A Survey of Oblivious Transfer Protocol

VIJAY KUMAR YADAV, Department of Computer Science Engineering and Technology, Bennett University, India
NITISH ANDOLA, Department of Computing Science and Engineering/Information Technology, Jaypee Institute of Information Technology, Noida, India
SHEKHAR VERMA and S. VENKATESAN, Indian Institute of Information Technology Allahabad, India

Oblivious transfer (OT) protocol is an essential tool in cryptography that provides a wide range of applications such as secure multi-party computation, private information retrieval, private set intersection, contract signing, and privacy-preserving location-based services. The OT protocol has different variants such as one-out-of-2, one-out-of-n, k-out-of-n, and OT extension. In the OT (one-out-of-2, one-out-of-n, and OT extension) protocol, the sender has a set of messages, whereas the receiver has a key. The receiver sends that key to the sender in a secure way; the sender cannot get any information about the received key. The sender encrypts every message by operating on every message using the received key and sends all the encrypted messages to the receiver. The receiver is able to extract only the required message using his key. However, in the k-out-of-n OT protocol, the receiver sends a set of k keys to the sender, and in replay, the sender sends all the encrypted messages. The receiver uses his keys and extracts the required messages, but it cannot gain any information about the messages that it has not requested. Generally, the OT protocol requires high communication and computation cost if we transfer millions of oblivious messages. The OT extension protocol provides a solution for this, where the receiver transfers a set of keys to the sender by executing a few numbers of OT protocols. Then, the sender encrypts all the messages using cheap symmetric key cryptography with the help of a received set of keys and transfers millions of oblivious messages to the receiver. In this work, we present different variants of OT protocols such as one-out-of-2, one-out-of-n, k-out-of-n, and OT extension. Furthermore, we cover various aspects of theoretical security guarantees such as semi-honest and malicious adversaries, universally composable, used techniques, computation, and communication efficiency aspects. From the analysis, we found that the semi-honest adversary-based OT protocols required low communication and computation costs as compared to malicious adversary-based OT protocols.

CCS Concepts: • **Security and privacy → Public key encryption**; **Access control**; **Privacy-preserving protocols;**

Additional Key Words and Phrases: Oblivious transfer protocol, multiparty computation, oblivious transfer extension protocol, pseudorandom generator, random oracle model

ACM Reference format:
Vijay Kumar Yadav, Nitish Andola, Shekhar Verma, and S. Venkatesan. 2022. A Survey of Oblivious Transfer Protocol. *ACM Comput. Surv.* 54, 10s, Article 211 (September 2022), 37 pages.
https://doi.org/10.1145/3503045

Authors' addresses: V. K. Yadav, Department of Computer Science Engineering and Technology, Bennett University, Greater Noida, India; email: vijaykumarmit55@gmail.com; N. Andola, Department of Computing Science and Engineering/Information Technology, Jaypee Institute of Information Technology, Noida, New Delhi, India; email: andola.nitish18@gmail.com; S. Verma and S. Venkatesan, Indian Institute of Information Technology Allahabad, Devghat, Jhalwa, Prayagraj, U.P., India; emails: {sverma, venkat}@iiita.ac.in.

0360-0300/2022/09-ART211 $15.00
https://doi.org/10.1145/3503045

1 INTRODUCTION

In *distributed computing*, several parties (or computing devices) jointly evaluate some function. For instance, we may assume that these devices to be servers that contain a distributed database, and that the function to be computed may be read-write database operations. ***Secure multiparty computation* (SMC)** is used to compute such kinds of distributed computing securely. Generally, in distributed computing, we mainly deal with inadvertent faults and the threat of machine crashes, while in SMC, we deal with the participating entities' malicious behaviors. The target of adversarial entities in the SMC is to determine the secret information or aim to the output of the function incorrectly. Any multiparty computation protocol is said to be secure from these adversarial activities if it satisfies two conditions: *privacy* and *correctness* [39, 91]. If the participating entity in the secure computation learns only the protocol's output, nothing else, then that protocol satisfies the privacy requirement. If the participating entity receives the correct output after the protocol's execution, then that protocol satisfies the correctness requirement. Some researchers have proposed different security definitions for secure multiparty computation protocol [6, 21, 35, 45]. Following are security definitions of the SMC protocol:

- *Privacy:* Every party should learn only the output of the protocol, nothing else.
- *Correctness:* After the execution of the protocol, every party should get the correct output.
- *Independent of Inputs:* Corrupted parties that follow the adversaries' instruction can choose their input independently with respect to the honest parties' inputs.
- *Guaranteed output delivery:* Under any circumstances, the corrupted party cannot prevent honest parties to receive their output.
- *Fairness:* If the corrupted entities receive their output, then honest entities also receive their output [42, 43].

The above security definitions for the SMC are only applicable to the protocol's security requirement. However, these security definitions are not suitable for the following reasons: It may be a chance that the crucial requirement has been missed because it is true that different applications have different security requirements. Moreover, the definition needs to be simple so it can cover all the security requirements of the SMC protocol. The standard security definition has been proposed to deal with these issues [4, 13, 40, 66]. By formalizing the security definition in the following way: There exists an *ideal world*, where a trusted party helps the participating parties in the multiparty computation protocol to compute the output of the protocol securely. All the parties send their private input to the trusted party via a secure channel in this world. The trusted party then evaluates the output of the received inputs' function and sends that output to every party. In this world, the adversary has the freedom to choose the corrupt parties' inputs while the honest parties can send their input to the trusted party. This ideal world computation technique satisfies all the security requirements, as the parties receive their prescribed output from the trusted party, nothing else. Hence, the privacy requirement is satisfied. Similarly, as the trusted party is honest, it always computes the function correctly and sends the parties' correct output. Hence, the correctness requirement is also satisfied. Indeed, no such trusted authority exists in the real world on which all the parties can trust. Therefore, all the parties want to execute some protocol without the trusted party. In other words, all the parties execute a *real protocol* (without a trusted party) and use their inputs to compute the output of the function securely. This real-world protocol is considered to be secure; if there exists an adversary that can break this protocol, then that can also break the ideal-world protocol. As no such adversary exists that can break the ideal world protocol. Hence, we conclude that no such adversary can break the real-world protocol. We formally define the security of the real-world protocol as follows:

Definition 1.1. Suppose there exists an adversary that can successfully break the real-world protocol. In that case, it can also break the ideal-world protocol such that the input/output distributions of the participating entities and adversary in the real and ideal-world execution are completely indistinguishable.

This definition satisfies all the security properties such as privacy, correctness, independence of inputs, guaranteed output delivery, and fairness [45].

An ***oblivious transfer*** **(OT)** protocol is a crucial tool used for the secure computation of any multiparty computation. Rabin introduced this protocol [82]. The OT protocol has three varieties.

(1) *1-out-of-2 OT protocol:* Two parties participate, sender and receiver. The sender has two inputs string $(x_0, x_1) \in \{0,1\}^l$, while the receiver has a selection bit $r \in \{0,1\}$. Once the protocol ends, the receiver learns x_r (without knowing anything about x_{1-r}), while the sender learns nothing about the bit r. This type of OT protocol mainly used in the secure evaluation of garbled circuit [7, 56, 57, 70, 79, 92].
(2) *1-out-of-n OT protocol:* Two parties participate sender and receiver. The sender has n inputs string $(x_0, x_1, \ldots, x_n) \in \{0,1\}^l$, while the receiver has a selection number $r \in \{0,1,\ldots,n-1\}$. Once the protocol ends, the receiver learns x_r (without knowing anything about x_ψ, where $\psi \in \{0,1,\ldots,n-1\}$ and $\psi \neq r$), while the sender learns nothing anything about the number r. This type of OT protocol mainly used in *private information retrieval* [15, 27, 83] and *location-based services* [10, 51].
(3) *k-out-of-n OT protocol:* Two parties participate, sender and receiver. The sender has n inputs string $(x_0, x_1, \ldots, x_n) \in \{0,1\}^l$, while the receiver has a set of k numbers $\{r_1, r_2, \ldots, r_k\} \subset \{0,1,\ldots,n-1\}$. Once the protocol ends, the receiver learns $\{x_{r_1}, x_{r_2}, \ldots, x_{r_k}\}$ (without knowing anything about x_ψ, where $\psi \subset \{0,1,\ldots,n-1\}$ and $\psi \nsubseteq \{r_1, r_2, \ldots, r_k\}$), while the sender learns nothing anything about the numbers $\{r_1, r_2, \ldots, r_k\}$. This type of OT protocol is mainly used in *private set intersection* [73, 78, 80, 81].

Millions of OT protocols are used for efficient and secure evaluation of multiparty protocol [24, 25, 29, 60]. To minimize the computation costs in the multiparty protocol, an extended version of the OT protocol, called *OT extension*, is used.

OT Extension Protocol: In the OT extension protocol, we use a few OT protocols to transfer millions of oblivious messages. In this protocol, the sender has m pairs of input strings $\{(x_0^j, x_1^j)\}_{j=1}^m$, and the size of every string is l bit while the receiver has m bit string $r = \{r_1, r_2, \ldots, r_m\}$ as input. Both sender and receiver execute a small number of one-out-of-2 OT protocol, called *base OT* protocol, to transfer a large number of OT protocols. After the end of this OT extension protocol, the receiver receives the following messages, $\{x_{r_j}^j\}_{j=1}^m$ while the sender gets nothing.

The above Definition 1.1 for SMC protocol is also applicable to OT protocol and OT extension. In other words, any OT protocol is considered to be secure against the malicious adversary if that OT protocol satisfies the security Definition 1.1.

1.1 Power of Adversaries

The above security Definition 1.1 skips out a crucial issue, such as adversarial power that attacks the protocol to extract the secret information. We consider the three types of adversarial power.

(1) **Corruption Approach:** The adversary controls a subset of the participating entities to get secret information from the protocol. Till now, we have not discussed how the participating entities come under the control of the adversary. The corruption approach describes how and when parties are corrupted. The adversary follows two types of corruption strategies to corrupt the entities.

- *Static Corruption Approach:* In this approach, before executing the protocol, the adversary chooses a set of corrupted parties. Throughout the protocol's execution, the corrupted parties remain corrupt, and the honest parties remain honest.
- *Adaptive Corruption Approach:* In this approach, the adversary can corrupt a set of parties during the execution of the protocol. The adversary can choose any set of parties according to its requirement during the execution of the protocol. Moreover, once the party gets corrupted in this approach, it remains corrupted until the completion of the protocol.
- *Proactive Corruption Approach:* In this approach, parties may be corrupted for a specific amount of time only. Moreover, in this approach, during the execution of the protocol, corrupted parties may become honest parties, and the honest parties may become corrupted [14, 74].

(2) **Allowed Adversarial Behavior:** The adversary controls the corrupted parties, giving instructions to the parties to passively gather private information from the protocol output or modify the protocol to get the secret information. In this scenario, there are three types of adversaries possible.

- *Semi-honest Adversaries:* These types of adversaries are also called *passive* and *honest-but-curious*. In this case, the adversaries instruct the corrupt parties to follow the rule of the protocol correctly. In contrast, the adversaries use the corrupted parties' output and try to learn the secret information from the protocol. This type of adversary is considered to be a weak adversary. We use this type of adversary where the server executes the correct program.
- *Malicious Adversaries:* These types of adversaries are also called *active* adversaries. In this case, adversaries instruct the corrupted parties to not follow the protocol rule, and the parties can modify the protocol and get the private information from the protocol. Most commonly, we prefer the protocol that is secure against malicious adversary presence so no adversarial attack can happen in the future. This type of adversary is considered to be a strong adversary. Generally, to achieve this level of security, we require high computation and communication costs. In other words, these types of protocols are inefficient.
- *Covert Adversaries:* If a protocol is secure in a semi-honest adversary's presence, then that protocol is considered a weakly secure protocol. However, if a protocol is secure in the presence of an active adversary, then that protocol is deemed to be a strongly secure. Generally, these protocols are inefficient. Therefore, an intermediate adversary is required; such type of adversary is called covert adversary. In this case, the covert adversary is allowed to behave maliciously, and if it does so, it will be caught by the honest parties with some given probabilities.

(3) **Computationally dependent adversaries:** According to the computational capacity of the adversaries, we consider two types of adversaries.

- *Polynomial time adversaries:* This type of adversary is allowed to break the protocol's security in probabilistic polynomial time (or finite time).
- *Computationally unbounded adversaries:* In this case, we give an unlimited amount of time to the adversaries to break the protocol's security.

The above distinction about the complexity of adversity introduces two different models for secure computation: *computational model* [39, 91] and *information-theoretic* model [9, 16]. In the computational model, we considered that the adversary tries to break the protocol's security in polynomial time. Therefore, in this model, we build the protocol under cryptographic assumptions like the trapdoor function. In this scenario, we consider that communication takes place between the parties' from the authenticated channel. The parties do not need to access private channels,

because the private channels are implemented using asymmetric key encryption techniques. However, the authentication channels are implemented using digital signatures [12, 32, 41, 58, 85] and public-key infrastructure. If the parties use the authenticated channel, then any adversaries who are listening to the medium cannot modify the message communicated between the parties.

In the information-theoretic case, the adversary does not depend on any time complexity like polynomial time. Therefore, in this model, we build the protocol that does not rely on any security assumption like a trapdoor function. We only consider that the parties communicate through a private channel and reconsider that adversary cannot eavesdrop on the channel when the honest parties communicate. The information-theoretic security in any of the protocols can be achieved if the number of honest parties in the protocol is higher than the corrupt parties. However, in the OT protocol, only two parties participate, so information-theoretic security cannot be achieved in OT protocol.

1.2 Applications of Oblivious Transfer Protocol

The OT protocol has been used in three crucial applications, such as **private set intersection (PSI), private information retrieval (PIR)**, and **location-based services (LBS)**.

In PSI protocol, two parties participate, sender and receiver. The sender has a set of n private inputs $X = \{x_1, x_2, \ldots, x_n\}$ while the receiver has a set of m private inputs $Y = \{y_1, y_2, \ldots, y_n\}$. Both parties execute the PSI protocol with the help of OT protocol on their inputs. At the end of the PSI protocol, the receiver securely gets the intersection of their input $X \cap Y$ while the sender receives nothing [20, 26, 48, 55]. The two most real-life applications of the private set intersection are child safety and password monitoring. With the help of PSI, we prevent the spread of **Child Sexual Abuse Material (CSAM)**. PSI warns children and their parents when receiving or sending sexually explicit photos. Before an image is stored in Cloud, an on-device matching process is performed for that image against the known CSAM hashes. This matching process is powered by a cryptographic technology called PSI, which determines if there is a match without revealing the result. Password monitoring informs users if any of their saved passwords have been found in a third-party breach. The underlying technology ensures the security and privacy of the user's passwords, which means that any other party cannot learn the user's passwords while they are being monitored.

In PIR protocol, two parties participate user and database administrator. The database administrator divides its database D into a set of n blocks $D = \{d_1, d_2, \ldots, d_n\}$. The user sends the query $\gamma \in [n]$ regarding any one of the blocks to the database administrator. In return, the database administrator sends all the blocks, $D = \{d_1, d_2, \ldots, d_n\}$ of the database to the user, but the user is only able to extract the required data block d_γ. Moreover, the database administrator is unable to get any information about the user query γ [23, 27, 54, 68]. PIR schemes allow users to extract information from a database while keeping their identities private. The real-life examples are as follows: e-health application or medical databases, databases with sensitive information, such as stocks, in which users are likely to be highly motivated to hide which record they are trying to retrieve. The PIR schemes aim to achieve this goal efficiently with the help of OT protocol, where the main cost measure has traditionally been the communication complexity.

In the LBS scheme also two parties participate, location server and LBS user. In this scheme, the location server divides a big area A into a set of n regions $A = \{a_1, a_2, \ldots, a_n\}$. The user sends a query $\gamma \in [n]$ regarding the information about any particular region to the location server. In return, the location server sends all the information of all the regions to the LBS user, but the LBS user is only able to extract the information about the required region a_γ. Moreover, the location server cannot get any information about the user query γ [10, 51, 76]. In **vehicular ad hoc networks (VANETs)**, the vehicle user sends his current location to the location server. In return,

Input of S**:** It has two messages $x_0 \in \{0,1\}^l$ and $x_1 \in \{0,1\}^l$.
Input of R**:** It has one bit $r \in \{0,1\}$.
Common Inputs: Security parameter λ.
Output of S**:** Nothing.
Output of R**:** x_r.

(1) The sender S randomly chooses two messages $m_0 \in \{0,1\}^l$, $m_1 \in \{0,1\}^l$, and sends them with E, pkS to the receiver R.
(2) The receiver R randomly chooses a message $K \in \{0,1\}^l$, besides, he chooses a bit $r \in \{0,1\}$. After that, he computes $q = E_{pkS}(K) \oplus m_r$ and sends it to S.
(3) S uses his secret key skS and computes the following $\{K_i = D_{skS}(q \oplus m_i)\}_{i=0}^{1}$. After that, he encrypts his messages as follows $\{C_i = x_i \oplus K_i\}_{i=0}^{1}$ and sends the ciphertexts C_0 and C_1 to R.
(4) R extracts his desired message $x_r = C_r \oplus K$.

Fig. 1. Public-key encryption based 1-out-of-2 OT protocol for semi-honest receiver [33].

the location server sends all the information corresponding to the vehicle user's current location, but the vehicle user is only able to extract the information about the required location. However, the location server cannot get any information about the vehicle user's location. In this scenario, the OT protocol helps to maintain the vehicle user's location privacy from the location server.

2 OBLIVIOUS TRANSFER PROTOCOL

In the OT protocol, two parties participate, sender and receiver. The sender has a set of inputs, while the receiver has only one input. The receiver sends his input (without revealing the content of this input) to the sender. The sender sends his set of all input to the receiver, but the receiver can only obtain the output message corresponding to the requested input. The OT protocol was originated by Rabin, whose security depends on the factorization of a composite number [82]. However, this protocol is not suitable for practical applications. In this section, we survey three flavors of OT protocols.

2.1 1-out-of-2 OT Protocol

The first practically feasible 1-out-of-2 OT protocol has been proposed under the assumption of public-key cryptography [33]. This public-key cryptographic scheme consists of three algorithms, key generation G, encryption E, and decryption D [28]. The algorithm G takes input as a security parameter λ, and it produces public key pk and private key sk. The algorithm E uses the public key pk and computes the ciphertext C on the message M. In other words, $C = E_{pk}(M)$. However, the decryption algorithm extracts the message M from the ciphertext with the help of private key sk. In other words, $M = D_{sk}(C)$. The description of 1-out-of-2 OT protocol [33] is given in Figure 1. As described in the figure, both sender and receiver choose their input. In item 1, the sender chooses a public key encryption technique and executes the key generation algorithm G to generate private and public keys, respectively, skS and pkS. After that, he randomly generates two messages m_0 and m_1 and sends them with the public key pkS, and encryption algorithm E to the receiver. In item 2, the receiver chooses the desired bit r and encrypts the randomly generated a message K by him using the received public key pkS. After that, he computes $q = E_{pkS}(K) \oplus m_r$ and sends it to S. Since the sender knows that if the receiver has chosen $r = b \in \{0,1\}$, then the received expression q is equal to $q = E_{pkS}(K) \oplus m_b$. Therefore, in item 3, the sender computes $\{K_i = D_{skS}(q \oplus m_i)\}_{i=0}^{1}$, because he knows that at $r = b$, he gets the correct value of $K_r = K$ used by the receiver, while at $r = 1 - b$ he gets a uniformly random key K_{1-b}. The sender uses the values of K_0, K_1 and encrypts the values of x_0 and x_1 and sends them to the receiver. In the last item, the receiver uses the input bit r and computes his required message, $x_r = C_r \oplus K$. This protocol is only secure in the presence

Common Inputs: Security parameter λ, a multiplicative group, $\mathcal{Z}_p^*$ of prime order p with generator g. Moreover, $C \in \mathcal{Z}_p^*$ known to all the users, but discrete log of C is unknown to all the users.

(1) The receiver randomly chooses a number $s \in \{0, 1, ...p-1\}$, besides he chooses an input bit $r \in \{0, 1\}$. After that, he sets $\beta_r = g^s$, and $\beta_{1-r} = C.(g^s)^{-1}$. The values of (β_0, β_1) and (r, s) are respectively public keys and secret keys of the receiver. Note that, anyone can verify the public key of the receiver by computing, $C = \beta_0\beta_1$.
(2) The sender randomly chooses two numbers $\gamma_0, \gamma_1 \in \{0, 1, ..., p-2\}$ and sends $\alpha_0 = g^{\gamma_0}$, $\alpha_1 = g^{\gamma_1}$ to R. After that, the sender computes, $y_0 = \beta_0^{\gamma_0}$, $y_1 = \beta_1^{\gamma_1}$ and sends $s_0 = x_0 \oplus y_0$ and $s_1 = x_1 \oplus y_1$ to R.
(3) R uses the values of α_0, α_1, and secret key (r, s) and extracts the required message as follows, $x_r = y_r \oplus s_r$, where, $\alpha_r^s = y_r$.

Fig. 2. Non-interactive 1-out-of-2 OT protocol for semi-honest receiver [8].

Common Inputs: Security parameter λ, a multiplicative group, $\mathcal{Z}_p^*$ of prime order p with generator g, ROM $H : \{0, 1\}^* \rightarrow \{0, 1\}^l$. Moreover, $C \in \mathcal{Z}_p^*$ known to all the users, but discrete log of C is unknown to all the users.

(1) The receiver randomly chooses a number $s \in \{0, 1, ...p-1\}$, besides he chooses an input bit $r \in \{0, 1\}$. After that, he sets $\beta_r = g^s$, and $\beta_{1-r} = C.(g^s)^{-1}$. The receiver sends the value of β_0 to the sender and keeps the values of (r, s) private.
(2) The sender computes $\beta_1 = C/\beta_0$ and verifies the public key of the receiver as follows, $C \stackrel{?}{=} \beta_0\beta_1$. If this condition holds true, then the sender continues the OT protocol; otherwise, he aborts. The sender randomly chooses two numbers $\gamma_0, \gamma_1 \in \{0, 1, ..., p-2\}$ and encrypts x_0 by $e_0 = (g^{\gamma_0}, H(\beta_0^{\gamma_0}) \oplus x_0)$, and encrypts x_1 by $e_1 = (g^{\gamma_1}, H(\beta_1^{\gamma_1}) \oplus x_1)$, and sends the values of (e_0, e_1) to R.
(3) R uses the values of (r, s) and extracts the required message as follows, $x_r = H((g^{\gamma_r})^s) \oplus H(\beta_r^{\gamma_r}) \oplus x_r$.

Fig. 3. ROM-based 1-out-of-2 OT protocol for semi-honest receiver [69].

of semi-honest adversaries. Moreover, due to the use of a public-key encryption technique, this protocol is computationally inefficient.

A new Diffie-Hellman assumption based OT protocol has been proposed to improve the efficiency of the OT protocol [8]. They have named this protocol as a non-interactive OT protocol. Description of this protocol is given in Figure 2. In this protocol, both sender and receiver take the same input as in the previous protocol. The output of this protocol is also similar to the previous protocol. Both sender and receiver agree on the following common inputs, k, p, g, Z_p^*, and C. The value of C is generated by the trusted authority so no user can know the discrete log of C. In item 1, the receiver randomly chooses an element s and uses his input bit r and computes the public keys, (β_0, β_1) in this way that any users cannot guess the secret bit r from the public keys. In item 2, the sender randomly chooses two numbers $\gamma_0, \gamma_1 \in \{0, 1, \ldots, p-2\}$ and sends $\alpha_0 = g^{\gamma_0}$, $\alpha_1 = g^{\gamma_1}$ to R. After that, the sender computes, $y_0 = \beta_0^{\gamma_0}$, $y_1 = \beta_1^{\gamma_1}$. The sender, then, uses the values of (y_0, y_1) and encrypts the values of (x_0, x_1) and sends them to the receiver. In the last item, the receiver uses his private key (s, r) and decrypts the required message, $x_r = y_r \oplus s_r$, where, $\alpha_r^s = y_r$.

In Reference [69], they claim that the non-interactive OT protocol is insecure, and it requires high communication and computation costs. Moreover, they have improved non-interactive OT protocol security and reduced communication and computation costs using a **random oracle model (ROM)**. The security of this protocol is based on the **computational Diffie-Hellman (CDH)** assumption. Description of ROM based 1-out-of-2 OT protocol is given in Figure 3. This protocol takes input similar to the non-interactive OT protocol, and the output produced by this is also identical to the non-interactive OT protocol. This protocol's common inputs are also similar

Common Inputs: Security parameter λ, a multiplicative group, $\mathcal{Z}_p^*$ of prime order p with generator g, ROM $H : \{0,1\}^* \rightarrow \{0,1\}^l$. Moreover, $C \in \mathcal{Z}_p^*$ known to all the users, but discrete log of C is unknown to all the users.

(1) The receiver randomly chooses a number $s \in \{0, 1, ...p-1\}$, besides he chooses an input bit $r \in \{0,1\}$. After that, he sets $\beta_r = g^s$, and $\beta_{1-r} = C.(g^s)^{-1}$. The receiver sends the value of β_0 to the sender and keeps the values of (r, s) private.
(2) The sender randomly chooses a number $\gamma \in \{0, 1, ..., p-2\}$ and computes $\beta_1^\gamma = C^\gamma / \beta_0^\gamma$. The sender, then encrypts x_0 by $e_0 = H(\beta_0^\gamma, 0) \oplus x_0$, and encrypts x_1 by $e_1 = H(\beta_1^\gamma, 1) \oplus x_1$, and sends the values of (e_0, e_1) with g^γ to R.
(3) R uses the values of (r, s) and extracts the required message as follows, $x_r = H((g^\gamma)^s, r) \oplus e_r$.

Fig. 4. Computationally efficient ROM-based 1-out-of-2 OT protocol for semi-honest receiver [69].

Common Inputs: Security parameter λ, a multiplicative group, $\mathcal{Z}_p^*$ of prime order p with generator g.

(1) The receiver randomly chooses two numbers $a, b \in \mathcal{Z}_p^*$, besides he chooses an input bit $r \in \{0,1\}$. After that, he sets $\beta_r = ab$, and chooses β_{1-r} randomly from $\mathcal{Z}_p^*$. The receiver then sends the values of $x = g^a$, $y = g^b$, $z_0 = g^{\beta_0}$, and $z_1 = g^{\beta_1}$ to the sender.
(2) The sender verifies $z_0 \neq z_1$ if it is true, then it continues the protocol otherwise aborts. After that, the sender randomly generates m_0, n_0, m_1, n_1 and computes, $\omega_0 = x^{n_0}.g^{m_0}$, $k_0 = z_0^{n_0}.y^{m_0}$ and $\omega_1 = x^{n_1}.g^{m_1}$, $k_1 = z_1^{n_1}.y^{m_1}$. The sender encrypts the value of x_0 using key k_0 while encrypts the value of x_1 using key k_1. In other words, he computes $e_0 = k_0.x_0$, $e_1 = k_1.x_1$, and sends the values of (e_0, ω_0) and (e_1, ω_1) to the receiver.
(3) R uses the values of (r, b) and extracts the required message as follows, $x_r = e_r.(k_r)^{-1}$, where $k_r = (\omega_r)^b$.

Fig. 5. Without ROM-based 1-out-of-2 OT protocol for semi-honest receiver [69].

to the previous protocol except that this protocol requires ROM H. Item 1 of this protocol is similar to the non-interactive OT protocol, except that the receiver sends the only β_0 to the sender instead of (β_0, β_1). In item 2, the sender verifies the public key of the receiver and randomly generates two elements γ_0, γ_1 from the multiplicative group $\mathcal{Z}_p^*$, and encrypts x_0 by using γ_0 while encrypting x_1 by using γ_1. After that, the sender sends the encrypted values e_0 and e_1 to the receiver. In the last item, the receiver uses his private key (s, r) and decrypts the required message x_r.

In the same work, they have further reduced the computation costs of their ROM-based OT protocol. The description of the computationally enhanced OT protocol is given in Figure 4. In this protocol, they have reduced one exponential operation than the previous version. Input, output, and item 1 of this protocol are similar to the previous version. In item 2, the sender randomly generates an element γ from the multiplicative group Z_q^* and computes $\beta_1^\gamma = C^\gamma / \beta_0^\gamma$. After that, the sender encrypts the value of x_0 using $H(\beta_0^\gamma, 0)$, while encrypting the value of x_1 using $H(\beta_1^\gamma, 1)$, then sends the encrypted values (e_0, e_1) to the receiver. In the last item, R uses his private key (r, s) and extracts the required message as follows: $x_r = H((g^\gamma)^s, r) \oplus H(\beta_r^\gamma, r) \oplus x_r$.

A trusted authority is required for the construction of C in the previous OT protocols, which are described in Figures 2, 3, and 4. Moreover, these protocols use the ROM. Therefore, to avoid the ROM and the trusted authority, another OT protocol has been proposed [69]. The security of this protocol relies on the **decisional Diffie-Hellman (DDH)** assumption. The description of this protocol is given in Figure 5. Input, common input, and output of this protocol are similar to the non-interactive OT protocol except that the value of C. In item 1, the receiver randomly generates two numbers (a, b) from the multiplicative group $\mathcal{Z}_p^*$ and uses his private input r and creates Diffie-Hellman tuples (x, y, z_0, z_1) and sends them to the sender so the sender is unable to

Common Inputs: Security parameter λ, a multiplicative group, $\mathcal{Z}_p^*$ of prime order p with generator g, and key distribution function (KDF) defined in [2].

(1) The receiver randomly chooses an element $a \in \mathcal{Z}_p^*$, besides he chooses an input bit $r \in \{0, 1\}$. After that, he sets $\beta_r = g^a$, and chooses β_{1-r} randomly from $\mathcal{Z}_p^*$. The receiver then sends the values of β_0, β_1 to the sender.
(2) The sender randomly generates an element $\gamma \in \mathcal{Z}_p^*$ and computes the keys as follows: $k_0 = \beta_0^\gamma, k_1 = \beta_1^\gamma$. After that, the sender uses the keys k_0 and k_1, and encrypts the messages x_0 and x_1 respectively as follows: $e_0 = x_0 \oplus KDF(k_0)$ and $e_1 = x_1 \oplus KDF(k_1)$. The sender then sends the ciphertexts (e_0, e_1) along with $u = g^\gamma$ to the receiver.
(3) R uses the values of (r, a) and extracts the required message as follows, $x_r = e_r \oplus KDF(k_r)$, where $k_r = (u)^a$.

Fig. 6. DDH assumption based 1-out-of-2 OT protocol for semi-honest receiver [2].

Common Inputs: Security parameter λ, a multiplicative group, $\mathcal{Z}_p^*$ of prime order p with generator g_0.

(1) The receiver randomly chooses two elements $a_0, y \in \mathcal{Z}_p^*$, besides he chooses an input bit $r \in \{0, 1\}$. After that, he sets $a_1 = a_0 + 1$ and computes $g_1 = (g_0)^y$, $h_0 = (g_0)^{a_0}$, and $h_1 = (g_1)^{a_1}$. The receiver then sends the values of (g_1, h_0, h_1) to S. Moreover, the receiver randomly chooses γ and computes $g = (g_r)^\gamma$, $h = (h_r)^\gamma$ and sends (g, h) to S.
(2) S performs the following operations:
- S defines a function $\mathcal{R}(u, v, w, x) = (y, z)$, where $y = u^s.w^t$ and $z = v^s.x^t$, and the values of $s, t \in \mathcal{Z}_p^*$.
- S evaluates $(y_0, z_0) = \mathcal{R}(g_0, g, h_0, h)$ and $(y_1, z_1) = \mathcal{R}(g_1, g, h_1, h)$.
- S encrypts the value of x_0 by computing $e_0 = x_0.z_0$ and encrypts the value of x_1 by computing $e_1 = x_1.z_1$. The sender then sends the values of (y_0, e_0) and (y_1, e_1) to R.

(3) R uses the values of (r, γ) and extracts the required message as follows, $x_r = e_r/(y_r)^\gamma$.

Fig. 7. DDH assumption based 1-out-of-2 OT protocol for semi-honest receiver [77].

guess the private input bit r. In item 2, the sender first tests whether $z_0 \neq z_1$. If it is false, then the sender aborts the protocol. Otherwise, the sender encrypts the values of x_0 and $_x1$ using k_0 and k_1, respectively, and sends the values of (e_0, ω_0) and (e_1, ω_1) to the receiver. In the last item, the receiver uses his private inputs (r, b) and decrypts the required message x_r.

The OT protocol described in Figure 5 works over the standard model that requires high computation costs. In Reference [2], they have proposed a new OT protocol under the DDH assumption and reduced the computation costs three times than the previous OT protocol [69]. Description of this protocol is given in Figure 6. Input, common input, and output of this protocol are similar to the previous protocol, but this OT protocol requires one extra **key distribution function (KDF)**. In item 1, the receiver randomly generates one element a from the multiplicative group $\mathcal{Z}_p^*$ and uses his private input r and creates Diffie-Hellman tuples (β_0, β_1) and sends them to the sender so the sender is unable to guess the private input bit r. In item 2, the sender randomly generates an element $\gamma \in \mathcal{Z}_p^*$ and uses the values of (β_0, β_1) and computes the keys (k_0, k_1). After that, the sender encrypts the value of x_0 using $KDF(k_0)$ while encrypting the value of x_1 using $KDF(k_1)$ and sends the encrypted values (e_0, e_1) along with $u = g^\gamma$ to the receiver. The receiver uses his secret inputs (a, r) and computes the required message x_r.

One more 1-out-of-2 OT protocol has been proposed whose security depends on the DDH assumption [77]. The description of this OT protocol is given in Figure 7. The input and output of this OT protocol are similar to the without random oracle model-based OT protocol (Figure 5). Moreover, common inputs are also similar to that protocol except that in this protocol, we have denoted the generator as g_0, while in that protocol, indicated as g. In item 1, the receiver randomly

Common Inputs: Security parameter λ, a multiplicative group, $\mathcal{Z}_p^*$ of prime order p with generator g.

(1) The receiver randomly chooses three elements $a_0, a_1, y \in \mathcal{Z}_p^*$, besides he chooses an input bit $r \in \{0,1\}$. After that, he computes $h_0 = (g)^{a_0}$, $h_1 = (g)^{a_1}$, and $a = g^y$. Moreover, he also computes $b_0 = (h_0)^y.g^r$ and $b_1 = (h_1)^y.g^r$. The receiver then sends the values of (h_0, h_1, a, b_0, b_1) to S.
(2) S verifies whether the tuples $(h_0, h_1, a, b_0, b_1) \in \mathcal{Z}_p^*$. If not, then he aborts the protocol.
(3) R computes $h = h_0/h_1$ and $b = b_0/b_1$, and he claims the sender that tuples (g, h, a, b) are Diffie-Hellman tuples by using zero-knowledge proof.
(4) S randomly chooses four elements m_0, n_0, m_1, n_1 from the multiplicative group $\mathcal{Z}_p^*$, and computes the following: $\omega_0 = a^{m_0}.g^{n_0}$, $k_0 = b_0^{m_0}.h_0^{n_0}$ and $\omega_1 = a^{m_1}.g^{n_1}$, and $k_1 = (\frac{b_1}{g})_0^{m_1}.h_1^{n_1}$. After that, he encrypts x_0 by computing $e_0 = x_0.k_0$ while encrypts x_1 by computing $e_1 = x_1.k_1$, and sends (e_0, ω_0), (e_1, ω_1) to R
(5) R uses the values of (r, a_r) and extracts the required message as follows, $x_r = e_r/(\omega_r)^{a_r}$.

Fig. 8. DDH assumption based 1-out-of-2 OT protocol for malicious receiver [45].

generates two numbers a_0, y from the multiplicative group $\mathcal{Z}_p^*$ and uses his input bit r and creates the following Diffie-Hellman tuples (g_1, h_0, h_1, g, h) so the sender cannot guess the input bit r of the receiver. The receiver sends the following Diffie-Hellman tuples (g_1, h_0, h_1, g, h) to the sender. In item 2, the sender defines a function $\mathcal{R}$ that takes four inputs in the form of the multiplicative group $\mathcal{Z}_p^*$ and produces two outputs in the form of the multiplicative group $\mathcal{Z}_p^*$. During the execution of this function, the sender needs to generate two random elements from the multiplicative group $\mathcal{Z}_p^*$. In the next item of item 2, the sender evaluates $(y_0, z_0) = \mathcal{R}(g_0, g, h_0, h)$ and $(y_1, z_1) = \mathcal{R}(g_1, g, h_1, h)$, where $y_0 = (g_0)^{m_0}.(h_0)^{n_0}$, $z_0 = g^{m_0}.h^{n_0}$, and $y_1 = (g_1)^{m_1}.(h_1)^{n_1}$, $z_1 = g^{m_1}.h^{n_1}$. The values of m_0, n_0, m_1, n_1 are randomly chosen by the sender from $\mathcal{Z}_p^*$. In the last item of item 2, the sender encrypts the value of x_0 by using z_0 while he encrypts the value of x_1 by using z_1 and sends the encrypted values (e_0, e_1) along with (y_0, y_1) to the receiver. The receiver uses his secret values (r, γ) and extracts the required message as follows, $x_r = e_r/(y_r)^\gamma$.

All the one-out-of-2 OT protocols from Figures 1 to 7 only provide security in the presence of a semi-honest receiver. In other words, these OT protocols fail to provide security in the presence of a malicious receiver. A new DDH assumption based 1-out-of-2 OT protocol has been proposed that provides security in the presence of a malicious receiver [45]. They have invoked the ***zero-knowledge proof* (ZKP)** protocol [34] in their OT protocol to provide security against the malicious receiver. The ZKP protocol forces the receiver to follow the OT protocol according to the prescribed rules. If the receiver executes the OT protocol without the ZKP protocol, then the receiver may extract more information than the prescribed output. The description of one-out-of-2 OT protocol, which is secure in the presence of a malicious receiver, is given in Figure 8. The input of the sender and receiver, the receiver's output, and the common inputs of this protocol are similar to the without random oracle-based OT protocol (Figure 5). In item 1 of this protocol, the receiver randomly generates three elements a_0, a_1, y from the multiplicative group $\mathcal{Z}_p^*$ and uses his input bit r and hides the input bit r by computing the following tuples: (h_0, h_1, a, b_0, b_1). After that, the receiver sends these tuples (h_0, h_1, a, b_0, b_1) to the sender. In item 2, the sender verifies whether these tuples (h_0, h_1, a, b_0, b_1) belong to the multiplicative group $\mathcal{Z}_p^*$. If not, then he aborts the protocol. In item 3, the receiver creates the following Diffie-Hellman tuples (g, h, a, b). It executes the ZKP protocol on these tuples (g, h, a, b), and it claims to the sender that it is correctly executing the rule of the OT protocol. In item 4, the sender generates four elements m_0, n_0, m_1, n_1 from the multiplicative group $\mathcal{Z}_p^*$ and computes (ω_0, ω_1) and (k_0, k_1). After that, the sender encrypts the value of x_0 using key k_0 while encrypts the value of x_1 using key k_1 and sends the ciphertexts

Common Inputs: Security parameter λ, a multiplicative group, $\mathcal{Z}_p^*$ of prime order p with generator g_0.

(1) The receiver randomly chooses two elements $a_0, y \in \mathcal{Z}_p^*$, besides he chooses an input bit $r \in \{0,1\}$. After that, he sets $a_1 = a_0 + 1$ and computes $g_1 = (g_0)^y$, $h_0 = (g_0)^{a_0}$, and $h_1 = (g_1)^{a_1}$. The receiver then sends the values of (g_1, h_0, h_1) to S. Moreover, The receiver randomly chooses an element γ and computes $g = (g_r)^\gamma$, $h = (h_r)^\gamma$ and sends (g, h) to S.

(2) Both sender and receiver execute the zero-knowledge proof (ZKP) protocol, where the receiver acts as a prover, and the sender acts as a verifier. In the ZKP protocol, the prover sends the Diffie-Hellman tuples $(g_0, g_1, h_0, h_1/g_1)$ to the verifier, and the verifier verifies whether the tuples are Diffie-Hellman tuples.

(3) S performs the following operations:
- S defines a function $\mathcal{R}(u, v, w, x) = (y, z)$, where $y = u^s.w^t$ and $z = v^s.x^t$, and the values of $s, t \in \mathcal{Z}_p^*$.
- S evaluates $(y_0, z_0) = \mathcal{R}(g_0, g, h_0, h)$ and $(y_1, z_1) = \mathcal{R}(g_1, g, h_1, h)$.
- S encrypts the value of x_0 by computing $e_0 = x_0.z_0$ and encrypts the value of x_1 by computing $e_1 = x_1.z_1$. The sender then sends the values of (y_0, e_0) and (y_1, e_1) to R.

(4) R uses the values of (r, γ) and extracts the required message as follows, $x_r = e_r/(y_r)^\gamma$.

Fig. 9. DDH assumption based 1-out-of-2 OT protocol for malicious receiver [62].

(e_0, e_1) along with (ω_0, ω_1) to the receiver. In the last item, the receiver uses his private values (r, a_r) and extracts the required message as follows: $x_r = e_r/(\omega_r)^{a_r}$.

The one-out-of-2 OT protocol described in Figure 8 provides security in the presence of a malicious receiver, but it requires high computation costs. Another one-out-of-2 OT protocol has been proposed to minimize computation costs [62]. This OT protocol security relies on the DDH assumption. This protocol invokes the ZKP protocol [36–38] in the existing OT protocol [77] and provides security against the malicious receiver. The description of this protocol is given in Figure 9. The input of the sender and receiver, the receiver's output, and the common inputs of this protocol are similar to the OT protocol described in the Figure 7. Item 1 of this OT protocol is similar to item 1 of the OT protocol of Figure 7. In item 2, both parties execute the ZKP protocol, where the receiver sends proof to the sender and claims that it correctly follows the OT protocol rules. However, the sender uses that proof to verify whether the receiver correctly follows the OT protocol. The sender needs to verify, because if the receiver does not correctly follow the OT protocol, then the receiver may extract more information than required from the sender. The receiver creates Diffie-Hellman tuples $(g_0, g_1, h_0, h_1/g_1)$ as proof and sends it to the sender. Items 3 and 4 of this OT protocol are, respectively, similar to items 2 and 3 of the OT protocol of Figure 7.

The security of these OT protocols (Figures 1 to 9) depends on the following assumptions: DDH, ROM, and number-theoretic public key encryption. These security assumptions are vulnerable to quantum attacks. To overcome this quantum-resistant OT protocols have been proposed. The first quantum-resistant OT protocol has been proposed using lattice-based public-key encryption technique [77]. This protocol provides universally composable security. The description of universally composable lattice-based OT protocol is given in Figure 10. Before the beginning of this protocol, both parties choose their inputs. For example, the sender chooses two input messages (x_0, x_1), session-id ssid, and its own id sid. However, the receiver chooses input bit $r \in \{0,1\}$, session-id ssid, and its own id rid. After that, both parties agree on common input, CRS, and lattice-based public-key encryption, decryption, and key generation algorithms. In item 1, both parties exchange their ids. In item 2, the receiver generates the public-private key pair (pk, sk) corresponding to the input bit r, sends the public key pk to the sender, and keeps the secret key sk private. The sender uses the receiver's public key pk, encrypts the messages (x_0, x_1), and sends them to the receiver. The receiver uses the secret key sk_r corresponding to the input bit r and extracts the desired message

Input of S**:** It has two messages $x_0 \in \{0,1\}^l$ and $x_1 \in \{0,1\}^l$, session id: ssid, sender id: sid.
Input of R**:** It has one bit $r \in \{0,1\}$, session id: ssid, receiver: rid.
Common Inputs: Both parties agree on common reference string crs, Dual-Mode Cryptosystem (Messy or Decryption): $mode$, $KeyGen(crs,\ r)$, $Enc(.)$, $Dec(.)$ function as described in [77].

(1) The sender S sends the query $\mathcal{F}_{CRS}^{mode}$ along with (ssid, sid) to the receiver and gets $(sid,\ crs)$ from the receiver. Similarly, receiver R sends the query $\mathcal{F}_{CRS}^{mode}$ along with (ssid, rid) to the sender and gets $(rid,\ crs)$ from the sender.
(2) The receiver uses his input r, generates the public private key pair using $KeyGen(crs,\ r)$ algorithm as follows, $(pk, sk) \leftarrow KeyGen(crs,\ r)$ and sends the values (sid,ssid, pk) to the sender and keeps the value (sid,ssid, sk) secret.
(3) The sender uses the received public key pk, encrypts both the messages as follows $y_b = Enc(pk, b, x_b)$, for all $b \in \{0,1\}$ and sends the encrypted values (y_0, y_1) to the receiver.
(4) The receiver uses his secret key sk and extracts the required message x_r as follows, $x_r = Dec(sk, y_r)$.

Fig. 10. Lattice-based universally composability 1-out-of-2 OT protocol [77].

Common Inputs: q and m are selected in the form of $q \equiv 1\ mod\ 2m$. Both parties agree on a uniformly generated polynomial $a \in \mathcal{U}(\mathcal{R}_q)$, Sampling algorithm: $Sample(\chi)$.

(1) The sender generates two vectors, $s, e \leftarrow Sample(\chi)^2$, computes $A \leftarrow a.s + e \in \mathcal{R}_q$, $T \leftarrow \mathcal{U}(\mathcal{R}_q)$, and sends both the polynomials T, A to the receiver.
(2) The receiver also generates three vectors, $s^{'}, e^{'}, e^{''} \leftarrow Sample(\chi)^3$, computes $b^{'} \leftarrow a.s^{'} + e^{'} \in \mathcal{R}_q$, $v \leftarrow A.s^{'} + e^{''} \in \mathcal{R}_q$, $\bar{v} = RR(v)\mathcal{R}_{2q}$, $c =< \bar{v} >_2$. If $r = 0$, then sets $B = b^{'}$. If $r = 1$, then sets $B = b^{'} + T$. After that, he sends (B, c) to the sender.
(3) The sender uses its secret value s, derives the shared secret keys as follows, $k_0 = reconcilation(2Bs, c)$, $k_1 = reconcilation(2(B - T)s, c)$, and encrypts both the messages $e_0 = x_0 \oplus k_0$, $e_1 = x_1 \oplus k_1$, and sends the encrypted values e_0, e_1 to the receiver.
(4) The receiver derives the shared secret key $k_r = \lfloor \bar{v} \rceil_2$, and extracts the required message $x_r = e_r \oplus k_r$.

Fig. 11. Universally composable 1-out-of-2 OT from ideal lattice [63].

x_r. This public-key encryption-based OT protocol requires high computation costs. In [61, 64, 65], they have described the security proof of this universally composable lattice-based OT protocol.

To reduce the computation costs, another ideal lattice-based universally composable OT protocol has been proposed [63]. This OT protocol uses the ring learning with errors (ring-LWE) key exchange mechanism and provides input privacy of both parties. This protocol also provides universally composable security. The description of universally composable ideal lattice-based OT protocol is given in Figure 11. Before the beginning of this protocol, both parties choose their inputs. For example, the sender chooses two input messages (x_0, x_1). However, the receiver chooses input bit $r \in \{0,1\}$. After that, both parties agree on common inputs, a prime number q, uniformly generated polynomial $a \in \mathcal{U}(\mathcal{R}_q)$, and sampling algorithm $Sample(\chi)$. This algorithm takes input as a standard deviation χ and generates output as a secret or error vector (s, e). In Reference [89], they have described the Gaussian sampling algorithm. In item 1, the sender generates two polynomials A, T. The first polynomial A is generated using a sampling algorithm, whereas the second polynomial T is generated using a uniform distribution. The sender sends both polynomials to the receiver. The receiver generates three vectors $(s^{'}, e^{'}, e^{'})$, uses the received polynomials, derives the shared secret key k_r, and creates a hint polynomial (B, c). It uses the **randomized rounding (RR)** function to remove the noise from the polynomial v. To derive the shared secret key from the polynomial $\bar{v}$, the receiver uses the modular rounding function $\lfloor \bar{v} \rceil_2$, while it derives the hint polynomial c, using cross rounding function $c =< \bar{v} >_2$. In item 3, the sender uses the

Common Inputs: q and m are selected in the form of $q \equiv 1 \ mod \ 2m$. Both parties agree on a pair of uniformly generated polynomials $(a^0, a^1) \in \mathcal{R}_q$.

(1) The receiver chooses his input $r \in \{0, 1\}$, computes a polynomial $b^r = a^r s^r + s^{1-r} \in \mathcal{R}_q$ and sends it to the sender, where $s^r, s^{1-r} \in Sample(\chi)^2$.

(2) The sender performs operation on received polynomial b^r and encrypts both the messages x_0, x_1 are as follows, $e^0, e^1, e^2 \in Sample(\chi)^3$, $u^0 = e^0 a^0 + e^1 \in \mathcal{R}_q$, $v^0 = e^0 b^r + e^2 \in \mathcal{R}_q$, $vv^0 = RR(v^0)$, $\mu^0 = \lfloor vv^0 \rceil_2$, $kk^0 = \mu^0 \oplus x_0$, $c^0 = (u^0, < vv^0 >_2)$, $u^1 = e^1 a^1 + e^2 \in \mathcal{R}_q$, $v^1 = e^1 b^r + e^0 \in \mathcal{R}_q$, $vv^1 = RR(v^1)$, $\mu^1 = \lfloor vv^1 \rceil_2$, $kk^1 = \mu^1 \oplus x_1$, $c^1 = (u^1, < vv^1 >_2)$, and sends the encrypted values (c^0, kk^0), (c^1, kk^1) to the receiver.

(3) The receiver uses his required input bit r and extracts the required message as follows $\omega^r = u^r s^r$, $\mu^r = reconciliation(\omega^r, < vv^r >_2)$, $x_r = \mu^r \oplus kk^r$.

Fig. 12. Universally composable ring-LWE based 1-out-of-2 OT protocol [88].

reconciliation function, derives the shared secret keys (k_0, k_1), encrypts the messages (x_0, x_1), and sends the encrypted messages (e_0, e_1) to the receiver. The receiver uses the input bit r, derived shared secret key k_r, and extracts the desired message x_r. This universally composable ideal lattice-based OT protocol requires low computation costs, but it requires a high communication overhead.

To reduce the communication overhead, another ring-LWE based universally composable OT protocol has been proposed [88]. This OT protocol also uses the **ring learning with errors (ring-LWE)** key exchange mechanism and provides input privacy of both parties. This protocol also provides universally composable security. The description of universally composable ring-LWE based OT protocol is given in Figure 12. Before the beginning of this protocol, both parties choose their inputs. For example, the sender chooses two input messages (x_0, x_1). However, the receiver chooses input bit $r \in \{0, 1\}$. After that, both parties agree on common inputs, a prime number q, two uniformly generated polynomials (a^0, a^1), and sampling algorithm $Sample(\chi)$. This algorithm takes input as a standard deviation χ and generates output as a secret or error vector. In item 1, the receiver generates a polynomial b^r using secret vectors (s^r, s^{1-r}) and sends it to the sender. The sender generates three vectors (e^0, e^1, e^2), uses the received polynomial b^r, and derives the shared secret key μ^0, μ^1, and creates hints polynomial c^0, c^1. It uses the **randomized rounding (RR)** function to remove the noise from the polynomials (v^0, v^1). To derive the shared secret keys (μ^0, μ^1) from the polynomial (vv^0, vv^1), the receiver uses the modular rounding function, while it derives the hint polynomial $(c^0 c^1)$, using cross rounding function. After that, the sender encrypts both the messages (x_0, x_1) and sends the encrypted messages (kk^0, kk^1) along with hint polynomial (c^0, c^1) to the receiver. In item 3, the receiver uses his input bit r, the reconciliation function, and derives the shared secret key kk_r. The receiver uses the derived shared secret key and extracts the desired message x_r. This universally composable ring-LWE-based OT protocol requires low computation and communication costs than the other existing universally composable lattice-based OT protocols.

The problem with the existing universally composable lattice-based OT protocols is that they required high communication overhead. To reduce the communication overhead, a number-theoretic public-key encryption-based universally composable OT protocol has been proposed [1]. The description of public-key encryption-based universally composable OT protocol is given in Figure 13. Both sender and receiver choose their inputs similar to Figure 11. Both parties agree on public-key encryption Enc, decryption Dec, and key generation algorithms $KeyGen$. Besides, they agree on two collision-resistant hash functions H_1 and H_2. In item 1, the receiver randomly generates a seed $t \in \mathcal{Z}$, and computes T using t, ssid, and sid. It uses the input bit $r \in \{0, 1\}$, executes the key generation algorithm, and generates the public-private key pair (pk_r, sk_r) corresponding to input bit $r \in \{0, 1\}$. After that, it sends the public key pk_0 along with random seed t to the sender and

Common Inputs: Both parties agree on a public key encryption: Enc, decryption: Dec, and key generation: $KeyGen$ algorithms. Two collision resistant hash functions, $H_1 : \{0,1\}\times\{0,1\}\times\{0,1\} \rightarrow \mathcal{Z}_q^*$ and $H_2 : \{0,1\} \times \{0,1\} \rightarrow \{0,1\}$.

(1) The receiver generates two public keys, but it knows only one secret key of the generated public keys. It generates a random seed t, sets $T = H_1(t, sid, ssid)$. After that, it chooses its input bit r, computes a pair of public private key $pk_r, sk_r = KeyGen(1^k)$ and sets $pk_{1-r} = T \oplus pk_r$, and sends the value (t, pk_0) to the sender and keeps the value (sk_r, r) secret.
(2) It encrypts both messages m_0, m_1 as follows, $pk_1 = pk_0 \oplus T$, for all $b \in (0,1)$, it picks a random key $k_b = \{0,1\}^\ell$, computes $k_b^{'} = H_2(k_b, sid)$, $C_b = Enc(pk_b, k_b, r_b)$, $v_b = x_b \oplus k_b^{'}$, sends (C_0, C_1, v_0, v_1) to the receiver and erases all the randomness.
(3) The receiver chooses his input bit r, and extracts the required message as follows, $k_r = Dec(sk_r, C_r)$, $x_r = v_r \oplus H_2(k_r, sid)$.

Fig. 13. Universally composable 1-out-of-2 OT protocol [1].

Common Inputs: Both parties agree on a multiplicative group element $X = g^x \in \mathcal{Z}_q$, where g is the generator of group $\mathcal{Z}_q$, besides x is unknown to both parties.

(1) The receiver randomly generates an element $m \leftarrow_R \mathcal{Z}_q$, computes $h_0 = g^m.X^{-r}$, and sends the value of h_0 to the sender.
(2) The sender performs the following operations, $h_1 = h_0 X$, $s \leftarrow_R \mathcal{Z}_q$, $S = g^s$, and encrypts both the messages as follows, $e_0 = h_0^s \oplus x_0$, $e_1 = h_1^s \oplus x_1$ and sends both the encrypted messages to the receiver.
(3) The receiver uses his input bit r, and extracts the required message as follows, $x_r = e_r \oplus S^m$.

Fig. 14. CDH assumption-based universally composable 1-out-of-2 OT protocol [30].

keeps the private key sk_r secret. In item 2, the sender encrypts both messages (x_0, x_1) using the received public key and sends the encrypted messages (v_0, v_1) to the receiver. Finally, in the last item, the receiver uses his input bit r and extracts the desired message x_r. This number-theoretic public-key encryption-based universally composable OT protocol requires high computation and communication costs.

To minimize the communication and computation overhead, a CDH assumption-based universally composable OT protocol has been proposed [30]. The description of this protocol is given in Figure 14. The input and output of this protocol are similar to the Figure 13 OT protocol. Before the beginning of this protocol, both parties agree on a multiplicative group element $X \in \mathcal{Z}$. In item 1, the receiver hides his input bit $r \in \{0,1\}$ in the CDH tuple h_0 and sends this tuple to the sender. In item 2, the sender creates two CDH tuples h_1, S, encrypts both messages (x_0, x_1) using these CDH tuples, and sends the encrypted messages (e_0, e_1) to the receiver. Finally, in item 3, the receiver uses his input bit r and extracts the required message x_r.

In Reference [59], they have proposed efficient and secure ROM-based universally composable OT protocol. The description of this protocol is given in Figure 15. The input and output of this protocol are similar to Figure 14. In this protocol also both parties agree on the following common input such as multiplicative group element $X \in \mathcal{Z}$ and a collision-resistant hash function H. In item 1, the receiver randomly generates an element m from the multiplicative group $\mathcal{Z}_q$, applies cryptographic operation on it with respect to input bit $r \in \{0,1\}$, and hides the input bit r. After that, it sends the hidden value C to the sender. The sender applies the hash function H on the hidden value C, creates a shared secret key, encrypts the messages (x_0, x_1) using this key, and sends them to the receiver. The receiver uses his input bit r and extracts the required message x_r.

2.2 1-out-of-2 OT Protocols

The first 1-out-of-2 OT protocol has been proposed using the **public-key encryption (PKE)** technique [33]. It provides security in the presence of a semi-honest receiver. It is more costly in terms

Common Inputs: Both parties agree on a multiplicative group element $X \in \mathcal{Z}_q$, and collision resistant hash function, $H : \{0,1\}^* \rightarrow \{0,1\}^l$.

(1) The receiver randomly generates an element $m \leftarrow_R \mathcal{Z}_q$, computes $C = m.X$, if $r = 0$, and $C = (m.X)^t$, if $r = 1$, sends the value of C to the sender.
(2) The sender performs the following operations, $s \leftarrow_R \mathcal{Z}_q$, $S = sX$, and encrypts both the messages as follows, $e_0 = H(s * C) \oplus x_0$, $e_1 = H(s * C^{-t}) \oplus x_1$ and sends both the encrypted messages along with S to the receiver.
(3) The receiver uses his input bit r, and extracts the required message as follows, $x_r = e_r \oplus H(m * S)$.

Fig. 15. Universally composable secure 1-out-of-2 isogeny-based OT protocol [59].

of communication and computation costs. To minimize these costs, a non-interactive discrete log-based 1-out-of-2 OT protocol has been proposed [8]. It also provides security in the presence of semi-honest adversaries. This protocol still required high communication and computation costs but less than that of PKE-based protocol. Two ROM-based 1-out-of-2 OT protocols have been proposed [69, 69], which required only a few numbers of cryptographic operations. In other words, these ROM-based OT protocols provide security in the presence of a semi-honest receiver at low communication and computation costs. Without ROM-based 1-out-of-2 OT protocols (also called DDH assumption-based OT protocols) have also been proposed for the semi-honest receiver, but they required high communication and computation costs [69]. Among all the PKE, NI, ROM, and DDH-based OT protocols, KDF based 1-out-of-2 OT protocol was the most efficient OT protocol [2]. This KDF-based OT protocol also provides security in the presence of a semi-honest receiver. Apart from efficiency, these PKE, NI, ROM, and DDH-based OT protocols provide security only in the presence of a semi-honest receiver, but they failed to provide security in the presence of a malicious receiver. To overcome this, a DDH assumption-based 1-out-of-2 OT protocol has been proposed [45, 77]. It used the zero-knowledge proof protocol to provide security in the presence of a malicious receiver; however, it required high communication and computation costs. To minimize these costs, another DDH assumption-based 1-out-of-2 OT protocol has been proposed [62]. This protocol efficiently provides security in the presence of a malicious receiver. All these number-theoretic based 1-out-of-2 OT protocols either provide security in the presence of a semi-honest or malicious receiver. However, they are vulnerable to a quantum computer attack; besides, they do not provide security in universally composability. The first lattice-based public-key encryption-based 1-out-of-2 OT protocol has been proposed to provide security in the presence of a quantum attack [77]. This protocol provided universal composability security, but it required high communication and computation overhead. An ideal lattice-based 1-out-of-2 OT protocol has been proposed that required low computation costs but required high communication costs [63]. To minimize communication overhead, a ring-LWE based 1-out-of-2 OT protocol has been proposed, but it still required high communication overhead [88]. It has also provided quantum-resistant and universal composability security. A combination of ROM and PKE-based universal composability 1-out-of-2 OT protocol has been proposed to mitigate the communication overhead [1]. This protocol has required low communication overhead than all the lattice-based OT protocols, but it required high computation costs. Moreover, it is vulnerable to quantum attacks. To minimize the computation costs, a CDH assumption-based 1-out-of-2 OT protocol has been proposed [30]. This protocol is further enhanced using the Isogeny technique to mitigate the computation costs [59]. Table 1 contains a summary of 1-out-of-2 OT protocols concerning the following criteria: used technique, semi-honest adversary, malicious adversary, and universal composability security. This table shows that only the OT protocol of References [45] and [62] provide security against the malicious receiver. However, except for the following OT protocols [1, 30, 59],

Table 1. Summary of 1-out-of-2 OT Protocols

	[33]	[8]	[69] (3)	[69] (4)	[69] (5)	[2]	[77]	[45]	[62]	[77]	[63]	[88]	[1]	[30]	[59]
Semi-honest Adversaries	YES	YES	YES	YES	YES	YES	YES	YES	YES	NO	NO	NO	NO	NO	NO
Malicious Adversaries	NO	NO	NO	NO	NO	NO	NO	YES	YES	NO	NO	NO	NO	NO	NO
Universally Composable	NO	NO	NO	NO	NO	NO	NO	NO	NO	YES	YES	YES	YES	YES	YES
Used Technique	PKE	NI	ROM	ROM	DDH	KDF	DDH	DDH	DDH	Ring-LWE	ideal lattice	Ring-LWE	ROM and PKE	CDH	Isogeny

Input of S**:** It has n messages $x_i \in \{0,1\}^l$, where $1 \leq i \leq n$.
Input of R**:** It has m bits number $r = (r_1, r_2, ..., r_m) \in \{0,1\}^m$.
Common Inputs: Pseudorandom function (PRF) $\mathcal{F}_k : \{0,1\}^l \rightarrow \{0,1\}^l | k \in \{0,1\}^t$, one-out-of-2 OT protocol, $n = 2^m$.

(1) The sender S randomly generates m pairs of stings $\{(k_0^i, k_1^i)\}_{i=1}^m$, where size of each k_b^i is t bits, and $b \in \{0,1\}$. For all $1 \leq i \leq m$, let i can be expressed as in the from of m bits, in other words, $i = \{i_1, i_2, ..., i_m\}$. However, the receiver chooses m bits number $r = (r_1, r_2, ..., r_m) \in \{0,1\}^m$.
(2) Both sender and receiver respectively execute the one-out-of-2 OT protocol on their respective inputs $\{(k_0^i, k_1^i)\}_{i=1}^m$ and $r = (r_1, r_2, ..., r_m) \in \{0,1\}^m$. Where the sender acts as a receiver, and the receiver acts as a sender. At the end of m executions of one-out-of-2 OT protocol, the receiver receives $\{k_{r_1}^1, k_{r_2}^2, ..., k_{r_m}^m\}$ while the sender gets nothing.
(3) The sender computes $y_i = x_i \oplus \bigoplus_{j=1}^m \mathcal{F}_{k_{i_j}^j}(i)$, for all $1 \leq i \leq n$, and sends the values of $\{y_1, y_2, ..., y_n\}$ to the receiver.
(4) The receiver uses his private input, $r = (r_1, r_2, ..., r_m)$, and extracts the required messages as follows: $x_r = y_r \oplus \bigoplus_{j=1}^m \mathcal{F}_{k_{r_j}^j}(r)$.

Fig. 16. Pseudorandom function based 1-out-of-n OT protocol for semi-honest receiver [68].

the rest of the number-theoretic OT protocols provide only security in the presence of the semi-honest adversaries. The following OT protocols [1, 30, 59] provide universal composability security. In addition, all the lattice-based OT protocols provide universal composability security. The number-theoretic-based OT protocols are vulnerable to quantum attacks and only lattice-based OT protocol provides security in the presence of quantum computer attacks. Moreover, it was found that the OT protocols that provided security in the presence of malicious receivers required high communication and computation costs than the semi-honest receiver.

2.3 1-out-of-n and k-out-of-n OT Protocol

Moni Naor and Benny Pinkas have introduced the one-out-of-n OT protocol [68]. In this protocol, they have applied the pseudorandom function and the one-out-of-2 OT protocol. Moreover, this protocol only provides security against the semi-honest receiver. The Description of this OT protocol is given in Figure 16. In this protocol, the sender has n input messages $\{x_1, x_2, \ldots, x_n\}$ while the receiver has a number $r \in [1, n]$. Moreover, both parties agree on the following common inputs: **pseudorandom function (PRF)**, one-out-of-2 OT protocol, $n = 2^m$. This PRF takes two inputs, t bit key, and l bit string, and it produces l bit output. In item 1 of this protocol, the sender randomly chooses m pairs of t bit strings, while the receiver selects a number $r \in [1, n]$. In item 2, the sender and receiver execute the one-out-of-2 OT protocol m times. During the first execution of the one-out-of-2 OT protocol, the sender uses a pair of inputs (k_0^1, k_1^1) while the receiver uses a bit $r_1 \in \{0,1\}$. At the end of the first execution of the one-out-of-2 OT protocol, the receiver gets $k_{r_1}^1$, while the sender receives nothing. Similarly, during the second execution of the one-out-of-2 OT protocol, the sender uses a pair of inputs (k_0^2, k_1^2) while the receiver uses a bit $r_2 \in \{0,1\}$. At the end of the second execution of the one-out-of-2 OT protocol, the receiver gets $k_{r_2}^2$, while the sender receives nothing. In item 3, the sender encrypts the values of x_i using the PRF and sends them to the receiver. In the last item, the receiver uses his secret value r and decrypts the required

Input of S**:** It has n messages $x_i \in \{0,1\}^l$, for all $0 \le i \le n-1$.
Input of R**:** It has an input $r \in \{0,1,...n-1\}$.
Common Inputs: Security parameter λ, a multiplicative group, $\mathcal{Z}_p^*$ of prime order p with generator g, ROM $H : \{0,1\}^* \rightarrow \{0,1\}^l$.

(1) The sender randomly chooses $n-1$ constants $\{C_i\}_{i=1}^{n-1}$ from the multiplicative group $\mathcal{Z}_p^*$. Moreover, he also randomly chooses an element $\gamma \in \mathcal{Z}_p^*$ and computes g^γ. After that, the sender sends the values of $\{C_1, C_2, ..., C_{n-1}\}$ along with g^γ to the receiver.
(2) The receiver randomly chooses a number $s \in \{0,1,...p-1\}$, besides he chooses an input $r \in \{0,1,..n-1\}$. After that, he sets $\beta_r = g^s$, if $s \neq 0$, then he computes $\beta_0 = C_r/\beta_r$. The receiver sends the value of β_0 to the sender and keeps the values of (r,s) private.
(3) The sender computes $\beta_i^\gamma = C_i^\gamma/\beta_0^\gamma$, for all $0 \le i \le n-1$. The sender, then randomly generates a number X and encrypts x_i by $e_i = H(\beta_i^\gamma, X, i) \oplus x_i$ for all $0 \le i \le n-1$, and sends them to the receiver.
(4) R uses the values of (r,s) and extracts the required message as follows: $x_r = H((g^\gamma)^s, X, r) \oplus e_r$.

Fig. 17. ROM based 1-out-of-n OT protocol for semi-honest receiver [69].

Input of S**:** It has n messages $x_i \in \mathcal{G}_q$, for all $1 \le i \le n$, where $\mathcal{G}_q$ is the multiplicative group of prime order q.
Input of R**:** It has an input $r \in \{1,...n\}$.
Common Inputs: Let g, h be the two generators in $\mathcal{G}_q$ of prime order q.

(1) The receiver randomly chooses a number $s \in \mathcal{Z}_q$, besides he chooses an input $r \in \{1,..n\}$. After that, he computes $\beta = g^s h^r$ and sends the value of β to the sender and keeps the values of (r,s) private.
(2) The sender randomly chooses $k_i \in \mathcal{Z}_q^*$, and computes $(a_i, b_i) = (g^{k_i}, x_i(\beta/h^i)^{k_i})$, for all $1 \le i \le n$. The sender then sends the values of $(a_1,b_1), (a_2,b_2), ..., (a_n,b_n)$ to the receiver.
(3) R uses the values of (r,s) and extracts the required message as follows: $x_r = b_r/(a_r)^s$.

Fig. 18. DDH based 1-out-of-n OT protocol for semi-honest receiver [86].

message x_r. In this protocol, they execute one-out-of-2 OT protocol $\log_2 n$ times and execute PRF $n \log_2 n$ times.

The 1-out-of-n OT protocol, described in Figure 16, requires high computation costs. A ROM-based 1-out-of-n OT protocol has been proposed in Reference [69] to improve the computational efficiency of the OT protocol. The description of ROM based 1-out-of-n OT protocol is given in Figure 17. In this protocol, the sender chooses a set of n messages, and the size of each message is l bit. However, the receiver selects an input $r \in [0, n-1]$. The common inputs of this protocol are similar to the OT protocol of Figure 4. At the end of this protocol, the sender gets nothing, while the receiver receives x_r. In item 1, the sender sends $(n-1)$ elements $\{C_1, C_2, \ldots, C_{n-1}\}$ along with g^γ to the receiver. In item 2, the receiver uses his private inputs (s,r) and computes the value of β_0 and sends it to the sender. In item 3, the sender computes $\beta_i^\gamma = C_i^\gamma/\beta_0^\gamma$, for all $0 \le i \le n-1$ and encrypts the values of x_i using $H(\beta_i^\gamma, X, i)$. In the last item, the receiver uses his private inputs (s,r) and decrypts the required messages x_r.

A new DDH assumption based one-out-of-n OT protocol has been proposed in Reference [86]. The description of this OT protocol is given in Figure 18. In this protocol, the sender has n messages, while the receiver has a number $r \in [1,n]$. Moreover, both parties agree on two generators g, h of the multiplicative group $\mathcal{G}_q$ of prime order q. At the end of this protocol, the sender gets nothing while the receiver receives the required message x_r. In item 1, the receiver randomly generates an element $s \in \mathcal{Z}_q^*$ and uses his input $r \in [1,n]$, and sends the value of β to the sender. In item 2, the sender randomly generates the n keys $k_i \in \mathcal{Z}_q^*$ and encrypts the input messages x_i with them and sends the encrypted messages (a_i, b_i) to the receiver. The receiver uses the private inputs (s,r) and decrypts the required message x_r.

Common Inputs: Let g, h be the two generators in $\mathcal{G}_q$ of prime order q, ROM $H : \mathcal{G}_q \rightarrow \mathcal{G}_q$.

(1) The receiver randomly chooses a number $s \in \mathcal{Z}_q$, besides he chooses an input $r \in \{1, ..n\}$. After that, he computes $\beta = g^s h^r$ and sends the value of β to the sender and keeps the values of (r, s) private.
(2) The sender randomly chooses $k \in \mathcal{Z}_q^*$, and computes $e_i = x_i \oplus H((\beta/h^i)^k, i))$, $a = g^k$, for all $1 \leq i \leq n$. The sender then sends the values of $e_1, e_2, ..., e_n$ along with a to the receiver.
(3) R uses the values of (r, s) and extracts the required message as follows: $x_r = e_r \oplus H(a^s, r)$.

Fig. 19. ROM based 1-out-of-n OT protocol for semi-honest receiver [86].

Input of S**:** It has n messages $x_i \in \mathcal{G}_q$, for all $1 \leq i \leq n$, where $\mathcal{G}_q$ is the multiplicative group of prime order q.
Input of R**:** It has k inputs $r_1, r_2, ...r_k \in \{1, ...n\}$.
Common Inputs: Let g, h be the two generators in $\mathcal{G}_q$ of prime order q.
Output of S**:** Nothing.
Output of R**:** x_{r_j}, for all $1 \leq j \leq k$.

(1) The receiver randomly chooses k numbers $s_i \in \mathcal{Z}_q$, besides he chooses an input $r_i \in \{1, ..n\}$. After that, he computes $\beta_i = g^{s_i} h^{r_i}$, for all $1 \leq i \leq k$ and sends the value of β_i to the sender and keeps the values of (r_i, s_i) private.
(2) The sender randomly chooses $k_{i,j} \in \mathcal{Z}_q^*$, and computes $(a_{i,j}, b_{i,j}) = (g^{k_{i,j}}, x_i(\beta_j/h^i)^{k_{i,j}})$, for all $1 \leq i \leq n$, $1 \leq j \leq k$. The sender then sends the values of $(a_{i,j}, b_{i,j})$ to the receiver.
(3) R uses the values of $(r_1, s_1), (r_2, s_2), ..., (r_k, s_k)$ and extracts the required messages as follows: $x_{r_j} = b_{r_j,j}/(a_{r_j,j})^{s_j}$, for all $1 \leq j \leq k$.

Fig. 20. DDH based k-out-of-n OT protocol for semi-honest receiver [86].

In the same work, they proposed another one-out-of-n OT protocol under the assumption of ROM [86]. The description of this protocol is given in Figure 19. Input, output, and common inputs of this protocol are similar to the previous protocol of Figure 18, except that this protocol uses the ROM H. Item 1 of this protocol is also identical to item 1 of the previous protocol of Figure 18. In item 2, the sender randomly generates an element k and encrypts the input messages x_i with keys $H((\beta/h^i)^k, i))$ and sends all the encrypted values $e_1, e_2, \ldots e_n$ along with g^k to the receiver. In the last item, the receiver uses his private inputs (r, s) and decrypts the required message x_r.

In the same work, they have also proposed k-out-of-n OT protocol under the DDH assumption. The description of this protocol is given in Figure 20. In this protocol, the sender has n input messages $\{x_1, x_2, \ldots, x_n\} \in \mathcal{G}_q$ while the receiver has k choice of numbers $\{r_1, r_2, \ldots, r_k\} \in [1, n]$. The common inputs of this protocol are similar to the one-out-of-n OT protocol of Figure 18. At the end of this OT protocol, the sender gets nothing from this protocol while the receiver receives k messages $\{x_{r_1}, x_{r_2}, \ldots, x_{r_k}\}$. In item 1 of this protocol, the receiver uses his private inputs $\{(r_1, s_1), (r_2, s_2), \ldots, (r_k, s_k)\}$ and computes the values of $\{\beta_1, \beta_2, \ldots, \beta_k\}$, and sends them to the sender. In item 2, the sender randomly generates the keys $k_{i,j}$ and encrypts the messages $\{x_1, x_2, \ldots, x_n\}$ using randomly generated keys and sends the encrypted messages $(a_{i,j}, b_{i,j})$ to the receiver. In the last item, the receiver uses his private inputs $\{(r_1, s_1), (r_2, s_2), \ldots, (r_k, s_k)\}$ and decrypts the required messages $\{x_{r_1}, x_{r_2}, \ldots, x_{r_k}\}$.

All the one-out-of-n OT protocols we discussed until now only provide security in the semi-honest adversaries' presence. Therefore, a new one-out-of-n OT protocol has been proposed under the DDH assumption that provides security in the presence of malicious adversaries [87]. The description of this protocol is given in Figure 21. Input, output, and common inputs of this protocol are similar to the OT protocol of Figure 18. In item 1 of this protocol, the receiver randomly generates three numbers $s, s', r' \in \mathcal{Z}_q$ and computes the following β and β' to hide the private inputs r. After that, the receiver sends the values of β and β' to the sender. In item 2, the sender

Common Inputs: Let g, h be the two generators in $\mathcal{G}_q$ of prime order q.

(1) The receiver randomly chooses three numbers $s, s^{'}, r^{'} \in \mathcal{Z}_q$, besides he chooses an input $r \in \{1, ..n\}$. After that, he computes $\beta = g^s h^r$, $\beta^{'} = g^{s^{'}} h^{r^{'}}$ and sends the value of $\beta, \beta^{'}$ to the sender and keeps the values of (r, s) private.
(2) The sender randomly chooses an element $c \in \mathcal{Z}_q^*$ and sends it to the receiver.
(3) The receiver computes the following: $z_1 = (s + s^{'}c) \mod q$, $z_2 = (r + r^{'}c) \mod q$ and sends them to the sender.
(4) If $\beta(\beta^{'})^c \neq g^{z_1}.h^{z_2}$, then the sender aborts the scheme; otherwise, randomly chooses $k_i \in \mathcal{Z}_q^*$, and computes $(a_i, b_i) = (g^{k_i}, x_i(\beta/h^i)^{k_i})$, for all $1 \leq i \leq n$. The sender, then sends the values of $(a_1, b_1), (a_2, b_2), ..., (a_n, b_n)$ to the receiver.
(5) R uses the values of (r, s), and extracts the required message as follows: $x_r = b_r/(a_r)^s$.

Fig. 21. DDH-based 1-out-of-n OT protocol for malicious receiver [87].

Input of S**:** It has n messages $x_i \in \{0,1\}^l$ where $0 \leq i \leq n-1$.
Input of R**:** It has a value $r \in \{0, 1, ..., n-1\}$.
Common Inputs: q and m are selected in the form of $q \equiv 1 \ mod \ 2m$. Both parties agree on a uniformly generated polynomial $a \in \mathcal{U}(\mathcal{R}_q)$, Sampling algorithm: $Sample(\chi)$.

(1) The sender generates two vectors, $s, e \leftarrow Sample(\chi)^2$, computes $A \leftarrow a.s + e \in \mathcal{R}_q$, $T_0 = 0^n$, $T_i \leftarrow \mathcal{U}(\mathcal{R}_q)$, $1 \leq i \leq n-1$ and sends all the polynomials T_0, T_i, A to the receiver.
(2) The receiver also generates three vectors, $s^{'}, e^{'}, e^{''} \leftarrow Sample(\chi)^3$, computes $b^{'} \leftarrow a.s^{'} + e^{'} \in \mathcal{R}_q$, $v \leftarrow A.s^{'} + e^{''} \in \mathcal{R}_q$, $\bar{v} = RR(v)\mathcal{R}_{2q}$, $c =< \bar{v} >_2$, sets $B = b^{'} + T_r$. After that, he sends (B, c) to the sender.
(3) The sender uses its secret value s, derives the shared secret keys as follows, $k_i = reconcilation(2(B - T_i)s, c)$, for all, $0 \leq i \leq n-1$ and encrypts all the messages $e_i = x_i \oplus k_i$, and sends all the encrypted values e_i to the receiver.
(4) The receiver derives the shared secret key $k_r = \lfloor \bar{v} \rceil_2$, and extracts the required message $x_r = e_r \oplus k_r$.

Fig. 22. Universally composable 1-out-of-n OT from ideal lattice [63].

randomly generates a challenge $c \in \mathcal{Z}_q^*$ and sends it to the receiver. In item 3, the receiver uses his randomly generated inputs $s, s^{'}, r^{'} \in \mathcal{Z}_q$ and private input r and computes the values of z_1 and z_2. The receiver sends the values of z_1 and z_2 as the response to the sender. In item 4, the sender verifies the received response. If the received response is invalid, then the sender aborts the protocol; otherwise, he randomly generates the keys $k_i \in \mathcal{Z}_q^*$ and encrypts the input messages x_i. After that, the sender sends the encrypted messages $(a_1, b_1), (a_2, b_2), \ldots, (a_n, b_n)$ to the receiver. In the last item, the receiver uses his secret inputs (r, s) and decrypts the required message x_r.

All the discussed 1-out-of-n OT protocols' security depends on the following assumptions, DDH, ROM, and number-theoretic public key encryption. These security assumptions are vulnerable to quantum attacks. To overcome this problem, several quantum-resistant 1-out-of-n OT protocols have been proposed. The first quantum-resistant OT protocol has been proposed using the **ring learning with errors (ring-LWE)** key exchange mechanism [63].

This protocol provides universally composable security. The description of the universally composable ideal lattice-based 1-out-of-n OT protocol is given in Figure 22. Before the beginning of this protocol, both parties choose their inputs. For example, the sender chooses n input messages $(x_0, x_1, \ldots, x_{n-1})$. However, the receiver chooses the input value $r \in \{1, n\}$. After that, both parties agree on common inputs, a prime number q, uniformly generated polynomial a, and sampling algorithm $Sample(\chi)$. This algorithm takes input as a standard deviation χ and generates output as a secret or error vector. In item 1, the sender generates $n + 1$ polynomials, $A, T_0, \ldots, T_{n-1}$. The first polynomial A is generated using secret vectors (e, s), the second polynomial T_0 is generated

Input of S**:** It has n messages $x_i \in \{0,1\}^\ell$, for all $1 \leq i \leq n$
Input of R**:** It has an input $\gamma \in_R (0, 1, ..., m-1)$.
Common Inputs: A prime number q such that, $q \equiv 1 \mod 2m$, error distribution : χ_{16}^m. $seed_i \leftarrow \{0, 1, ..., 255\}^{32}$ $\hat{a}_i \leftarrow Parse(Shake128(seed_i))$, polynomial $\hat{a}_i$ is precomputed and made available to both parties for $0 \leq i \leq m-1$.

(1) The receiver generates two secret vectors s_γ $s_{(\gamma+1) \mod m} \leftarrow \{\chi_{16}^m\}^2$, computes $\hat{b}_\gamma = \hat{a}_\gamma.NTT(s_\gamma) + NTT(s_{(\gamma+1)mod(m)}) \in R_q$, and sends it to the sender.
(2) The sender generates $m+1$ error vectors e_0, e_1,... $e_m \leftarrow \{\chi_{16}^m\}^{m+1}$, computes $\hat{e}_i \leftarrow NTT(e_i)$, $\hat{u}_i = \hat{a}_i.\hat{e}_i + NTT(e_{(i+1)mod(m+1)}) \in R_q$, $v_i = NTT^{-1}(\hat{b}_\gamma.\hat{e}_i) + e_{(i+2)mod(m+1)} \in R_q$, $w_i \leftarrow helprec(v_i)$, $k_i \leftarrow rec(v_i, w_i)$, $msg_i = k_i \oplus x_i$, $c_i \leftarrow (\hat{u}_i, w_i)$, and sends these c_i, msg_i to receiver, where $0 \leq i \leq m-1$.
(3) The receiver uses his input γ, and extracts the required message as follows, $\hat{u}_\gamma, w_\gamma \leftarrow c_\gamma$, $v_\gamma^{'} = NTT^{-1}(\hat{u}_\gamma.\hat{s}_\gamma) \in R_q$, $k_\gamma \leftarrow rec(v_\gamma^{'}, w_\gamma)$, $x_\gamma \leftarrow k_\gamma \oplus msg_\gamma$.

Fig. 23. Ring-LWE based 1-out-of-n OT protocol for semi-honest receiver [90].

using 0^n, and the remaining polynomials are generated using a uniform distribution $\mathcal{U}(\mathcal{R}_q)$. The sender sends all the generated polynomials $(A, T_0, \ldots, T_{n-1})$ to the receiver. The receiver generates three vectors $(s^{'}, e^{'}, e^{''})$, uses the received polynomials, and derives the shared secret key k_r, and creates hint polynomial (B, c). It uses the **randomized rounding (RR)** function to remove the noise from the polynomial v. To derive the shared secret key from the polynomial $\bar{v}$, the receiver uses the modular rounding function, while it derives the hint polynomial c, using cross rounding function. In item 3, the sender uses the reconciliation function, derives the shared secret keys k_i, and encrypts all the messages x_i, and sends all the encrypted messages e_i to the receiver, for all $0 \leq i \leq n-1$. The receiver uses the input value $r \in \{0, 1, \ldots, n-1\}$, derived shared secret key k_r, and extracts the desired message x_r. This universally composable ideal lattice-based 1-out-of-n OT protocol requires low computation costs, but it requires a high communication overhead. Apart from communication overhead, this protocol generates a secret or error vector, which requires high computation costs. Moreover, during the execution of OT protocol, it uses **Fast Fourier Transform (FFT)**-based polynomial multiplication technique, which also requires high computation costs.

To overcome these problems, a ring-LWE based 1-out-of-n OT protocol has been proposed [90]. This protocol uses a new hope key exchange mechanism and provides the input privacy of the sender and the receiver. The description of this protocol is given in Figure 23. The input and output of this protocol are similar to the Figure 22 protocol. At the beginning of this protocol, both parties agree on a prime number q, error distribution function χ_{16}^m, and parse algorithm. For the efficient polynomial multiplication, they have chosen the prime number q with the following condition ($q \equiv 1 \; mod 2m$). The error distribution function uses the Binomial distribution technique and generates the error or secret vector. Thus, it requires low computation costs as compared to Gaussian distribution. The parse algorithm takes input as a uniformly generated seed, $\{seed_i\}_{i=0}^{m-1}$ and generates output as a set of polynomials $\{\hat{a}_i\}_{i=0}^{m-1}$. In item 1, the receiver generates two secret vectors $(s_\gamma, s_{(\gamma+1)mod\ m})$ using the error distribution technique χ_{16}^m, computes a polynomial $\hat{b}_\gamma$, and sends it to the sender. During computation of the polynomial $\hat{b}_\gamma$, the receiver applies the **number-theoretic transform (NTT)** technique in the secret vectors $((s_\gamma, s_{(\gamma+1)mod\ m}))$ for efficient polynomial multiplication. In item 2, the sender generates $m+1$ error vectors $e_0, e_1, \ldots, e_m$ and derives the shared secret keys k_i and hint polynomials w_i. It uses the *rec* function to derive the shared secret key k_i from the polynomial w_i, while it uses the *helprec* function to derive the mask polynomial. The *rec* function is an efficient version of the modular rounding function, while the *helprec* function is an efficient version of the cross rounding function. The sender uses the shared

Input of S**:** It has n messages $x_i \in \mathcal{Z}_q^*$, for all $1 \leq i \leq n$.
Input of R**:** It has k inputs $r_1, r_2, ...r_k \in \{1, ...n\}$.
Common Inputs: Let g be a generator in $\mathcal{G}_q$ of prime order q.

(1) The receiver randomly chooses k numbers $m_i \in \mathcal{Z}_q^*$, besides he chooses k inputs $r_i \in \{1, ..n\}$, for all $1 \leq i \leq k$, and computes $\beta_{r_i} = g^{r_i}$. After that, he randomly generates $n - k$ numbers $y_j \in \mathcal{Z}_q^*$ and computes $\beta_j = m_1 y_j + ... + m_k y_j^k$, for all $k + 1 \leq j \leq n$, and sends the values of β_i to the sender for all $1 \leq i \leq n$.
(2) The sender randomly chooses $k_i \in \mathcal{Z}_q^*$, and computes $(a_i, b_i) = (g^{k_i}, x_i(\beta_i)^{k_i})$, for all $1 \leq i \leq n$. The sender then sends the values of (a_i, b_i) to the receiver.
(3) R uses the values of $r_1, r_2, ..., r_k$ and extracts the required messages as follows: $x_{r_j} = b_{r_j}/(a_{r_j})^{r_j}$, for all $1 \leq j \leq k$.

Fig. 24. Public-key encryption based k-out-of-n OT protocol for semi-honest receiver [67].

secret keys k_i and encrypts all the messages x_i with these keys k_i and sends them along with hint polynomials w_i to the receiver. The receiver uses his input value γ and derives the shared secret key k_γ and extracts the required message x_γ. This OT protocol is computationally efficient and requires a low communication overhead.

Another k-out-of-n OT protocol has been proposed that uses the public-key encryption technique [67]. This OT protocol requires fewer computation costs than the other k-out-of-n OT protocols [68, 86]. The description of this protocol is given in Figure 24. In this protocol, the sender has n input messages, $x_i \in \mathcal{Z}_q^*$, for all $1 \leq i \leq n$, whereas the receiver has k choice of numbers $\{r_1, r_2, \ldots, r_k\} \in [1, n]$. The generator g is the common input of this protocol. Once this protocol end, the receiver receives k messages $\{x_{r_1}, x_{r_2}, \ldots, x_{r_k}\}$ from this protocol, whereas the sender gets nothing from this protocol. In item 1 of this protocol, the receiver randomly generates k numbers $\{m_1, m_2, \ldots, m_k\}$ and hides his inputs as follows: $\beta_{r_i} = g^{r_i}$, for all $1 \leq i \leq k$. After that, the sender randomly generates $(n - k)$ numbers $\{y_{k+1}, y_{k+2}, \ldots, y_n\}$ and creates the following dummy messages, $\beta_j = m_1 y_j + \cdots + m_k y_j^k$, for all $k + 1 \leq j \leq n$, so the sender is unable to distinguish the real input messages of the receiver. The receiver sends all the messages β_i to the sender. In item 2, the sender randomly generates the keys $k_i \in \mathcal{Z}_q^*$ and encrypts his input messages $\{x_1, x_2, \ldots, x_n\}$ and sends the encrypted messages (a_i, b_i) to the receiver. The receiver uses his secret inputs $r_1, r_2, \ldots, r_k$ and decrypts the required messages $\{x_{r_1}, x_{r_2}, \ldots, x_{r_k}\}$.

The k-out-of-n OT protocol, described in Figure 24, requires high communication costs at the receiver side. To overcome this issue, another CDH assumption-based OT protocol has been proposed [18, 19]. All the discussed k-out-of-n OT protocols [67, 68, 86] till now provide only security against the semi-honest receiver. A new CDH-based OT protocol has been proposed to provide security against the malicious receiver [18, 19]. This protocol uses two different ROMs to achieve security in the presence of malicious receivers. The description of this protocol is given in Figure 25. Input and output of this protocol are similar to the k-out-of-n OT protocol of Figure 20. The common inputs of this protocol are as follows: Both parties agree on generator g and two ROMs H_1 and H_2. The ROM H_1 takes the input of variable length and produces output as a group element; however, the ROM H_2 takes input as an element of a multiplicative group $\mathcal{G}_q$ and produces output as a l bit string. In item 1 of this protocol, the receiver randomly generates k elements $\{s_1, s_2, \ldots, s_k\}$ from the multiplicative group $\mathcal{Z}_q^*$. The receiver then hides input number $r_j \in [1, n]$, for all $1 \leq j \leq k$ as follows: first hashes the input numbers r_j using ROM H_1 and then uses the output of H_1 and randomly generated elements $\{s_1, s_2, \ldots, s_k\}$ to hide the inputs $r_j \in [1, n]$ of the receiver. After that, the receiver sends the hidden values $\beta_1, \beta_2, \ldots, \beta_k$ to the sender. In item 2, the sender encrypts his input messages x_i using $H_2(\omega_i^x)$, where $\omega_i = H_1(i)$, $x \in \mathcal{Z}_q^*$ and sends the

Common Inputs: Let g be the generator in $\mathcal{G}_q$ of prime order q. Hash functions $H_1 : \{0,1\}^* \rightarrow \mathcal{G}_q$, $H_2 : \mathcal{G}_q \rightarrow \{0,1\}^l$.

(1) The receiver randomly chooses k numbers $s_i \in \mathcal{Z}_q^*$, besides he chooses inputs $r_i \in \{1, 2, ...n\}$ and computes $\omega_{r_i} = H_1(r_i)$, $\beta_i = \omega_{r_i} g^{s_i}$, for all $1 \leq i \leq k$ and sends the values of $\beta_1, \beta_2, ..., \beta_k$ to the sender.
(2) The sender randomly chooses a number $x \in \mathcal{Z}_q^*$, and computes, $y = g^x$, $d_j = (\beta_j)^x$, $\omega_i = H_1(i)$, and $e_i = x_i \oplus H_2(\omega_i^x)$, for all $1 \leq j \leq k$, $1 \leq i \leq n$. After that, he sends the values of y, d_j, e_i to the receiver.
(3) R uses the values of $(r_1, s_1), (r_2, s_2)..., (r_k, s_k)$ and extracts the required messages as follows: $x_{r_j} = e_{r_j} \oplus H_2(K_j)$, for all $1 \leq j \leq k$, where $K_j = d_j / y^{s_j}$.

Fig. 25. CDH based k-out-of-n OT protocol for semi-honest receiver [18, 19].

Common Inputs: Let g be the generator in $\mathcal{G}_q$ of prime order q.

(1) The sender randomly generates an element $\gamma \in \mathbf{Z}_q$, and computes the public-key $pk = g^\gamma$. After that, he encrypts all the messages as follows: $e_i = x_i . f_\gamma(i)$, for all $1 \leq i \leq n$ and sends the encrypted values $e_1, e_2, ..., e_n$ to the receiver.
(2) The receiver securely evaluates the functions $f_\gamma(r_1)$, $f_\gamma(r_2)$,...,$f_\gamma(r_k)$ respectively on his inputs $r_1, r_2, ...r_k$.
(3) R uses the values of $\{r_1, , r_2, ..., r_k\}$ and extracts the required messages as follows: $x_{r_j} = e_{r_j} / f_\gamma(r_j)$, for all $1 \leq j \leq k$.

Fig. 26. OPRF-based k-out-of-n OT protocol for malicious receiver [52].

encrypted messages e_i, along with y, d_j to the receiver. In the last item, the receiver uses his secret values $(r_1, s_1), (r_2, s_2) \ldots, (r_k, s_k)$ and decrypts the required messages.

A new **oblivious pseudorandom function (OPRF)**-based k-out-of-n OT protocol has been proposed in Reference [52]. In OPRF, the sender has a private input, and the receiver also has a private input. Both parties execute the OPRF scheme on their inputs. Once the scheme execution ends, the receiver receives the function's output, but he does not get any information about the sender's input. Moreover, the sender gets nothing from this scheme. Besides, it also does not get any information about the receiver's input. The description of OPRF-based k-out-of-n OT protocol is given in Figure 26. Input, output, and common inputs of this k-out-of-n OT protocol are similar to the k-out-of-n OT protocol of Figure 24. In item 1 of this protocol, the sender randomly generates a private input γ and computes the public key $pk = g^\gamma$. The sender then evaluates a set of functions on his private input γ as follows, $f_\gamma(i) = g^{1/(\gamma+i)}$, for all $1 \leq i \leq n$. After that, the sender encrypts his messages x_i using these functions $f_\gamma(i)$ as follows, $e_i = x_i . f_\gamma(i)$, for all $1 \leq i \leq n$ and sends the encrypted values $e_1, e_2, \ldots, e_n$ to the receiver. In item 2, the receiver securely evaluates the OPRF on their inputs $\{r_1, r_2, \ldots r_k\}$ and gets the following output, $f_\gamma(r_j) = g^{1/(\gamma+r_j)}$, for all $1 \leq j \leq k$ (description of OPRF evaluation is described in Reference [52]). In the last item, the receiver uses the output of OPRF functions $f_\gamma(r_j)$ and decrypts the required messages. This OPRF-based OT protocol provides security in the presence of a malicious receiver, but it requires several rounds of communication during the computation of OPRF.

2.4 Discussion on 1-out-of-n OT Protocols

The first one-out-of-n OT protocol was based on PRF [68]. This protocol offers security in the presence of a semi-honest receiver at high communication overhead. To minimize the communication overhead, a ROM-based 1-out-of-n OT protocol has been proposed [69]. It also offers security in the presence of a semi-honest receiver at a low communication cost but requires high computation overhead. The DDH assumption-based 1-out-of-n OT protocol has been proposed to minimize computation costs [86]. This protocol tries to minimize the computation costs but still suffers from

Table 2. Summary of One-out-of-n OT Protocols

	[68]	[69]	[86] (18)	[86] (19)	[87]	[63]	[90]
Semi-honest Adversaries	YES	YES	YES	YES	YES	NO	YES
Malicious Adversaries	NO	NO	NO	NO	YES	NO	NO
Universally Composable	NO	NO	NO	NO	NO	YES	NO
Used Technique	PRF	ROM	DDH	ROM	DDH	Ideal Lattice	Ring-LWE

high computation costs. They have provided another ROM-based one-out-of-n OT protocol that offers security in the presence of a semi-honest receiver at low communication and computation costs [86]. All the discussed one-out-of-n OT protocols only provide security in the presence of a semi-honest receiver, but they fail to provide security in the presence of a malicious receiver. A DDH assumption-based one-out-of-n OT protocol has been proposed using a zero-knowledge proof protocol that offers security in the presence of a malicious receiver [87]. All these number-theoretic based 1-out-of-n OT protocols either provide security in the presence of a semi-honest or malicious receiver. However, they are vulnerable to a quantum attack; besides, they do not provide security in universally composability. The first lattice-based public-key encryption-based 1-out-of-n OT protocol has been proposed to provide security in the presence of a quantum attack [10]. In this OT protocol, they have used NTRU public-key encryption technique and offer security in the presence of a semi-honest receiver, but it requires high communication and computation overhead. An ideal lattice-based 1-out-of-n OT protocol has been proposed that offers universal composable security at low computation costs, but it requires high communication costs [63]. To minimize communication overhead, a ring-LWE based 1-out-of-n OT protocol has been proposed that also offers universal composable security at low communication and computation costs [90]. In Table 2, we described the summary of one-out-of-n OT protocols. Table 2 shows that in the number-theoretic-based OT protocols, only the protocol in Reference [87] provides security in the presence of malicious receivers; rest of the number-theoretic based OT protocols provide security in the presence of semi-honest receivers. In the lattice-based OT protocols, only ideal lattice-based OT protocol offers universally composable security [63]; other lattice-based OT protocols provide security in the presence of a semi-honest receivers. 1-out-of-n OT protocols that provide security in the presence of semi-honest receivers require low communication and computation costs; however, the 1-out-of-n OT protocols that provide security in the presence of malicious receivers need high communication and computation costs.

2.5 Discussion on k-out-of-n OT Protocols

The first k-out-n OT protocol was proposed using PRF [68]. It offers security in the presence of a semi-honest receivers at high computation and communication costs. To minimize the communication and computation costs, a DDH assumption-based k-out-n OT protocol has been proposed [86]. This protocol tries to achieve security in the presence of a semi-honest receivers at low computation and communication costs, but it failed to minimize these costs. A PKE-based k-out-n OT protocol has been proposed to minimize these costs [67]. This protocol also offers security in the presence of a semi-honest receivers. Another DDH assumption-based OT protocol has been proposed to provide security in the presence of a semi-honest receivers [18, 19]. This scheme uses the polynomial multiplication technique to minimize communication and computation costs. A CDH assumption-based k-out-n OT protocol efficiently provides security in the presence of semi-honest receivers [18, 19]. All the discussed k-out-n OT protocols provide security in the presence of semi-honest receivers only [18, 19, 67, 68, 86] and fail to provide security in the presence of malicious receivers. An OPRF-based k-out-n OT protocol has been proposed to provide security in the presence of a malicious receivers, but it requires high communication and computation costs [52]. To

Table 3. Summary of k-out-of-n OT Protocols

	[68]	[86]	[67]	[18, 19](25)	[52]	[17]
Semi-honest Adversaries	YES	YES	YES	YES	YES	YES
Malicious Adversaries	NO	NO	NO	NO	YES	YES
Universally Composable	NO	NO	NO	NO	NO	NO
Used Technique	PRF	DDH	PKE	CDH	OPRF	BP

minimize these costs, a BP-based k-out-n OT protocol has been proposed [17]. This protocol efficiently offers security in the presence of a malicious receivers. Table 3 shows that only protocols in References [52] and [17] provide security in the presence of malicious receivers; however, the rest of the OT protocols only offer security in the presence of semi-honest adversaries. Similar to the 1-out-of-2 and 1-out-of-n OT protocols, we found that the k-out-of-n OT protocols that provide security in the presence of semi-honest receivers require low communication and computation costs; however, the k-out-of-n OT protocols that provide security in the presence of malicious receivers need high communication and computation costs.

3 OT EXTENSION PROTOCOL

The first OT extension protocol was introduced by Beaver [5]. In this work, they have suggested how to get many oblivious transfer messages using a few numbers of OT protocols. This OT extension protocol depends on the one-way function. Since the one-way function used in this protocol is based on public-key cryptography, it requires high computation cost. Therefore, this OT extension protocol is inefficient. This one-way function-based OT extension protocol provides security against the semi-honest adversaries.

Public-key encryption technique made the OT extension protocol inefficient. To make that protocol efficient, another OT extension protocol [50] based on idea similar to the *hybrid encryption* scheme has been proposed. A hybrid encryption scheme can be described as follows: Suppose that sender wants to encrypt the long message using the public-key encryption technique, then it requires huge computation costs. To send the encrypted message efficiently, the sender uses the combination of public-key [22, 32, 47, 75] and *symmetric key encryption* techniques [11, 53, 84]. The sender first uses the public-key encryption technique to encrypt the private key and sends the encrypted private key to the receiver. After that, the sender breaks the long messages into a set of blocks, encrypts each block using symmetric key encryption with the help of a private key, and sends it to the receiver. The receiver first decrypts the encrypted private key using the public key decryption algorithm. It then applies this private key to decrypt each block of encrypted messages. Similar to the hybrid encryption scheme, the OT extension protocol uses the combination of a ROM and symmetric key encryption to send a large number of oblivious messages using a few numbers of OT protocols. In this OT extension protocol, the receiver first generates a set of pairs of secret keys corresponding to the required messages' input. After that, the receiver sends those secret keys to the sender using a small number of OT protocols. The sender first applies those secret keys on the ROM and gets the outputs. After that, the sender uses the ROM outcomes in the symmetric key encryption technique to encrypt the oblivious messages and sends them to the receiver. To decrypt the received encrypted oblivious messages, the receiver applies the secret keys on the ROM and uses the output of the ROM on the symmetric key decryption technique and extracts the oblivious messages. A description of the OT extension protocol [50] is given in Figure 27. For convenience, we call this protocol as IKNP. In this protocol, the sender chooses input as m pairs of l bits string (x_0^j, x_1^j), while the receiver chooses m bits string, $\mathbf{r} = (r_1, r_2, \ldots, r_m)$.

Input of S**:** m pairs (x_0^j, x_1^j) of l-bit strings, where $1 \leq j \leq m$.
Input of R**:** Selection of m bits vector $\mathbf{r} = (r_1, r_2, ..., r_m)$.
Common Inputs: A ROM $H : [m] \times \{0,1\}^k \rightarrow \{0,1\}^l$, security parameter k, and an ideal OT_m^k.
Output of S**:** Nothing.
Output of R**:** $x_{r_j}^j$, where $1 \leq j \leq m$.

(1) The sender S randomly chooses k bits vector $s = \{s_1, s_2, ..., s_k\} \in \{0,1\}^k$; however, the receiver R randomly chooses $m \times k$ bits matrix T.
(2) Both parties execute the OT_m^k protocol, where S behaves as a receiver corresponding to the input s and R behaves as a sender corresponding to k pairs of input string $(t^i, \mathbf{r} \oplus t^i)$, where, $1 \leq i \leq k$, and t^i is the i-th column of the matrix T.
(3) Assume that, the matrix Q of order $m \times k$ is received by the sender S. It is noted that $q^i = (s_i.\mathbf{r}) \oplus t^i$ and $q_j = (r_j.s) \oplus t_j$. Where q^i and q_j respectively denote the i-th column and j-th row of the matrix Q.
(4) The sender S sends (y_0^j, y_1^j) to R, where $y_0^j = x_0^j \oplus H(j, q_j)$ and $y_1^j = x_1^j \oplus H(j, q_j \oplus s)$, for all $1 \leq j \leq m$.
(5) The receiver extracts his output $x_{r_j}^j = y_{r_j}^j \oplus H(j, t_j)$.

Fig. 27. PRF-based OT extension protocol for semi-honest adversaries or IKNP protocol [50].

Both parties sender and receiver agree on common inputs, ROM H, security parameter k, and OT_m^k protocol. The ROM takes two inputs, $j \in [1, m]$ and length of k bits string, and produces output as the length of l bits string. After the execution of IKNP protocol, the receiver gets $x_{r_j}^j$ as outputs, while the sender receives nothing. The description of the IKNP protocol is as follows: In item 1, the sender chooses k bits random vector $s \in \{0,1\}^k$, and the receiver chooses a random matrix T of order $m \times k$. In item 2, the receiver uses the random matrix T and the secret vector $\mathbf{r}$ and generates k pairs of m bits string, i.e., $(t^i, \mathbf{r} \oplus t^i)$, where $1 \leq i \leq k$, and t^i is the ith column of the matrix T. After that, both parties execute the OT_m^k protocol. In this protocol, the sender behaves as a receiver with input vector s while the receiver behaves as a sender with input k pairs of m bits string. After the completion of OT_m^k protocol, the sender receives a matrix Q of order $m \times k$. The ith row of the matrix Q can be written as $q^i = (s_i.\mathbf{r}) \oplus t^i$, while the jth column of the matrix can be written as $q_j = (r_j.s) \oplus t_j$. In item 3, since the sender knows that, if the r_j-th bit of the receiver is 0, then the expression q_j becomes $q_j = t_j$. If r_j-th bit is 1, then the expression q_j becomes $q_j = s \oplus t_j$, or $t_j = q_j \oplus s$. Therefore, in item 4, the sender encrypts the value of x_0^j using q_j and the value of x_1^j using $q_j \oplus s$, for all $1 \leq j \leq m$. For example, S computes $y_0^j = x_0^j \oplus H(j, q_j)$ and $y_1^j = x_1^j \oplus H(j, q_j \oplus s)$, for all $1 \leq j \leq m$ and sends the ciphertext (y_0^j, y_1^j) to the receiver. In the last step, the receiver uses his secret vector $\mathbf{r} = (r_1, r_2, \ldots, r_m)$ and decrypts $x_{r_j}^j = y_{r_j}^j \oplus H(j, t_j)$, for all $1 \leq j \leq m$.

In Reference [50], they claim that if the malicious receiver uses the different value of $\mathbf{r}$ during every computation of $\{(t^i, t^i \oplus \mathbf{r})\}_{i=1}^m$ in the IKNP protocol, then he can learn all the inputs of the server, which makes the IKNP protocol insecure. To overcome this issue, they have also proposed an OT extension protocol for the malicious receiver. To prevent the adversarial attack, they have used the *cut-and-choose* technique [50]. sends $(y_{(p),0}^j, y_{(p),1}^j)$, for all $1 \leq j \leq m$, $1 \leq p \leq \sigma$ to R, where $y_{(p),b}^j = x_{(p),b}^j \oplus H(p, j, q_{(p),j} \oplus b.s)$. In return, R sends the correction bit $c_{(p)}^j = r_j \oplus r_{j,(p)}$, for all $1 \leq j \leq m$ and $p \notin P$. In item 5, S uses this correction bit and $(x_{(p),0}^j, x_{(p),1}^j)$ and encrypts the pairs of messages (x_0^j, x_1^j), then sends them to R. In the last item, R uses his private input $\mathbf{r}$ and decrypts the desired messages.

In IKNP protocol, we get m parallel OTs of l bits string using only the execution of k OTs of m bits string, which requires huge communication costs. In Reference [46], they have proposed an OT extension protocol to reduce the communication costs of the IKNP protocol. In this protocol,

Common Inputs: A ROM $H : \{0,1\}^{\lceil \log m \rceil} \times \{0,1\}^k \rightarrow \{0,1\}^l$, PRP $G : \{0,1\}^M \times \{0,1\}^k \rightarrow \{0,1\}^M$, and an ideal OT_k^k. Where, k is a security parameter, and $M = m/B$, where B is size of the block.

(1) The sender S randomly chooses k bits vector $c = \{c_1, c_2, ..., c_k\} \in \{0,1\}^k$; however, the receiver R randomly chooses $M \times k$ bits matrix T. Moreover, R randomly chooses k pairs (s_0^j, s_1^j) of k-bits strings, where $1 \leq j \leq k$.
(2) Both parties execute the OT_k^k protocol, where S behaves as a receiver corresponding to the input c and R behaves as a sender corresponding to the input (s_0^j, s_1^j), where, $1 \leq j \leq k$. After the execution of OT_k^k protocol, S receives $s_{c_j}^j$.
(3) For all $0 \leq b < B$, computes $u_0^j = G(b, s_0^j) \oplus t^j$, $u_1^j = G(b, s_1^j) \oplus t^j \oplus (r_{bM+1}, ..., r_{bM+M})$ and sends these values to S. Where t^j is the j-th column of the matrix T, $1 \leq j \leq k$, R, and $m = BM$.
(4) S computes a matrix V of order $M \times k$ as follows, $V^j = u_{c_j}^j \oplus G(b, s_{c_j}^j)$, for all $1 \leq j \leq k$, where V^j is the j-th column of the matrix V. After that, he computes, $y_0^i = x_0^{bM+i} \oplus H(bM + i, V_i)$, $y_1^i = x_1^{bM+i} \oplus H(bM + i, V_i \oplus c)$ for all $1 \leq i \leq M$, where V_i is the i-th row of the matrix V. S sends (y_0^i, y_1^i) to R, for all $1 \leq i \leq M$, and $0 \leq b < B$.
(5) R extracts the $x_{r_{bM+i}}^i = y_{r_{bM+i}}^i \oplus H(bM + i, t_i)$, for all $1 \leq i \leq M$, $0 \leq b < B$, and t_i is the i-th row of the matrix T.

Fig. 28. PRP-based OT extension protocol for semi-honest adversaries [46].

they transfer m parallel OTs of l bits string using only the execution of k OTs of k bits string. To reduce communication costs, they have used AES and SHA-1, respectively, as **pseudo-random permutation (PRP)** and ROM. We have described their OT extension protocol in Figure 28, which is only secure for the semi-honest adversaries, and it is used in a fast garbled circuit [49]. This protocol takes input similar to the IKNP protocol, and it produces output similar to the IKNP protocol. The common input H, takes $(\lceil \log m \rceil + k)$-bits string as input and produces l-bits output, while G takes $(M + k)$-bits input and produces M-bits output. Both S and R agree on block size B and divide the value of m by B and get M so both parties execute the OT extension protocol parallelly. In step 1, S randomly chooses k bits string c, while R chooses k pairs of string (s_0^j, s_1^j), where size of every string is k bits. Also, R randomly chooses $M \times k$ bits matrix T. In step 2, both parties execute the OT_k^k protocol and after the execution of OT_k^k protocol, S receives $s_{c_j}^j$. In step 3, the receiver R uses his input matrix T and parallelly computes $u_0^j = G(b, s_0^j) \oplus t^j$, $u_1^j = G(b, s_1^j) \oplus t^j \oplus (r_{bM+1}, \ldots, r_{bM+M})$, for all $0 \leq b < B$, $1 \leq j \leq k$ and sends the values of (u_0^j, u_1^j) to S. In step 4, S uses his input vector c and the output of OT_k^k protocol s_{c_j} and creates a matrix V of order $M \times k$. S encrypts m pairs of l-bits string (x_0^j, x_1^j), parallelly using matrix V and sends (y_0^i, y_1^i) to R. In the last step, R uses his secret vector $\mathbf{r}$ and decrypts the desired messages.

Since the IKNP protocol uses the cut-and-choose technique to gain security against the malicious receiver, they achieve the security of $2^{-\sigma}$ by performing σ parallel computations of the underlying OT extension protocol. Due to these parallel computations, the IKNP protocol for the malicious receiver suffers from additional overhead. A new OT extension protocol, ABM box, for malicious receiver has been proposed [71]. This protocol uses the ROM to reduce the additional overhead. The description of the ABM box is given in Figure 29. This protocol also takes input similar to the IKNP protocol, and it produces output similar to the IKNP protocol. In this protocol, both parties agree on two ROMs and one collision-resistant hash function. Both ROMs H and $\tilde{H}$ take input as k bits string and produce output as l bits string. The description of this OT extension is as follows: item 1 to 3 of this OT extension protocol is similar to the IKNP protocol. In the OT_m^k protocol, if the receiver uses different values of $\mathbf{r}$ in their k pairs of input string $\{(t^i, \mathbf{r} \oplus t^i)\}_{i=1}^k$, then at the end of the OT extension protocol, he gets more information than he required. In other words, if the receiver is malicious, then he may cheat the sender inputs s and extract all the inputs

Common Inputs: A ROMs $H : \{0,1\}^k \rightarrow \{0,1\}^l$, and $\tilde{H} : \{0,1\}^k \rightarrow \{0,1\}^l$. $hash$: collision resistant hash function, security parameter k, and an ideal OT_m^k.

(1) The sender S randomly chooses k bits vector $s = \{s_1, s_2, ..., s_n\} \in \{0,1\}^k$; however, the receiver R randomly chooses $m \times k$ bits matrix T.
(2) Both parties execute the OT_m^k protocol, where S behaves as a receiver corresponding to the input s and R behaves as a sender corresponding to the k pairs of input string $(t^i, \mathbf{r} \oplus t^i)$, where, $1 \leq i \leq k$, and t^i is the i-th column of the matrix T.
(3) Assume that, the matrix Q of order $m \times k$ is received by the sender S. It is noted that $q^i = (s_i.\mathbf{r}) \oplus t^i$ and $q_j = (r_j.s) \oplus t_j$. Where q^i and q_j respectively denote the i-th column and j-th row of the matrix Q.
(4) **Consistency Check:**
   - For all $1 \leq j \leq m$, S computes $e_0^j = H(q_j)$, $e_1^j = H(q_j \oplus s)$, $f^j = e_0^j \oplus e_1^j$ and stores, $E_S = (e_0^1, e_0^2, ...e_0^m)$ and sends $F = (f^1, f^2, ...f^m)$ to R.
   - For all $1 \leq j \leq m$, the receiver R computes $e_{r_j}^j = H(t_j)$, $e_{1-r_j}^j = H(e_{r_j}^j \oplus f^j)$, $E_R = (e_0^1, e_0^2, ...e_0^m)$, and $h_R = hash(E_R)$. After that, R sends the value of h_R to the sender S.
   - S checks that, if $h_R \neq hash(E_S)$ then he terminates the protocol, otherwise continue.
(5) The sender S sends (y_0^j, y_1^j) to R, where $y_0^j = x_0^j \oplus \tilde{H}(q_j)$ and $y_1^j = x_1^j \oplus \tilde{H}(q_j \oplus s)$, for all $1 \leq j \leq m$.
(6) The receiver extracts his output $x_{r_j}^j = y_{r_j}^j \oplus \tilde{H}(t_j)$, for all $1 \leq j \leq m$.

Fig. 29. The ABM box based OT extension protocol for the malicious receiver [71].

$\{(x_0^j, x_1^j)\}_{j=1}^m$ of the sender. To deal with this cheating activity of the receiver, the sender executes the consistency check algorithm. The sender uses this algorithm to check whether the receiver has used the same value of $\mathbf{r}$ in each pair of input string $\{(t^i, \mathbf{r} \oplus t^i)\}_{i=1}^k$ during the execution of the OT_m^k protocol. Since the sender knows that the jth row of the matrix Q can be expressed as $q_j = (r_j.s) \oplus t_j$. Moreover, he also knows that if the receiver chooses r_j-th bit 0, then the jth row of the matrix Q can be expressed as $q_j = (r_j.s) \oplus t_j = t_j$; otherwise, $q_j = (r_j.s) \oplus t_j = s \oplus t_j$ or $t_j = s \oplus q_j$. Therefore, in item 1 of the consistency check algorithm, the sender computes $e_0^j = H(q_j)$ under the assumption if the r_j bit is 0; otherwise, he computes $e_1^j = H(q_j \oplus s)$, for all $1 \leq j \leq m$. Moreover, the sender also computes $f^j = e_0^j \oplus e_1^j$, for all $1 \leq j \leq m$ and stores $E_S = (e_0^1, e_0^2, \ldots, e_0^m)$ and sends $F = (f^1, f^2, ...f^m)$ to R. In item 2 of the consistency check algorithm, if the receiver has used the same value of $\mathbf{r}$ in the OT_m^k protocol, then he computes $e_{r_j}^j = H(t_j)$, $e_{1-r_j}^j = H(e_{r_j}^j \oplus f^j)$ for all $1 \leq j \leq m$.

After that, he sends $h_R = hash(E_R)$ to S, where $E_R = (e_0^1, e_0^2, ...e_0^m)$. In item 3 of the consistency check algorithm, the sender verifies whether the receiver has used the same value of $\mathbf{r}$ during the execution of the OT_m^k protocol. To do this, S performs the following test: $h_R \stackrel{?}{=} hash(E_S)$. If the receiver has used the same value of $\mathbf{r}$ during the execution of the OT_m^k protocol, then he successfully passes this test; otherwise, the sender aborts the scheme. Item 5 of this OT extension protocol is similar to item 4 of the IKNP protocol except for the ROM $\tilde{H}$. Similarly, item 6 of this OT extension protocol is identical to item 5 of the IKNP protocol except for the ROM $\tilde{H}$. In items 5 and 6 of this OT extension protocol, the sender uses $\tilde{H}$ instead of H because the receiver may use the received values of e_0^j and e_1^j and gets more information than required.

Another commitment-based OT extension protocol for the malicious receiver [44] is described in Figure 30. The description is as follows: This OT extension protocol uses only one ROM H, which takes k bits string as input and produces l bits string. Items 1 to 3 of this OT extension protocol are similar to the IKNP protocol. Since the receiver may cheat the input of the sender s and extract all the message $\{(x_0^j, x_1^j)\}_{j=1}^m$ of the sender, the sender executes the commitment phase so the receiver cannot cheat the sender's input. In item 1 of the commitment phase, S computes $e_0^j = H(q_j)$, and

Common Inputs: A ROMs $H : \{0,1\}^k \rightarrow \{0,1\}^l$. Security parameter k, and an ideal OT_m^k.

(1) The sender S randomly chooses k-bits vector $s = \{s_1, s_2, ..., s_n\} \in \{0,1\}^k$; however, the receiver R randomly chooses $m \times k$ bits matrix T.
(2) Both parties execute the OT_m^k protocol, where S behaves as a receiver corresponding to the k pairs of input strings s and R behaves as a sender corresponding to the input $(t^i, \mathbf{r} \oplus t^i)$, where, $1 \leq i \leq k$, and t^i is the i-th column of the matrix T.
(3) Assume that, the matrix Q of order $m \times k$ is received by the sender S. It is noted that $q^i = (s_i.\mathbf{r}) \oplus t^i$ and $q_j = (r_j.s) \oplus t_j$. Where q^i and q_j respectively denote the i-th column and j-th row of the matrix Q.
(4) **Commitment Phase:**
- For all $1 \leq j \leq m$, S computes $e_0^j = H(q_j)$, and $e_1^j = H(q_j \oplus s)$, while R computes, $e_{r_j}^j = H(t_j)$.
- S randomly chooses m pairs of l-bits strings (s_0^j, s_1^j) and computes $d_0^j = e_0^j \oplus s_0^j$, $d_1^j = e_1^j \oplus s_1^j$ and sends the values of (d_0^j, d_1^j) to R, for all $1 \leq j \leq m$.
- R computes $s_{r_j}^j = d_{r_j}^j \oplus e_{r_j}^j$, for all $1 \leq j \leq m$.

(5) The sender S sends (y_0^j, y_1^j) to R, where $y_0^j = x_0^j \oplus H(s_0^j)$ and $y_1^j = x_1^j \oplus H(s_1^j)$, for all $1 \leq j \leq m$.
(6) The receiver R extracts his output $x_{r_j}^j = y_{r_j}^j \oplus H(s_{r_j}^j)$, for all $1 \leq j \leq m$.

Fig. 30. Commitment-based OT extension protocol for the malicious receiver [44].

$e_1^j = H(q_j \oplus s)$, while R computes, $e_{r_j}^j = H(t_j)$, for all $1 \leq j \leq m$. In item 2 of this phase, S randomly chooses m pairs of l-bits strings (s_0^j, s_1^j) and encrypts the values of s_0^j and s_1^j, respectively, using e_0^j and e_1^j, for all $1 \leq j \leq m$. In other words, S computes, $d_0^j = e_0^j \oplus s_0^j$, $d_1^j = e_1^j \oplus s_1^j$ and sends the values of (d_0^j, d_1^j) to R, for all $1 \leq j \leq m$. In the last item of this commitment phase, the receiver uses his private key $\mathbf{r}$ and decrypts the values of $s_{r_j}^j = d_{r_j}^j \oplus e_{r_j}^j$, for all $1 \leq j \leq m$. During the execution of OT_m^k protocol, if the receiver has not used the same value of $\mathbf{r}$ in the input of OT_m^k protocol, then in this phase, the receiver will be unable to decrypt the correct values of $s_{r_j}^j$ for $1 \leq j \leq m$. In item 5, the sender encrypts the values of x_0^j using the output of the hash values of $H(s_0^j)$ while encrypting the values of x_1^j using the output of the hash values of $H(s_1^j)$ and sends the values of (y_0^j, y_1^j) to R, for all $1 \leq j \leq m$. The receiver uses the output of the commitment phase and decrypts his required message, $x_{r_j}^j = y_{r_j}^j \oplus H(s_{r_j}^j)$, for all $1 \leq j \leq m$.

To achieve security against the malicious receiver, all the previous OT extension protocols [44, 50, 71] execute OT_m^k protocol, which requires high communication cost. Existing OT extension protocols for the semi-honest adversaries require high communication costs [46, 50]. Moreover, current OT extension protocols require transpose of a matrix T of order $m \times k$, in which they have created the matrix column-wise and hashed the matrix row-wise. This transpose operation is computationally inefficient. A new pseudo-random based OT extension protocol for the semi-honest adversaries has been proposed to reduce communication costs [2]. In this OT extension protocol they have used the Eklundh algorithm [31] to perform efficient transpose operation. This algorithm reduced the number of swapping costs from $O(n^2)$ to $O(n \log n)$. Since the OT extension protocol works over the bit-matrix, they have used the register and compute the transpose of a matrix parallelly. Using register and parallel computation, they have reduced the swap operation costs from $O(n \log n)$ to $O(\lceil \frac{n}{r} \rceil \log n)$, where $r = 64$ is the size of register. We have named this OT extension protocol as ALSZ13. We describe this ALSZ13 protocol in Figure 31. The description of this ALSZ13 protocol is as follows: The ALSZ13 protocol also takes input similar to the IKNP protocol, and it produces output similar to the IKNP protocol. Both parties agree on collision robust hash function H, PRG G. The function H takes m bits string as input and generates l bits string as output, while G grabs k bits string as input and produces m bits string as output. Item 1 of the

Common Inputs: Collision robust function $H : [m] \times \{0,1\}^m \rightarrow \{0,1\}^l$, pseudo random generator $G : \{0,1\}^k \rightarrow \{0,1\}^m$ and an ideal OT_k^k, where k is a security parameter.

(1) S randomly chooses k bits vector $s = \{s_1, s_2, ..., s_k\} \in \{0,1\}^k$; however, the receiver R randomly chooses k pairs of keys (k_0^i, k_1^i), where size of every key is k bits.
(2) Both parties execute the OT_k^k protocol, where S behaves as a receiver corresponding to the input s and R behaves as a sender corresponding to the k pairs of input string (k_0^i, k_1^i), where, $1 \leq i \leq k$. After the execution of OT_k^k protocol, the receiver computes a matrix $T = [t^1|t^2|...|t^k]$ of order $m \times k$. Where $t^i = G(k_0^i)$ is the i-th column of the matrix T, for all $1 \leq i \leq k$. Assume that, t_j denote the j-th row of the matrix T, for all $1 \leq j \leq m$.
(3) R computes $u^i = t^i \oplus G(k_1^i) \oplus \mathbf{r}$, and sends u^i to S, for all $1 \leq i \leq k$.
(4) The sender uses the values of u^i and computes $q^i = (s_i.u^i) \oplus G(k_{s_i}^i) = (s_i.\mathbf{r}) \oplus t^i$, for all $1 \leq i \leq k$. Assume that, the matrix Q of order $m \times k$ is created by the sender S. Where q^i and q_j respectively denote the i-th column and j-th row of the matrix Q. Where q_j of the matrix can be expressed as $q_j = (r_j.s) \oplus t_j$, for all $1 \leq j \leq m$.
(5) The sender S sends (y_0^j, y_1^j) to R, where $y_0^j = x_0^j \oplus H(j, q_j)$ and $y_1^j = x_1^j \oplus H(j, q_j \oplus s)$, for all $1 \leq j \leq m$.
(6) The receiver R extracts his output $x_{r_j}^j = y_{r_j}^j \oplus H(j, t_j)$, for all $1 \leq j \leq m$.

Fig. 31. PRG-based OT extension protocol for the semi-honest receiver or ALSZ13 protocol [2].

ALSZ13 protocol is similar to item 1 of the IKNP protocol except that sender chooses k bits vector instead of the n bits vector, and the receiver chooses k pairs of string instead of n pairs of the string. Item 2 of the ALSZ13 is also similar to item 2 of the IKNP, but the only difference is that the ALSZ13 protocol executes OT_k^k protocol while the IKNP protocol runs OT_k^n. Moreover, in the ALSZ13 protocol, the receiver creates a matrix of order $m \times k$; however, in the IKNP protocol, the receiver creates a $m \times n$ matrix. In item 3, the receiver encrypts his secret vector $\mathbf{r}$ using pairs of string (k_0^i, k_1^i) and sends the values of $u^i = t^i \oplus G(k_1^i) \oplus \mathbf{r}$ to the sender, for all $1 \leq i \leq k$. In item 4, the sender uses his secret vector $s \in \{0,1\}^k$, the output $k_{s_i}^i$ of the OT protocol, and the received values of u^i to create a matrix Q of order $m \times k$. In item 5, the sender uses jth row of matrix Q to encrypt the value of x_0^j while he uses the output of ex-or operation of the secret vector s and the jth row of the matrix Q to encrypt the value of x_1^j. After that, S sends the encrypted values (y_0^j, y_1^j) to the receiver. In the last item, the receiver uses his secret vector $\mathbf{r}$ and decrypts the following messages $x_{r_j}^j = y_{r_j}^j \oplus H(j, t_j)$, for all $1 \leq j \leq m$. This AlSZ13 protocol requires low communication and computation costs due to the minimum number of the base OT protocols' execution and the minimum number of swapping operations during the computation of the transpose of a matrix. The drawback of this ALSZ13 protocol is that it only provides security against the semi-honest receiver.

To achieve security against malicious receiver with better communication and computation costs, a new OT extension protocol has been proposed [3]. This OT extension protocol used the footprint of the ALSZ13 protocol. For convenience, we have named this OT extension protocol ALSZ15. We describe the ALSZ15 protocol in Figure 32. The description of this protocol is as follows: The ALSZ15 protocol takes inputs similar to the ALSZ13 protocol and generates output identical to that protocol. Moreover, the ALSZ15 protocol uses a similar collision robust function as the ALSZ13 has used. In ALSZ15 protocol, both parties agree on a PRG G that takes k bits string as input and generates $m + k$ bits string as output. Item 1 of this protocol is similar to item 1 of the ALSZ13 protocol except that sender generates n bits vector instead of k bits vector while the receiver generates n pairs of string instead of k pairs of string, where $n = k + \rho$. In ALSZ15, they have chosen the value of $k = 128$ and $\rho = 40$ in their experimental analysis. Moreover, they claimed that these constant parameters provide security beyond 2020. To provide security of this protocol

Common Inputs: Collision robust function $H : [m] \times \{0,1\}^m \rightarrow \{0,1\}^l$, pseudo random generator $G : \{0,1\}^k \rightarrow \{0,1\}^{m+k}$ and an ideal OT_k^n, where k is a security parameter. Moreover, $n = k + \rho$, where ρ is a statistical security parameter.

(1) S randomly chooses n bits vector $s = \{s_1, s_2, ..., s_n\} \in \{0,1\}^n$; however, the receiver R randomly chooses n pairs of keys (k_0^i, k_1^i), where size of every key is k bits.
(2) Both parties execute the OT_k^n protocol, where S behaves as a receiver corresponding to the input s and R behaves as a sender corresponding to the k pairs of input string (k_0^i, k_1^i), where, $1 \leq i \leq n$. After the execution of OT_k^n protocol, the receiver computes a matrix $T = [t^1|t^2|...|t^n]$ of order $(m+k) \times n$. Where $t^i = G(k_0^i)$ is the i-th column of the matrix T, for all $1 \leq i \leq n$. Assume that, t_j denote the j-th row of the matrix T, for all $1 \leq j \leq m$.
(3) R randomly chooses k bits string $\tau = \{0,1\}^k$ and computes $u^i = t^i \oplus G(k_1^i) \oplus \mathbf{r}^{'}$, and sends u^i to S, for all $1 \leq i \leq n$. Where $\mathbf{r}^{'} = \mathbf{r}||\tau$.
(4) **Consistency Check of $\mathbf{r}^{'}$**
   (a) For every pair $a, b \subseteq [n]^2$, R defines four tuples
   - $h_{0,0}^{a,b} = H(G(k_0^a) \oplus G(k_0^b))$
   - $h_{0,1}^{a,b} = H(G(k_0^a) \oplus G(k_1^b))$
   - $h_{1,0}^{a,b} = H(G(k_1^a) \oplus G(k_0^a))$
   - $h_{1,1}^{a,b} = H(G(k_1^a) \oplus G(k_1^b))$

   then sends $\mathcal{H} = (h_{0,0}^{a,b}, h_{0,1}^{a,b}, h_{1,0}^{a,b}, h_{1,1}^{a,b})$ to S.
   (b) For every pair $(a, b) \in [n]^2$, S knows s_a, s_b, $k_{s_a}^a$, $k_{s_b}^b$, u^a, u^b and performs three checks such that:
   - $h_{s_a,s_b}^{a,b} = H(G(k_{s_a}^a) \oplus G(k_{s_b}^b))$
   - $h_{s_a,s_b}^{\bar{a},\bar{b}} = H(G(k_{s_a}^a) \oplus G(k_{s_b}^b) \oplus u^a \oplus u^b) = H(G(k_{s_a}^{\bar{a}}) \oplus G(k_{s_b}^{\bar{b}}) \oplus \mathbf{r}^a \oplus \mathbf{r}^b)$
   - $u^a \neq u^b$

   If all the above checks are positive, then S continues the protocol; otherwise, he aborts the protocol.
(5) S uses the values of u^i and computes $q^i = (s_i.u^i) \oplus G(k_{s_i}^i) = (s_i.\mathbf{r}^{'}) \oplus t^i$, for all $1 \leq i \leq n$. Assume that, the matrix Q of order $(m+k) \times n$ is created by the sender S. Where q^i and q_j respectively denote the i-th column and j-th row of the matrix Q. Where q_j of the matrix can be expressed as $q_j = (r_j^{'}.s) \oplus t_j$, for all $1 \leq j \leq (m+k)$.
(6) The sender S sends (y_0^j, y_1^j) to R, where $y_0^j = x_0^j \oplus H(j, q_j)$ and $y_1^j = x_1^j \oplus H(j, q_j \oplus s)$, for all $1 \leq j \leq m$.
(7) The receiver R extracts its output $x_{r_j}^j = y_{r_j}^j \oplus H(j, t_j)$, for all $1 \leq j \leq m$.

Fig. 32. PRG-based OT extension protocol for the malicious receiver or ALSZ15 protocol [3].

under these parameters, they have followed the work of Reference [62]. Item 2 of the ALSZ15 protocol is also similar to item 2 of the ALSZ13 protocol, but the only difference is that the ALSZ15 protocol runs OT_k^n protocol while the ALSZ13 protocol executes OT_k^n. Moreover, in the ALSZ15 protocol, the receiver uses the values of k_0^i and generates a matrix of order $(m+k) \times n$; however, in the ALSZ13 protocol, the receiver creates a matrix of order $m \times k$. In item 3, the receiver first randomly chooses k bits string $\tau \in \{0,1\}^k$ and then appends his secret vector $\mathbf{r}$ to $\mathbf{r}^{'} = \mathbf{r}||\tau$. After that, he computes $u^i = t^i \oplus G(k_1^i) \oplus \mathbf{r}^{'}$, for all $1 \leq i \leq n$ and sends them to the sender. In item 3, if the receiver does not use the consistent value of $\mathbf{r}^{'}$ during every computation of u^i, then the receiver may extract the secret vector s of the sender, and he may use this secret vector to obtain the entire message of the sender. The sender executes a graph-based consistency check algorithm to force the receiver to use the consistent value of $\mathbf{r}^{'}$ during every computation of u^i. In the consistency check algorithm, they have considered that $\{u^i\}_{i=1}^n$ are the vertices of the complete graph, and the consistency check between two vertices has considered being an edge of the graph. For instance, we assume that $i = 3$, hence the vertices of the graph are u^1, u^2, and u^3, while the edges of the

graph are (u^1, u^2), (u^2, u^3), and (u^3, u^1). Since the values of u^i vary from 1 to n, therefore, n number of vertices are possible in the graph, so $n(n-1)/2$ pairs of consistency checks are possible. In item (a) of the consistency check algorithm, the receiver creates four tuples $(h_{0,0}^{a,b}, h_{0,1}^{a,b}, h_{1,0}^{a,b}, h_{1,1}^{a,b})$ for every pair of vertices and sends them to the receiver. The sender uses his secret vector s, the output of the OT_k^n protocol, and the received tuples and performs the check whether the receiver is consistent with the same value of $\mathbf{r}$ during every computation of u^i. In item (b) of this algorithm, the sender performs those tests. If all the tests are positive, then the sender continues the protocol; otherwise, he aborts. In item 5, the sender uses the received values of u^i and the secret vector s to compute the matrix Q of order $(m+k) \times n$. In item 6, the sender uses the jth row of the matrix Q and encrypts the value of x_0^j while he encrypts the value of x_1^j using secret vector s and jth row of the matrix Q. In the last item, the receiver uses his secret vector $\mathbf{r}$ and decrypts the following messages $x_{r_j}^j = y_{r_j}^j \oplus H(j, t_j)$, for all $1 \leq j \leq m$. Since the consistency check algorithm requires $O(n^2)$ checks in the ALSZ15 protocol, that increases the computation costs of the protocol. To reduce the computation cost, they further improved the consistency check algorithm. The improved algorithm reduces the consistency check from $O(n^2)$ to $O(n)$ [3].

3.1 Discussion on OT Extension Protocols

The first OT extension protocol has been proposed using a public-key encryption technique [5]. This technique is highly inefficient due to the involvement of one-way function. It offers security in the presence of semi-honest adversaries. To improve the efficiency of the OT extension protocol, a PRF-based OT extension protocol has been proposed [50]. It also offers security in the presence of semi-honest adversaries at high communication overhead but fails to provide security in the presence of malicious adversaries. They apply the cut-and-choose technique in the PRF-based OT extension protocol and offer security in the presence of malicious adversaries at high communication and computation costs [50]. A combination of PRP and AES encryption-based OT extension protocol has been proposed that offers security in the presence of semi-honest adversaries [46]. In this protocol, they execute base OT protocols parallelly to minimize the communication cost. The main drawback of this OT extension protocol was that it fails to provide security in the presence of malicious adversaries. An ABM Box-based OT extension protocol has been proposed that offers security in the presence of malicious adversaries using a consistency check algorithm [71]. In this protocol, they require three collision-resistant hash functions, which decrease the computational efficiency of the OT extension protocol. To overcome this problem, a commitment-based OT extension protocol has been proposed [44]. In this protocol, they execute a commitment phase to provide security in the presence of malicious adversaries. This commitment phase requires only one hash function that makes it computationally efficient. However, this protocol requires to execute a large number of base OT protocols that degrade the efficiency of this protocol. A combination of PRP and ROM-based OT extension protocol has been proposed to minimize the execution of base OT protocols [72]. This protocol also offers security in the presence of malicious adversaries. A PRG-based OT extension protocol has been proposed, and it requires to execute only a few numbers of base OT protocols[2]. This protocol offers security in the presence of semi-honest adversaries but fails to provide security in the presence of malicious adversaries. They apply a graph-based consistency check algorithm in the PRG-based OT extension protocol and offer security in the presence of malicious adversaries [3]. We summarized the security of OT extension protocols in Table 4. This figure shows that only protocols [2, 5, 46] do not provide security against the malicious adversaries while the rest of the protocols offer security against the malicious adversaries. We found that the OT extension protocols that offered security in the presence of semi-honest adversaries required low communication and computation cost; however, the protocols that offered security in the presence of malicious adversaries required high communication and computation cost.

Table 4. Summary of the OT Extension Protocols

	[5]	[50]	[46]	[71]	[44]	[2]	[3]
Semi-honest Adversaries	YES	YES	YES	YES	YES	YES	YES
Malicious Adversaries	NO	YES	NO	YES	YES	NO	YES
Used Technique	one-way function	cut-and-choose	PRP and AES	ABM Box	Commitment	PRG	Graph

4 PERFORMANCE AND SECURITY OF OT PROTOCOLS

In this section, we carry out a high-level comparative analysis of a different variant of OT protocols with respect to performance and security. We also discuss the advantages and disadvantages in terms of security and efficiency.

4.1 Security

The security of 1-out-of-2 OT protocols depend on public-key encryption, non-interactive proof system, DDH assumption, ROM, and pseudorandom function (KDF). We found that only DDH assumption based one-out-of-2 OT protocols are able to provide security in the presence of malicious adversaries. However, the rest of the one-out-of-2 OT protocols can only provide security in the presence of semi-honest adversaries. The security of one-out-of-n OT protocols depend on PRF, DDH assumption, and ROM. In this scenario, only the DDH assumption-based protocol can provide security in the presence of malicious adversaries. However, rest of the OT protocols are able to provide security in the presence of semi-honest adversaries only. Several k-out-of-n OT protocols have been proposed under the following security assumption, pseudorandom function, public-key encryption, DDH, CDH, oblivious pseudorandom function, and bilinear pairing. In this scenario, the only bilinear pairing, and oblivious pseudorandom function based OT protocols are able to provide security in the presence of malicious adversaries. However, public-key encryption, CDH, and pseudorandom function based k-out-of-n OT protocols are unable to provide security in the presence of malicious adversaries. Even DDH assumption-based OT protocol does not provide security in the presence of malicious adversaries. To transfer millions of oblivious messages, several OT extension protocols have been proposed. To transfer such messages, each OT extension protocol requires the execution of a few numbers of one-out-of-2 OT protocol and cheap symmetric key cryptography. OT extension protocols' security depends on one-way function, cut-and-choose, a combination of PRP and AES, ABM box, commitment, PRG, and graph. Some of the OT extension protocols such as one-way function, PRG, and combination PRP and AES are only secure in the presence of semi-honest adversaries. The rest of the OT extension protocols are secure in the presence of semi-honest and malicious adversaries.

4.2 Performance

We consider the performance of the OT protocol in terms of computation and communication costs. In other words, if a protocol requires high communication and computation costs, then we say that the performance of the protocol is low. In the case of one-out-of-2 OT protocol, ROM and PRF-based protocols performance are higher than that of DDH, because the DDH-based protocols require a large number of exponential operations than the ROM based protocol. However, the ROM and PRF-based protocols insecure against malicious adversaries. Similarly, in a one-out-of-n OT protocol scenario, we found that the PRF and ROM based protocols performed better than that of DDH based protocol. In terms of security, only DDH based OT protocols secure in the presence of malicious adversaries. In the case of k-out-of-n OT protocols, the following techniques like PRF, DDH, public-key encryption, and CDH-based scheme requires low communication and computation costs, but they fail to provide security against malicious adversaries. However, OPRF and BP-based schemes provide security against malicious adversaries, but they require high

computation costs like pairing and exponential operations. The next variety of OT protocol is OT extension protocol. The one-way function-based OT extension protocol neither efficient nor secure against malicious adversaries. However, the combination of PRP and AES-based OT extension protocol is efficient, but it only provides security against the semi-honest adversaries. The PRP-based OT extension protocol behaves similar to the combination of PRP and AES-based scheme. The rest of the OT extension protocols such as cut-and-choose, ABM box, commitment, PRP, and graph-based schemes provide security in the presence of malicious adversaries, but they require high communication and computation costs. Among all the maliciously secure OT extension protocols, the graph-based scheme is the most efficient OT extension scheme.

4.3 Advantages and Disadvantages

Advantages:

- OT protocol offers a wide range of one-out-of-2, one-out-of-n, k-out-of-n, and OT extension with varying tradeoffs between efficiency and security guarantees.
- Theoretical security guarantees in the presence of semi-honest and malicious adversaries of every OT protocols are well studied. However, the concrete security analysis of the OT protocols needs to be studied.
- The performance of each variety of OT protocols is well studied with respect to communication and computation costs. The OT protocol's main advantage is that it can transfer variable length of oblivious message efficiently as per the user's requirements. For example, if a user wants to send up to 1,024 bits message, then one-out-of-2 OT protocol is suitable. If the user wants to efficiently transfer millions of oblivious messages, then OT extension protocol is appropriate.

Disadvantages:

- The security of one-out-of-2, one-out-of-n, and k-out-of-n OT protocols depends on the following assumptions, such as: DDH, CDH, ROM, public-key encryption, and PRF. Every security assumption except lattice-based public-key encryption is based on the number-theoretic assumption. It is known that the number-theoretic assumption-based cryptographic protocols are vulnerable to the quantum attack.
- The lattice-based public-key encryption-based OT protocols are secure from the quantum attacks, but these protocols only provide security in the presence of semi-honest adversaries.
- The OT extension protocols' security depends on either ROM or PRG. Both ROM- and PRF-based cryptographic protocols also depend on number-theoretic assumption. Hence, both OT extension protocols are vulnerable to quantum attack.

5 CONCLUSION

We presented a detailed survey of different oblivious transfer protocols such as one-out-of-2, one-out-of-n, k-out-of-n, and OT extension. We evaluated the performance of each protocol in terms of communication and computation overhead. Moreover, we also analyzed each protocol's security features in terms of semi-honest adversaries, malicious adversaries, and universal composable. In one-out-of-2 OT protocol, we found that the PRG- and ROM-based protocols performed better than the DDH assumption-based protocols. However, in terms of security, only DDH-based protocols provided security in the presence of malicious adversaries. We found that the performance and security of one-out-of-n OT protocols behaved identically to the one-out-of-2 OT protocols. In k-out-of-n OT protocol, only OPRF and BP-based protocols provided security against malicious adversaries, but they required high communication and computation costs. However, the rest of

the DDH, CDH, PRF, and public-key encryption-based protocols required low costs, but they only provided security against semi-honest adversaries. In the OT extension protocol, only graph-based OT protocol efficiently provided security against malicious adversaries. However, the rest of the protocols either fail to provide security against malicious adversaries or require high communication and computation cost.

REFERENCES

[1] Nicolas Aragon, Olivier Blazy, Neals Fournaise, and Philippe Gaborit. 2020. CROOT: Code-based round-optimal oblivious transfer. In *Proceedings of the 17th International Joint Conference on E-Business and Telecommunications*. ScitePress, 76–85.

[2] Gilad Asharov, Yehuda Lindell, Thomas Schneider, and Michael Zohner. 2013. More efficient oblivious transfer and extensions for faster secure computation. In *Proceedings of the ACM SIGSAC Conference on Computer & Communications Security*. 535–548.

[3] Gilad Asharov, Yehuda Lindell, Thomas Schneider, and Michael Zohner. 2015. More efficient oblivious transfer extensions with security for malicious adversaries. In *Proceedings of the Annual International Conference on the Theory and Applications of Cryptographic Techniques*. Springer, 673–701.

[4] Donald Beaver. 1991. Foundations of secure interactive computing. In *Proceedings of the Annual International Cryptology Conference*. Springer, 377–391.

[5] Donald Beaver. 1996. Correlated pseudorandomness and the complexity of private computations. In *Proceedings of the 28th Annual ACM Symposium on Theory of Computing*. 479–488.

[6] Donald Beaver and Shaft Goldwasser. 1989. Multiparty computation with faulty majority. In *Proceedings of the Conference on the Theory and Application of Cryptology*. Springer, 589–590.

[7] Donald Beaver, Silvio Micali, and Phillip Rogaway. 1990. The round complexity of secure protocols. In *Proceedings of the 22nd Annual ACM Symposium on Theory of Computing*. 503–513.

[8] Mihir Bellare and Silvio Micali. 1989. Non-interactive oblivious transfer and applications. In *Proceedings of the Conference on the Theory and Application of Cryptology*. Springer, 547–557.

[9] Michael Ben-Or, Shafi Goldwasser, and Avi Wigderson. 2019. Completeness theorems for non-cryptographic fault-tolerant distributed computation. In *Providing Sound Foundations for Cryptography: On the Work of Shafi Goldwasser and Silvio Micali*. 351–371.

[10] Bo Bi, Darong Huang, Bo Mi, Zhenping Deng, and Hongyang Pan. 2019. Efficient LBS security-preserving based on NTRU oblivious transfer. *Wirel. Person. Commun.* 108, 4 (2019), 2663–2674.

[11] Eli Biham and Adi Shamir. 2012. *Differential Cryptanalysis of the Data Encryption Standard*. Springer Science & Business Media.

[12] Dan Boneh, Ben Lynn, and Hovav Shacham. 2001. Short signatures from the Weil pairing. In *Proceedings of the International Conference on the Theory and Application of Cryptology and Information Security*. Springer, 514–532.

[13] Ran Canetti. 2000. Security and composition of multiparty cryptographic protocols. *J. Cryptol.* 13, 1 (2000), 143–202.

[14] Ran Canetti and Amir Herzberg. 1994. Maintaining security in the presence of transient faults. In *Proceedings of the Annual International Cryptology Conference*. Springer, 425–438.

[15] Yan-Cheng Chang. 2004. Single database private information retrieval with logarithmic communication. In *Proceedings of the Australasian Conference on Information Security and Privacy*. Springer, 50–61.

[16] David Chaum, Claude Crépeau, and Ivan Damgard. 1988. Multiparty unconditionally secure protocols. In *Proceedings of the 20th Annual ACM Symposium on Theory of Computing*. 11–19.

[17] Jue-Sam Chou. 2012. A novel k-out-of-n oblivious transfer protocol from bilinear pairing. *Adv. Multim.* 2012 (2012).

[18] Cheng-Kang Chu and Wen-Guey Tzeng. 2005. Efficient k-out-of-n oblivious transfer schemes with adaptive and non-adaptive queries. In *Proceedings of the International Workshop on Public Key Cryptography*. Springer, 172–183.

[19] Cheng-Kang Chu, Wen-Guey Tzeng, et al. 2008. Efficient k-out-of-n oblivious transfer schemes. *J. UCS* 14, 3 (2008), 397–415.

[20] Michele Ciampi and Claudio Orlandi. 2018. Combining private set-intersection with secure two-party computation. In *Proceedings of the International Conference on Security and Cryptography for Networks*. Springer, 464–482.

[21] Richard Cleve. 1986. Limits on the security of coin flips when half the processors are faulty. In *Proceedings of the 18th Annual ACM Symposium on Theory of Computing*. 364–369.

[22] Ronald Cramer and Victor Shoup. 1998. A practical public key cryptosystem provably secure against adaptive chosen ciphertext attack. In *Proceedings of the Annual International Cryptology Conference*. Springer, 13–25.

[23] Claude Crépeau. 1987. Equivalence between two flavours of oblivious transfers. In *Proceedings of the Conference on the Theory and Application of Cryptographic Techniques*. Springer, 350–354.

[24] Ivan Damgård, Rasmus Lauritsen, and Tomas Toft. 2014. An empirical study and some improvements of the MiniMac protocol for secure computation. In *Proceedings of the International Conference on Security and Cryptography for Networks*. Springer, 398–415.
[25] Ivan Damgård and Sarah Zakarias. 2013. Constant-overhead secure computation of Boolean circuits using preprocessing. In *Proceedings of the Theory of Cryptography Conference*. Springer, 621–641.
[26] Emiliano De Cristofaro and Gene Tsudik. 2010. Practical private set intersection protocols with linear complexity. In *Proceedings of the International Conference on Financial Cryptography and Data Security*. Springer, 143–159.
[27] Giovanni Di Crescenzo, Tal Malkin, and Rafail Ostrovsky. 2000. Single database private information retrieval implies oblivious transfer. In *Proceedings of the International Conference on the Theory and Applications of Cryptographic Techniques*. Springer, 122–138.
[28] Whitfield Diffie and Martin Hellman. 1976. New directions in cryptography. *IEEE Trans. Inf. Theor.* 22, 6 (1976), 644–654.
[29] Changyu Dong, Liqun Chen, and Zikai Wen. 2013. When private set intersection meets big data: An efficient and scalable protocol. In *Proceedings of the ACM SIGSAC Conference on Computer & Communications Security*. 789–800.
[30] Nico Döttling, Sanjam Garg, Mohammad Hajiabadi, Daniel Masny, and Daniel Wichs. 2020. Two-round oblivious transfer from CDH or LPN. *Adv. Cryptol.–EUROCRYPT 2020* 12106 (2020), 768.
[31] Jan-Olof Eklundh. 1972. A fast computer method for matrix transposing. *IEEE Trans. Comput.* 100, 7 (1972), 801–803.
[32] Taher ElGamal. 1985. A public key cryptosystem and a signature scheme based on discrete logarithms. *IEEE Trans. Inf. Theor.* 31, 4 (1985), 469–472.
[33] Shimon Even, Oded Goldreich, and Abraham Lempel. 1985. A randomized protocol for signing contracts. *Commun. ACM* 28, 6 (1985), 637–647.
[34] Uriel Feige and Adi Shamir. 1989. Zero knowledge proofs of knowledge in two rounds. In *Proceedings of the Conference on the Theory and Application of Cryptology*. Springer, 526–544.
[35] Zvi Galil, Stuart Haber, and Moti Yung. 1987. Cryptographic computation: Secure fault-tolerant protocols and the public-key model. In *Proceedings of the Conference on the Theory and Application of Cryptographic Techniques*. Springer, 135–155.
[36] Oded Goldreich. 2002. Concurrent zero-knowledge with timing, revisited. In *Proceedings of the 34th Annual ACM Symposium on Theory of Computing*. 332–340.
[37] Oded Goldreich. 2007. *Foundations of Cryptography: Volume 1, Basic Tools*. Cambridge University Press.
[38] Oded Goldreich. 2009. *Foundations of Cryptography: Volume 2, Basic Applications*. Cambridge University Press.
[39] Oded Goldreich, Silvio Micali, and Avi Wigderson. 2019. How to play any mental game, or a completeness theorem for protocols with honest majority. In *Providing Sound Foundations for Cryptography: On the Work of Shafi Goldwasser and Silvio Micali*. 307–328.
[40] Shafi Goldwasser and Leonid Levin. 1990. Fair computation of general functions in presence of immoral majority. In *Proceedings of the Conference on the Theory and Application of Cryptography*. Springer, 77–93.
[41] Shafi Goldwasser, Silvio Micali, and Ronald L. Rivest. 1988. A digital signature scheme secure against adaptive chosen-message attacks. *SIAM J. Comput.* 17, 2 (1988), 281–308.
[42] S. Dov Gordon, Carmit Hazay, Jonathan Katz, and Yehuda Lindell. 2011. Complete fairness in secure two-party computation. *J. ACM* 58, 6 (2011), 1–37.
[43] S. Dov Gordon and Jonathan Katz. 2010. Partial fairness in secure two-party computation. In *Proceedings of the Annual International Conference on the Theory and Applications of Cryptographic Techniques*. Springer, 157–176.
[44] Danny Harnik, Yuval Ishai, Eyal Kushilevitz, and Jesper Buus Nielsen. 2008. OT-combiners via secure computation. In *Proceedings of the Theory of Cryptography Conference*. Springer, 393–411.
[45] Carmit Hazay and Yehuda Lindell. 2010. *Efficient Secure Two-party Protocols: Techniques and Constructions*. Springer Science & Business Media.
[46] Wilko Henecka and Thomas Schneider. 2013. Faster secure two-party computation with less memory. In *Proceedings of the 8th ACM SIGSAC Symposium on Information, Computer and Communications Security*. 437–446.
[47] Jeffrey Hoffstein, Jill Pipher, and Joseph H. Silverman. 1998. NTRU: A ring-based public key cryptosystem. In *Proceedings of the International Algorithmic Number Theory Symposium*. Springer, 267–288.
[48] Yan Huang, David Evans, and Jonathan Katz. 2012. Private set intersection: Are garbled circuits better than custom protocols? In *Proceedings of the Network and Distributed Security Symposium*.
[49] Yan Huang, David Evans, Jonathan Katz, and Lior Malka. 2011. Faster secure two-party computation using garbled circuits. In *Proceedings of the USENIX Security Symposium*, Vol. 201. 331–335.
[50] Yuval Ishai, Joe Kilian, Kobbi Nissim, and Erez Petrank. 2003. Extending oblivious transfers efficiently. In *Proceedings of the Annual International Cryptology Conference*. Springer, 145–161.
[51] Hoda Jannati and Behnam Bahrak. 2017. An oblivious transfer protocol based on Elgamal encryption for preserving location privacy. *Wirel. Person. Commun.* 97, 2 (2017), 3113–3123.

[52] Stanisław Jarecki and Xiaomin Liu. 2009. Efficient oblivious pseudorandom function with applications to adaptive OT and secure computation of set intersection. In *Proceedings of the Theory of Cryptography Conference*. Springer, 577–594.
[53] Daemen Joan and Rijmen Vincent. 2002. The design of Rijndael: AES-the advanced encryption standard. In *Information Security and Cryptography*. Springer.
[54] Aggelos Kiayias and Moti Yung. 2001. Secure games with polynomial expressions. In *Proceedings of the International Colloquium on Automata, Languages, and Programming*. Springer, 939–950.
[55] Vladimir Kolesnikov, Ranjit Kumaresan, Mike Rosulek, and Ni Trieu. 2016. Efficient batched oblivious PRF with applications to private set intersection. In *Proceedings of the ACM SIGSAC Conference on Computer and Communications Security*. 818–829.
[56] Vladimir Kolesnikov, Payman Mohassel, and Mike Rosulek. 2014. FleXOR: Flexible garbling for XOR gates that beats free-XOR. In *Proceedings of the Annual Cryptology Conference*. Springer, 440–457.
[57] Vladimir Kolesnikov and Thomas Schneider. 2008. Improved garbled circuit: Free XOR gates and applications. In *Proceedings of the International Colloquium on Automata, Languages, and Programming*. Springer, 486–498.
[58] Kaoru Kurosawa and Wakaha Ogata. 1999. Efficient Rabin-type digital signature scheme. *Des., Codes Cryptog.* 16, 1 (1999), 53–64.
[59] Yi-Fu Lai, Steven D. Galbraith, and Cyprien Delpech de Saint Guilhem. 2021. Compact, efficient and UC-secure isogeny-based oblivious transfer. In *Proceedings of the Annual International Conference on the Theory and Applications of Cryptographic Techniques*. Springer, 213–241.
[60] Enrique Larraia, Emmanuela Orsini, and Nigel P. Smart. 2014. Dishonest majority multi-party computation for binary circuits. In *Proceedings of the Annual Cryptology Conference*. Springer, 495–512.
[61] Zengpeng Li, Can Xiang, and Chengyu Wang. 2018. Oblivious transfer via lossy encryption from lattice-based cryptography. *Wirel. Commun. Mob. Comput.* 2018 (2018).
[62] Yehuda Lindell and Benny Pinkas. 2012. Secure two-party computation via cut-and-choose oblivious transfer. *J. Cryptol.* 25, 4 (2012), 680–722.
[63] Momeng Liu and Yupu Hu. 2019. Universally composable oblivious transfer from ideal lattice. *Front. Comput. Sci.* 13, 4 (2019), 879–906.
[64] Momeng Liu and Yu-Pu Hu. 2017. Equational security of a lattice-based oblivious transfer protocol. *J. Netw. Intell.* 2, 3 (2017), 231–249.
[65] Mo-Meng Liu, Juliane Krämer, Yu-Pu Hu, and Johannes Buchmann. 2017. Quantum security analysis of a lattice-based oblivious transfer protocol. *Front. Inf. Technol. Electron. Eng.* 18, 9 (2017), 1348–1369.
[66] Silvio Micali and Phillip Rogaway. 1991. Secure computation. In *Proceedings of the Annual International Cryptology Conference*. Springer, 392–404.
[67] Yi Mu, Junqi Zhang, and Vijay Varadharajan. 2002. m out of n oblivious transfer. In *Proceedings of the Australasian Conference on Information Security and Privacy*. Springer, 395–405.
[68] Moni Naor and Benny Pinkas. 1999. Oblivious transfer and polynomial evaluation. In *Proceedings of the 31st Annual ACM Symposium on Theory of Computing*. 245–254.
[69] Moni Naor and Benny Pinkas. 2001. Efficient oblivious transfer protocols. In *Proceedings of the 12th Annual ACM-SIAM Symposium on Discrete Algorithms*, Vol. 1. 448–457.
[70] Moni Naor, Benny Pinkas, and Reuban Sumner. 1999. Privacy preserving auctions and mechanism design. In *Proceedings of the 1st ACM Conference on Electronic Commerce*. 129–139.
[71] Jesper Buus Nielsen. 2007. Extending oblivious transfers efficiently-How to get robustness almost for free. *IACR Cryptol. ePrint Arch.* 2007 (2007), 215.
[72] Jesper Buus Nielsen, Peter Sebastian Nordholt, Claudio Orlandi, and Sai Sheshank Burra. 2012. A new approach to practical active-secure two-party computation. In *Proceedings of the Annual Cryptology Conference*. Springer, 681–700.
[73] Michele Orrù, Emmanuela Orsini, and Peter Scholl. 2017. Actively secure 1-out-of-n OT extension with application to private set intersection. In *Proceedings of the Cryptographers' Track at the RSA Conference*. Springer, 381–396.
[74] Rafail Ostrovsky and Moti Yung. 1991. How to withstand mobile virus attacks. In *Proceedings of the 10th Annual ACM Symposium on Principles of Distributed Computing*. 51–59.
[75] Pascal Paillier and David Pointcheval. 1999. Efficient public-key cryptosystems provably secure against active adversaries. In *Proceedings of the International Conference on the Theory and Application of Cryptology and Information Security*. Springer, 165–179.
[76] Russell Paulet, Md Golam Kaosar, Xun Yi, and Elisa Bertino. 2013. Privacy-preserving and content-protecting location based queries. *IEEE Trans. Knowl. Data Eng.* 26, 5 (2013), 1200–1210.
[77] Chris Peikert, Vinod Vaikuntanathan, and Brent Waters. 2008. A framework for efficient and composable oblivious transfer. In *Proceedings of the Annual International Cryptology Conference*. Springer, 554–571.

[78] Benny Pinkas, Mike Rosulek, Ni Trieu, and Avishay Yanai. 2019. Spot-light: Lightweight private set intersection from sparse OT extension. In *Proceedings of the Annual International Cryptology Conference*. Springer, 401–431.
[79] Benny Pinkas, Thomas Schneider, Nigel P. Smart, and Stephen C. Williams. 2009. Secure two-party computation is practical. In *Proceedings of the International Conference on the Theory and Application of Cryptology and Information Security*. Springer, 250–267.
[80] Benny Pinkas, Thomas Schneider, and Michael Zohner. 2014. Faster private set intersection based on OT extension. In *Proceedings of the 23rd USENIX Security Symposium (USENIX Security'14)*. 797–812.
[81] Benny Pinkas, Thomas Schneider, and Michael Zohner. 2018. Scalable private set intersection based on OT extension. *ACM Trans. Priv. Secur.* 21, 2 (2018), 1–35.
[82] Michael O. Rabin. 2005. How to exchange secrets with oblivious transfer. *IACR Cryptol. ePrint Arch.* 2005, 187 (2005).
[83] Zulfikar Amin Ramzan and Craig B. Gentry. 2009. Method and Apparatus for Communication Efficient Private Information Retrieval and Oblivious Transfer. US Patent 7,620,625.
[84] R. P. Ratnadewi, Y. Hutama Adhie, A. Saleh Ahmar, and M. I. Setiawan. 2018. Implementation cryptography data encryption standard (DES) and triple data encryption standard (3DES) method in communication system based near field communication (NFC). In *J. Phys. Conf. Ser*, Vol. 954. 12009.
[85] Ronald L. Rivest, Adi Shamir, and Leonard Adleman. 1978. A method for obtaining digital signatures and public-key cryptosystems. *Commun. ACM* 21, 2 (1978), 120–126.
[86] Wen-Guey Tzeng. 2002. Efficient 1-out-n oblivious transfer schemes. In *Proceedings of the International Workshop on Public Key Cryptography*. Springer, 159–171.
[87] Wen-Guey Tzeng. 2004. Efficient 1-out-of-n oblivious transfer schemes with universally usable parameters. *IEEE Trans. Comput.* 53, 2 (2004), 232–240.
[88] Vijay Kumar Yadav, Shekhar Verma, and S. Venkatesan. 2020. Efficient and secure location-based services scheme in VANET. *IEEE Trans. Vehic. Technol.* 69, 11 (2020), 13567–13578.
[89] Vijay Kumar Yadav, Shekhar Verma, and S. Venkatesan. 2021. An efficient and light weight polynomial multiplication for ideal lattice-based cryptography. *Multim. Tools Applic.* 80, 2 (2021), 3089–3120.
[90] Vijay Kumar Yadav, Shekhar Verma, and Subramanian Venkatesan. 2021. Linkable privacy-preserving scheme for location-based services. *IEEE Trans. Intell. Transport. Syst.* (2021).
[91] Andrew Chi-Chih Yao. 1986. How to generate and exchange secrets. In *Proceedings of the 27th Annual Symposium on Foundations of Computer Science (SFCS'86)*. IEEE, 162–167.
[92] Samee Zahur, Mike Rosulek, and David Evans. 2015. Two halves make a whole. In *Proceedings of the Annual International Conference on the Theory and Applications of Cryptographic Techniques*. Springer, 220–250.

Received March 2021; revised October 2021; accepted November 2021