, $|y|$ refers to the bit length of y; 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 secure channel. We define a parse function as $\mathsf{parse}(\lambda, y) \rightarrow (u_1, u_2)$, 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 $\mathsf{H} : \{0,1\}^* \rightarrow \{0,1\}^\sigma$ and $\mathsf{G} : \{0,1\}^* \rightarrow \{0,1\}^{\sigma+\lambda}$ modelled as random oracles [15].

[1] Oblivis is derived from the Latin verb oblivisci (to forget). We use it to reflect that any information seen by parties other than the receiver is uninformative and can be regarded as forgettable since it reveals nothing about the message or the query.

OT variant	Party	Input	Output
DQ–OT	S	(m_0, m_1)	ϵ
	P_1	ϵ	ϵ
	P_2	ϵ	ϵ
	R	$s \in \{0,1\}$	m_s
DUQ–OT	S	(m_0, m_1)	ϵ
	T	$s \in \{0,1\}$	ϵ
	P_1	ϵ	ϵ
	P_2	ϵ	ϵ
	R	ϵ	m_s
DQ^{MR}–OT	S	$\vec{m}$	ϵ
	P_1	$v \in \{0, \ldots, z-1\}$	z
	P_2	ϵ	ϵ
	R	$s \in \{0,1\}$	$m_{s,v}$
DUQ^{MR}–OT	S	$\vec{m}$	ϵ
	T	(v, s, z)	ϵ
	P_1	ϵ	z
	P_2	ϵ	ϵ
	R	ϵ	$m_{s,v}$
Compiler	S	$(m_0, \ldots, m_{n-1})$	ϵ
	R	$s \in \{0, \ldots, n-1\}$	m_s
Supersonic OT	S	(m_0, m_1)	ϵ
	P	ϵ	ϵ
	R	$s \in \{0,1\}$	m_s

Table 1: Summary of parties, inputs, and outputs for the OT variants in *Oblivis* (ideal-functionality view). Here, $\vec{m} = [(m_{0,0}, m_{1,0}), \ldots, (m_{0,z-1}, m_{1,z-1})]$, ϵ denotes an empty input/output. In multi-receiver variants, v identifies the receiver (or dataset slice) among z slices.

Table 1 summarizes our protocols' input and output values for each party involved. In this paper, we use the simulation-based paradigm of secure multi-party computation [35] to define and prove the protocols. We consider semi-honest adversaries that do not collude with each other. Appendix A (in the supplemental material file) restates the formal definition in this model.

2.2 Random Permutation

A random permutation $\pi(e_0, ..., e_n) \rightarrow (e'_0, ..., e'_n)$ is a probabilistic function that takes a set $A = \{e_0, ..., e_n\}$ and returns the same set of elements in a permuted order $B = \{e'_0, ..., e'_n\}$. The security of π requires that given a 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 [44] can permute a set of n elements in time $O(n)$. We will use π in the protocols presented in Figures 3 and 4.

2.3 Controlled Swap

The idea of controlled swap was introduced by Fredkin and Toffoli [32]. It can be defined as function $\bar{\pi}$ which takes two inputs: a binary value s and a pair (c_0, c_1). When $s = 0$, it returns the input pair (c_0, c_1), i.e., it does not swap the elements. When $s = 1$, it returns (c_1, c_0), effectively swapping the elements. It is clear that if s is uniformly chosen at random, then $\bar{\pi}$ represents a random permutation, implying that the probability of swapping or not swapping is $\frac{1}{2}$. We will use $\bar{\pi}$ in the Supersonic OT, presented in Figure 7.

2.4 The Original OT of Naor and Pinkas

As the DQ-OT protocol in Section 4.3 builds on the OT by Naor and Pinkas [52, pp. 450–451], we restate their construction in Figure 1.

(1) *S-side Initialization:* $\mathtt{S.Init}(1^\lambda) \rightarrow pk$
  (a) choose a random large prime number p (where $\log_2 p = \lambda$), a random element $C \xleftarrow{\$} \mathbb{Z}_p$ and random generator g. It is only important that the receiver R will not know a, which is the discrete logarithm of C to the base g, i.e., $C = g^a$.
  (b) publish $pk = (C, g, p)$.
(2) *R-side Query Generation:* $\mathtt{R.GenQuery}(pk, s) \rightarrow (q, sp)$
  (a) pick a random value $r \xleftarrow{\$} \mathbb{Z}_p \setminus \{0\}$ and sets $sp = r$.
  (b) set $\beta_s = g^r$ and $\beta_{1-s} = \frac{C}{\beta_s}$.
  (c) send $q = \beta_0$ to S and locally stores sp.
(3) *S-side Response Generation:* $\mathtt{S.GenRes}(m_0, m_1, pk, q) \rightarrow res$
  (a) compute $\beta_1 = \frac{C}{\beta_0}$.
  (b) choose two random values, $y_0, y_1 \xleftarrow{\$} \mathbb{Z}_p$.
  (c) encrypt the elements of the pair (m_0, m_1) as follows:
$$e_0 := (e_{0,0}, e_{0,1}) = (g^{y_0}, \mathsf{H}(\beta_0^{y_0}) \oplus m_0)$$
$$e_1 := (e_{1,0}, e_{1,1}) = (g^{y_1}, \mathsf{H}(\beta_1^{y_1}) \oplus m_1)$$
  (d) send $res = (e_0, e_1)$ to R.
(4) *R-side Message Extraction:* $\mathtt{R.Retrieve}(res, sp, pk, s) \rightarrow m_s$
  • retrieve the related message m_s by computing
$$m_s = \mathsf{H}((e_{s,0})^r) \oplus e_{s,1}$$

Figure 1: Original OT proposed by Naor and Pinkas [52, 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).

2.5 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 the 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 [29].

Convention. Exponents (e.g., x, r_1, r_2, a) are elements of $\mathbb{Z}_p$ and all arithmetic on them is modulo p; we omit mod p for readability. Group expressions such as g^r remain elements of the group $\mathbb{G}$.

2.6 Secret Sharing

A (threshold) secret sharing $\mathsf{SS}^{(t,n)}$ 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 $\mathsf{SS}^{(t,n)}$ for brevity. In the case where $t = n$, there exists a highly efficient XOR-based secret sharing [7]. In this case, to share the secret s, the dealer first picks $n-1$ random bit strings $r_1, ..., r_{n-1}$ of the same length as the secret. Then, it computes $r_n = r_1 \oplus, ..., \oplus r_n \oplus s$. It considers each $r_i \in \{r_1, .., r_n\}$ as a share of the secret. To reconstruct the secret, one can easily compute $r_1 \oplus, ..., \oplus r_n$. Any subset of fewer 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, $\mathsf{SS}(1^\lambda, s, n, t) \rightarrow (r_1, ..., r_n)$: to share a secret and $\mathsf{RE}(r_1, ..., r_t, n, t) \rightarrow s$ to reconstruct the secret.

2.7 Additive Homomorphic Encryption

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

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

We require that the encryption scheme satisfy indistinguishability against chosen-plaintext attacks (IND-CPA). We refer readers to [43] for a formal definition. One such scheme that meets the above features is the Paillier public key cryptosystem [54]. We will use an additive homomorphic encryption only in Sections 6.2 and 7.

3 Related Work

The traditional 1-out-of-2 OT ($\mathcal{OT}_1^2$) is a protocol that involves a sender S and a receiver R [31, 48]. The sender S has a pair of messages (m_0, m_1) and R has an index s. Informally, $\mathcal{OT}_1^2$ allows R to obtain m_s, without revealing anything about s to S, and without allowing R to learn anything about m_{1-s}. The $\mathcal{OT}_1^2$ functionality is defined as $\mathcal{F}_{\mathcal{OT}_1^2} : ((m_0, m_1), s) \rightarrow (\epsilon, m_s)$.

There exist many variants of OT. 1-out-of-n OTs [47, 50, 59] allow R to pick one entry out of n entries held by S, k-out-of-n OTs [16, 18, 41] enable R to pick k entries out of n entries held by S, where $k \geq 1$. OT extensions [5, 38, 39, 53] support efficient executions of OT (that mainly rely on symmetric-key operations) in the case where OT must be invoked many times. Distributed OTs [21, 51, 62] allow 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 [48].

3.1 Distributed OT

Naor and Pinkas [51] proposed several distributed OTs 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 functions. In these distributed OTs, the security against the servers holds as long as fewer than a predefined number of these servers collude. Later, various distributed OTs have been proposed[2]. For instance, Corniaux and Ghodosi [21] proposed a verifiable 1-out-of-n distributed OT that considers the case where a threshold of the servers is potentially active adversaries. The scheme is based on a sparse n-variate polynomial, verifiable secret sharing, and error-correcting codes.

Moreover, Zhao *et al.* [62] has proposed a distributed version of OT extension that aims to preserve the efficiency of OT extension while delegating the role of S to multiple servers, a threshold of which can be potentially semi-honest. The scheme is based on a hash function and an oblivious pseudorandom function.

However, there exists no OT that supports secure delegation of the query computation to third-party servers.

3.2 Multi-Receiver OT

Camenisch *et al.* [11] proposed a protocol for "OT with access control". It involves a set of receivers and a sender that 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 [10, 12, 13].

Nevertheless, in all the aforementioned schemes, the size of the entire database is revealed to the receivers.

3.3 OT with Constant Response Size

Researchers have proposed several OTs [14, 37, 61] that enable a receiver to obtain a constant-size response to its query. To achieve this level of communication efficiency, these OTs 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.

[2] Distributed OT has also been called proxy OT in [60].

3.4 Information-Theoretic OT

There have been efforts to design (both-sided) information-theoretic OTs. Some schemes [9, 21, 51] rely on multiple servers/senders that maintain an identical copy of the database. Other ones [24, 25, 40] rely on a specific network structure, i.e., a noisy channel, to achieve information-theoretic OT. There is also a scheme [57] that achieves information-theoretic OT using a fully trusted initializer. Hence, there exists no (efficient) information-theoretic OT that does not use noisy channels, multi-server, and a fully trusted initializer.

4 Delegated-Query OT

In this section, we present the notion of Delegated-Query 1-out-of-2 OT ($\mathcal{DQ}$–$\mathcal{OT}^2_1$) and a protocol that realizes it. $\mathcal{DQ}$–$\mathcal{OT}^2_1$ involves sender S, receiver R, and two proxy servers P_1 and P_2 that assist R to compute the query. $\mathcal{DQ}$–$\mathcal{OT}^2_1$ enables R to delegate (i) the computation of query and (ii) the interaction with S to P_1 and P_2, who jointly compute R's query and send it to S. $\mathcal{DQ}$–$\mathcal{OT}^2_1$ (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.

4.1 Functionality Definition

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

4.2 Security Definition

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

Definition 4.1 ($\mathcal{DQ}$–$\mathcal{OT}^2_1$). Let $\mathcal{F}_{\mathcal{DQ}\text{-}\mathcal{OT}^2_1}$ be the delegated-query OT functionality defined above. Protocol Γ realizes $\mathcal{F}_{\mathcal{DQ}\text{-}\mathcal{OT}^2_1}$ in the presence of passive adversaries, if for every non-uniform PPT adversary $\mathcal{A}$ in the real model, there exists a non-uniform PPT adversary (or simulator) $\mathtt{Sim}$ in the ideal model, such that:

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

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

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

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

A $\mathcal{DQ}$–$\mathcal{OT}^2_1$ scheme meets *efficiency* and *sender-push communication* (SPC). Efficiency states that the query generation of the receiver is faster compared to traditional (non-delegated) OT. SPC states that the sender sends responses to the receiver without requiring the receiver to directly initiate a query to the sender.

Definition 4.2 (Efficiency). A $\mathcal{DQ}$–$\mathcal{OT}^2_1$ scheme is efficient if the running time of the receiver-side request (or query) generation algorithm, denoted as $\mathtt{Request}(1^\lambda, \mathsf{s}, \mathsf{pk})$, 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 4.3 (Sender-push communication). Let (a) $\mathtt{Action}_R(t)$ represents 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, a $\mathcal{DQ}$–$\mathcal{OT}^2_1$ 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 scheme's execution), 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 can push messages to the receiver R without receiving an explicit request directly from R. Formally,
$$\forall t : \mathtt{Action}_S(t) \subseteq \{\mathtt{SendMessage}(S,R)\}$$

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

4.3 Protocol

Now, we present an efficient 1-out-of-2 OT protocol, called DQ-OT, that realizes $\mathcal{DQ}$–$\mathcal{OT}^2_1$. We build DQ-OT upon the $\mathcal{OT}^2_1$ proposed by Naor and Pinkas, presented in Figure 1. Our motivation for this choice is primarily didactic.

The DQ-OT relies on a key observation (which to our knowledge has not appeared in prior OTs): if the receiver secret-shares its index (see Figure 1) into two random shares, and the proxies sequentially compute and order their partial queries according to theses shares, the final query pair sent to the sender has an identical structure to the receiver's query in the Naor–Pinkas OT, i.e., as if the receiver had generated both queries (β_0, β_1) itself. Below, we explain how DQ-OT operates.

Initially, R splits its desired index into two binary shares, (s_1, s_2). Then, it picks two random values, (r_1, r_2), and sends each pair (s_i, r_i) to each P_i. 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 [52]). 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.

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 S. Given the queries, S computes the response in the same manner it does in the original OT in [52] 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 2.

(1) *S-side Initialization:* $\texttt{Init}(1^\lambda) \rightarrow pk$
  (a) choose a sufficiently large prime number p.
  (b) select a random element $C \xleftarrow{\$} \mathbb{Z}_p$ and generator g.
  (c) publish $pk = (C, p, g)$.
(2) *R-side Delegation:* $\texttt{Request}(1^\lambda, s, pk) \rightarrow req = (req_1, req_2)$
  (a) split 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) pick two uniformly random values: $r_1, r_2 \xleftarrow{\$} \mathbb{Z}_p$.
  (c) send $req_1 = (s_1, r_1)$ 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) compute a pair of partial queries:

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

  (b) send $q_2 = (\delta_0, \delta_1)$ to P_1.
(4) *P_1-side Query Generation:* $\texttt{P}_1.\texttt{GenQuery}(req_1, q_2, pk) \rightarrow q_1$
  (a) compute a pair of final queries as:

$$\beta_{s_1} = \delta_0 \cdot g^{r_1}, \quad \beta_{1-s_1} = \frac{\delta_1}{g^{r_1}}$$

  (b) send $q_1 = (\beta_0, \beta_1)$ to S.
(5) *S-side Response Generation:* $\texttt{GenRes}(m_0, m_1, pk, q_1) \rightarrow res$
  (a) abort if $C \neq \beta_0 \cdot \beta_1$.
  (b) pick two uniformly random values: $y_0, y_1 \xleftarrow{\$} \mathbb{Z}_p$.
  (c) compute a response pair (e_0, e_1) as follows:

$$e_0 := (e_{0,0}, e_{0,1}) = (g^{y_0}, \mathsf{H}(\beta_0^{y_0}) \oplus m_0)$$

$$e_1 := (e_{1,0}, e_{1,1}) = (g^{y_1}, \mathsf{H}(\beta_1^{y_1}) \oplus m_1)$$

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

Figure 2: 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, SS is the share-generation algorithm, H is a hash function, and $ denotes sampling a value uniformly at random.

Theorem 4.4. *Let $\mathcal{F}_{\mathcal{DQ}\text{-}\mathcal{OT}_1^2}$ be the functionality defined in Section 4.2. If Discrete Logarithm (DL), Computational Diffie-Hellman (CDH), and Random Oracle (RO) assumptions hold, then DQ-OT (presented in Figure 2) securely computes $\mathcal{F}_{\mathcal{DQ}\text{-}\mathcal{OT}_1^2}$ in the presence of semi-honest adversaries, w.r.t. Definition 4.1.*

Proof sketch. We argue security via simulation for each party in Definition 4.1. Each proxy receives only a one-time-pad share s_i of s (uniform on $\{0,1\}$) and random r_i, and its remaining view consists of group elements whose exponents are hidden under DL, so its view can be simulated without knowledge of s. Moreover, S sees only a random β_0 and $\beta_1 = C/\beta_0$, so its view is independent of s and is simulatable. For R, the ciphertext component for m_{1-s} is a one-time pad under key $\mathsf{H}(\beta_{1-s}^{y_{1-s}})$, and computing $\beta_{1-s}^{y_{1-s}}$ from $(\beta_{1-s}, g^{y_{1-s}})$ breaks CDH; as H is modelled as a RO, this pad is indistinguishable from random, so the transcript is simulatable given only m_s. □

Appendices B and C present the full proof of Theorem 4.4 and the proof of DQ-OT's correctness, respectively.

Note that in the protocol, we send β_1 to maintain uniformity in the query format across our OT protocols. Equivalently, P_1 can send only β_0 and S can reconstruct β_1.

Naïve Approach. One might instead let the receiver generate a standard OT query, encrypt it, and send the ciphertext through a proxy. Under semantic security, the proxy learns nothing from such a ciphertext, so this naïve design offers no security disadvantage relative to DQ-OT. However, it does not achieve delegation: the receiver must construct the OT query locally, and the proxy performs no part of the query-generation task. In contrast, DQ-OT (and its variants) shifts all heavy operations in the query-generation phase to the proxies and leaves the receiver with only basic operations.

5 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.

5.1 Security Definition

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

Definition 5.1 ($\mathcal{DUQ}\text{-}\mathcal{OT}_1^2$). Let $\mathcal{F}_{\mathcal{DUQ}\text{-}\mathcal{OT}_1^2}$ be the functionality defined above. A protocol Γ realizes $\mathcal{F}_{\mathcal{DUQ}\text{-}\mathcal{OT}_1^2}$ in the presence of passive adversaries, if for every PPT adversary $\mathcal{A}$ in the real model, there exists a non-uniform PPT simulator Sim in the ideal model, such that:

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

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

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

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

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 4.2.

5.2 Protocol

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

5.2.1 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 obtains a value indistinguishable from a random value (otherwise, it would learn extra information about m_{s-1}). Thus, if S imposes a certain publicly known structure on messages (m_0, m_1), then after decrypting S's response, only m_s would preserve the same structure.

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

5.2.2 An Overview. 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 received from R and T, each P_i generates queries the same way they do in DQ-OT. Given the final query pair and r_3, S first appends r_3 to m_0 and m_1 and then computes the response the same way it does in DQ-OT, with the difference that it also randomly permutes the elements of the response pair. Given the response pair and r_3, R decrypts each element in the pair and accepts the result that contains r_3. Figure 3 presents DUQ-OT in more detail.

Theorem 5.2. *If DL, CDH, and RO assumptions hold and random permutation π is secure, DUQ-OT (presented in Figure 3) securely computes $\mathcal{F}_{\mathcal{DUQ}\text{-}\mathcal{OT}^2_1}$ (defined in Section 5.1) in the presence of semi-honest adversaries, w.r.t. Definition 5.1.*

Proof sketch. When S is corrupted, the received queries are distributed independently of s (under DL); therefore, a simulator can sample a matching transcript without knowing s. When R is corrupted, learning anything about m_{1-s} would require computing the unchosen DH key (CDH); modelling G as a random oracle, the unchosen pad is indistinguishable from uniform; moreover, the pad r_3 is chosen uniformly at random and independent of the index s; thus, the view can be simulated given only m_s. If a proxy is corrupted, it sees only uniformly random shares and fresh randomness, plus group elements with hidden exponents; thus, its view is simulatable under DL. The view of T can be easily simulated; it has input s; however, it receives no messages from its counterparts (other than the public parameter pk) and receives no output from the protocol. □

Appendix D presents full proof of Theorem 5.2.

(1) *S-side Initialization:* $\mathtt{Init}(1^\lambda) \to pk$
  (a) choose a sufficiently large prime number p.
  (b) select a random element $C \xleftarrow{\$} \mathbb{Z}_p$ and generator g.
  (c) publish $pk = (C, p, g)$.
(2) *R-side Delegation:* $\mathsf{R.Request}(pk) \to req = (req_1, req_2)$
  (a) pick two uniformly random values: $r_1, r_2 \xleftarrow{\$} \mathbb{Z}_p$.
  (b) send $req_1 = r_1$ to P_1 and $req_2 = r_2$ to P_2.
(3) *T-side Query Generation:* $\mathsf{T.Request}(1^\lambda, s, pk) \to (req'_1, req'_2, sp_S)$
  (a) split the private index s into two shares (s_1, s_2) by calling $\mathtt{SS}(1^\lambda, s, 2, 2) \to (s_1, s_2)$.
  (b) pick a uniformly random value: $r_3 \xleftarrow{\$} \{0,1\}^\lambda$.
  (c) send $req'_1 = s_1$ 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:* $\mathsf{P_2.GenQuery}(req_2, req'_2, pk) \to q_2$
  (a) compute a pair of partial queries:
$$\delta_{s_2} = g^{r_2}, \quad \delta_{1-s_2} = \frac{C}{g^{r_2}}$$
  (b) send $q_2 = (\delta_0, \delta_1)$ to P_1.
(5) *P_1-side Query Generation:* $\mathsf{P_1.GenQuery}(req_1, req'_1, q_2, pk) \to q_1$
  (a) compute a pair of final queries as:
$$\beta_{s_1} = \delta_0 \cdot g^{r_1}, \quad \beta_{1-s_1} = \frac{\delta_1}{g^{r_1}}$$
  (b) send $q_1 = (\beta_0, \beta_1)$ to S.
(6) *S-side Response Generation:* $\mathtt{GenRes}(m_0, m_1, pk, q_1, sp_S) \to res$
  (a) abort if $C \neq \beta_0 \cdot \beta_1$.
  (b) pick two uniformly random values: $y_0, y_1 \xleftarrow{\$} \mathbb{Z}_p$.
  (c) compute a response pair (e_0, e_1) as follows:
$$e_0 := (e_{0,0}, e_{0,1}) = (g^{y_0}, \mathsf{G}(\beta_0^{y_0}) \oplus (m_0 || r_3))$$
$$e_1 := (e_{1,0}, e_{1,1}) = (g^{y_1}, \mathsf{G}(\beta_1^{y_1}) \oplus (m_1 || r_3))$$
  (d) randomly permute the elements of the pair (e_0, e_1) as follows: $\pi(e_0, e_1) \to (e'_0, e'_1)$.
  (e) send $res = (e'_0, e'_1)$ to R.
(7) *R-side Message Extraction:* $\mathtt{Retrieve}(res, req, pk, sp_R) \to m_s$
  (a) set $x = r_2 + r_1 \cdot (-1)^{s_2}$.
  (b) retrieve message m_s as follows. $\forall i, 0 \leq i \leq 1$:
    (i) set $y = \mathsf{G}((e'_{i,0})^x) \oplus e'_{i,1}$.
    (ii) call $\mathtt{parse}(\gamma, y) \to (u_1, u_2)$.
    (iii) set $m_s = u_1$, if $u_2 = r_3$.

Figure 3: DUQ-OT: Our 1-out-of-2 OT that supports query delegation while preserving the privacy of query from R. In the protocol, G is a hash function, π is a random permutation, and \$ denotes picking a value uniformly at random.

6 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 ($\mathcal{DUQ}^{\mathcal{MR}}$–$\mathcal{OT}^2_1$). They are suitable for the *multi-receiver* setting in which the sender maintains a (large) database containing z pairs of messages $\boldsymbol{m} = [(m_{0,0}, m_{1,0}), ..., (m_{0,z-1}, m_{1,z-1})]$.

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

6.1 Delegated-Query Multi-Receiver OT

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

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

Definition 6.1 ($\mathcal{DQ}^{\mathcal{MR}}$–$\mathcal{OT}^2_1$). 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 adversaries, if for every non-uniform PPT adversary $\mathcal{A}$ in the real model, there exists a non-uniform PPT simulator $\mathtt{Sim}$ in the ideal model, such that:

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

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

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

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

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

6.1.2 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 [39]*, employing one of the following approaches:
   - *Approach 1*: receiver R 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 which discards all pairs from the response except for v-th pair. R 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.
   - *Approach 2*: R 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 is interested.

(2) *Using an existing multi-receiver OT, e.g., in [11].* 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 3.2). 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.

6.1.3 Protocol. We present DQ$^{\text{MR}}$–OT that realizes $\mathcal{DQ}^{\mathcal{MR}}$–$\mathcal{OT}^2_1$. We build DQ$^{\text{MR}}$–OT upon DQ-OT (presented in Figure 2). 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 6.2. *Let g be a generator of a group $\mathbb{G}$ (defined in Section 2.5) 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 2.1), where its output size is δ-bit. Let $\gamma = r_1 \stackrel{+}{-} r_2$, $\beta_0 = g^{a+\gamma}$, 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 $\mu(\lambda)$.*

Appendix E presents the proof of Lemma 6.2. 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, 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 and discards the rest. Given the pair, R computes the result the same way a receiver does in DQ-OT. Figure 10 in Appendix F presents DQ$^{\text{MR}}$–OT in detail.

6.2 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).

6.2.1 Security Definition. The functionality that $\mathcal{DUQ}^{\mathcal{MR}}$–$\mathcal{OT}^2_1$ computes takes as input (i) a vector of messages $\boldsymbol{m} = [(m_{0,0}, m_{1,0}), ...,$

$(m_{0,z-1}, m_{1,z-1})]$ 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: $\mathcal{F}_{\mathcal{DUQ}^{\mathcal{MR}}\text{-}\mathcal{OT}_1^2} : \left([(m_{0,0}, m_{1,0}), ..., (m_{0,z-1}, m_{1,z-1})], (v,s,z), \epsilon, \epsilon, \epsilon\right) \rightarrow (\epsilon, \epsilon, z, \epsilon, m_{s,v})$, where $v \in \{0, ..., z-1\}$. Next, we present a formal definition of $\mathcal{DUQ}^{\mathcal{MR}}$–$\mathcal{OT}_1^2$.

Definition 6.3 ($\mathcal{DUQ}^{\mathcal{MR}}$–$\mathcal{OT}_1^2$). Let $\mathcal{F}_{\mathcal{DUQ}^{\mathcal{MR}}\text{-}\mathcal{OT}_1^2}$ be the functionality defined above. We assert that protocol Γ realizes $\mathcal{F}_{\mathcal{DUQ}^{\mathcal{MR}}\text{-}\mathcal{OT}_1^2}$ in the presence of passive adversaries, if for every non-uniform PPT adversary $\mathcal{A}$ in the real model, there exists a non-uniform PPT simulator $\mathtt{Sim}$ in the ideal model, such that:

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

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

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

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

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

6.2.2 Protocol. We proceed to present DUQ$^{\text{MR}}$–OT that realizes $\mathcal{DUQ}^{\mathcal{MR}}$–$\mathcal{OT}_1^2$. We build DUQ$^{\text{MR}}$–OT upon protocol DUQ-OT (presented in Figure 3). DUQ$^{\text{MR}}$–OT mainly relies on Lemma 6.2 and the following technique.

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

(1) construct vector $\boldsymbol{b} = [b_0, ..., b_{z-1}]$, where all b_is are set to zero except for v-th element b_v which is set to 1.
(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.[3]

In DUQ$^{\text{MR}}$–OT, $\boldsymbol{b'}$ is not sent for each query to S. Instead, it 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 which decrypts its and finds the message it was interested. Thus, P_1 *obliviously filters out* all other records of field elements that do not belong to R and sends to R only the messages that R is allowed to fetch. Figures 4 and 5 present DUQ$^{\text{MR}}$–OT in detail.

[3]Such a technique was previously used by Devet *et al.* [28] in the "private information retrieval" research line.

(1) *S-side Initialization:* $\mathtt{Init}(1^\lambda) \rightarrow pk$
It chooses a large random prime number p, random element $C \overset{\$}{\leftarrow} \mathbb{Z}_p$, and generator g. It publishes $pk = (C, p, g)$.
(2) *R-side One-off Setup:* $\mathtt{R.Setup}(1^\lambda) \rightarrow (pk_j, sk_j)$
It generates a key pair for the homomorphic encryption, by calling $\mathtt{KGen}(1^\lambda) \rightarrow (sk_j, pk_j)$. It send pk_j to T and S.
(3) *T-side One-off Setup:* $\mathtt{T.Setup}(z, pk_j) \rightarrow \boldsymbol{w}_j$
  (a) initialize an empty vector $\boldsymbol{w}_j = []$ of size z.
  (b) create a compressing vector, by setting v-th position of $\boldsymbol{w}_j$ to encrypted 1 and setting the rest of $z-1$ positions to encrypted 0. $\forall t, 0 \le t \le z-1$:
    (i) set $d = 1$, if $t = v$; set $d = 0$, otherwise.
    (ii) append $\mathtt{Enc}(pk_j, d)$ to $\boldsymbol{w}_j$.
  (c) send $\boldsymbol{w}_j$ to P_1.
(4) *R-side Delegation:* $\mathtt{R.Request}(pk) \rightarrow req = (req_1, req_2)$
  (a) pick random values: $r_1, r_2 \overset{\$}{\leftarrow} \mathbb{Z}_p$.
  (b) send $req_1 = r_1$ 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) split the private index s into two shares (s_1, s_2) by calling $\mathtt{SS}(1^\lambda, s, 2, 2) \rightarrow (s_1, s_2)$.
  (b) pick a uniformly random value: $r_3 \overset{\$}{\leftarrow} \{0,1\}^\lambda$.
  (c) send $req'_1 = s_1$ 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) compute queries: $\delta_{s_2} = g^{r_2}$, $\delta_{1-s_2} = \frac{C}{g^{r_2}}$.
  (b) send $q_2 = (\delta_0, \delta_1)$ to P_1.
(7) *P_1-side Query Generation:*
$\mathtt{P_1.GenQuery}(req_1, req'_1, q_2, pk) \rightarrow q_1$
  (a) compute queries as: $\beta_{s_1} = \delta_0 \cdot g^{r_1}, \beta_{1-s_1} = \frac{\delta_1}{g^{r_1}}$
  (b) send $q_1 = (\beta_0, \beta_1)$ 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) abort if $C \neq \beta_0 \cdot \beta_1$.
  (b) compute a response as follows. $\forall t, 0 \le t \le z-1$:
    (i) pick two random values $y_{0,t}, y_{1,t} \overset{\$}{\leftarrow} \mathbb{Z}_p$.
    (ii) compute response:
      $e_{0,t} := (e_{0,0,t}, e_{0,1,t}) = (g^{y_{0,t}}, \mathtt{G}(\beta_0^{y_{0,t}}) \oplus (m_{0,t} || r_3))$
      $e_{1,t} := (e_{1,0,t}, e_{1,1,t}) = (g^{y_{1,t}}, \mathtt{G}(\beta_1^{y_{1,t}}) \oplus (m_{1,t} || r_3))$
    (iii) randomly permute the elements of each pair $(e_{0,t}, e_{1,t})$ as $\pi(e_{0,t}, e_{1,t}) \rightarrow (e'_{0,t}, e'_{1,t})$.
  (c) send $res = (e'_{0,0}, e'_{1,0}), \ldots, (e'_{0,z-1}, e'_{1,z-1})$ to P_1.

Figure 4: Phases 1–8 of DUQ$^{\text{MR}}$–OT.

Theorem 6.4. *Let $\mathcal{F}_{\mathcal{DUQ}^{\mathcal{MR}}\text{-}\mathcal{OT}_1^2}$ be the functionality defined in Section 6.2.1. If DL, CDH, and RO assumptions hold, and additive homomorphic encryption satisfies IND-CPA, then DUQ$^{\text{MR}}$–OT (presented in Figures 4 and 5) securely computes $\mathcal{F}_{\mathcal{DUQ}^{\mathcal{MR}}\text{-}\mathcal{OT}_1^2}$ in the presence of semi-honest adversaries, w.r.t. Definition 6.3.*

(9) P_1-*side Oblivious Filtering:* $\texttt{OblFilter}(res, pk_j, \boldsymbol{w}_j) \rightarrow res'$
  (a) compress 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) send $res' = (o_{0,0}, o_{0,1}), (o_{1,0}, o_{1,1})$ to R.
(10) R-side *Message Extraction:* $\texttt{Retrieve}(res', req, sk_j, pk, sp_R) \rightarrow m_s$
  (a) decrypt the response from P_1 as follows: $\forall i, i', 0 \leq i, i' \leq 1 : \texttt{Dec}(sk_j, o_{i,i'}) \rightarrow o'_{i,i'}$.
  (b) set $x = r_2 + r_1 \cdot (-1)^{s_2}$.
  (c) retrieve message $m_{s,v}$ as follows. $\forall i, 0 \leq i \leq 1$:
    (i) set $y = \mathsf{G}((o'_{i,0})^x) \oplus o'_{i,1}$.
    (ii) call $\texttt{parse}(\gamma, y) \rightarrow (u_1, u_2)$.
    (iii) set $m_{s,v} = u_1$, if $u_2 = r_3$.

Figure 5: Phases 9 and 10 of DUQ$^{\text{MR}}$–OT.

We refer readers to Appendix H for Theorem 6.4's proof.

7 A Compiler for Generic OT with Constant Size Response

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

The main technique we rely on is the encrypted binary vector that we used in Section 6.2. The high-level idea is as follows. During query computation, R (along with its vector that encodes its index $s \in \{0, n-1\}$) computes a binary vector of size n, where all elements of the vector are set to 0 except for the 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 (in 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.

7.1 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 ($\mathcal{OT}_1^n$) have the following algorithms:

- $\mathsf{S.Init}(1^\lambda) \rightarrow pk$: a probabilistic algorithm run by S. It takes as input security parameter 1^λ and returns a public key pk.
- $\mathsf{R.GenQuery}(pk, n, s) \rightarrow (q, sp)$: a probabilistic algorithm run by R. It takes as input pk, n, and a secret index s. It returns a query (vector) q and a secret parameter sp.
- $\mathsf{S.GenRes}(m_0, \ldots, m_{n-1}, pk, q) \rightarrow res$: a probabilistic algorithm run by S. It takes as input pk and q. It generates an encoded response (vector) res.
- $\mathsf{R.Retrieve}(res, q, sp, pk, s) \rightarrow m_s$: 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: $\mathcal{F}_{\mathcal{OT}_1^n} : \big((m_0, ..., m_{n-1}), s\big) \rightarrow (\epsilon, m_s)$. 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 $(m_0, ..., m_{n-1})$. We refer readers to [35] for further discussion on 1-out-of-n OT.

7.2 The Compiler

We present the compiler in detail in Figure 6. We highlight that in the case where each $e_i \in res$ contains more than one value, e.g., $e_i = [e_{0,i}, ..., e_{w-1,i}]$ (due to a certain protocol design), then each element of e_i is separately multiplied and added by the element of vector $\boldsymbol{b'}$, e.g., the j-st element of the response is $e_{j,0} \overset{H}{\times} \boldsymbol{b'}[0] \overset{H}{+} \ldots \overset{H}{+} e_{j,n-1} \overset{H}{\times} \boldsymbol{b'}[n-1]$, for all j, $0 \leq j \leq w-1$. In this case, only w elements are sent to R.

Theorem 7.1. *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, generic OT with constant size response (presented in Figure 6) (i) securely computes $\mathcal{F}_{\mathcal{OT}_1^n}$ in the presence of semi-honest adversaries and (ii) offers $O(1)$ response size, regarding n.*

Appendix I presents the proof of Theorem 7.1. This compiler is most beneficial in the generic database setting, where each item is a record with multiple attributes. Let the sender hold n records $\{m_i\}_{i=0}^{n-1}$, where $m_i = (m_{i,1}, \ldots, m_{i,u})$ has u attributes. A conventional 1-out-of-n OT requires the receiver to download all n encrypted records, i.e., $O(n \cdot u)$ ciphertext per query. Our compiler instead returns only the selected record, reducing the receiver download to $O(u)$. For $u > 1$, this also reduces overall communication from $O(|q| + n \cdot u)$ to $O(|q| + n + u)$, since the extra $O(n)$ upload (the encrypted one-hot selection vector) is independent of u.

8 Supersonic OT

In this section, we introduce a 1-out-of-2 OT, called "Supersonic OT", which (i) operates at high speed by eliminating the need for public-key-based cryptography, (ii) delivers a response of size $O(1)$ to the recipient, R, (iii) ensures information-theoretic security, making it post-quantum secure, and (iv) is simple (but elegant), thus facilitating a simple analysis of its security and implementation.

8.1 Security Definition

Supersonic OT involves a sender S, a receiver R, and a server P. Each party might be corrupted by an independent passive adversary. The functionality $\mathcal{F}_{\mathcal{OT}_1^2}$ that Supersonic OT computes is similar to that of conventional OT, with the difference that now an additional party P is introduced, having no input and receiving no output. The functionality is defined as: $\mathcal{F}_{\mathcal{OT}_1^2} : \big((m_0, m_1), \epsilon, s\big) \rightarrow (\epsilon, \epsilon, m_s)$. Next, we present a formal definition of $\mathcal{OT}_1^2$.

Definition 8.1 ($\mathcal{OT}_1^2$). Let $\mathcal{F}_{\mathcal{OT}_1^2}$ be the OT functionality defined above. We assert that protocol Γ realizes $\mathcal{F}_{\mathcal{OT}_1^2}$ in the presence of passive adversaries, if for every non-uniform PPT adversary $\mathcal{A}$ in

(1) *S-side Initialization:* $\texttt{Init}(1^\lambda) \to pk$
This phase involves S.
(a) call $\texttt{S.Init}(1^\lambda) \to pk$.
(b) publish pk.
(2) *R-side Setup:* $\texttt{Setup}(1^\lambda) \to (sk_R, pk_R)$
This phase involves R.
(a) call $\texttt{KGen}(1^\lambda) \to (sk_R, pk_R)$.
(b) publish pk_R.
(3) *R-side Query Generation:* $\texttt{GenQuery}(pk_R, n, s) \to (q, sp, \boldsymbol{b}')$
This phase involves R.
(a) call $\texttt{R.GenQuery}(pk, n, s) \to (q, sp)$.
(b) construct a vector $\boldsymbol{b} = [b_0, ..., b_{n-1}]$, as:
(i) set every element b_i to zero except for s-th element b_s which is set to 1.
(ii) encrypt each element of $\boldsymbol{b}$ using additive homomorphic encryption, $\forall 0 \le i \le n-1 : b'_i = \texttt{Enc}(pk_R, b_i)$. Let $\boldsymbol{b}'$ be the vector of the encrypted elements.
(c) send q and $\boldsymbol{b}'$ to S and locally store sp.
(4) *S-side Response Generation:*
$\texttt{GenRes}(m_0, \ldots, m_{n-1}, pk, pk_R, q, \boldsymbol{b}') \to res$
This phase involves S.
(a) call $\texttt{S.GenRes}(m_0, \ldots, m_{n-1}, pk, q) \to res$. Let $res = [e_0, ..., e_{n-1}]$.
(b) compress the response using vector $\boldsymbol{b}'$ as follows.
$\forall i, 0 \le i \le n-1 :$
$$e = (e_0 \overset{H}{\times} \boldsymbol{b}'[0]) \overset{H}{+} \ldots \overset{H}{+} (e_{n-1} \overset{H}{\times} \boldsymbol{b}'[n-1])$$
(c) send $res = e$ to R.
(5) *R-side Message Extraction.*
$\texttt{Retrieve}(res, q, sp, pk, sk_R, s) \to m_s$
This phase involves R.
(a) call $\texttt{Dec}(sk_R, res) \to res'$.
(b) call $\texttt{R.Retrieve}(res', q, sp, pk, s) \to m_s$.

Figure 6: 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 $O(1)$.

the real model, there is a non-uniform PPT simulator $\texttt{Sim}$ in the ideal model, where:

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

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

$$\left\{\texttt{Sim}_R\Big(s, \mathcal{F}_{OT_1^2}\big((m_0, m_1), \epsilon, s\big)\Big)\right\}_{m_0,m_1,s} \equiv \left\{\texttt{View}_R^\Gamma\big((m_0, m_1), \epsilon, s\big)\right\}_{m_0,m_1,s} \tag{18}$$

8.2 The Protocol

Figure 7 presents Supersonic OT in detail. At a high level, the protocol works as follows. Initially, R and S agree on a pair of keys. In the query generation phase, R splits its private index into two binary shares. It sends one share to S and the other to P. Given the share/query, S encrypts every message m_i (using a one-time pad) under one of the keys it agreed with R. S permutes the encrypted messages using $\bar{\pi}$ and its share. It sends the resulting pair to P, which permutes the received pair using $\bar{\pi}$ and its share. P sends only the first element of the resulting pair (which is a ciphertext) to R and discards the second element of the pair. Next, R decrypts the ciphertext and learns the message it was interested in.

(1) *R-side Setup:* $\texttt{Setup}(1^\lambda) \to (k_0, k_1)$
- R picks two random keys $(k_0, k_1) \xleftarrow{\$} \{0,1\}^\sigma$ and sends them to S.

(2) *R-side Query Generation:* $\texttt{GenQuery}(1^\lambda, s) \to q = (q_1, q_2)$
(a) split the private index s into two shares (s_1, s_2) by calling $\texttt{SS}(1^\lambda, s, 2, 2) \to (s_1, s_2)$.
(b) send $q_1 = s_1$ to S and $q_2 = s_2$ to P.
(3) *S-side Response Generation:* $\texttt{GenRes}(m_0, m_1, k_1, k_2, q_1) \to res$
(a) encrypt each message as follows.
$$\forall i, 0 \le i \le 1 : m'_i = m_i \oplus k_i$$
Let $e = (m'_0, m'_1)$ contain the encrypted messages.
(b) permute the elements of e as: $\bar{\pi}(s_1, e) \to e'$.
(c) send $res = e'$ to P.
(4) *P-side Oblivious Filtering:* $\texttt{OblFilter}(res, q_2) \to res'$
(a) permute the elements of e' as: $\bar{\pi}(s_2, e') \to e''$.
(b) send (always) the first element in e'', say $res' = e''_0$, to R and discard the second element in e''.
(5) *R-side Message Extraction:* $\texttt{Retrieve}(res', k_s) \to m_s$
- retrieve the final related message m_s by decrypting e''_0 as: $m_s = e''_0 \oplus k_s$.

Figure 7: Supersonic OT.

Theorem 8.2. *Let $\mathcal{F}_{OT_1^2}$ be the functionality defined in Section 8.1. Then, Supersonic OT (presented in Figure 7) securely computes $\mathcal{F}_{OT_1^2}$ in the presence of semi-honest adversaries, w.r.t. Definition 8.1.*

Proof sketch. If R is corrupted, its view contains $e''_0 = m_s \oplus k_s$ where $k_s \leftarrow \{0,1\}^\sigma$ is uniform, so a simulator picks $k \leftarrow \{0,1\}^\sigma$ and outputs $e = m_s \oplus k$. If S is corrupted, its view is (s_1, k_0, k_1); by security of $(2,2)$ secret sharing, s_1 is indistinguishable from a uniform bit, and (k_0, k_1) are uniform, so the simulator samples fresh independent values. If P is corrupted, its view is (s_2, e'); s_2 is uniform, and e' is a pair of one-time-pad ciphertexts, hence indistinguishable from two uniform σ-bit strings. Also, the ordering is hidden from P because the permutation depends on the unknown share s_1. Therefore, the real and ideal views are indistinguishable. □

We refer readers to Appendices J and K for full proof of Theorem 8.2 and proof of Supersonic OT's correctness, respectively. Note that Supersonic OT is independent of the other protocols proposed in this paper; however, it targets the same proxy-mediated thin-client theme.

9 Evaluation

We have implemented Supersonic OT in C++ and evaluated its concrete runtime. The source code for the implementation is publicly available in [2]. We used two platforms for the experiments: (1) a

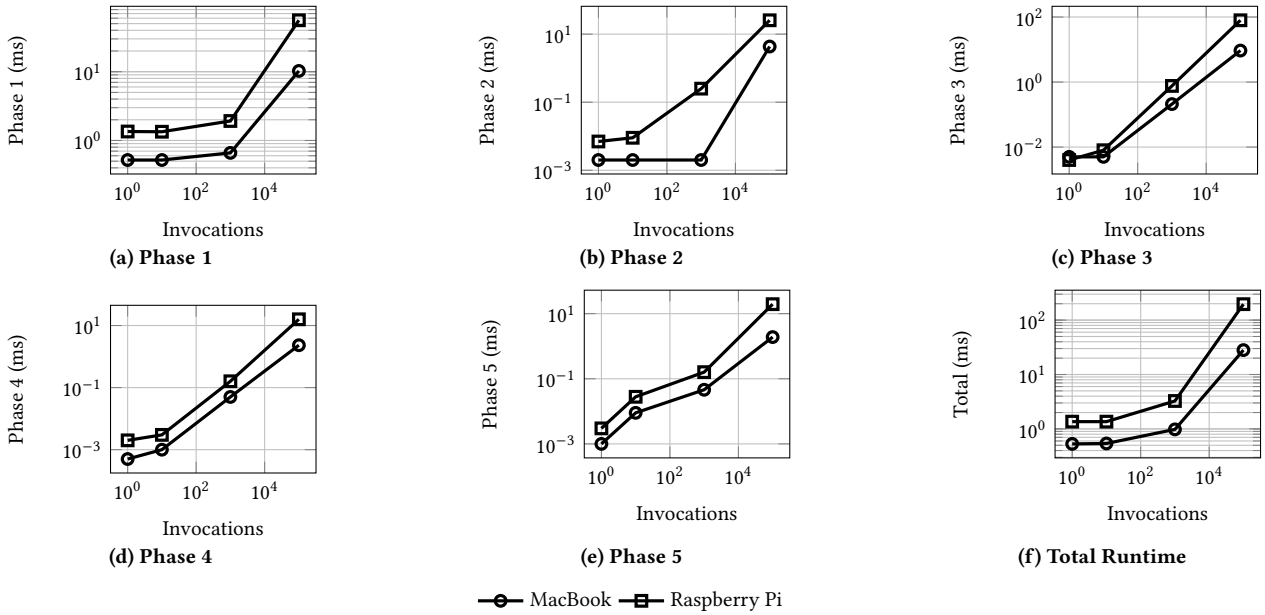

Figure 8: Supersonic OT runtime breakdown across phases on MacBook and Raspberry Pi.

MacBook Pro with an Apple M3 Pro CPU and 36 GB of RAM, and (2) a Raspberry Pi 4 with 4 GB RAM. We did not use parallelization or any other optimization. Each experiment was repeated 50 times and we report the mean. We utilized the GMP library [34] for big-integer arithmetic.

9.0.1 Supersonic OT Runtime. We analyzed the runtime of various phases of Supersonic OT across different invocation frequencies (1, 10, 10^3, and 10^5 times) and different platforms. We successfully ran the entire end-to-end Supersonic OT on the Raspberry Pi. Table 2 shows the performance of Supersonic OT.

Table 2: Supersonic OT's runtime (in ms), categorized by various phases and execution environment.

	Phases	Number of OT Invocations			
		1	10	10^3	10^5
MacBook	Phase 1	0.52	0.52	0.66	10.26
	Phase 2	0.0020	0.0020	0.0020	4.32
	Phase 3	0.0050	0.0050	0.21	9.29
	Phase 4	0.0005	0.0010	0.05	2.32
	Phase 5	0.0010	0.0090	0.046	1.91
	Total	0.53	0.54	0.98	28.11
Raspberry Pi	Phase 1	1.35	1.34	1.92	55.96
	Phase 2	0.0070	0.0090	0.25	25.52
	Phase 3	0.0040	0.0080	0.76	79.43
	Phase 4	0.0020	0.0030	0.16	15.89
	Phase 5	0.0030	0.028	0.16	19.58
	Total	1.36	1.36	3.26	196.39

A single invocation completes in 0.53 ms on the MacBook and 1.36 ms on the Raspberry Pi. Even under large batches of 10^5 invocations, the runtime remains modest: 28.11 ms on the MacBook and 196.39 ms on the Raspberry Pi. These results suggest that the protocol remains lightweight even on constrained hardware.

Figure 8 shows the scaling behaviour of the five phases and the total runtime. The plots reveal a consistent pattern across both platforms. Across the measured invocation counts, the Raspberry Pi shows a slowdown of about 3–4× compared to the MacBook; however, the relative behaviour of all phases remains consistent on both devices. The dominant cost on both platforms is Phase 1, which performs the key-related setup and share operations. The remaining phases incur minimal overhead, and their contribution stays small even at 10^5 invocations.

9.0.2 Runtime Comparison. We compared Supersonic OT's runtime to the runtimes of efficient: a base OT (i.e., Simplest OT) [17] and an OT extension (i.e., IKPN OT) [39]. We report only the MacBook runtimes because we were unable to successfully build the LibOTe library [56] (which contains the implementations of [17, 39]) on the Raspberry Pi despite extensive installation attempts. The source code implementaiton of [39] is available at [1].

Supersonic Versus Base OTs. For completeness, in addition to the runtime of Simplest OT, we include the reported runtimes of previous base OT protocols in [5, 52]; these protocols' source code was available but could not be successfully executed in our MacBook. For the runtime of STD–OT in [5], STD–OT in [52], and RO–OT in [52], we derived the reported figures from [5], specifically from Table 3, where the GMP library was employed. Table 3 summarizes the results. As shown, Supersonic OT achieves a substantial speedup, running at least 92.8× faster (than Simplest OT of [17]).

Supersonic OT Versus OT Extensions. OT extensions are efficient when they are invoked many times. To determine how Supersonic OT performs compared to the OT extensions, we compared its runtime with one of the most efficient OT extensions: IKPN OT [39]. We instantiated it using the Simplest OT as a base. We invoked the Supersonic OT and IKPN OT from 200 to 100,000 times. The source code implementaiton of [39] is available at [1]. Our implementation executes Supersonic OT sequentially without employing

Table 3: Comparing the runtime (in ms) of Supersonic OT with the base: standard OT (STD–OT) [5], STD–OT [52], and the random oracle OT (RO–OT) [52]. The runtime is based on 128 calls of each scheme. The speedup factor is the improvement that Supersonic OT offers compared to each scheme. The protocols tagged with ‡ were run on the same machine. Other values are reported from the related original papers.

Scheme	Runtime (ms)	Speedup Factor
STD–OT in [5]	1,217	1,622
STD–OT in [52]	1,681	2,241
RO–OT in [52]	288	384
Simplest OT‡ [17]	69.6	92.8
Supersonic OT‡	**0.75**	1

parallelization techniques. This contrasts with optimized libraries like libOTe, which leverage parallel execution (particularly for base OT operations) to achieve higher throughput. Thus, our sequential approach provides a conservative baseline for performance comparison. Table 4 and Figure 9 summarize the results.

Supersonic OT achieves notable speedups across all invocation counts, ranging from 106× faster for 200 invocations to 2.6× faster for 100,000 invocations. As shown in Figure 9, the IKNP-based OT extension exhibits the expected near-constant runtime, yet Supersonic OT retains better performance across the entire tested range. The converging trend suggests that for a large number of invocations beyond those tested, the IKNP OT's minimal marginal cost may allow it to outperform our Supersonic OT implementation. Our experiment results suggest a crossover beyond 10^5 (i.e., $\approx 5 \times 10^5$), after which IKNP OT outperforms.

Table 4: End-to-end runtime of Supersonic OT versus the IKNP OT extension across different invocation counts. It shows the speedup achieved by Supersonic OT.

Number of Invocations	Supersonic OT (ms)	IKNP OT Extension (ms)	Speedup Factor
200	0.65	70.05	**106**
1,000	1.08	70.46	65
4,500	2.24	70.59	31
20,000	8.50	71.89	8.4
100,000	28.11	74.41	2.6
500,000	83.25	74.59	-1.1

9.0.3 Features Comparison. For the base OTs or OT extensions to achieve information-theoretical security, they typically require multiple replicas of the database, a noisy channel, or the involvement of a fully trusted initializer, all of which contribute to increased deployment costs or stronger trust assumptions. In contrast, Supersonic OT attains information-theoretic security without relying on such settings or assumptions. Unlike base OTs or OT extensions that typically only involve the sender and receiver, Supersonic OT involves a proxy that might be corrupted by a semi-honest adversary who does not collude with other parties. Table 5 compares the features of Supersonic OT with several state-of-the-art OTs.

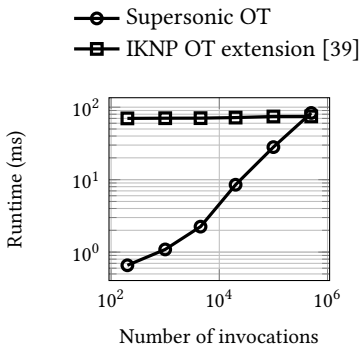

Figure 9: Runtime comparison between Supersonic OT and an IKNP-based OT extension across different invocation counts.

Table 5: Feature comparison of 1-out-of-2 OTs. I.T. stands for information-theoretic, and P.Q. stands for post-quantum.

Protocol	I.T. security	P.Q. security	No database replications	No noisy channel	No trusted party
[5]	×	×	✓	✓	✓
[52]	×	×	✓	✓	✓
[17]	×	×	✓	✓	✓
[51]	✓	✓	×	✓	✓
[24]	✓	✓	✓	×	✓
[25]	✓	✓	✓	×	✓
[40]	✓	✓	✓	×	✓
[57]	✓	✓	✓	✓	×
[55]	×	✓	✓	✓	✓
[45]	×	✓	✓	✓	✓
[30]	×	✓	✓	✓	✓
[6]	×	✓	✓	✓	✓
Supersonic OT	✓	✓	✓	✓	✓

10 Conclusion and Future Work

This work introduced Oblivis, a framework of Oblivious Transfer protocols that address gaps in delegated, metadata-hiding, and lightweight retrieval settings. We presented new OT variants that support secure query outsourcing, hidden-query execution, and multi-receiver privacy in merged cloud datasets, tasks that cannot be handled by classical OT. We further developed a generic compiler for achieving constant-size responses without imposing large local storage requirements. Our performance-oriented component, Supersonic OT, shows that information-theoretic OT can be both practical and fast. Our implementation achieves notable speedups over state-of-the-art base OTs and OT extensions. It operates smoothly even on constrained hardware such as a Raspberry Pi 4. Because it depends only on the GMP library, the implementation is lightweight and portable, which enables straightforward deployment across a wide range of environments.

Multiple directions remain open. Extending Supersonic OT to richer query classes (e.g., range predicates) is an important next step. Another direction is exploring browser-based deployments using lightweight client-side cryptography. Combining our delegated OT mechanisms with federated or multi-service architectures, or with edge-computing platforms, also appears promising.

Acknowledgments

Aydin Abadi was supported in part by (a) UKRI grant: EP/V011189/1 and Kuwait Foundation for the Advancement of Sciences (KFAS) under project code: PA24-6TE-2493. Yvo Desmedt thanks the Jonsson Endowment. AI-based writing assistants were used only for language and presentation editing.

References

[1] Aydin Abadi. 2025. Source Code of State-of-the-Art Efficient OTs. https://github.com/AydinAbadi/Supersonic-OT-Related-Work.
[2] Aydin Abadi. 2025. Source code of Supersonic OT. https://github.com/AydinAbadi/Supersonic-OT-Compatible-for-Raspberry-Pi.
[3] Aydin Abadi, Jack Liddell, Bradley Doyle, Francesco Gini, Kieron Guinamard, Sasi Kumar Murakonda, Paul Mellor, Steven J. Murdoch, Mohammad Naseri, Hector Page, George Theodorakopoulos, and Suzanne Weller. 2024. Starlit: Privacy-Preserving Federated Learning to Enhance Financial Fraud Detection. (2024).
[4] Sunpreet S. Arora, Andrew Beams, Panagiotis Chatzigiannis, Sebastian Meiser, Karan Patel, Srinivasan Raghuraman, Peter Rindal, Harshal Shah, Yizhen Wang, Yuhang Wu, Hao Yang, and Mahdi Zamani. 2024. Privacy-Preserving Financial Anomaly Detection via Federated Learning & Multi-Party Computation. In *Annual Computer Security Applications Conference, ACSAC 2024 - Workshops*. IEEE.
[5] Gilad Asharov, Yehuda Lindell, Thomas Schneider, and Michael Zohner. 2013. More efficient oblivious transfer and extensions for faster secure computation. In *CCS'13*.
[6] Paulo Barreto, Glaucio Oliveira, and Waldyr Benits. 2018. Supersingular Isogeny Oblivious Transfer. *IACR Cryptol. ePrint Arch.* (2018).
[7] George Robert Blakley. 1980. One time Pads are Key Safegaurding Schemes, not Cryptosystems. Fast Key Safeguarding Schemes (Threshold Schemes) Exist. In *IEEE S&P*.
[8] Bloomberg Terminal. 2025. There is a reason customers turn to the Bloomberg Terminal to address industry challenges.
[9] Carlo Blundo, Paolo D'Arco, Alfredo De Santis, and Douglas R. Stinson. 2007. On Unconditionally Secure Distributed Oblivious Transfer. *J. Cryptol.* (2007).
[10] Jan Camenisch, Maria Dubovitskaya, Robert R. Enderlein, and Gregory Neven. 2012. Oblivious Transfer with Hidden Access Control from Attribute-Based Encryption. In *Security and Cryptography for NetworksSCN*.
[11] Jan Camenisch, Maria Dubovitskaya, and Gregory Neven. 2009. Oblivious transfer with access control. In *CCS*.
[12] Jan Camenisch, Maria Dubovitskaya, and Gregory Neven. 2010. Unlinkable Priced Oblivious Transfer with Rechargeable Wallets. In *FC*.
[13] Jan Camenisch, Maria Dubovitskaya, Gregory Neven, and Gregory M. Zaverucha. 2011. Oblivious Transfer with Hidden Access Control Policies. In *PKC*.
[14] Jan Camenisch, Gregory Neven, and Abhi Shelat. 2007. Simulatable Adaptive Oblivious Transfer. In *Advances in Cryptology - EUROCRYPT*.
[15] Ran Canetti. 1997. Towards realizing random oracles: Hash functions that hide all partial information. *IACR Cryptol. ePrint Arch.* (1997).
[16] Yalin Chen, Jue-Sam Chou, and Xian-Wu Hou. 2010. A novel k-out-of-n Oblivious Transfer Protocols Based on Bilinear Pairings. *IACR Cryptol. ePrint Arch.* (2010).
[17] Tung Chou and Claudio Orlandi. 2015. The Simplest Protocol for Oblivious Transfer. In *Progress in Cryptology - LATINCRYPT 2015 - 4th International Conference on Cryptology and Information Security in Latin America, Guadalajara, Mexico, August 23-26, 2015, Proceedings*. Springer.
[18] Cheng-Kang Chu and Wen-Guey Tzeng. 2005. Efficient k-Out-of-n Oblivious Transfer Schemes with Adaptive and Non-adaptive Queries. In *PKC*.
[19] Cloudflare, Inc. 2022. Cloudflare's investigation of the January 2022 Okta compromise. https://blog.cloudflare.com/cloudflare-investigation-of-the-january-2022-okta-compromise/. Blog post; accessed 2025-11-25.
[20] Cloudflare, Inc. 2025. How Cloudflare works. https://developers.cloudflare.com/fundamentals/concepts/how-cloudflare-works/. Documentation; accessed 2025-11-25.
[21] Christian L. F. Corniaux and Hossein Ghodosi. 2013. A Verifiable 1-out-of-n Distributed Oblivious Transfer Protocol. *IACR Cryptol. ePrint Arch.* (2013).
[22] CoStar. 2025. The most comprehensive platform for commercial real estate information, analytics and news.
[23] General Medical Council. 2020. The dialogue leading to a decision. t.ly/UA_Yn.
[24] Claude Crépeau and Joe Kilian. 1988. Achieving Oblivious Transfer Using Weakened Security Assumptions (Extended Abstract). In *FoCS*.
[25] Claude Crépeau, Kirill Morozov, and Stefan Wolf. 2004. Efficient Unconditional Oblivious Transfer from Almost Any Noisy Channel. In *Security in Communication Networks, 4th International Conference, SCN*.
[26] Dan Milmo and Alex Hern. 2023. BA, Boots and BBC cyber-attack: who is behind it and what happens next?
[27] Department of Justice–U.S. Attorney's Office. 2018. Former JP Morgan Chase Bank Employee Sentenced to Four Years in Prison for Selling Customer Account Information.
[28] Casey Devet, Ian Goldberg, and Nadia Heninger. 2012. Optimally Robust Private Information Retrieval. In *USENIX Security*.
[29] Whitfield Diffie and Martin E. Hellman. 1976. New directions in cryptography. *IEEE Trans. Inf. Theory* (1976).
[30] Rafael Dowsley, Jeroen van de Graaf, Jörn Müller-Quade, and Anderson C. A. Nascimento. 2012. Oblivious Transfer Based on the McEliece Assumptions. *IEICE Trans. Fundam. Electron. Commun. Comput. Sci.* (2012).
[31] Shimon Even, Oded Goldreich, and Abraham Lempel. 1985. A Randomized Protocol for Signing Contracts. *Commun. ACM* (1985).
[32] Edward Fredkin and Tommaso Toffoli. 1982. Conservative Logic. In *International Journal of Theoretical Physics*.
[33] General Medical Council. 2022. Withholding Information from Patients.
[34] GNU Project. 1991. The GNU Multiple Precision Arithmetic Library. https://gmplib.org.
[35] Oded Goldreich. 2004. *The Foundations of Cryptography - Volume 2, Basic Applications*. Cambridge University Press.
[36] Shafi Goldwasser, Silvio Micali, and Charles Rackoff. 1985. The Knowledge Complexity of Interactive Proof-Systems (Extended Abstract). In *STOC*.
[37] Matthew Green and Susan Hohenberger. 2008. Universally Composable Adaptive Oblivious Transfer. In *ASIACRYPT*.
[38] Wilko Henecka and Thomas Schneider. 2013. Faster secure two-party computation with less memory. In *CCS*.
[39] Yuval Ishai, Joe Kilian, Kobbi Nissim, and Erez Petrank. 2003. Extending Oblivious Transfers Efficiently. In *CRYPTO,*.
[40] Yuval Ishai, Eyal Kushilevitz, Rafail Ostrovsky, Manoj Prabhakaran, Amit Sahai, and Jürg Wullschleger. 2011. Constant-Rate Oblivious Transfer from Noisy Channels. In *CRYPTO*.
[41] Stanislaw Jarecki and Xiaomin Liu. 2009. Efficient Oblivious Pseudorandom Function with Applications to Adaptive OT and Secure Computation of Set Intersection. In *TCC*.
[42] Maithilee P. Joshi, Sudip Mittal, Karuna P. Joshi, and Tim Finin. 2017. Semantically Rich, Oblivious Access Control Using ABAC for Secure Cloud Storage. In *IEEE International Conference on Edge Computing, EDGE*.
[43] Jonathan Katz and Yehuda Lindell. 2014. *Introduction to Modern Cryptography, Second Edition*. CRC Press.
[44] Donald E. Knuth. 1981. *The Art of Computer Programming, Volume II: Seminumerical Algorithms, 2nd Edition*. Addison-Wesley.
[45] Nibedita Kundu, Sumit Kumar Debnath, and Dheerendra Mishra. 2020. 1-out-of-2: post-quantum oblivious transfer protocols based on multivariate public key cryptography. *Sādhanā* (2020).
[46] David Leigh, James Ball, Juliette Garside, and David Pegg. 2015. HSBC files timeline: From Swiss bank leak to fallout. *The Guardian* (2015).
[47] Momeng Liu and Yupu Hu. 2019. Universally composable oblivious transfer from ideal lattice. *Frontiers Comput. Sci.* (2019).
[48] Michael O. Rabin. 1981. How to Exchange Secrets with Oblivious Transfer.
[49] MLS. 2025. MLS Multiple Listing Service Listings. https://www.mls.com.
[50] Moni Naor and Benny Pinkas. 1999. Oblivious Transfer and Polynomial Evaluation. In *STOC*.
[51] Moni Naor and Benny Pinkas. 2000. Distributed Oblivious Transfer. In *ASIACRYPT*. Springer.
[52] Moni Naor and Benny Pinkas. 2001. Efficient Oblivious Transfer Protocols *(SODA)*.
[53] Jesper Buus Nielsen. 2007. Extending Oblivious Transfers Efficiently - How to get Robustness Almost for Free. *IACR Cryptol. ePrint Arch.* (2007).
[54] Pascal Paillier. 1999. Public-Key Cryptosystems Based on Composite Degree Residuosity Classes. In *EUROCRYPT*.
[55] Chris Peikert, Vinod Vaikuntanathan, and Brent Waters. 2008. A Framework for Efficient and Composable Oblivious Transfer. In *CRYPTO*.
[56] Peter Rindal and Mike Rosulek. 2017. libOTe: Oblivious Transfer Extension Library. https://github.com/osu-crypto/libOTe. Accessed: October 2025.
[57] Ronald Rivest. 1999. Unconditionally secure commitment and oblivious transfer schemes using private channels and a trusted initializer. *technical report* (1999).
[58] Slalom LLC. 2023. RMIT University Data & Analytics Platform: Transforming Education Through Data Innovation. Online case study. https://www.slalon.com/us/en/customer-stories/rmit-university-data-analytics-platform
[59] Wen-Guey Tzeng. 2002. Efficient 1-Out-n Oblivious Transfer Schemes. In *PKC*, David Naccache and Pascal Paillier (Eds.).
[60] Gang Yao and Dengguo Feng. 2006. Proxy Oblivious Transfer Protocol. In *International Conference on Availability, Reliability and Security, ARES*.
[61] Bingsheng Zhang, Helger Lipmaa, Cong Wang, and Kui Ren. 2013. Practical Fully Simulatable Oblivious Transfer with Sublinear Communication. In *FC*.
[62] Shengnan Zhao, Xiangfu Song, Han Jiang, Ming Ma, Zhihua Zheng, and Qiuliang Xu. 2020. An Efficient Outsourced Oblivious Transfer Extension Protocol and Its Applications. *Secur. Commun. Networks* (2020).

A Security Model

In this paper, we rely on the simulation-based model of secure multi-party computation [35, 36] to define and prove the proposed protocols. Below, we restate the formal security definition within this model.

A.0.1 Two-party Computation. A two-party protocol Γ problem is captured by specifying a random process that maps pairs of inputs to pairs of outputs, one for each party. Such process is referred to as a functionality denoted by $f : \{0,1\}^* \times \{0,1\}^* \rightarrow \{0,1\}^* \times \{0,1\}^*$, where $f := (f_1, f_2)$. For every input pair (x, y), the output pair is a random variable $(f_1(x, y), f_2(x, y))$, such that the party with input x wishes to obtain $f_1(x, y)$ while the party with input y wishes to receive $f_2(x, y)$. In the setting where f is asymmetric and only one party (say the first one) receives the result, f is defined as $f := (f_1(x, y), \epsilon)$.

A.0.2 Security in the Presence of Passive Adversaries. In the passive adversarial model, the party corrupted by such an adversary correctly follows the protocol specification. Nonetheless, the adversary obtains the internal state of the corrupted party, including the transcript of all the messages received, and tries to use this to learn information that should remain private. Loosely speaking, a protocol is secure if whatever can be computed by a party in the protocol can be computed using its input and output only. In the simulation-based model, it is required that a party's view in a protocol's execution can be simulated given only its input and output. This implies that the parties learn nothing from the protocol's execution. More formally, party i's view (during the execution of Γ) on input pair (x, y) is denoted by $\mathsf{View}_i^{\Gamma}(x, y)$ and equals $(w, r_i, m_1^i, ..., m_t^i)$, 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 A.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:

$$\{\mathsf{Sim}_1(x, f_1(x, y))\}_{x,y} \stackrel{c}{\equiv} \{\mathsf{View}_1^{\Gamma}(x, y)\}_{x,y}$$
$$\{\mathsf{Sim}_2(y, f_2(x, y))\}_{x,y} \stackrel{c}{\equiv} \{\mathsf{View}_2^{\Gamma}(x, y)\}_{x,y}$$

Definition A.2 (Efficiency). A $\mathcal{DQ}$-$\mathcal{OT}_1^2$ scheme is efficient if the running time of the receiver-side request (or query) generation algorithm, denoted as $\mathtt{Request}(1^\lambda, \mathsf{s}, \mathsf{pk})$, 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 A.3 (Sender-push communication). Let (a) $\mathtt{Action}_R(t)$ represents 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, a $\mathcal{DQ}$-$\mathcal{OT}_1^2$ 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 scheme's execution), 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 can push messages to the receiver R without receiving an explicit request directly from R. Formally,
$$\forall t : \mathtt{Action}_S(t) \subseteq \{\mathtt{SendMessage}(S, R)\}$$

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

B DQ-OT's Security Proof

Below, we prove DQ-OT's security, i.e., Theorem 4.4.

Proof. We consider the case where each party is corrupt.

B.0.1 Corrupt Receiver R. In the real execution, R's view is: $\mathsf{View}_R^{DQ\text{-}OT}(m_0, m_1, \epsilon, \epsilon, s) = \{r_R, g, C, p, e_0, e_1, m_s\}$, 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 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: $C', r'_1, r'_2, y'_0, y'_1 \stackrel{\$}{\leftarrow} \mathbb{Z}_p$.
   - 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 $u \stackrel{\$}{\leftarrow} \mathbb{Z}_p$ and then sets $e'_s = (g^{y'_s}, \mathsf{H}(\beta_s'^{y'_s}) \oplus m_s)$ 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 $e_{1-s} =$

$(g^{y_{1-s}}, \mathsf{H}(\beta_{1-s}^{y_{1-s}}) \oplus m_{1-s})$, where $\beta_{1-s}^{y_{1-s}} = \frac{C}{g^x} = g^{a-x}$. Since y_{1-s} in the real model and y'_{1-s} in the ideal model have been picked uniformly at random and unknown to the adversary/distinguisher, $g^{y_{1-s}}$ and $g^{y'_{1-s}}$ have identical distributions.

Furthermore, in the real model, given $C = g^a$, due to the DL problem, a cannot be computed by a PPT adversary. Also, due to CDH assumption, R cannot compute $\beta_{1-s}^{y_{1-s}}$ (i.e., the input of H), given $g^{y_{1-s}}$ and g^{a-x}. We also know that H is modelled as a random oracle and its output is indistinguishable from a random value. Thus, $\mathsf{H}(\beta_{1-s}^{y_{1-s}}) \oplus m_{1-s}$ 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 ideal 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 an identical distribution in both models.

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

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

(1) constructs an empty view and appends uniformly random coin r'_S to it, where r'_S will be used to generate random values for S.
(2) picks random values $C', r' \xleftarrow{\$} \mathbb{Z}_p$.
(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 the DL assumption, β_s and β_{1-s} have identical distributions and are indistinguishable. In the ideal model, r' has been picked uniformly at random and we know that a' in $C' = g^{a'}$ is a uniformly random value, unknown to the adversary; therefore, due to DL assumption, β'_0 and β'_1 have identical distributions too. Moreover, values $\beta_s, \beta_{1-s}, \beta'_0$, and β'_1 have been defined over the same field, $\mathbb{Z}_p$. Thus, they have identical distributions and are indistinguishable.

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

B.0.3 Corrupt Server P_2. In the real execution, P_2's view is: $\mathtt{View}_{P_2}^{DQ\text{-}OT}\big((m_0, m_1), \epsilon, \epsilon, s\big) = \{g, C, p, s_2, r_2\}$. Below, we show how an ideal-model simulator $\mathtt{Sim}_{P_2}$ 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 $\mathtt{SS}$.
(3) appends s'_2, C' and r'_2 to the view and outputs the view.

Next, we explain why the views in the ideal and real models are indistinguishable.

Since 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 $\mathtt{SS}$ each share s_2 is indistinguishable from a random value s'_2, where $s'_2 \in \mathbb{U}$. Also, both C and C' have been picked uniformly at random from $\mathbb{Z}_p$; therefore, they have identical distribution.

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

B.0.4 Corrupt Server P_1. In the real execution, P_1's view is: $\mathtt{View}_{P_1}^{DQ\text{-}OT}\big((m_0, m_1), \epsilon, \epsilon, s\big) = \{g, C, p, s_1, r_1, \delta_0, \delta_1\}$. Ideal-model $\mathtt{Sim}_{P_1}$ 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 $\mathtt{SS}$.
(4) appends $s'_1, C', r'_1, \delta'_0, \delta'_1$ to the view and outputs the view.

Now, we explain why the views in the ideal and real models are indistinguishable. 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 4.2) holds. □

C 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 [52], the random value a (i.e., the discrete logarithm of random value C) is inserted by receiver R into the query β_{1-s} 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 6 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 [52].

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.

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

Table 6: 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.

Moreover, given $g^{y_s} \in e_s$, R can recompute $\mathsf{H}(g^{y_s})^x$, as it knows the value of s, s_1, and s_2. Specifically, as Table 6 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 $\mathsf{H}((g^{y_0})^x) = \mathsf{H}((g^{y_0})^{r_2+r_1}) = q$. Also, $e_0 = \mathsf{H}(\beta_0^{y_0}) \oplus m_0 = \mathsf{H}((g^{r_2+r_1})^{y_0}) \oplus m_0$. Thus, $q \oplus e_0 = m_0$.
  - In Case 2, it holds $\mathsf{H}((g^{y_1})^x) = \mathsf{H}((g^{y_1})^{r_2+r_1}) = q$. Moreover, $e_1 = \mathsf{H}(\beta_1^{y_1}) \oplus m_1 = \mathsf{H}((g^{r_2+r_1})^{y_1}) \oplus m_1$. 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 $\mathsf{H}((g^{y_0})^x) = \mathsf{H}((g^{y_0})^{r_2-r_1}) = q$. On the other hand, $e_0 = \mathsf{H}(\beta_0^{y_0}) \oplus m_0 = \mathsf{H}((g^{r_2-r_1})^{y_0}) \oplus m_0$. Therefore, $q \oplus e_0 = m_0$.
  - In Case 4, it holds $\mathsf{H}((g^{y_1})^x) = \mathsf{H}((g^{y_1})^{r_2-r_1}) = q$. Also, $e_1 = \mathsf{H}(\beta_1^{y_1}) \oplus m_1 = \mathsf{H}((g^{r_2-r_1})^{y_1}) \oplus m_1$. 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.

D DUQ-OT's Security Proof

Below, we prove DUQ-OT's security theorem, i.e., Theorem 5.2. 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.

D.0.1 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) initiate 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 $\bar{e}_0 = (g^{y'_0}, \mathsf{G}(\beta_0'^{y'_0}) \oplus (m_s || r'_3))$ 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 $e'_i = e_{1-s} = (g^{y_{1-s}}, \mathsf{G}(\beta_{1-s}^{y_{1-s}}) \oplus (m_{1-s} || r_3))$, 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 $e_{1-s} = (g^{y_{1-s}}, \mathsf{G}(\beta_{1-s}^{y_{1-s}}) \oplus (m_{1-s} || r_3))$, 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 the 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 G), given $g^{y_{1-s}}$ and g^{a-x}. We know that G is considered a random oracle and its output is indistinguishable from a random value. Therefore, $\mathsf{G}(\beta_{1-s}^{y_{1-s}}) \oplus (m_{1-s} || r_3)$ 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 identical distribution in both models. Both C and C' have also been picked uniformly at random from $\mathbb{Z}_p$; therefore, they have identical distribution.

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

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

D.0.2 Corrupt Sender S. In the real model, S's view is:

$$\mathtt{View}_S^{DUQ\text{-}OT}\big((m_0, m_1), \epsilon, \epsilon, \epsilon, s\big) = \{r_S, C, r_3, \beta_0, \beta_1\}$$

where r_S is the outcome of the internal random coin of S. Next, we construct an idea-model simulator $\mathtt{Sim}_S$ which receives $\{m_0, m_1\}$ from S.

(1) constructs an empty view and appends uniformly random coin r'_S to it, where r'_S will be used to generate random values for S.
(2) picks random values $C', r' \xleftarrow{\$} \mathbb{Z}_p, r'_3 \xleftarrow{\$} \{0,1\}^\lambda$.
(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 the DL assumption, β_s and β_{1-s} have identical distributions and are indistinguishable.

In the ideal model, r' and C' have been picked uniformly at random and we know that a' in $C' = g^{a'}$ is a uniformly random value, unknown to the adversary; thus, due to DL assumption, β'_0 and β'_1 have identical distributions too. The same holds for values C and C'.

Moreover, values $\beta_s, \beta_{1-s}, \beta'_0$, and β'_1 have been defined over the same field, $\mathbb{Z}_p$. Thus, they have identical distributions and are indistinguishable. The same holds for values r_3 in the real model and r'_3 in the ideal model.

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

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

$$\texttt{View}_{P_2}^{DUQ\text{-}OT}\big((m_0, m_1), \epsilon, \epsilon, \epsilon, s\big) = \{g, C, p, s_2, r_2\}$$

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 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 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 5.1) holds.

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

$$\texttt{View}_{P_1}^{DUQ\text{-}OT}\big((m_0, m_1), \epsilon, \epsilon, \epsilon, s\big) = \{g, C, p, s_1, r_1, \delta_0, \delta_1\}$$

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 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 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 5.1) holds.

D.0.5 Corrupt T. T's view can be easily simulated. It has an 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

$$\texttt{View}_{T}^{DUQ\text{-}OT}\big((m_0, m_1), \epsilon, \epsilon, \epsilon, s\big) = \{r_T, g, C, p\}$$

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 5.1) holds. □

E Proof of Lemma 6.2

Below, we prove Lemma 6.2 presented in Section 6.1.3.

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

F DQ$^{\text{MR}}$–OT in more Detail

Figure 10 presents the DQ$^{\text{MR}}$–OT that realizes $\mathcal{DQ}^{\mathcal{MR}}\text{–}\mathcal{OT}_1^2$.

Theorem F.1. *Let $\mathcal{F}_{\mathcal{DQ}^{\mathcal{MR}}\text{–}\mathcal{OT}_1^2}$ be the functionality defined in Section 6.1.1. If DL, CDH, and RO assumptions hold, then DQ$^{\text{MR}}$–OT (presented in Figure 10) 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.*

Appendix G presents the proof of Theorem F.1.

G Proof of Theorem F.1

Below, we prove the security of DQ$^{\text{MR}}$–OT, i.e., Theorem F.1.

(1) *S-side Initialization:* $\mathtt{Init}(1^\lambda) \rightarrow pk$
  (a) chooses a sufficiently large prime number p.
  (b) selects random element $C \xleftarrow{\$} \mathbb{Z}_p$ and generator g.
  (c) publish $pk = (C, p, g)$.
(2) *R-side Delegation:* $\mathtt{R.Request}(1^\lambda, s, pk) \rightarrow req = (req_1, req_2)$
  (a) split the private index s into two shares (s_1, s_2) by calling $\mathtt{SS}(1^\lambda, s, 2, 2) \rightarrow (s_1, s_2)$.
  (b) pick two uniformly random values: $r_1, r_2 \xleftarrow{\$} \mathbb{Z}_p$.
  (c) send $req_1 = (s_1, r_1)$ to P_1 and $req_2 = (s_2, r_2)$ to P_2.
(3) *P_2-side Query Generation:* $\mathtt{P_2.GenQuery}(req_2, , pk) \rightarrow q_2$
  (a) compute a pair of partial queries:
$$\delta_{s_2} = g^{r_2}, \quad \delta_{1-s_2} = \frac{C}{g^{r_2}}$$
  (b) send $q_2 = (\delta_0, \delta_1)$ to P_1.
(4) *P_1-side Query Generation:* $\mathtt{P_1.GenQuery}(req_1, q_2, pk) \rightarrow q_1$
  (a) compute a pair of final queries as:
$$\beta_{s_1} = \delta_0 \cdot g^{r_1}, \quad \beta_{1-s_1} = \frac{\delta_1}{g^{r_1}}$$
  (b) send $q_1 = (\beta_0, \beta_1)$ to S.
(5) *S-side Response Generation:* $\mathtt{GenRes}(m_{0,0}, m_{1,0}, \ldots, m_{0,z-1}, m_{1,z-1}, pk, q_1) \rightarrow res$
  (a) abort if $C \neq \beta_0 \cdot \beta_1$.
  (b) compute a response as follows. $\forall t, 0 \leq t \leq z-1$:
    (i) pick two random values $y_{0,t}, y_{1,t} \xleftarrow{\$} \mathbb{Z}_p$.
    (ii) compute response:
$e_{0,t} := (e_{0,0,t}, e_{0,1,t}) = (g^{y_{0,t}}, \mathsf{H}(\beta_0^{y_{0,t}}) \oplus m_{0,t})$
$e_{1,t} := (e_{1,0,t}, e_{1,1,t}) = (g^{y_{1,t}}, \mathsf{H}(\beta_1^{y_{1,t}}) \oplus m_{1,t})$
  (c) send $res = (e_{0,0}, e_{1,0}), ..., (e_{0,z-1}, e_{1,z-1})$ to P_1.
(6) *P_1-side Oblivious Filtering:* $\mathtt{OblFilter}(res) \rightarrow res'$
  - forward $res' = (e_{0,v}, e_{1,v})$ to R and discard the rest of the messages received from S.
(7) *R-side Message Extraction:* $\mathtt{Retrieve}(res', req, pk) \rightarrow m_s$
  (a) set $x = r_2 + r_1 \cdot (-1)^{s_2}$.
  (b) retrieve message $m_{s,v}$ by setting:
$$m_{s,v} = \mathsf{H}((e_{s,0,v})^x) \oplus e_{s,1v}$$

Figure 10: DQ$^{\mathrm{MR}}$–OT: Our protocol that realizes $\mathcal{DQ}^{\mathcal{MR}}$–$\mathcal{OT}_1^2$.

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

G.0.1 Corrupt R. Recall that in DQ$^{\mathrm{MR}}$–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: $\mathtt{View}_R^{\mathrm{DQ^{MR}-OT}}(m_0, m_1, v, \epsilon, s) = \{r_R, g, C, p, e_{0,v}, e_{1,v}, m_{s,v}\}$, 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 $\mathtt{Sim}_R$ 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. $\mathtt{Sim}_R$ 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: $\mathtt{SS}(1^\lambda, s, 2, 2) \rightarrow (s'_1, s'_2)$.
  - 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 $e'_s = (g^{y'_s}, \mathsf{H}(\beta'^{y'_s}_s) \oplus m_s)$ 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 B) 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 6.1.1) holds.

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

Therefore, Relation 8 (in Section 6.1.1) holds.

G.0.3 Corrupt P_2. This case is identical to the corrupt P_2 case in the proof of DQ-OT. So, Relation 10 (in Section 6.1.1) holds.

G.0.4 Corrupt P_1. In the real execution, P_1's view is: $\mathtt{View}_{P_1}^{\mathrm{DQ^{MR}-OT}}((m_0, m_1), v, \epsilon, s) = \{g, C, p, s_1, r_1, \delta_0, \delta_1, (e_{0,0}, e_{1,0}), \ldots, (e_{0,z-1}, e_{1,z-1})\}$. Ideal-model $\mathtt{Sim}_{P_1}$ 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 $\delta'_0, \delta'_1 \xleftarrow{\$} \mathbb{Z}_p$.
(3) picks two uniformly random values $s'_1 \xleftarrow{\$} \mathbb{U}$ and $C', r'_1 \xleftarrow{\$} \mathbb{Z}_{p-1}$, where $\mathbb{U}$ is the output range of SS.
(4) picks z pairs of random values as follows $(a_{0,0}, a_{1,0}), ..., (a_{0,z-1}, a_{1,z-1}) \xleftarrow{\$} \mathbb{Z}_p$.

(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 B) 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 6.2, 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 6.1.1) holds. □

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

We prove the security of DUQ$^{\text{MR}}$–OT, i.e., Theorem 6.4.

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

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

$\mathtt{View}_R^{\text{DUQ}^{\text{MR}}\text{–OT}}(\boldsymbol{m}, (v, s, z)\ \epsilon, \epsilon, \epsilon) = \{r_R, g, C, p, r_3, s_2, o_0, o_1,\ m_{s,v}\}$, 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 $\mathtt{Sim}_R$ 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. $\mathtt{Sim}_R$ 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 $\bar{e}_0 := \left(\bar{e}_{0,0} = g^{y'_0}, \bar{e}_{0,1} = \mathtt{G}(\beta_0'^{y'_0}) \oplus (m_{s,v}||r'_3)\right)$ 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. $\forall i, i', 0 \leq i, i' \leq 1 : \bar{o}_{i,i'} = \mathtt{Enc}(pk, \bar{e}_{i,i'})$. 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.

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 the 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.

Further, (i) $y_{s,v}$ in the real model and y'_0 in the ideal model have been picked uniformly at random and (ii) the decryption of both e'_{1-i} and $\bar{e}'_{1-j}$ contain $m_{s,v}$; therefore, e'_{1-i} and $\bar{e}'_{1-j}$ have identical distributions. Also, $m_{s,v}$ has an 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 6.2) holds. That means, even though S holds z pairs of messages and generates a response for all of them, R's view is still identical to the case where S holds only two pairs of messages.

H.0.2 Corrupt S. This case is identical to the corrupt S in the proof of DUQ-OT (in Appendix D) 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 Appendix D) to argue why real model and ideal model views are indistinguishable (when S is corrupt), can be used in this case as well.

Therefore, Relation 12 (in Section 6.2) holds.

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

H.0.4 Corrupt P_1. In the real execution, P_1's view is: $\mathtt{View}_{P_1}^{\mathrm{DUQ^{MR}\text{-}OT}}(\boldsymbol{m}, (v, s, z), \epsilon, \epsilon, \epsilon) = \{g, C, p, s_1, \boldsymbol{w}_j, r_1, \delta_0, \delta_1, (e'_{0,0}, e'_{1,0}), ..., (e'_{0,z-1}, e'_{1,z-1})\}$. 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 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 Appendix D) 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 6.2, 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, $0 \le i \le z-1$. Also, due to the IND-CPA property of the additive homomorphic encryption scheme, the elements of $\boldsymbol{w}_j$ in the real model are indistinguishable from the elements of $\boldsymbol{w}'$ in the ideal model.

Hence, Relation 13 (in Section 6.2) holds.

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

I Proof of Theorem 7.1

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). □

J Proof of Theorem 8.2

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

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

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

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

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

(1) constructs an empty view and appends a uniformly random coin r'_S to the view.
(2) picks a binary random value $s' \xleftarrow{\$} \{0,1\}$.
(3) picks two uniformly random keys $(k'_0, k'_1) \xleftarrow{\$} \{0,1\}^\sigma$.
(4) appends s', k'_0, k'_1 to the view and outputs the view.

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

been picked uniformly at random from the same domain. Hence, Relation 16 (in Section 8.1) holds.

J.0.3 Corrupt Server P. In the real execution, P's view is: $\texttt{View}_P^{Supersonic\text{-}OT}\big((m_0, m_1), \epsilon, s\big) = \{r_P, s_2, e'\}$, where r_P is the outcome of the internal random coin of P and is used to generate its random values and e' is a pair (e'_0, e'_1) and is an output of $\bar{\pi}$. Next, we construct an idea-model simulator $\texttt{Sim}_P$.

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

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

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

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

K Proof of Correctness

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

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

Table 7: Relation between query s and behaviour of permutation $\bar{\pi}$ from the perspective of S and P. When $s_i = 1$, $\bar{\pi}$ swaps the elements of its input pairs and when $s_i = 0$, $\bar{\pi}$ does not swap the elements of the input pairs.

As Table 7 indicates, when $s = 0$, then (i) either both S and P permute their pairs or (ii) neither does. In the former case, since both swap the elements of their pair, then the final permuted pair e'' will have the same order as the original pair e (before it was permuted). In the latter case, again e'' will have the same order as the original pair e because neither party has permuted it. Thus, in both of the above cases (when $s = 0$), the first element of e'' will be the encryption of m_0. Moreover, as Table 7 indicates, when $s = 1$, then only one of the parties S and P will permute their input pair. This means that the first element of the final permuted pair e'' will always equal the encryption of m_1.