Efficient Oblivious Transfer from Lossy Threshold Homomorphic Encryption

Isheeta Nargis

Department of Computer Science,
University Of Calgary, Calgary, Canada.
Email Address: inargis@ucalgary.ca.

No Institute Given

Abstract. In this article, a new oblivious transfer (OT) protocol, secure in the presence of erasure-free one-sided active adaptive adversaries is presented. The new bit OT protocol achieves better communication complexity than the existing bit OT protocol in this setting. The new bit OT protocol requires fewer number of public key encryption operations than the existing bit OT protocol in this setting. As a building block, a new two-party lossy threshold homomorphic public key cryptosystem is designed. It is secure in the same adversary model. It is of independent interest.

1 Introduction

Oblivious transfer (OT) is a fundamental primitive of cryptography. An 1-out-of-2 bit OT is a protocol between two parties. The input of the sender S is a pair of bits $\{x_0, x_1\}$, and the input of the receiver R is a choice bit σ. The goal is that R learns only x_σ without learning $x_{1-\sigma}$, and S remains oblivious to which bit was requested. OT is 'complete' for secure multiparty computation in the sense that if an implementation of OT is given, then it is possible to securely evaluate any polynomial time computable function, without any additional primitive [23]. Besides multiparty computation, OT can be used in private information retrieval, privacy preserving data mining, signing contract protocols, randomized coin flipping protocols, and certified email transfer protocols.

In the *passive adversary model*, the corrupted parties try to learn the inputs and outputs of the honest parties but still follow the protocol. In the *active adversary model*, the corrupted parties may behave in any possible way, including the violation of the protocol. The active adversary model portrays the real world better. In the *covert adversary model*, a corrupted party may cheat in an arbitrary way but it is guaranteed to get caught with a fixed probability. This probability is called the *deterrence factor* of the model.

In the *static adversary model*, the adversary fixes the set of parties to corrupt before the protocol starts, and this set remains fixed throughout the execution of the protocol. In the *adaptive adversary model*, the adversary may corrupt a party at any time of the protocol, and even after the protocol has finished its execution. The adaptive adversary model captures the real-world scenario better than the static adversary model. In the *adaptive adversary model with erasure*, it is assumed that parties can erase some of its local data and randomness. In the *erasure-free adaptive adversary model*, it is assumed that the adversary can see all history of a party when it corrupts that party. Assuming erasure is unrealistic as complete erasure is sometimes impossible to achieve. Erasure is a property of a party that cannot be verified in any way by another party.

Achieving adaptive security for OT is significantly harder than achieving static security for OT [25]. Adaptively secure OT protocol is an important building block in the design of two-party computation protocols secure against adaptive adversaries.

In the common reference string (CRS) model, it is assumed that all parties have access to a common string that is selected from some specified distribution.

The *communication complexity* of a protocol denotes the total size of all the message transferred between the two parties during the protocol. Sometimes the number of public key encryption (PKE) operations is used as an estimate on the computational complexity of a protocol, since the PKE operations constitute the main portion of the computational complexity of a protocol.

"Opening a ciphertext" means supplying the plaintext and randomness used during the generation of that ciphertext. In a traditional cryptosystem, encryption is binding; meaning that a given ciphertext can be opened to only the original plaintext with which it was created. In a *lossy public key encryption (PKE) scheme*, there are two modes of operation – injective mode and lossy mode. In the *injective mode*, the encryption operation is binding. In the *lossy mode*, if the private key is known, then a ciphertext can be opened to any plaintext of choice.

In a *two-party threshold PKE scheme*, parties jointly generate a key pair and the private key is shared between the parties. The parties can encrypt alone but the participation of both parties is necessary for decrypting any ciphertext.

In *one-sided active adaptive adversary model* for two-party computation, it is assumed that the adversary is active, adaptive and it can corrupt at most one party [21]. This is a relaxation from the standard adaptive adversary model for two-party computation, where the adversary can corrupt both parties. This relaxed model is used to achieve more efficient protocols. Let n denote the security parameter. Garay et al. [20] designed the most efficient OT protocol secure against active adaptive adversaries. For string OT of size q bits, their protocol requires $O(q)$ PKE operations in the worst case. Here, q is a polynomial of n. Hazay and Patra [21] designed an OT protocol for one-sided active adaptive adversary model. For string OT of size q bits, their protocol requires constant number of PKE operations in the expected case. So, relaxing the notion of security has resulted in a protocol requiring significantly smaller number of PKE operations, in the expected case. In the *erasure-free adaptive adversary model*, it is assumed that the adversary can see all history of a party when it corrupts that party. Assuming erasure is unrealistic as complete erasure is sometimes impossible to achieve.

Hazay and Patra [21] designed an OT protocol for one-sided active adaptive adversary model. The OT protocol of [21] requires $O(n^2)$ communication complexity for bit OT. One research goal is to improve the communication complexity in this setting.

1.1 Contribution of this Article

In this article, a new OT protocol secure against erasure-free one-sided active adaptive adversaries is presented. The new bit OT protocol needs $O(n)$ communication complexity, which is a significant improvement over the $O(n^2)$ communication complexity of the bit OT protocol of [21]. The bit OT protocol of [21] requires $O(n)$ PKE operations in the worst case, and the new bit OT protocol needs a constant number of PKE operations in the worst case. The OT protocol of [21] is secure in the universally composable (UC) framework. The new OT protocol is secure according to the simulation-based security definition of Canetti [9], which satisfies sequential composition.
As a building block, a new two-party lossy threshold homomorphic PKE scheme is designed. It is secure against erasure-free one-sided active adaptive adversaries. This encryption scheme is of independent interest. It can be used in other two-party protocols.

1.2 Techniques

Aumann and Lindell [2] designed an OT protocol secure against covert adversaries. Their OT protocol is secure against static adversaries. The new OT protocol is designed by converting their OT protocol. It is secure against erasure-free one-sided active adaptive adversaries. The new OT protocol achieves a much stronger notion of security than the OT protocol of [2] in two senses. Firstly, the active adversary model is a stronger security model than the covert adversary model [2]. Secondly, the adaptive adversary model is more secure than the static adversary model [9]. The OT protocol of [2] is modified in two ways, to obtain the new OT protocol. The protocol of [2] uses a traditional homomorphic PKE scheme and the new OT protocol uses a two-party lossy threshold homomorphic PKE scheme. For verification, the protocol of [2] uses cut-and-choose technique and the new OT protocol uses adaptive zero-knowledge arguments.
The reason for using these tools in the new OT protocol is described below.
In the protocol of [2], a traditional homomorphic encryption scheme suffices. As the adversary is static, the adversary selects the party to corrupt before the protocol starts. If a party is corrupted, the simulator gets its input and prepares its ciphertexts based on its input.
In the new OT protocol, the adaptive adversary can corrupt any party at any time. Whenever a party gets corrupted, the simulator has to supply a view that is consistent with the actual input of that party. A lossy encryption scheme is used to cope with adaptive adversaries. In the protocol, the parties work in the injective mode. The lossy mode is used only by the simulator. The simulator supplies ciphertexts based on dummy inputs for the honest parties. If an honest

party is corrupted after the simulator already sent ciphertexts on behalf of that party, the simulator can open those ciphertexts to the plaintexts that corresponds to the true input of the corrupted party, due to the lossy mode.

The property of a threshold encryption scheme ensures that the private key is not fully disclosed to a single party. The reason behind this is that from the key pair (which consists of the public key and the private key) of a lossy encryption scheme, it can be efficiently detected whether the key pair is an injective pair or a lossy key pair. If a non-threshold lossy encryption scheme is used, then an adversary controlling one party can detect whether lossy key pair or injective key pair is used, and thereby distinguish between the ideal world and the real world.

To prevent this, a combination of lossy encryption scheme and threshold encryption scheme is needed. The encryption scheme has to be secure against erasure-free one-sided active adaptive adversaries. Currently there exists no such encryption scheme. So, a new lossy threshold homomorphic PKE scheme $ELTA2E$ is designed as a building block.

In covert adversary model, cut-and-choose technique is sufficient for verification since the any attempt to cheat can go undetected with a probability of $(1-\epsilon)$. In the active adversary model, any attempt to cheat must be detected with probability very close to one. To achieve security against active adaptive adversaries, adaptive zero-knowledge arguments are used.

2 Background

2.1 Notation

Let n denote the security parameter. Let $\mathbb{Z}_q = \{0, 1, \ldots, q-1\}$ where q is a prime. Let $\mathbb{Z}_q^* = \{1, 2, \ldots, q-1\}$. For all elements a and $b \neq 1$ in group $\mathbb{G}$, the discrete logarithm of a in base b is denoted by $\log_b(a)$. For a set R, let $r \overset{\$}{\leftarrow} R$ denote that r is selected uniformly at random from R. Let A be a probabilistic polynomial-time algorithm. Let $coins(A)$ denote the distribution of the internal randomness of A. $y \leftarrow A(x)$ means that y is computed by running A on input x and randomness r where $r \overset{\$}{\leftarrow} coins(A)$. Let $E_{pk}(m, r)$ denote the result of encryption of plaintext m using encryption key pk and randomness r. Let $D_{sk}(c)$ denote the result of decryption of ciphertext c using decryption key sk. Let $Com_\mu(a, r)$ denote the commitment of secret a using commitment key μ and randomness r.

2.2 The DDH Assumption

The decisional Diffie-Hellmann (DDH) assumption for cyclic group $\mathbb{G}$ of order prime q says that, for random generator $g \in \mathbb{G}^*$ ($\mathbb{G}^*$ denotes the generators of $\mathbb{G}$), the tuples (g, g^a, g^b, g^{ab}) and (g, g^a, g^b, g^c) are computationally indistinguishable, where $a, b, c \overset{\$}{\leftarrow} \mathbb{Z}_q$.

2.3 Trapdoor Commitment Scheme

A *trapdoor commitment scheme* is a commitment scheme such that a trapdoor is generated during the key generation. With the trapdoor, one can efficiently compute a randomness to open a given commitment to any value of choice. Without the trapdoor, the binding property of the commitment scheme holds. Pedersen [28] designed a trapdoor commitment scheme based on the DDH assumption. In Pedersen's commitment scheme, the commitment key is $\mu = g^{\delta}$, and δ is the trapdoor.

2.4 Adaptive Zero-Knowledge Arguments and Σ-Protocols

An interactive proof is a protocol between two parties. One party is called the *prover* and the other party is called the *verifier*. There is a common input of an interactive proof. The objective of an interactive proof is that the prover can convince the verifier that a mathematical statement is true, without disclosing the prover's secret. Adaptive zero-knowledge argument is a type of interactive proof. Σ-protocol is another type of interactive proof. Non-erasure Σ-protocols are Σ-protocols with a special property called *non-erasure*. The non-erasure property is necessary when a Σ-protocol is used in a multiparty protocol secure against erasure-free adaptive adversaries. The adaptive zero-knowledge arguments used in this article are designed by converting non-erasure Σ-protocols into adaptive zero-knowledge arguments. The adaptive zero-knowledge arguments designed in this way are communication efficient.
The definition of zero-knowledge argument, Σ-protocol, non-erasure Σ-protocol and proof of knowledge are presented in AppendixAppDefn.

2.5 Converting Σ-Protocol to Adaptive Zero-Knowledge Argument

Damgård [15] described how to convert a Σ-protocol for a given relation to a zero-knowledge proof for the same relation. Damgård described this conversion in the common reference string model. The conversion needs a trapdoor commitment scheme. The commitment key μ is used as the common reference string. By definition, a Σ-protocol is an honest-verifier zero-knowledge proof. The honesty required from the verifier is that it generates its challenge e, irrespective of the first message a of the prover. The commitment scheme is used to ensure this honesty, thereby converting to zero-knowledge. In the first step, the prover computes its first message a in the original Σ-protocol, selects a string $r_c \overset{\$}{\leftarrow} coins(Com)$, computes a commitment c of its first message a using randomness r_c, and then sends the commitment c to the verifier. The verifier selects its challenge $e \overset{\$}{\leftarrow} \{0,1\}^t$, and then sends e to the prover. Since a is hidden by the hiding property of the commitment scheme, the verifier cannot generate its challenge e based on the value of a. Then, the prover computes its second message z of the original Σ-protocol, and sends a, z, and the randomness r_c used in the commitment, to the verifier. The verifier checks that (a, e, z) is an

Adaptive Zero-Knowledge Argument Π^{R}_{AdZKA} implementing functionality $\mathcal{F}^{R}_{ZK}$.
The protocol Π^{R}_{AdZKA} runs with two parties – the prover P and the verifier V. P acts as the requester of the key of V for a trapdoor commitment scheme Com. The common reference string μ is the commitment key and δ is the trapdoor.

Common Reference String : $\mu \in \mathbb{Z}_q$.
Original Σ-Protocol for Relation R : $\Pi_{\Sigma R}$.
Security Parameter : k.
Inputs :

1. **Common Input :** x.
2. **Witness of P :** w.

1. On input (x, w), P computes its first message a of the Σ-protocol $\Pi_{\Sigma R}$.
   P selects $r_c \overset{\$}{\leftarrow} coins(Com)$, and computes $c = Com_{\mu}(a, r_c)$.
   P sends c to V.
2. V selects its challenge $e \overset{\$}{\leftarrow} \{0, 1\}^t$.
3. P computes its second message z for challenge e for the Σ-protocol $\Pi_{\Sigma R}$.
   P sends (z, a, r_c) to V.
   V accepts if and only if (a, e, z) is an accepting conversation for the original Σ-protocol $\Pi_{\Sigma R}$ and $c = Com_{\mu}(a, r_c)$.

Fig. 1: Adaptive Zero-Knowledge Argument designed from a Σ-protocol [15,27].

accepting conversation for the original Σ-protocol, and also that committing a using randomness r_c yields c. The full protocol is given in Figure 1.
The trapdoor property of the commitment is needed for simulating the conversation. The simulator generates a commitment key along with its trapdoor. In order to ensure security against an adaptive adversary, it is maintained that the simulator uses the trapdoor to generate randomness only if the verifier is corrupted [27]. If the prover is corrupted after sending the commitment c, the simulator uses the trapdoor to compute randomness r_a such that everything is consistent. In this way, it is secure against an adaptive adversary. The security proof of this type of zero-knowledge proof against an adaptive adversary can be found in [[27],Chapter 5].
In the zero-knowledge proof systems where the prover commits in the first step and later opens the commitment, the perfect soundness condition holds only if the commitment scheme is perfectly binding. If the zero-knowledge proof has to be secure against adaptive adversaries, then the simulator has to open commitments to any possible value for the case where the adversary corrupts verifier at start and prover at the end. It is possible using a trapdoor commitment scheme. However, in that case, the soundness condition of the zero-knowledge proof is violated; since a computationally unbounded prover can find the trapdoor and use that trapdoor to make the verifier accept a false claim. That means, when a trapdoor commitment scheme is used in a zero-knowledge proof, it only achieves

computational soundness; that is, a polynomial-time prover cannot make the verifier accept a false claim. By definition, the resulting system is a zero-knowledge argument.

The security proof of Π^R_{AdZKA} is given in Appendix B.

2.6 Additive Homomorphic PKE Scheme

In an *additive homomorphic PKE scheme*, one can efficiently compute an encryption c of (m_1+m_2) from ciphertexts c_1 and c_2 encrypting plaintexts m_1 and m_2, respectively. This is called *homomorphic addition* and denoted by $c = c_1 +_h m_2$. In an additive homomorphic PKE scheme, one can also efficiently compute an encryption c_2 of $(m_1 \times m_2)$ from an encryption c_1 of m_1 and the plaintext m_2. This is called *homomorphic multiplication by constant*, and denoted by $c_2 = m_2 \times_h c_1$.

2.7 Randomizable PKE Scheme

In a *randomizable PKE scheme*, there exists a probabilistic polynomial-time algorithm *Blind*, which, on input public key pk and an encryption c of plaintext m, produces another encryption c_1 of plaintext m such that c_1 is distributed identically to $E_{pk}(m, r)$ where $r \overset{\$}{\leftarrow} Coins(E)$.

2.8 Lossy Encryption Scheme

Bellare et al. [6] defined *lossy encryption schemes*, extending the definition of *dual-mode encryption schemes* by Peikert et al. [29], and the definition of *meaningful/meaningless encryption schemes* by Kol and Naor [24].

Definition 1. *(Lossy PKE Scheme [6])*
A ***lossy PKE scheme*** *is a tuple (G, E, D) of probabilistic polynomial time algorithms such that keys generated by $G(1^k, 1)$ and $G(1^k, 0)$ are called injective keys and lossy keys,respectively. The algorithms must satisfy the following properties.*

1. ***Correctness on Injective Keys:*** *For all plaintexts m,*

$$Pr\Big[(pk, sk) \leftarrow G(1^k, 1); r \overset{\$}{\leftarrow} coins(E) : D_{sk}\big(E_{pk}(m, r)\big) = m\Big] = 1.$$

2. ***Indistinguishability of Keys:*** *The lossy public keys are computationally indistinguishable from the injective public keys. If $proj : (pk, sk) \to pk$ is the projection map, then $\{proj(KG(1^k, 1))\} \overset{c}{\equiv} \{proj(KG(1^k, 0))\}$.*
3. ***Lossiness on Lossy Keys:*** *If $(pk_\ell, sk_\ell) \leftarrow G(1^k, 0)$, then, for all m_0, m_1, the distributions $E_{pk_\ell}(m_0, R)$ and $E_{pk_\ell}(m_1, R)$ are statistically indistinguishable.*
4. ***Openability:*** *If $(pk_\ell, sk_\ell) \leftarrow G(1^k, 0)$ and $r_0 \overset{\$}{\leftarrow} coins(E)$, then, for all m_0, m_1, with overwhelming probability, there exists $r_1 \in coins(E)$ such that $E_{pk_\ell}(m_0, r_0) = E_{pk_\ell}(m_1, r_1)$. That is, there exists a (possibly inefficient) algorithm Opener that can open a lossy ciphertext to any arbitrary plaintext with all but negligible probability.*

The semantic security of a lossy encryption scheme is implied by definition [6]. For a lossy key pair, the semantic security follows from the fact that the encryptions of two different plaintexts are statistically indisinguishable by the "lossiness on lossy keys" property. For an injective key pair, the semantic security follows from the "indistinguishability of keys" property and "lossiness on lossy keys" property, as follows.
For any m_0, m_1,

$$\begin{aligned} E_{proj(G(1^k,1))}(m_0, R) &\overset{c}{\equiv} E_{proj(G(1^k,0))}(m_0, R) \overset{s}{\equiv} E_{proj(G(1^k,0))}(m_1, R) \\ &\overset{c}{\equiv} E_{proj(G(1^k,1))}(m_1, R). \end{aligned}$$

Here, $\overset{c}{\equiv}$ denotes computational indistinguishability, and $\overset{s}{\equiv}$ denotes statistical indistinguishability.

2.9 Single Inconsistent Party Technqiue and Persistently Inconsistent Party Technqiue

Canetti et al. [10] and Frankel et al. [19] independently introduced the *single inconsistent party (SIP)* technique for proving the security of distributed key generation protocols in the presence of adaptive adversaries with erasure. At the start of simulation of a subprotocol of a threshold scheme, the simulator generates the identity of the single inconsistent party (SIP) uniformly at random from the set of participating parties. The simulator is constructed in such a way that, the view of any party, except the SIP, in the simulation is guaranteed to be computationally indistinguishable from its view in the real world. It is guaranteed that the view of the adversary is independent from the choice of the SIP. During the simulation, if the adversary corrupts that party, then the simulator rewinds to the start step of that subprotocol, generates a new party uniformly at random from the set of parties, and proceeds again. To ensure that the simulations of consecutive subprotocols of a threshold scheme remain independent, the parties erase most of the local data produced during each subprotocol.
Jarecki and Lysyanskaya [22] extended the SIP technique to *persistently inconsistent party* technique for proving the security of distributed key generation protocol for erasure-free adaptive adversary model. In the *persistently inconsistent party* technique, the identity of an inconsistent player remains fixed throughout the simulation of a threshold scheme. At start of the simulation of the whole threshold scheme, the identity of the SIP is picked uniformly at random from the set of parties. If the adversary does not corrupt that party, then the simulation proceeds without rewinding. If the adversary corrupts that party at some point, then whole simulation of the threshold scheme is started from scratch.
The persistently inconsistent party technique of Jarecki and Lysyanskaya [22] is used in the security proofs in this article. The technique was used for multiparty case, but the technique is applicable in the one-sided adaptive adversary model for two-party case as well. Let $\mathcal{A}$ be a one-sided active adaptive adversary. At start of the simulation of the whole encryption scheme, the simulator selects I,

the identity of the SIP, uniformly at random from $\{P_1, P_2\}$. During the simulation, if $\mathcal{A}$ corrupts I, then the simulator generates a new I uniformly at random from $\{P_1, P_2\}$, and starts the simulation of the whole encryption scheme again. $\mathcal{A}$ corrupts at most one party; so the probability of a randomly selected party I being corrupted is at most $\frac{1}{2}$. That means, the expected number of trials for successful simulation of the encryption scheme is at most two, and the simulation can be performed in expected polynomial time. To bound the running time of simulation to strictly polynomial time, execution can be continued up to k^{ℓ_1} steps where ℓ_1 is a predetermined constant. If the simulation is not finished within k^{ℓ_1} steps, then the simulator fails. Note that the probability of failure of simulation is negligible.

2.10 Security Model

The security of the new protocols are proved following the simulation based security definition by Canetti [9].

3 Definition of Two-Party Lossy Threshold PKE Scheme

A definition of two-party lossy threshold PKE scheme secure against one-sided active adaptive adversaries is presented below.

Definition 2. *(Lossy Threshold PKE Scheme Secure against Erasure-Free One-Sided Active Adaptive Adversaries)*
A ***lossy threshold PKE scheme secure against erasure-free one-sided active adaptive adversaries*** *for the set of parties $P = \{P_1, P_2\}$, and security parameter n, is a 4-tuple (K, KG, E, Π_{DEC}) having the following properties.*

Key Space: *The key space K is a family of finite sets (pk, sk_1, sk_2). pk is the public key and sk_i is the secret key share of P_i. Let M_{pk} denote the message space for public key pk.*

Key Generation: *There exists a probabilistic polynomial-time key generation algorithm KG, which, on input $(1^n, mode)$, generates public output pk and a list $\{vk, vk_1, vk_2\}$ of verification keys, and secret output sk_i for P_i, where $(pk, sk_1, sk_2) \in K$. By setting mode to zero and one, key in lossy mode and injective mode can be generated, respectively. vk is called the verification key, vk_i is called the verification key of P_i.*

Encryption: *There exists a probabilistic polynomial-time encryption algorithm E, which, on input pk, $m \in M_{pk}$, $r \overset{\$}{\leftarrow} coins(E)$, outputs an encryption $c = E_{pk}(m, r)$ of m.*

Decryption: *There exists a two-party decryption protocol Π_{DEC} secure against erasure-free one-sided active adaptive adversaries. On common public input (c, pk, vk, vk_1, vk_2), and secret input sk_i for each $P_i, i \in \{1, 2\}$, where sk_i is the secret key share of P_i for the public key pk (as generated by KG), and c is an encrypted message, Π_{DEC} returns a message m, or the symbol $\perp$ denoting a decryption failure, as a common public output.*

Lossy Encryption Properties: *The encryption scheme is a lossy encryption scheme according to Definition 1.*

Threshold Semantic Security: *Consider the following game G for an erasure-free one-sided active adaptive adversary $\mathcal{A}$.*

G1. *$\mathcal{A}$ may corrupt at most one party. If $\mathcal{A}$ corrupts P_i, then $\mathcal{A}$ learns the history of P_i.*

G2. *The challenger executes algorithm KG. The challenger broadcasts the public key and the verification keys. For each $i \in \{1, 2\}$, the challenger sends sk_i to P_i. If there is a corrupted party P_i, then $\mathcal{A}$ learns sk_i.*

G3. *$\mathcal{A}$ adaptively makes the following types of queries.*

1. *Corruption query*
   $\mathcal{A}$ may corrupt a party, if no party was corrupted before. If $\mathcal{A}$ corrupts P_i, then $\mathcal{A}$ learns sk_i and the history of P_i.
2. *Decryption query*
   $\mathcal{A}$ selects a message $m \in M_{pk}$, and sends it to the challenger. The challenger sends $\mathcal{A}$ the decryption shares and the validity proofs of P_1 and P_2, for an encryption of m.

$\mathcal{A}$ repeats step G3 as many times as it wishes.

G4. *$\mathcal{A}$ selects two message m_0 and m_1 from M_{pk}, and sends them to the challenger. The challenger randomly selects a bit b, and sends an encryption c of m_b, to $\mathcal{A}$.*

G5. *$\mathcal{A}$ repeats step G3 as many times as it wishes. $\mathcal{A}$ cannot request message m_0 or m_1 in step G3(2).*

G6. *$\mathcal{A}$ outputs a guess bit b_1.*

A threshold encryption scheme is said to be ***semantically secure against erasure-free one-sided active adaptive adversaries*** *if, for any probabilistic polynomial-time erasure-free one-sided active adaptive adversary, $b = b_1$ with probability only negligibly greater than $\frac{1}{2}$.*

The verification keys are used for validity proofs in Π_{DEC}. During Π_{DEC}, each party P_i uses *validity proof* such that P_i can convince the remaining party that P_i performed its calculation in Π_{DEC} correctly, without disclosing its secret.

Note that $\mathcal{A}$ can only request for ciphertexts for which it knows the plaintext. It is not like the chosen ciphertext attack (CCA) where the adversary can ask for decryption shares for any chosen ciphertext.

Step G3(2) is used in game G to denote that, despite learning all the decryption shares and validity proofs for several chosen plaintexts, the adversary still does not gain any advantage in guessing the plaintext from the ciphertext.

Let $\mathcal{F}_{KG}$ be the ideal functionality for the key generation. In a two-party lossy threshold encryption scheme, there may exist a two-party distributed key generation (DKG) protocol that computes $\mathcal{F}_{KG}$ securely against erasure-free one-sided active adaptive adversaries.

4 A New Two-Party Lossy Threshold Homomorphic Encryption Scheme

In this section, a new two-party lossy threshold homomorphic public key encryption scheme $ELTA2E = (K, KG, E, \Pi_{DEC})$ is presented. The name $ELTA2E$ denotes an encryption scheme that is lossy, threshold, secure against adaptive adversaries, for two parties and based on the ElGamal encryption scheme. $ELTA2E$ is based on the DDH assumption. All protocols of $ELTA2E$ work in the CRS model.
Bellare and Yilek [7] designed a non-threshold lossy encryption scheme with efficient *Opener* algorithm, based on the DDH assumption. Let *EncLossy* denote their encryption scheme. $ELTA2E$ is created by adding the threshold properties to *EncLossy*.

4.1 Group

One possible group $\mathbb{G}$ for $ELTA2E$ is as follows. *Safe primes* are primes of the form $p = 2q + 1$ where q is also a prime. On input n, using known methods to generate safe primes, an n-bit safe prime p is generated, with generator g_0 of $\mathbb{Z}_p^*$. There is exactly one subgroup $\mathbb{G}$ of $\mathbb{Z}_p^*$ of order q. Let g be the generator of $\mathbb{G}$. $g = {g_0}^{\frac{p-1}{q}} = (g_0)^2$. (p, q, g) is the description of group $\mathbb{G}$. Unless otherwise specified, all computations are performed in group $\mathbb{G}$. Pedersen commitment scheme [28] is used as the trapdoor commitment scheme in $ELTA2E$.

4.2 Generation of the Description of the Group

At first, parties have to generate the group $\mathbb{G}$ for $ELTA2E$. One possible way to perform this operation is as follows. P_1 generates a description (p, q, g) of group $\mathbb{G}$ using Bach's algorithm [3]. P_1 sends (p, q, g) to P_2. P_2 checks whether it is valid or not. If the description is invalid, then P_1 and P_2 repeat the same process.

4.3 Key Generation

The key generation algorithm KG of $ELTA2E$ is described in Figure 2.

4.4 Encryption

The encryption algorithm E is presented in Figure 3.
The result of encryption is $c = (y, z)$. So, the ciphertext space of $ELTA2E$ is $\mathbb{G} \times \mathbb{G}$. $\mathbb{G}$ is a subgroup of $\mathbb{Z}_p^*$ and p is a k-bit prime. So, the size of ciphertext is $2k$.

Key Generation Algorithm, KG.

Common Reference String: $\mu \in \mathbb{Z}_p$.
Group description: (p, q, g).
Input:

1. **Security Parameter:** n.
2. **Mode:** $mode \in \{0, 1\}$.
   $mode = 1$ denotes injective mode,
   $mode = 0$ denotes lossy mode.

```
1. Select $\alpha_1, \alpha_2 \xleftarrow{\$} \mathbb{Z}_q$.
   Set
       $sk_1 = \alpha_1,$
       $sk_2 = \alpha_2,$
       $\alpha = (\alpha_1 + \alpha_2) \bmod q,$
       $h = g^{\alpha},$
       $vk = g,$
       $vk_1 = g^{\alpha_1},$
       $vk_2 = g^{\alpha_2}.$
2. Select $\gamma \xleftarrow{\$} \mathbb{Z}_q$.
   Set $j = g^{\gamma}$.
3. If $mode = 1$, then set
       $\ell = g^{\gamma\alpha}.$
   If $mode = 0$, then select $\rho \xleftarrow{\$} \mathbb{Z}_q \setminus \{\alpha\}$, and set
       $\ell = g^{\gamma\rho}.$
4. Set $pk = (q, g, j, h, \ell)$.
5. Send $(pk, vk, vk_1, vk_2, sk_1)$ as the output to $P_1$.
   Send $(pk, vk, vk_1, vk_2, sk_2)$ as the output to $P_2$.
```

Fig. 2: Key generation algorithm KG.

Encryption Algorithm, E.

Group description: (p, q, g).
Input:

- **Public Key:** $pk = (q, g, j, h, \ell)$.
- **Plaintext:** $m \in \{0, 1\}$.
- **Randomness:** $s, t \in \mathbb{Z}_q \times \mathbb{Z}_q$.

1. Set $y = g^s j^t$.
2. Set $z = h^s \ell^t g^m$.
3. Return $c = (y, z)$.

Fig. 3: Encryption algorithm E.

Threshold Decryption Protocol, Π_{DEC}.

Common Reference String: $\mu \in \mathbb{Z}_p$.
Group description: (p, q, g).
Common Inputs:

- **Public Key:** $pk = (q, g, j, h, \ell)$.
- **Verification Keys:** (vk, vk_1, vk_2).
- **Ciphertext:** $c = (y, z)$.

Input of P_1 : Secret Key Share sk_1.
Input of P_2 : Secret Key Share sk_2.

1. P_1 sends $ds_1 = y^{sk_1}$.
2. P_1 proves that $\log_y(ds_1) = \log_{vk}(vk_1)$. If P_1 fails, then P_2 aborts.
3. P_2 sends $ds_2 = y^{sk_2}$.
4. P_2 proves that $\log_y(ds_2) = \log_{vk}(vk_2)$. If P_2 fails, then P_1 aborts.
5. P_1 and P_2 compute
$$w = \frac{z}{ds_1 \cdot ds_2}.$$
6. From w, P_1 and P_2 compute m where $m \in \{0, 1\}$, and $g^m = w$ in $\mathbb{G}$.
If there is such a value m, then P_1 and P_2 output m.
Otherwise, P_1 and P_2 output $\perp$, denoting decryption failure.

Fig. 4: Threshold decryption protocol Π_{DEC}.

4.5 Protocol for Threshold Decryption

The protocol Π_{DEC} for threshold decryption is presented in Figure 4.
Adaptive zero-knowledge argument for equality of discrete logarithm is used as the validity proof in Π_{DEC}. The communication cost of the adaptive zero-knowledge argument for equality of discrete logarithm is $7k$. The communication cost of the threshold decryption protocol Π_{DEC} is $16k$.

Private Threshold Decryption to One Party Similar to the protocol Π_{DEC}, where both parties learn the output, it is possible to perform a private threshold decryption of a ciphertext known to both parties to just one party P_i. In this case, only P_i is supposed to learn the result of decryption. The remaining party P_{2-i} computes $ds_{2-i} = y^{(sk_{2-i})}$, and sends ds_{2-i} to P_i. Then P_{2-i} proves the correctness of this calculation by the zero-knowledge argument as above, to P_i. If P_{2-i} fails in the proof, then P_i aborts. P_i computes $ds_i = y^{sk_i}$. Then P_i computes w and the output as above.

4.6 Distributed Key Generation Protocol

$ELTA2E$ has a distributed key generation protocol Π_{DKG}. The objective of protocol Π_{DKG} is that parties P_1 and P_2 together generate the public key and the secret key shares according to the distribution given by encryption scheme $ELTA2E$ without any trusted dealer in such a way that the secret key is not stored in a single location. Along with the public key and the private key shares, Π_{DKG} also generates the verification keys. For encryption scheme $ELTA2E$, protocol Π_{DKG} has the following requirements.

1. $\alpha \overset{\$}{\leftarrow} \mathbb{Z}_q^*$. α is additively shared between P_1 and P_2. $h = g^{\alpha}$ is made public.
2. $\gamma \overset{\$}{\leftarrow} \mathbb{Z}_q$. $j = g^{\gamma}$ is made public.
3. If $mode = 1$, then $\ell = g^{\gamma\alpha}$. If $mode = 0$, then $\ell = g^{\gamma\rho}$ where $\rho \overset{\$}{\leftarrow} \mathbb{Z}_q \setminus \{\alpha\}$. ℓ is made public.

The protocol Π_{DKG} is presented in Figure 5.
The proofs in steps 2,4,9, and 11 are performed using adaptive zero-knowledge arguments. For simplicity of presentation, the adaptive zero-knowledge arguments are not described with the explicit communication with the trusted party computing the zero-knowledge argument. In each case, the prover sends the message $(ZK-prover, V, x, w)$ to $\mathcal{F}_{ZK}^R$ where V is the identity of the verifier, x is the common input and w is the witness of the prover. If $(x, w) \in R$, then $\mathcal{F}_{ZK}^R$ sends $(ZK-proof, x)$ to the verifier. There is no communication between prover and verifier in the adaptive zero-knowledge argument in the hybrid world. The CRS μ is used as the commitment key for Pedersen commitment scheme. The CRS μ also acts as the CRS for the zero-knowledge arguments.
The reason for using commitments in Π_{DKG} is to ensure that no party can affect the distribution of the generated key. If commitments are not used, then

Distributed Key Generation Protocol, Π_{DKG}.

Common Reference String: $\mu \in \mathbb{Z}_p$.
Group description: (p, q, g).
Inputs:

1. **Security Parameter:** k.
2. **Mode:** $mode \in \{0, 1\}$.
   $mode = 1$ denotes injective mode,
   $mode = 0$ denotes lossy mode.

1. P_1 selects $\alpha_1, \gamma_1, \beta_1, \theta_1 \overset{\$}{\leftarrow} \mathbb{Z}_q$. P_1 sets $sk_1 = \alpha_1$. P_1 computes $h_1 = g^{\alpha_1}, j_1 = g^{\gamma_1}$. P_1 computes commitments $b_1 = Com_\mu(h_1, \beta_1), c_1 = Com_\mu(j_1, \theta_1)$. P_1 sends (b_1, c_1).
2. P_1 proves the knowledge of committed secret for commitments b_1 and c_1. If P_1 fails in any proof, then P_2 aborts.
3. P_2 selects $\alpha_2, \gamma_2 \overset{\$}{\leftarrow} \mathbb{Z}_q$. P_2 sets $sk_2 = \alpha_2$. P_2 computes $h_2 = g^{\alpha_2}, j_2 = g^{\gamma_2}$. P_2 sends (h_2, j_2).
4. P_2 proves knowledge of discrete logarithm of h_2 and j_2. If P_2 fails in any proof, then P_1 aborts.
5. P_1 sends the openings (h_1, β_1) and (j_1, θ_1) of its commitments.
6. P_2 verifies that $b_1 = Com_\mu(h_1, \beta_1)$, and $c_1 = Com_\mu(j_1, \theta_1)$. If any of these two equalities does not hold, then P_2 aborts.
7. P_1 and P_2 set $vk = g, vk_1 = h_1, vk_2 = h_2, h = h_1h_2, j = j_1j_2$.
8. If $mode = 0$, then P_1 selects $\tau_1 \overset{\$}{\leftarrow} \mathbb{Z}_q \setminus \{\alpha_1\}$, and sets $\ell_1 = j^{\tau_1}$.
   If $mode = 1$, then P_1 sets $\ell_1 = j^{\alpha_1}$. P_1 sends ℓ_1.
9. If $mode = 1$, then P_1 proves that $\log_j(\ell_1) \neq \log_{vk}(vk_1)$.
   If $mode = 1$, then P_1 proves that $\log_j(\ell_1) = \log_{vk}(vk_1)$.
   If P_1 fails, then P_2 aborts.
10. P_2 sends $\ell_2 = j^{\alpha_2}$.
11. P_2 proves that $\log_j(\ell_2) = \log_{vk}(vk_2)$. If P_2 fails, then P_1 aborts.
12. P_1 and P_2 set $\ell = \ell_1\ell_2, pk = (q, g, j, h, \ell)$.
13. P_1 outputs $(pk, sk_1, (vk, vk_1, vk_2))$.
14. P_2 outputs $(pk, sk_2, (vk, vk_1, vk_2))$.

Fig. 5: Distributed Key Generation Protocol Π_{DKG}.

P_1 selects α_1, γ_1, and sends $h_1 = g^{\alpha_1}$, and $j_1 = g^{\gamma_1}$ to P_2. After viewing h_1 and j_1, P_2 can select its shares α_2 and γ_2 in such a way that it can affect the distribution of the generated key h and j. To prevent this, a commitment scheme is used. At first, P_1 sends commitments of h_1 and j_1 to P_2. By the hiding property of the commitment scheme, P_2 does not learn about h_1 and j_1. Then, P_2 selects α_2, γ_2, and sends $h_2 = g^{\alpha_2}$, and $j_2 = g^{\gamma_2}$ to P_1. Since P_2 does not know h_1 and j_1, P_2 cannot affect the distribution of h or j. After that, P_1 opens its commitments to P_2. By the binding property of the commitment scheme, P_1 cannot open its commitments to anything other than what it used in the original commitments. That means P_1 cannot affect the distribution of h or j, after seeing h_2 and j_2. As a result, the generated key h and j are uniformly distributed in $\mathbb{G}$.
If injective mode is chosen, then $\ell = \ell_1\ell_2 = j^{\alpha_1}j^{\alpha_2} = j^{\alpha_1+\alpha_2} = j^{\alpha}$. Then, $\log_g(h) = \log_j(\ell) = \alpha$, as required for an injective key pair. If lossy mode is selected, then $\ell = \ell_1\ell_2 = j^{\tau_1}j^{\alpha_2} = j^{\tau_1+\alpha_2} \neq j^{\alpha_1+\alpha_2}$ since $\tau_1 \neq \alpha_1$. Then, $\log_g(h) \neq \log_j(\ell)$, as required for a lossy key pair.

4.7 *ELTA2E* is Additive Homomorphic

The following lemma proves the additive homomorphic property of $ELTA2E$.

Lemma 1. *$ELTA2E$ is additive homomorphic.*

Proof. **Homomorphic Addition.** Let $c_1 = (g^{s_1}j^{t_1}, h^{s_1}\ell^{t_1}g^{m_1})$, and $c_2 = (g^{s_2}j^{t_2}, h^{s_2}\ell^{t_2}g^{m_2})$ be two ciphertexts encrypting plaintexts m_1 and m_2, respectively.
$c = c_1 +_h c_2 = (g^{s_1}j^{t_1} \cdot g^{s_2}j^{t_2}, h^{s_1}\ell^{t_1}g^{m_1} \cdot h^{s_2}\ell^{t_2}g^{m_2})$
$= (g^{s_1+s_2}j^{t_1+t_2}, h^{s_1+s_2}\ell^{t_1+t_2}g^{m_1+m_2})$.
Homomorphic Multiplication by Constant. Let $c_1 = (y_1, z_1) = (g^{s_1}j^{t_1}, h^{s_1}\ell^{t_1}g^{m_1})$ be a ciphertext encrypting plaintext m_1. Let m_2 be a known plaintext.
$c_2 = c_1 \times_h m_2 = \big((g^{s_1}j^{t_1})^{m_2}, (h^{s_1}\ell^{t_1}g^{m_1})^{m_2}\big) = (g^{s_1m_2}j^{t_1m_2}, h^{s_1m_2}\ell^{t_1m_2}g^{m_1m_2})$.

4.8 *ELTA2E* is Randomizable.

Let $c = (y, z) = (g^s j^t, h^s \ell^t g^m)$ be a ciphertext encrypting plaintext m. The *Blind* function on input $(pk, c) = ((q, g, j, h, \ell), (y, z))$ works as follows. Select $s_1, t_1 \xleftarrow{\$} \mathbb{Z}_q \times \mathbb{Z}_q$. Set $y_1 = y \cdot g^{s_1}j^{t_1}, z_1 = z \cdot h^{s_1}\ell^{t_1}$. Return $c_1 = (y_1, z_1)$. Then, $c_1 = (g^s j^t \cdot g^{s_1}j^{t_1}, h^s\ell^t g^m \cdot h^{s_1}\ell^{t_1}) = (g^{s+s_1}j^{t+t_1}, h^{s+s_1}\ell^{t+t_1}g^m)$.

Thus, c_1 encrypts m. Since $s_1, t_1 \xleftarrow{\$} \mathbb{Z}_q \times \mathbb{Z}_q$, $(s + s_1)$ and $(t + t_1)$ are uniformly distributed in $\mathbb{Z}_q$.

5 Security of the DKG Protocol Π_{DKG}

The following thoerem describes the security of protocol Π_{DKG}.

Theorem 1. *Provided that the DDH assumption holds, and trapdoor commitment scheme and adaptive zero-knowledge arguments exist, protocol Π_{DKG} computes $\mathcal{F}_{KG}$ securely against erasure-free one-sided active adaptive adversaries.*

The proof of Theorem 1 is presented in Appendix C.

6 Security of Encryption Scheme *ELTA2E*

The following lemma proves the lossy encryption property of *ELTA2E*.

Lemma 2. *If the decisional Diffie-Hellman assumption holds, then ELTA2E is a lossy encryption scheme. ELTA2E has an efficient (polynomial-time) Opener algorithm.*

Proof. **Correctness of Decryption in the Injective Mode.** In the injective mode, $pk = (q, g, j, h, \ell) = (q, g, g^{\gamma}, g^{\alpha}, g^{\alpha \cdot \gamma})$. Then, $w = \frac{z}{ds_1 \cdot ds_2} = \frac{z}{y^{sk_1} \cdot y^{sk_2}} = \frac{z}{y^{\alpha_1+\alpha_2}} = \frac{z}{y^{\alpha}} = \frac{h^s \ell^t g^m}{(g^s j^t)^{\alpha}} = \frac{(g^{\alpha})^s (g^{\gamma\alpha})^t g^m}{\left(g^s (g^{\gamma})^t\right)^{\alpha}} = \frac{g^{\alpha s+\alpha\gamma t+m}}{g^{\alpha s+\alpha\gamma t}} = g^m$.
Indistinguishability of Keys. In the injective mode, $pk = (q, g, j, h, \ell) = (q, g, g^{\gamma}, g^{\alpha}, g^{\gamma\alpha})$. In the lossy mode, $pk = (q, g, j, h, \ell) = (q, g, g^{\gamma}, g^{\alpha}, g^{\gamma\rho})$. By the DDH assumption, the public key in injective mode is computationally indistinguishable from the public key in lossy mode.
Lossiness on Lossy Keys. Let $pk = (q, g, j, h, \ell) = (q, g, g^{\gamma}, g^{\alpha}, g^{\gamma\rho})$ be a lossy public key. Encryption of a message m with randomness (s, t) is $c = (y, z) = (g^{s+\gamma t}, g^{\alpha s+\gamma\rho t} \cdot g^m)$. Since $\rho \neq \alpha$, $(s+\gamma t)$ and $(\alpha s+\gamma\rho t)$ are linearly independent combinations of s and t. Then, y and z are uniformly random group elements.
Efficient *Opener* **Algorithm.** Let $pk = (q, g, j, h, \ell) = (q, g, g^{\gamma}, g^{\alpha}, g^{\gamma\rho})$ be a lossy public key. Let the corresponding secret key be $sk = (\gamma, \rho, \alpha)$. Let $c = (y, z)$ be an encryption of plaintext m with randomness $r = (s, t)$. Then, $c = (y, z) = (g^{s+\gamma t}, g^{\alpha s+\gamma\rho t} \cdot g^m)$. Let m_1 be the plaintext with which the ciphertext c has to be opened. On input $(pk, sk, (y, z), m, (s, t), m_1)$, the algorithm *Opener* has to find randomness $r_1 = (s_1, t_1) \in \mathbb{Z}_q \times \mathbb{Z}_q$ such that $s + \gamma t = s_1 + \gamma t_1$, and $\alpha s + \gamma\rho t + m = \alpha s_1 + \gamma\rho t_1 + m_1$. These two equations are two linear equations on the variables (s_1, t_1). The *Opener* algorithm solves these two equations to find s_1 and t_1 in polynomial time. There is only one such pair $(s_1, t_1) \in \mathbb{Z}_q \times \mathbb{Z}_q$ such that these two equalities hold.
Note that without the knowledge of the secret key $sk = (\gamma, \rho, \alpha)$, finding such s_1 and t_1 takes exponential time. In that case, to solve for s_1, t_1, it is necessary to solve for five variables $(s_1, t_1, \alpha, \gamma, \rho)$ from two equations, which takes exponential time. In other words, without the knowledge of the secret key, no polynomial-time machine can open a given ciphertext created under a lossy key to any plaintext of its choice. Without the knowledge of the secret key, encryption is binding for lossy keys.

The following lemma proves the threshold semantic security of *ELTA2E*.

Lemma 3. *Provided that the decisional Diffie-Hellman assumption holds, and trapdoor commitment scheme and adaptive zero-knowledge arguments exist, the encryption scheme ELTA2E achieves threshold semantic security in the presence of erasure-free one-sided active adaptive adversaries.*

The proof of Lemma 3 is presented in Appendix D. The following theorem describes the security of encryption scheme *ELTA2E*.

Theorem 2. *Provided that the DDH assumption holds, and trapdoor commitment scheme and adaptive zero-knowledge arguments exist, the encryption scheme $ELTA2E$ is a two-party lossy threshold encryption scheme secure against erasure-free one-sided active adaptive adversaries.*

Proof. By Lemma 2, $ELTA2E$ satisfies the lossy encryption properties. By Lemma 3, $ELTA2E$ satisfies the threshold semantic security requirement given in Definition 2. Then, $ELTA2E$ is a two-party lossy threshold encryption scheme secure against erasure-free one-sided active adaptive adversaries.

7 Oblivious Transfer against One-Sided Active Adaptive Adversaries

In this section, a new protocol Π_{OTAA} for bit OT is presented. In Figure 6 the ideal functionality $\mathcal{F}_{OT}$ for oblivious transfer is presented, following [12]. Protocol Π_{OTAA} uses the following tools.

Functionality $\mathcal{F}_{OT}$.
Common Reference String : $\mu \in \mathbb{Z}_p$.
$\mathcal{F}_{OT}$ proceeds as follows, running with a sender S, a receiver R, and an adversary $\mathcal{A}$:

- Upon receiving $(\text{sender}, x_0, x_1)$ from S, where each $x_j \in \{0,1\}$, record (x_0, x_1).
- Upon receiving $(\text{receiver}, \sigma)$ from S, where $\sigma \in \{0,1\}$, send x_σ to R, and received to $\mathcal{A}$, and halt.
  If there was no previous message of the form $(\text{sender}, ..)$, then send nothing to R.

Fig. 6: Ideal functionality $\mathcal{F}_{OT}$.

1. A two-party lossy threshold homomorphic public key cryptosystem secure against one-sided active adaptive adversaries with efficient (polynomial-time) *Opener* algorithm. For this purpose, a new encryption scheme $ELTA2E = (K, KG, E, \Pi_{DEC})$ is designed. $ELTA2E$ is presented in Section 4.
   As mentioned in Section 4, $\mathcal{F}_{KG}$ denotes the ideal functionality for the key generation of $ELTA2E$. A two-party protocol Π_{DKG} that implements this functionality is described in Section 4.6. Π_{DKG} is secure against erasure-free one-sided active adaptive adversaries.
2. Adaptive zero-knowledge arguments. The adaptive zero-knowledge arguments used in the protocol are described in Section 11.

The relations of the adaptive zero-knowledge arguments used in the OT protocol are described below. Let p, q be primes such that q divides $(p-1)$. Let g be an element of order q in $\mathbb{Z}_p^*$.

Let R_{EQ} be the relation denoting equality of discrete logarithm. Let the common input be $(y, z, g, h) = (g^w \bmod p, h^w \bmod p, g, h)$. The prover P knows witness $w \in \mathbb{Z}_q$.

$$R_{EQ} = \{((y, z, g, h), w) : y \equiv g^w \bmod p, z \equiv h^w \bmod p\}.$$

Let R_{MULT} be the relation denoting correctness of multiplication for $ELTA2E$. Let $c_1 = (u_1, v_1)$ be a ciphertext of $ELTA2E$. Let $m_2 \in \mathbb{Z}_q$ be a known plaintext to prover P. P computes ciphertext $c_2 = (u_2, v_2)$ by performing homomorphic multiplication of c_1 by m_2, that is, $c_2 = m_2 \times_h c_1$. Then P obtains ciphertext $c_3 = (u_3, v_3)$ by applying $Blind$ function on ciphertext c_2. Let the randomness used in the $Blind$ operation be (s_3, t_3). The relationship R_{MULT} is used to denote a relationship between the ciphertexts c_1 and c_3. Let the common input be $(c_1, c_3) = (u_1, v_1, u_3, v_3)$. The witness w for relation R_{MULT} consists of the known plaintext m_2 and the randomness (s_3, t_3) used in the $Blind$ operation. That is, $w = (m_2, s_3, t_3)$. The prover P knows witness $w = (m_2, s_3, t_3) \in \mathbb{Z}_q \times \mathbb{Z}_q \times \mathbb{Z}_q$.
$R_{MULT} =$
$\{((u_1, v_1, u_3, v_3), (m_2, s_3, t_3)) : u_3 \equiv (u_1)^{m_2} g^{s_3} j^{t_3} \bmod p, v_3 \equiv (v_1)^{m_2} h^{s_3} \ell^{t_3} \bmod p\}$.
Let R_{ZERO} be the relation denoting that a given ciphertext encrypts zero, for $ELTA2E$. If $m = 0$, then the encryption for the injective mode of $ELTA2E$ is $c = (u, v) = \left(g^{s+\gamma t}, g^{\alpha s+\alpha\gamma t} g^0\right) = (g^{s+\gamma t}, h^{s+\gamma t})$. Proving that a given ciphertext $c = (u, v)$ is an encryption of zero is equivalent to prove that $\log_g(u) = \log_h(v)$. Let the common input be $c = (u, v)$. The prover P knows witness $w \in \mathbb{Z}_q$.

$$R_{ZERO} = \{((u, v), w) : u \equiv g^w \bmod p, v \equiv h^w \bmod p\}.$$

For proving that one of two given ciphertexts encrypts zero without disclosing which one, the OR-construction of Σ-protocols are performed [14]. Let $R_{OR-ZERO}$ be the relation denoting the OR-composition of the relation R_{ZERO}, for $ELTA2E$. Let the common input be $(c_0, c_1) = ((u_0, v_0), (u_1, v_1))$. Let $b \in \{0, 1\}$ be such that c_b is an encryption of zero. Then the prover P knows witness $w_b \in \mathbb{Z}_q$. $R_{OR-ZERO}((u_0, v_0, u_1, v_1), (w_0, w_1))$
$= R_{ZERO}((u_0, v_0), w_0) \vee R_{ZERO}((u_1, v_1), w_1)$.
The protocol Π_{OTAA} is presented in Figure 7. The protocol Π_{OTAA} works in the common reference string model. The protocol Π_{OTAA} works in the $(\mathcal{F}_{KG}, \mathcal{F}_{ZK}^{R})$-hybrid world. For simplicity of presentation, the zero-knowledge arguments are not described with the communication of parties to the ideal functionality $\mathcal{F}_{ZK}^{R}$. One possibility to generate the auxiliary inputs p, q, g is as follows. S generates the description (p, q, g) of the group $\mathbb{G}$ for $ELTA2E$, using Bach's algorithm [3]. S sends (p, q, g) to R. R checks its validity. If the description is invalid, then S and R repeat the same process. The proofs in steps 3, 4(b) and 5(b) of Π_{OTAA} are performed by adaptive zero-knowledge arguments. The CRS μ acts as the CRS for functionality $\mathcal{F}_{KG}$ and all the zero-knowledge arguments. In step 3, R proves that one of c_0 and c_1 encrypts zero, without disclosing which one. If R could set both ciphertexts c_0 and c_1 to encryptions of one, then R could learn both

Protocol Π_{OTAA}.

Common Reference String: $\mu \overset{\$}{\leftarrow} \mathbb{Z}_p$.

Input of $S : (x_0, x_1) \in \{0,1\}^2$.

Input of $R : \sigma \in \{0,1\}$.

Auxiliary Input : (k, p, q, g) where k is the security parameter, and (p, q, g) is a representation of a group $\mathbb{G}$ for the encryption scheme $ELT2_{DDH}$.

1. S and R jointly generate an injective key for $ELT2_{DDH}$, by executing $\mathcal{F}_{KG}$ with input $(1^n, 1)$.
   Here, S and R acts as P_1 and P_2, respectively.
   Both parties get the public key $pk = (q, g, j, h, \ell)$ and the verification keys (vk, vk_1, vk_2).
   S gets its secret key share sk_1 and R gets its secret key share sk_2.
2. R selects $s_0, t_0, s_1, t_1 \overset{\$}{\leftarrow} \mathbb{Z}_q$.
   R computes
   $$\begin{aligned} c_0 &= E_{pk}\left(1-\sigma, (s_0, t_0)\right), \text{ and} \\ c_1 &= E_{pk}\left(\sigma, (s_1, t_1)\right). \end{aligned}$$
   R sends (c_0, c_1) to S.
3. R proves by an adaptive zero-knowledge argument that one of (c_0, c_1) is an encryption of zero.
   If R fails in the proof, then S aborts.
4. (a) For each $i \in \{0,1\}, S$ computes
   $$\begin{aligned} d_i &= x_i \times_h c_i, \\ v_i &= Blind(pk, d_i). \end{aligned}$$
   S sends (v_0, v_1) to R.

   (b) S proves the correctness of homomorphic multiplication for v_0 and v_1, by using adaptive zero-knowledge arguments.
   If S fails in any of the proofs, then R aborts.

Fig. 7: Protocol Π_{OTAA}.

Protocol Π_{OTAA}.

5. For each $i \in \{0,1\}$, S and R jointly perform private decryption of v_i to R, so only R gets the output w_i of decryption, as follows.
   (a) Let $v_i = (y_i, z_i)$.
   S computes
   $$ds_{1,i} = (y_i)^{(sk_1)} .$$
   S sends $ds_{1,i}$ to R.
   (b) S proves, by an adaptive zero-knowledge argument, that
   $$\log_{(y_i)}(ds_{1,i}) = \log_g(vk_1) .$$
   If S fails in the proof, then R aborts.
   (c) R performs the following steps.
      i. R computes
      $$ds_{2,i} = (y_i)^{(sk_2)} .$$
      ii. R computes
      $$\theta_i = \frac{z_i}{ds_{1,i} \cdot ds_{2,i}} .$$
      iii. From θ_i, R computes w_i where $w_i \in \{0,1\}$ and $g^{w_i} = \theta_i$ in $\mathbb{G}$.
6. R outputs w_σ.

Fig. 7: Protocol Π_{OTAA} (continued).

x_0 and x_1 at step 5. This zero-knowledge argument is incorporated to prevent this type of cheating by R. In step 4, S computes $d_i = x_i \times_h c_i, v_i = Blind(d_i)$. S sends v_i. R knows the ciphertext c_i. The $Blind$ function is included so that new randomness is added to the result d_i. Then, R cannot learn the constant x_i after seeing v_i.

7.1 Correctness of Protocol Π_{OTAA}.

If S and R both follow the protocol, then the following events occur. S and R generate an injective key for $ELTA2E$. R honestly computes c_0 and c_1. c_σ encrypts one, and $c_{1-\sigma}$ encrypts zero. R passes in the proof in step 3. S honestly performs step 4, and passes in the proofs. v_σ encrypts x_σ and $v_{1-\sigma}$ encrypts zero. In step 5, S and R honestly perform two private decryption processes. By the "correctness on injective keys" property of $ELTA2E$, $w_\sigma = x_\sigma$ and $w_{1-\sigma} = 0$. Therefore, R learns x_σ.

7.2 Extension to String OT.

In a string OT, S has a pair of bit strings of length q as input : $(\overline{x_0}, \overline{x_1}) = \left(\left\{x_0^1, x_0^2, \ldots, x_0^q\right\}, \left\{x_1^1, x_1^2, \ldots, x_1^q\right\}\right)$. Here q is a polynomial of n. R has input $\sigma \in \{0,1\}$. The output of R is $\overline{x_\sigma} = \left\{x_\sigma^1, x_\sigma^2, \ldots, x_\sigma^q\right\}$, and the output of S is an empty string. The bit OT protocol Π_{OTAA} is extended to a string OT protocol as follows. In step 4, for each $i \in \{0,1\}, j \in \{1,2,\dots,q\}$, S computes $v_i^j = x_i^j \times_h c_i^j$. In step 5, for each $i \in \{0,1\}, j \in \{1,2,\dots,q\}$, S and R jointly perform private decryption of v_i^j to R, so R obtains result w_i^j. R outputs $\{w_\sigma^1, w_\sigma^2, \dots, w_\sigma^q\}$.

8 Security of Protocol Π_{OTAA}

The following theorem describes the security of protocol Π_{OTAA}.

Theorem 3. *Assume that the DDH assumption holds and there exists a trapdoor commitment scheme and adaptive zero-knowledge arguments. Assume that there exists a two-party lossy threshold public key cryptosystem which is secure against erasure-free one-sided active adaptive adversaries, is additive homomorphic, randomizable, and has an efficient (polynomial-time) Opener algorithm. Then, protocol Π_{OTAA} is a protocol for oblivious transfer secure under sequential composition, in the presence of erasure-free one-sided active adaptive adversaries. Protocol Π_{OTAA} requires $O(n)$ communication complexity and constant number of public-key operations in the worst case.*

The proof of Theorem 3 is presented in Appendix E.

9 Comparison with Related Work

Hazay and Patra [21] designed an OT protocol for erasure-free one-sided active adaptive adversaries. Their protocol for bit OT requires $(288n^2 + 100n + 16) \in O(n^2)$ communication complexity. Protocol Π_{OTAA} needs $101n \in O(n)$ communication complexity. The worst case number of PKE operations of the protocol of [21] for bit OT is $(16n+6) \in O(n)$. The worst case number of PKE operations of Π_{OTAA} is constant (only four).
For OT of strings of size n, the OT protocol of [21] requires $(288n^2 + 110n + 16)$ communication complexity and $(16n + 6)$ PKE operations in the worst case. For OT of strings of size n, protocol Π_{OTAA} requires $(38n^2 + 98n)$ communication complexity and $(2n + 2)$ PKE operations in the worst case. For string OT of size n, protocol Π_{OTAA} requires seven times less communication complexity and eight times less PKE operations than the OT protocol of [21].

10 Main Factor in the Communication Complexity of the OT Protocol by Hazay and Patra [21]

In this section, the main factor of the complexity of the OT protocol by Hazay and Patra [21] is described. They have different efficiency for polynomial-size message space and exponential-size message space, with respect to the security parameter n. Here, the efficiency of bit OT, which falls in the category of polynomial-size message space, is described.
The OT protocol of [21] uses a non-committing encryption (NCE) scheme secure against one-sided active adaptive adversaries. They designed a protocol

Π_{OSC} that for this purpose. Π_{OSC} uses the somewhat non-committing encryption (SNCE) of [20]. The SNCE protocol of [20] uses the non-committing encryption scheme (NCE) of [16]. There was another more recent NCE scheme [13] which is error-free but requires more communication complexity and computational complexity than the NCE of [16] [31]. The NCE scheme of [16] uses a subroutine named *attempt*. In [Theorem 2, [16]], it is mentioned that the NCE scheme of [16] has to repeat $4n$ calls of *attempt* in order to ensure that the probability of failure of subroutine *attempt* remains negligible in n. That means, the worst case number of repeats of *attempt* is $4n$. Each call of *attempt* has communication cost $(12n + 1)$. The communication complexity of the NCE scheme of [16] is $O(n^2)$ for message size of one bit. Each call of *attempt* uses one encryption operation of a simulatable PKE scheme, so the number of PKE operations for *attempt* is 1. Then, the NCE scheme of [16] needs $4n$ PKE operations in the worst case. The communication complexity of the SNCE protocol of [20], with equivocality parameter $\ell = 2$, is $O(n^2)$. It uses the NCE protocol of [16] for sending an index $i \in \{1, \ldots, \ell\}$. As mentioned in [20], the expected number of PKE operations for this step is $O(\log \ell)$. In the worst case, this step requires $4n \in O(n)$ PKE operations. The communication complexity of the bit OT protocol of [21] is $O(n^2)$. The number of PKE operations of the bit OT of [21], in the worst case, is $O(n)$.

Hazay and Patra claims that their OT protocol needs a constant number of PKE operations [Theorem 2,[21]]. One possibility is that they were counting one encryption of the NCE scheme Π_{OSC} secure against one-sided adaptive adversaries (or one encryption of the dual-mode encryption scheme of [29]), each of them as a single PKE operation. But the encryption scheme Π_{OSC} uses other PKE schemes (the non-committing encryption scheme for the sender (NCES) of [6], the non-committing encryption scheme for the receiver (NCER) of [22] and the SNCE scheme of [20]) as its subroutines inside its implementation. The notion of *atomic* PKE scheme is necessary for the analysis of the number of PKE operations. An *atomic* PKE scheme denotes a PKE scheme that does not use any other PKE scheme as a subroutine in its implementation. To get the actual number of PKE scheme of a protocol, it should be counted that how many operations of atomic PKE scheme are invoked inside that protocol.

More details on the efficiency analysis of the OT protocol of [21] is provided in Appendix H.

11 Adaptive Zero Knowledge Arguments

In this section, the adaptive zero-knowledge arguments used in the protocols are described. First, the non-erasure Σ-protocols for the corresponding relations are presented. Then, it is described how to convert them to adaptive zero-knowledge arguments.

Non-Erasure Σ-Protocol for Knowledge of Discrete Logarithm. Scnorr [30] suggested a non-erasure Σ-protocol for knowledge of discrete logarithm [14].

Non-Erasure Σ-Protocol for Equality of Discrete Logarithm. A non-erasure Σ-protocol for equality of discrete logarithm is given in [14].

Non-Erasure Σ-Protocol for Knowledge of Committed Secret for Pedersen Commitment Scheme. Damgård [14] presented a non-erasure Σ-protocol for proving knowledge of committed secret for Pedersen commitment scheme.

Non-Erasure Σ-Protocol for Proving that one of two given Ciphertexts Encrypts Zero. If $m = 0$, then the encryption for the injective mode of $ELTA2E$ is $c = (x, y) = \left(g^{s+\gamma t}, g^{\alpha s+\alpha\gamma t} g^0\right) = (g^{s+\gamma t}, h^{s+\gamma t})$. Proving that a given ciphertext $c = (x, y)$ is an encryption of zero is equivalent to prove that $\log_g(x) = \log_h(y)$. For proving that one of two given ciphertexts encrypts zero without disclosing which one, the OR-construction of Σ-protocols [14] is performed.

11.1 Σ-Protocol for Inequality of Discrete Logarithm

If $mode = 0$, then, in step 2 of stage 2 of Π_{DKG}, P_1 has to prove that $\log_j(\ell_1) \neq \log_{vk}(vk_1)$. This can be called a proof for inequality of discrete logarithm. The relation can be defined as follows.

Let R_{NEQ} be the relation denoting inequality of discrete logarithm. Let the common input be
$\left(x_1, x_2, y_1, y_2\right) = \left((y_1)^{w_1} \bmod p, (y_2)^{w_2} \bmod p, y_1, y_2\right)$. The prover P knows witness $w_1, w_2 \in \mathbb{Z}_q$.

$R_{NEQ} =$
$$\left\{\left((x_1, x_2, y_1, y_2), (w_1, w_2)\right) : x_1 \equiv (y_1)^{w_1} \bmod p, x_2 \equiv (y_2)^{w_2} \bmod p, w_1 \neq w_2\right\}.$$
A new Σ-protocol $\Pi_{\Sigma NEQ}$ for proving the inequality of discrete logarithm is designed. The protocol is presented in Figure 8.

The security proof of protocol $\Pi_{\Sigma NEQ}$ is given below.

Lemma 4. *The protocol $\Pi_{\Sigma NEQ}$ is a non-erasure Σ-protocol for R_{NEQ}.*

The proof of Lemma 4 is given in Appendix **??**.

11.2 Non-Erasure Σ-Protocol for Proving Multiplication Correct

A new Σ-protocol for relation R_{MULT} for $ELTA2E$ is designed. The Σ-protocol $\Pi_{\Sigma MULT}$ is presented in Figure 9.

Lemma 5. *The protocol $\Pi_{\Sigma MULT}$ is a non-erasure Σ-protocol for R_{MULT} for $ELTA2E$.*

The proof of Lemma 5 is given in Appendix G.

Σ **Protocol** $\Pi_{\Sigma NEQ}$ **for** R_{NEQ}**.**

Inputs:

1. Common Input: $(x_1, x_2, y_1, y_2) \in \mathbb{Z}_p \times \mathbb{Z}_p \times \mathbb{Z}_p \times \mathbb{Z}_p$.
2. Witness of P: $(w_1, w_2) \in \mathbb{Z}_q \times \mathbb{Z}_q$.

1. P chooses $r \overset{\$}{\leftarrow} \mathbb{Z}_q$.
   P computes
   $$\begin{aligned} a_1 &= (y_1)^r \bmod p, \\ a_2 &= (y_2)^r \bmod p. \end{aligned}$$
   P sends $a = (a_1, a_2)$.
2. V chooses a challenge $e \overset{\$}{\leftarrow} \mathbb{Z}_q$ and sends it.
3. P computes
   $$\begin{aligned} z_1 &= r + ew_1 \bmod q, \\ z_2 &= r + ew_2 \bmod q. \end{aligned}$$
   P sends (z_1, z_2).
   V accepts if and only if
   $$\begin{aligned} (y_1)^{z_1} &= a_1(x_1)^e \bmod p, \\ (y_2)^{z_2} &= a_1{x_2}^e \bmod p, \\ (y_1)^{z_2} &\neq a_1(x_1)^e \bmod p, \text{ and} \\ (y_2)^{z_1} &\neq a_2(x_2)^e \bmod p. \end{aligned}$$

Fig. 8: Σ Protocol $\Pi_{\Sigma NEQ}$ for R_{NEQ}.

Σ **Protocol** $\Pi_{\Sigma MULT}$ **for** R_{MULT} **for** $ELTA2E$**.**

Inputs:

1. Common Input: $(c_1, c_3) = (u_1, v_1, u_3, v_3)) \in \mathbb{G} \times \mathbb{G}$.
2. Witness of P: $m_2, s_3, t_3 \in \mathbb{Z}_q$.

1. P chooses $r_1, r_2, r_3 \xleftarrow{\$} \mathbb{Z}_q$.
   P computes
$$a_1 = (u_1)^{r_1} g^{r_2} j^{r_3} \bmod p,$$
$$a_2 = (v_1)^{r_1} h^{r_2} \ell^{r_3} \bmod p.$$
   P sends $a = (a_1, a_2)$.
2. V chooses a challenge $e \xleftarrow{\$} \mathbb{Z}_q$ and sends it.
3. P computes
$$z_1 = r + em_2 \bmod q,$$
$$z_2 = r_2 + es_3 \bmod q,$$
$$z_3 = r_3 + et_3 \bmod q.$$
   P sends $z = (z_1, z_2, z_3)$.
   V accepts if and only if
$$(b_1)^{z_1} g^{z_1} j^{z_3} = a_1 (b_3)^e \bmod p,$$
   and
$$(d_1)^{z_1} h^{z_1} \ell^{z_3} = a_2 (d_3)^e \bmod p.$$

Fig. 9: Σ Protocol $\Pi_{\Sigma MULT}$ for R_{MULT} for $ELTA2E$.

12 Future Work

One future research work is to design an efficient two-party computation protocol for one-sided active adaptive adversary model, using the new efficient oblivious transfer protocol. Another research direction is to design efficient oblivious transfer protocol for the fully adaptive adversary model, that is, when the adversary may corrupt both parties at some point.

References

1. Yonatan Aumann and Yehuda Lindell. Security Against Covert Adversaries: Efficient Protocols for Realistic Adversaries. In *Proceedings of the 4th Conference on Theory of Cryptography*, TCC'07, pages 137–156, Berlin, Heidelberg, 2007. Springer-Verlag.
2. Yonatan Aumann and Yehuda Lindell. Security Against Covert Adversaries: Efficient Protocols for Realistic Adversaries. *Journal of Cryptology*, 23(2):281–343, 2010. (Preliminary version in TCC 2007, [1]).
3. Eric Bach. *Analytic Methods in the Analysis and Design of Number-theoretic Algorithms*. Massachusetts Institute of Technology, Cambridge, MA, USA, 1985.
4. Donald Beaver. Adaptive Zero Knowledge and Computational Equivocation (Extended Abstract). In *Proceedings of the Twenty-eighth Annual ACM Symposium on Theory of Computing*, STOC '96, pages 629–638, New York, NY, USA, 1996. ACM.
5. Mihir Bellare and Oded Goldreich. On Defining Proofs of Knowledge. In *Proceedings of the 12th Annual International Cryptology Conference on Advances in Cryptology*, CRYPTO '92, pages 390–420, London, UK, UK, 1993. Springer-Verlag.
6. Mihir Bellare, Dennis Hofheinz, and Scott Yilek. Possibility and Impossibility Results for Encryption and Commitment Secure under Selective Opening. In *Proceedings of the 28th Annual International Conference on Advances in Cryptology: the Theory and Applications of Cryptographic Techniques*, EUROCRYPT '09, pages 1–35, Berlin, Heidelberg, 2009. Springer-Verlag.
7. Mihir Bellare and Scott Yilek. Encryption Schemes Secure under Selective Opening Attack. Cryptology ePrint Archive, Report 2009/101, 2009. http://eprint.iacr.org/.
8. Gilles Brassard, David Chaum, and Claude Crépeau. Minimum Disclosure Proofs of Knowledge. *Journal of Computer and System Sciences*, 37(2):156–189, October 1988.
9. Ran Canetti. Security and Composition of Multiparty Cryptographic Protocols. *Journal of Cryptology*, 13(1):143–202, 2000.
10. Ran Canetti, Rosario Gennaro, Stanisław Jarecki, Hugo Krawczyk, and Tal Rabin. Adaptive Security for Threshold Cryptosystems. In *Proceedings of the 19th Annual International Cryptology Conference on Advances in Cryptology*, CRYPTO'99, pages 98–115, London, UK, 1999. Springer-Verlag.
11. Ran Canetti, Shai Halevi, and Jonathan Katz. Adaptively-Secure, Non-Interactive Public-key Encryption. In *Proceedings of the Second International Conference on Theory of Cryptography*, TCC'05, pages 150–168, Berlin, Heidelberg, 2005. Springer-Verlag.

12. Ran Canetti, Yehuda Lindell, Rafail Ostrovsky, and Amit Sahai. Universally Composable Two-Party and Multi-Party Secure Computation. Cryptology ePrint Archive, Report 2002/140, 2002. http://eprint.iacr.org/.
13. Seung Geol Choi, Dana Dachman-Soled, Tal Malkin, and Hoeteck Wee. Improved Non-committing Encryption with Applications to Adaptively Secure Protocols. In *Proceedings of the 15th International Conference on the Theory and Application of Cryptology and Information Security: Advances in Cryptology*, ASIACRYPT '09, pages 287–302, Berlin, Heidelberg, 2009. Springer-Verlag.
14. Ivan Damgård. On Σ-Protocols. www.cs.au.dk/ ivan/Sigma.pdf.
15. Ivan Damgård. Efficient Concurrent Zero-Knowledge in the Auxiliary String Model. In Bart Preneel, editor, *Advances in Cryptology EUROCRYPT 2000*, volume 1807 of *Lecture Notes in Computer Science*, pages 418–430. Springer Berlin Heidelberg, 2000.
16. Ivan Damgård and Jesper Buus Nielsen. Improved Non-committing Encryption Schemes Based on a General Complexity Assumption. In *Proceedings of the 20th Annual International Cryptology Conference on Advances in Cryptology*, CRYPTO '00, pages 432–450, London, UK, 2000. Springer-Verlag.
17. Taher El Gamal. A Public Key Cryptosystem and a Signature Scheme Based on Discrete Logarithms. In *Proceedings of CRYPTO 84 on Advances in Cryptology*, pages 10–18, New York, NY, USA, 1985. Springer-Verlag New York, Inc.
18. Pierre-Alain Fouque, Guillaume Poupard, and Jacques Stern. Sharing Decryption in the Context of Voting or Lotteries. In *Proceedings of the 4th International Conference on Financial Cryptography*, FC'00, pages 90–104, London, UK, UK, 2001. Springer-Verlag.
19. Yair Frankel, Philip D. MacKenzie, and Moti Yung. Adaptively-Secure Optimal-Resilience Proactive RSA. In *Proceedings of the International Conference on the Theory and Applications of Cryptology and Information Security: Advances in Cryptology*, ASIACRYPT '99, pages 180–194, London, UK, UK, 1999. Springer-Verlag.
20. Juan A. Garay, Daniel Wichs, and Hong-Sheng Zhou. Somewhat Non-committing Encryption and Efficient Adaptively Secure Oblivious Transfer. In *Proceedings of the 29th Annual International Cryptology Conference on Advances in Cryptology*, CRYPTO '09, pages 505–523, Berlin, Heidelberg, 2009. Springer-Verlag.
21. Carmit Hazay and Arpita Patra. One-Sided Adaptively Secure Two-Party Computation. In Yehuda Lindell, editor, *Theory of Cryptography*, volume 8349 of *Lecture Notes in Computer Science*, pages 368–393. Springer Berlin Heidelberg, 2014.
22. Stanisław Jarecki and Anna Lysyanskaya. Adaptively Secure Threshold Cryptography: Introducing Concurrency, Removing Erasures. In *Proceedings of the 19th International Conference on Theory and Application of Cryptographic Techniques*, EUROCRYPT'00, pages 221–242, Berlin, Heidelberg, 2000. Springer-Verlag.
23. Joe Kilian. Founding Crytpography on Oblivious Transfer. In *Proceedings of the Twentieth Annual ACM Symposium on Theory of Computing*, STOC '88, pages 20–31, New York, NY, USA, 1988. ACM.
24. Gillat Kol and Moni Naor. Cryptography and Game Theory: Designing Protocols for Exchanging Information. In *Proceedings of the 5th Conference on Theory of Cryptography*, TCC'08, pages 320–339, Berlin, Heidelberg, 2008. Springer-Verlag.
25. Yehuda Lindell and Hila Zarosim. Adaptive Zero-Knowledge Proofs and Adaptively Secure Oblivious Transfer. In *Proceedings of the 6th Theory of Cryptography Conference on Theory of Cryptography*, TCC '09, pages 183–201, Berlin, Heidelberg, 2009. Springer-Verlag.

26. Moni Naor, Rafail Ostrovsky, Ramarathnam Venkatesan, and Moti Yung. Perfect Zero-Knowledge Arguments for NP Using any One-Way Permutation. *Journal of Cryptology*, 11:87–108, 1998.
27. Jesper Buus Nielsen. *On Protocol Security in the Cryptographic Model*. PhD thesis, University of Aarhus, 2004.
28. Torben Pryds Pedersen. Non-Interactive and Information-Theoretic Secure Verifiable Secret Sharing. In *Proceedings of the 11th Annual International Cryptology Conference on Advances in Cryptology*, CRYPTO'91, pages 129–140, London, UK, 1992. Springer-Verlag.
29. Chris Peikert, Vinod Vaikuntanathan, and Brent Waters. A Framework for Efficient and Composable Oblivious Transfer. In *Proceedings of the 28th Annual Conference on Cryptology: Advances in Cryptology*, CRYPTO'08, pages 554–571, Berlin, Heidelberg, 2008. Springer-Verlag.
30. C.P. Schnorr. Efficient Signature Generation by Smart Cards. *Journal of Cryptology*, 4(3):161–174, 1991.
31. Huafei Zhu, Tadashi Araragi, Takashi Nishide, and Kouichi Sakurai. *Adaptive and Composable Non-committing Encryptions*, pages 135–144. Springer Berlin Heidelberg, Berlin, Heidelberg, 2010.

A Some Definitions

A.1 Zero-Knowledge Arguments

Brassard et al. [8] introduced zero-knowledge arguments. Let P denote the prover and V denote the verifier.

Definition 3. *(Zero-Knowledge Argument [26])*
A pair of interactive machines (P, V) is a ***zero-knowledge argument for a language*** *L if both machines are probabilistic polynomial time and the followings hold.*

1. ***Completeness.*** *For every $x \in L$, there exists a witness y such that for every string z,*
$$Pr\left[\langle P(y), V(z)\rangle(x) = 1\right] \geq \frac{2}{3}.$$
It is said that the ***completeness is perfect*** *if, for every $x \in L$, there exists a witness y such that for every string z,*
$$Pr\left[\langle P(y), V(z)\rangle(x) = 1\right] = 1.$$
2. ***Computational Soundness.*** *For every probabilistic polynomial-time interactive machine B and all sufficiently long $x \notin L$, and every y and z,*
$$Pr\left[\langle B(y), V(z)\rangle(x) = 1\right] \leq \frac{1}{3}.$$
3. ***Zero-Knowledge.*** *For every probabilistic polynomial-time interactive machine V^*, there exists a probabilistic algorithm M^*, running in time polynomial in the length of its first input, such that, on any positive instance $x \in L$ and auxiliary input y for the prover and z for the verifier, the following two ensembles are computationally indistinguishable (when the distinguishing gap is considered as a function of $|x|$):*

- $\{\langle P(y), V^*(z)\rangle(x)\}_{x\in L, z\in\{0,1\}^*}$
- $\{M^*(x,z)\}_{x\in L, z\in\{0,1\}^*}$

A.2 Adaptive Zero-Knowledge Arguments

Beaver defined adaptive zero-knowledge proofs [4]. An adaptive zero-knowledge proof system is a zero-knowledge proof system secure against adaptive adversaries. It is defined through the existence of polynomial-time simulator for adaptive adversaries. Similarly, an *adaptive zero-knowledge argument* is a zero-knowledge argument secure against adaptive adversaries.
Figure 10 defines the ideal functionality $\mathcal{F}_{ZK}^{R}$ for adaptive zero-knowledge argument. $\mathcal{F}_{ZK}^{R}$ returns 1 only if the instance, witness pair (x, w) satisfies the relation R. $\mathcal{F}_{ZK}^{R}$ works in the common reference string model.

Functionality $\mathcal{F}_{ZK}^{R}$.
Common Reference String : μ.
$\mathcal{F}_{ZK}^{R}$ proceeds as follows, running with a prover P, a verifier V, the common reference string μ, an adaptive adversary $\mathcal{A}$, and parametrized with a relation R:

- Upon receiving $(\text{ZK-prover}, V, x, w)$ from P, do:
  if $R(x, w) = 1$, then
  send $(\text{ZK-proof}, x)$ to V and $\mathcal{A}$, and halt.
  Otherwise, halt.

Fig. 10: Ideal functionality $\mathcal{F}_{ZK}^{R}$.

A.3 Σ-Protocols

Damgård [14] defined Σ-protocols. In this dissertation, Σ-protocol is used as a tool in the design of adaptive zero-knowledge arguments. The definition of Σ-protocols is given below, following [14,27].
Let R be a binary relation; that is, R is a subset of $\{0,1\}^* \times \{0,1\}^*$, where the only restriction is that if $(x, w) \in R$, then the length of w is at most $p(|x|)$, for some polynomial $p()$. For some$(x, w) \in R$, x may be thought as an instance of some computational problem, and w as the solution to that instance. w is called a *witness* for x.

Definition 4. *(Σ-Protocols [14,27])*
*A protocol Π is said to be a **Σ-protocol for relation** R if the following properties hold.*

- *Let x be the common input to P, V. P has a private input w such that $(x, w) \in R$. Protocol Π is of the following form:*

1. P sends a message a.
2. V sends a challenge $e \xleftarrow{\$} \{0,1\}^t$.
3. P sends a reply z.
V decides to accept or reject based on the data it has seen, that is, (x, a, e, z).

– ***Completeness:***
If P, V follow the protocol on input x and private input w to P where $(x, w) \in R$, V always accepts.
– ***Special Soundness:***
From any x and any pair $\{(a, e, z), (a, e', z')\}$ of accepting conversations on input x where $e \neq e'$, one can efficiently compute w such that $(x, w) \in R$.
– ***Special Honest-Verifier Zero-Knowledge:***
There exists a polynomial-time simulator hvs, which on input x and a random e, outputs an accepting conversation of the form (a, e, z), with the same probability distribution as conversations between the honest P, V on input x. The algorithm hvs is called the honest verifier simulator. hvs takes as input $x \in L(R)$, $e \in \{0,1\}^t$, and a uniformly random bit-string r_{hvs}, and produces as output (a, z) which is supposed to be distributed as the (a, z) produced by a honest prover with instance x receiving challenge e.

A.4 Non-Erasure Σ-Protocols

Nielsen [27] defined non-erasure Σ-protocols.

Definition 5. *(Non-Erasure Σ-Protocols [27])*
A protocol Π is said to be a ***non-erasure Σ-protocol for relation*** *R if Π is a Σ-protocol and the following property holds.*
Special Non-Erasure Honest Verifier Zero-Knowledge:
Let $EXEC(x, w, e) = (x, w, a, r_p, e, z)$ where (a, e, z) is the sequence of messages of the Σ-protocol Π, and r_p denotes the randomness used by P in protocol Π. Let $SIM(x, w, e) = (x, w, a', r'_p, e, z')$ where $(a', z') = hvs(x, e, r_{hvs})$, r_{hvs} denotes the randomness used by simulator hvs, and
$r'_p = rbs(x, w, e, r_{hvs})$. The algorithm rbs is called the random bits simulator. rbs takes as input $(x, w) \in R$, a challenge $e \in \{0,1\}^t$ and the randomness r_{hvs} used by hvs in a run $(a', z') \leftarrow hvs(x, e, r_{hvs})$. rbs produces as output a bit-string r'_p where r'_p is the randomness used by P in protocol Π. Then, the random variables $EXEC(x, w, e)$ and $SIM(x, w, e)$ are identically distributed for all $(x, w) \in R$ and all $e \in \{0,1\}^t$.

That means, in the definition of non-erasure Σ-protocols, the extra property *non-erasure* that is required from the definition of Σ-protocols is the simulation of random bits. This property is necessary when a Σ-protocol is used in a multi-party protocol secure against erasure-free adaptive adversaries. For erasure-free adaptive adversaries, the simulator has to supply randomness for a party when a party gets corrupted in the middle of a protocol or after the protocol is finished. The random bit simulator algorithm *rbs* is useful to generate randomness in this case.

Proof of Knowledge Bellare and Goldreich [5] defined the concept of proof of knowledge. Their definition is given below.

Definition 6. *(Proof of Knowledge [5]) Let $\kappa()$ be a function from bit strings to the interval $[0 \dots 1]$. The protocol Π is called a* ***proof of knowledge*** *for relation R with knowledge error κ, if the following two properties are satisfied.*

1. ***Completeness:*** *On common input x, if the honest prover P gets w as private input where $(x, w) \in R$, then the verifier V always accepts.*
2. ***Knowledge Soundness:*** *There exists a probabilistic polynomial-time algorithm M called the* ***knowledge extractor****. The algorithm M gets input x and rewindable black-box access to the prover and tries to compute w such that $(x, w) \in R$. For any prover P^*, let $\epsilon(x)$ be the probability that V accepts on input x. There exists a constant c such that whenever $\epsilon(x) > \kappa(x)$, M outputs a correct w in expected time at most*

$$\frac{|x|^c}{\epsilon(x) - \kappa(x)}$$

where access to P^* counts as one step only.

Damgård [14] proved that a Σ-protocol for relation R with challenge length t is a proof of knowledge with knowledge error 2^{-t}. That means, an adaptive zero-knowledge argument converted from a Σ-protocol is also a proof of knowledge. The knowledge extractor of this type of adaptive zero-knowledge arguments are used in the security proofs later.

B Security Proof of Protocol Π^R_{AdZKA}

Protocol Π^R_{AdZKA} is presented in Figure 1. The security proof of Π^R_{AdZKA} against an adaptive adversary is given below, following [27].

Lemma 6. *If trapdoor commitment scheme exists and $\Pi_{\Sigma R}$ is a non-erasure Σ-protocol for relation R, then Π^R_{AdZKA} is an adaptive zero-knowledge argument for relation R.*

Proof. The simulator $\mathcal{S}_{AdZKA}$ for Π^R_{AdZKA} works as follows. $\mathcal{S}_{AdZKA}$ selects $\delta \overset{\$}{\leftarrow} \mathbb{Z}_q$ and sets common reference string to $\mu = g^\delta \bmod p$. The common reference string μ is used as the commitment key for Pederesn commitment scheme, so $\mathcal{S}_{AdZKA}$ knows the trapdoor of the commitment key μ.

1. If P is corrupted, then receive c from $\mathcal{A}$.
   If P is honest, then generate a challenge $e_1 \overset{\$}{\leftarrow} \{0,1\}^t$. Select $r_{hvs} \overset{\$}{\leftarrow} coins(hvs)$ and compute $(a_1, z_1) \leftarrow hvs(x, e_1, r_{hvs})$. Select $r_{c1} \overset{\$}{\leftarrow} coins(Com)$. Compute $c = Com_\mu(a_1, r_{c1})$. Send c to V.
   If $\mathcal{A}$ corrupts P after this step, then corrupt P in the ideal world. Receive (x, w) from $\mathcal{Z}$. Compute $r_{p1} \leftarrow rbs(x, w, e_1, r_{hvs})$. Send (r_{p1}, r_{c1}) as the randomness used by P to environment $\mathcal{Z}$.

2. If V is corrupted, then receive e from $\mathcal{A}$.
   If V is honest, then send challenge $e = e_1$.
   If $\mathcal{A}$ corrupts V after this step, then corrupt V in the ideal world.
3. If V is honest and P is honest, then send (z_1, a_1, r_{c1}).
   If $\mathcal{A}$ corrupts P after this step, then perform the same steps if P was corrupted after step 1.
   If V is honest and P is corrupted, then receive (z, a, r_c) from $\mathcal{A}$. Act as an honest verifier. If (a, e_1, z) is not an accepting conversation of $\Pi_{\Sigma R}$ or if $Com_\mu(a, r_c) \neq c$, then send $abort_P$ to the trusted party and halt.
   If $\mathcal{A}$ corrupts V after this step, then corrupt V in the ideal world.
   If V is corrupted, then, with overwhelming probability, $e \neq e_1$. If P is honest, then proceed as follows. Generate $(a_2, z_2) \leftarrow hvs(x, e, r_{hvs})$. Using trapdoor δ of the commitment key μ, compute r_{c2} such that $c = Com_\mu(a_2, r_{c2})$. Send (a_2, r_{c2}, z_2) to V.
   If $\mathcal{A}$ corrupts P after this step, then corrupt P in the ideal world. Receive (x, w) from $\mathcal{Z}$. Compute $r_{p2} \leftarrow rbs(x, w, e, r_{hvs})$. Send (r_{p2}, r_{c2}) as the randomness used by P to environment $\mathcal{Z}$.

Next, it is proved why the global output generated by $\mathcal{S}_{AdZKA}$ is computationally indistinguishable from the global output in the real world.

1. If P is corrupted, then $\mathcal{S}_{AdZKA}$ receives c from $\mathcal{A}$.
   If P is honest, then $\mathcal{S}_{AdZKA}$ generates a challenge $e_1 \overset{\$}{\leftarrow} \{0,1\}^t$. Then, $\mathcal{S}_{AdZKA}$ selects $r_{hvs} \overset{\$}{\leftarrow} coins(hvs)$ and computes $(a_1, z_1) \leftarrow hvs(x, e_1, r_{hvs})$. $\mathcal{S}_{AdZKA}$ selects $r_{c1} \overset{\$}{\leftarrow} coins(Com)$ and computes $c = Com_\mu(a_1, r_{c1})$. $\mathcal{S}_{AdZKA}$ sends c to V. In the real world, honest P computes its first message a using its input and randomness. Then, honest P selects $r_c \overset{\$}{\leftarrow} coins(Com)$ and computes $c = Com_\mu(a, r_c)$. Honest P sends c to V. By the unconditional hiding property of Pedersen commitment scheme, the distribution of c in two worlds are identical.
   If $\mathcal{A}$ corrupts P after this step, then $\mathcal{S}_{AdZKA}$ corrupts P in the ideal world. $\mathcal{S}_{AdZKA}$ receives the input (x, w) of P from $\mathcal{Z}$. As P was honest, it holds that $(x, w) \in R$. $\mathcal{S}_{AdZKA}$ computes $r_{p1} \leftarrow rbs(x, w, e_1, r_{hvs})$. $\mathcal{S}_{AdZKA}$ sends (r_{p1}, r_{c1}) as the randomness used by P to environment $\mathcal{Z}$. The "special non-erasure honest verifier zero-knowledge" property of $\Pi_{\Sigma R}$ implies that r_{p1} is identically distributed to the randomness used by honest P in step 1 of $\Pi_{\Sigma R}$ in the real world. In the ideal world, $r_{c1} \overset{\$}{\leftarrow} coins(Com)$. In the real world, $r_c \overset{\$}{\leftarrow} coins(Com)$. So, the distribution of r_{c1} and r_c are identical.
2. If V is honest, then $\mathcal{S}_{AdZKA}$ sends challenge $e = e_1$. The challenge in two worlds are identically distributed.
   If $\mathcal{A}$ corrupts V after this step, then $\mathcal{S}_{AdZKA}$ corrupts V in the ideal world. There is no private input of V in the protocol, so $\mathcal{S}_{AdZKA}$ does not receive any input.
3. If V is honest and P is honest, then $\mathcal{S}_{AdZKA}$ sends (z_1, a_1, r_{c1}). $\mathcal{S}_{AdZKA}$ computed $(a_1, z_1) \leftarrow hvs(x, e_1, r_{hvs})$ in step 1. The "special honest verifier

zero-knowledge" property of $\Pi_{\Sigma R}$ implies that (a_1, z_1) is identically distributed to (a, z) where (a, e_1, z) is the sequence of message between honest P and honest V in the real world, for challenge e. As argued in step 1, the distribution of r_{c1} and r_c are identical.

If $\mathcal{A}$ corrupts P after this step, then $\mathcal{S}_{AdZKA}$ performs the same steps if P was corrupted after step 1.

If V is honest and P is corrupted, then $\mathcal{S}_{AdZKA}$ receives (z, a, r_c) from $\mathcal{A}$. $\mathcal{S}_{AdZKA}$ acts as an honest verifier. If (a, e_1, z) is not an accepting conversation of $\Pi_{\Sigma R}$ or if $Com_\mu(a, r_c) \neq c$, then $\mathcal{S}_{AdZKA}$ sends $abort_P$ to the trusted party and halts. In this case, honest V detects the same problem in the real world and aborts.

If $\mathcal{A}$ corrupts V after this step, then $\mathcal{S}_{AdZKA}$ corrupts V in the ideal world.

If V is corrupted, then, with overwhelming probability, $e \neq e_1$. If P is honest, then $\mathcal{S}_{AdZKA}$ generates $(a_2, z_2) \leftarrow hvs(x, e, r_{hvs})$. Using trapdoor δ of the commitment key μ, $\mathcal{S}_{AdZKA}$ computes r_{c2} such that $c = Com_\mu(a_2, r_{c2})$. $\mathcal{S}_{AdZKA}$ sends (a_2, r_{c2}, z_2) to V. By the "special honest verifier zero-knowledge" property of $\Pi_{\Sigma R}$, $(a2, z_2)$ is identically distributed to (a_3, z_3) where (a_3, e, z_3) is the sequence of message between honest P and honest V in the real world, for challenge e. By the trapdoor property of Pedersen commitment scheme, the opening r_{c2} computed is consistent. In the ideal world, P passes the proof. In the real world, honest P passes the proof. The randomness r_{c2} is identically distributed to the randomness used by honest P in the ideal world during commitment.

If $\mathcal{A}$ corrupts P after this step, then $\mathcal{S}_{AdZKA}$ corrupts P in the ideal world. $\mathcal{S}_{AdZKA}$ receives the input (x, w) of P from $\mathcal{Z}$. As P was honest, it holds that $(x, w) \in R$. $\mathcal{S}_{AdZKA}$ computes $r_{p2} \leftarrow rbs(x, w, e, r_{hvs})$. $\mathcal{S}_{AdZKA}$ sends (r_{p2}, r_{c2}) as the randomness used by P to environment $\mathcal{Z}$. By the "special non-erasure honest verifier zero-knowledge" property of $\Pi_{\Sigma R}$, r_{p2} is identically distributed to the randomness used by honest P in step 1 of $\Pi_{\Sigma R}$ in the real world. As argued above, the randomness r_{c2} is identically distributed to the randomness used by honest P in the ideal world during commitment.

C Proof of Theorem 1

Let $\mathcal{A}$ be an one-sided adaptive adversary and $\mathcal{Z}$ be the environment. Let $\mathcal{S}_{DKG}$ be the simulator of protocol Π_{DKG} for adversary $\mathcal{A}$ and environment $\mathcal{Z}$. The inputs of $\mathcal{S}_{DKG}$ are the public key $pk = (q, g, j, h, \ell)$, mode parameter $mode$, and the identity I of the SIP.

At start of simulation, the simulator $\mathcal{S}_{DKG}$ generates $\delta \overset{\$}{\leftarrow} \mathbb{Z}_p$, and sets the common reference string to $\mu = g^\delta$. As the common reference string μ is used as the commitment key, so $\mathcal{S}_{DKG}$ knows the trapdoor δ of the commitment key μ. The simulator for the two cases where the SIP is P_1 and P_2, are presented separately.

Simulation for the case where P_1 is the SIP

Simulator $\mathcal{S}_{DKG}$ for protocol Π_{DKG} for the case $I = P_1$.

1. Select $\alpha_2, \gamma_2 \xleftarrow{\$} \mathbb{Z}_q$. Compute $h_2 = g^{\alpha_2}, j_2 = g^{\gamma_2}$. Compute $h_1 = \frac{h}{h_2}, j_1 = \frac{j}{j_2}$.
   Select $\beta_1, \theta_1 \xleftarrow{\$} \mathbb{Z}_q$. Compute $b_1 = Com_\mu(h_1, \beta_1), c_1 = Com_\mu(j_1, \theta_1)$.
2. Act as an honest prover in the proofs.
3. If P_2 is honest, then use h_2, j_2 computed in step 1.
4. If P_2 is corrupted, then act as an honest verifier.
   If P_2 passes the proofs, then extract α_2 and γ_2.
   If P_2 fails in any proof, then send $abort_{P_2}$ to the trusted party and halt.
   If P_2 is honest, then act as an honest prover in the proofs.
5. Compute $\hat{h}_1 = \frac{h}{h_2}, \hat{j}_1 = \frac{j}{j_2}$.
   Using trapdoor δ, compute randomness $\hat{\beta}_1$ and $\hat{\theta}_1$ for commitments that are consistent to $\hat{h}_1$ and $\hat{j}_1$.
   Use $(\hat{h}_1, \hat{\beta}_1), (\hat{j}_1, \hat{\theta}_1)$ as the message from P_1 for this step.
6. Proceed to next step.
7. Set $vk = g, vk_1 = \hat{h}_1, vk_2 = h_2$.
   Use (h, j) of the input.
8. Compute $\hat{\ell}_1 = \frac{\ell}{j^{(\alpha_2)}}$.
9. If $mode = 1$, then generate a proof transcript using the simulator of the zero-knowledge argument and trapdoor δ.
   If P_2 is honest, then act as an honest verifier.
10. If P_2 is honest, then honestly compute ℓ_2.
11. If P_2 is corrupted, then act as an honest verifier.
    If P_2 fails, then send $abort_{P_2}$ to the trusted party and halt.
    If P_2 is honest, then act as an honest prover.
12. Use (ℓ, pk) of input.
13. Set $(pk, vk, vk_1, vk_2, \alpha_1)$ as the output of P_1.
    If P_2 is honest, then set $(pk, vk, vk_1, vk_2, \alpha_2)$ as the output of P_2.

Fig. 11: Simulator $\mathcal{S}_{DKG}$ for protocol Π_{DKG} for the case $I = P_1$.

The simulator $\mathcal{S}_{DKG}$ for the case where P_1 is the SIP, that is, $I = P_1$, is presented in Figure 11.

Since P_1 is the SIP, P_1 is uncorrupted in this case. If $\mathcal{A}$ corrupts P_2 after any step of protocol Π_{DKG}, then $\mathcal{S}_{DKG}$ corrupts P_2 in the ideal world. There is no input of any party in this protocol, so after corrupting a party, $\mathcal{S}_{DKG}$ does not learn any input.

Next, it is proved why the global output produced by $\mathcal{S}_{DKG}$ is computationally indistinguishable from the global output in the real world. If P_2 is honest, then $\mathcal{S}_{DKG}$ performs the actions on behalf of P_2 in the simulation. The simulator is designed in such a way that if, $\mathcal{A}$ corrupts P_2 after any step of Π_{DKG}, then $\mathcal{A}$ would see that P_2 performed up to that step honestly. If P_2 is corrupted and P_2 fails in some proof, then the honest P_1 aborts in the real world. In the ideal world, $\mathcal{S}_{DKG}$ sends $abort_{P_2}$ to the trusted party and halts. Then, the trusted party sends $abort_{P_2}$ to P_1 and the honest P_1 aborts. In that case, P_2 fails in the real world and honest P_1 aborts in the real world.

If P_2 does not fail in any proof, then the following things happen.

1. $\mathcal{S}_{DKG}$ selects $\alpha_2, \gamma_2 \overset{\$}{\leftarrow} \mathbb{Z}_q$, computes $h_2 = g^{\alpha_1}, j_2 = g^{\gamma_2}, h_1 = \frac{h}{h_2}, j_1 = \frac{j}{j_2}$. $\mathcal{S}_{DKG}$ selects $\beta_1, \theta_1 \overset{\$}{\leftarrow} \mathbb{Z}_q$ and computes $b_1 = Com_\mu(h_1, \beta_1), c_1 = Com_\mu(j_1, \theta_1)$. By the hiding property of the commitment scheme, the distribution of (b_1, c_1) in two worlds are identical.
2. $\mathcal{S}_{DKG}$ honestly performs step 2. The proof transcript in two worlds are identically distributed.
3. If P_2 is honest, then $\mathcal{S}_{DKG}$ uses h_2, j_2 computed in step 1. Since $\mathcal{S}_{DKG}$ honestly computed them, the distribution of (h_2, j_2) are identical in two worlds.
4. If P_2 is corrupted, $\mathcal{S}_{DKG}$ acts as an honest verifier. If P_2 passes the proofs, then $\mathcal{S}_{DKG}$ extracts α_2 and γ_2 using the knowledge extractor of the zero-knowledge argument. Since a Σ-protocol is a proof of knowledge [14], any adaptive zero-knowledge argument converted from a Σ-protocol is a proof of knowledge. By definition, there exists a probabilistic polynomial-time knowledge extractor that, given common input x, can extract an input, witness pair $(x, w) \in R$. If P_2 is honest, then $\mathcal{S}_{DKG}$ acts as an honest prover.
5. $\mathcal{S}_{DKG}$ computes $\hat{h}_1 = \frac{h}{h_2}, \hat{j}_1 = \frac{j}{j_2}$. Using the trapdoor δ of the commitment key μ, $\mathcal{S}_{DKG}$ computes $\hat{\beta}_1, \hat{\theta}_1$ such that $b_1 = Com_\mu(\hat{h}_1, \hat{\beta}_1)$, and $c_1 = Com_\mu(\hat{j}_1, \hat{\theta}_1)$. $\mathcal{S}_{DKG}$ uses $(\hat{h}_1, \hat{\beta}_1), (\hat{j}_1, \hat{\theta}_1)$ as the message from P_1.
   If $\mathcal{A}$ corrupts P_2 before step 3, then, corrupted P_2 sends h_2 and j_2 in step 3. The value of h and j are fixed since they are part of the input of $\mathcal{S}_{DKG}$. $\mathcal{A}$ sees that the openings of the commitments are consistent, and $h = \hat{h}_1 h_2$ and $j = \hat{j}_1 j_2$, as required. h_2 is supplied by $\mathcal{A}$ in step 3 in both worlds. Since the message seen by $\mathcal{A}$ up to step 2 is identically distributed in two worlds, the distribution of h_2 in two worlds are identical. Then, the value of $\hat{h}_1 = \frac{h}{h_2}$ is identically distributed to the value of h_1. Similarly, $\hat{j}_1$ and j_1 are identically distributed. By the trapdoor property of the commitment scheme, the values $(\hat{\beta}_1, \hat{\theta}_1)$ are consistent.

If P_2 is honest up to step 3, then $\hat{h}_1 = \frac{h}{h_2} = h_1$. $\hat{\beta}_1 = \beta_1$ since $\hat{h}_1 = h_1$. Similarly, $\hat{j}_1 = j_1$, and $\hat{\theta}_1 = \theta_1$. In both cases, openings of the commitments are identically distributed in two worlds.

6. P_1 passes the verification tests in the ideal world.
7. $\mathcal{S}_{DKG}$ sets $vk = g, vk_1 = \hat{h}_1, vk_2 = h_2$, and uses (h, j) of the input. As argued in step 5 and 3, either $\hat{h}_1 = h_1$ or $\hat{h}_1$ is identically distributed to h_1, and the distribution of h_2 in two worlds are identical.
8. $\mathcal{S}_{DKG}$ computes $\hat{\ell}_1 = \frac{\ell}{j^{(\alpha_2)}}$ where α_2 is the value extracted in step 3. In the real world, if $\ell_2 \neq j^{\alpha_2}$, then P_2 would fail the proof in step 11 and honest P_1 would abort. Since the full key is given as input, so Π_{DKG} executed fully, $\ell_2 = j^{(\alpha_2)}$, so $\ell_1 = \frac{\ell}{j^{(\alpha_2)}}$ in the real world. ℓ is fixed in both worlds.

   If $\mathcal{A}$ corrupts P_2 before step 3, then $\mathcal{A}$ selects α_2 in step 3 in both worlds. Since the message seen by $\mathcal{A}$ up to step 2 is identically distributed in two worlds, the distribution of α_2 in two worlds are identical. Then, $\hat{\ell}_1$ and ℓ_1 are identically distributed.

   If P_2 is honest up to step 3, then $\mathcal{S}_{DKG}$ selects $\alpha_2 \xleftarrow{\$} \mathbb{Z}_q$ in step 1. The distribution of α_2 in two worlds are identical. Then, $\hat{\ell}_1$ and ℓ_1 are identically distributed.
9. $\mathcal{S}_{DKG}$ generates a proof transcript using trapdoor δ. By definition of zero-knowledge argument, the proof transcript in two worlds are computationally indistinguishable.
10. If P_2 is honest, then $\mathcal{S}_{DKG}$ honestly performs step 10. The distribution of ℓ_1 in the two worlds are identical.
11. If P_2 is corrupted, $\mathcal{S}_{DKG}$ acts as an honest verifier. If P_2 is honest, then $\mathcal{S}_{DKG}$ acts as an honest prover.
12. $\mathcal{S}_{DKG}$ uses (ℓ, pk) of input. They are identical in two worlds.
13. The output of honest P_1 is $(pk, vk, vk_1, vk_2, \alpha_1)$. Then, the output of the honest P_1 in two worlds are identically distributed.
14. If P_2 is honest, then $\mathcal{S}_{DKG}$ sets $(pk, vk, vk_1, vk_2, \alpha_2)$ as the output of P_2. Only the last part α_2 of the output of P_2 is different from the last part α_1 of the output of P_1. $\mathcal{S}_{DKG}$ computed $sk_2 = \alpha_2$ honestly on behalf of P_2 in step 5, so the distribution of sk_2 in two worlds are identical.

If $\mathcal{A}$ corrupts P_2 after the execution of Π_{DKG} is finished, then $\mathcal{S}_{DKG}$ corrupts P_2 in the ideal world, and gets the output $(pk, vk, vk_1, vk_2, sk_2)$ of P_2.

Simulation for the case P_2 is the SIP

The simulator $\mathcal{S}_{DKG}$ for the case where P_2 is the SIP, that is, $I = P_2$, is presented in Figure 12.

Next, it is proved why the global output produced by $\mathcal{S}_{DKG}$ is computationally indistinguishable from the global output in the real world. If P_1 is honest, then $\mathcal{S}_{DKG}$ performs the actions on behalf of P_1 in the simulation. The simulator is designed in such a way that if, $\mathcal{A}$ corrupts P_1 after any step of Π_{DKG}, then $\mathcal{A}$ would see that P_1 performed up to that step honestly. If P_1 is corrupted and P_1 fails in some proof, then the honest P_2 aborts in the real world. In the ideal world, $\mathcal{S}_{DKG}$ sends $abort_{P_1}$ to the trusted party and halts. Then, the trusted

Simulator $\mathcal{S}_{DKG}$ for protocol $\mathit{\Pi}_{DKG}$ for the case $I = P_2$.

1. If P_1 is honest, then honestly perform step 1.
2. If P_1 is corrupted, then act as an honest verifier in the proofs.
   If P_1 fails in any proof, then send $abort_{P_1}$ to the trusted party and halt.
   If P_1 passes the proofs, then extract h_1 and j_1.
   If P_1 is honest, then act as an honest prover.
3. Set $\hat{h}_2 = \frac{h}{h_1}$, and $\hat{j}_2 = \frac{j}{j_2}$.
   Use $(\hat{h}_2, \hat{j}_2)$ as the message from P_2 for this step.
4. Generate proof transcripts using the simulator of the zero-knowledge argument and trapdoor δ.
5. If P_1 is honest, then use $((h_1, \beta_1), (j_1, \theta_1))$ selected in step 1 as the message from P_1 for this step.
6. If P_1 is corrupted, then honestly verify the equalities on behalf of P_2. If any of them does not hold, then send $abort_{P_1}$ to the trusted party and halt.
   If P_1 is honest, then proceed to next step.
7. Set $vk = g, vk_1 = h_1, vk_2 = \hat{h}_2$.
   Use h, j of input.
8. If P_1 is honest, then compute ℓ_1 honestly.
9. If P_1 is corrupted and $mode = 1$, then act as an honest verifier in the proof. If P_1 fails the proof, then send $abort_{P_1}$ to the trusted party and halt.
   If P_1 is honest, then act as an honest prover.
10. Compute $\hat{\ell}_2 = \frac{\ell}{\ell_1}$.
11. Generate a proof transcript using the simulator of the zero-knowledge argument and trapdoor δ.
12. Use (ℓ, pk) of input.
13. If P_1 is honest, then set $(pk, vk, vk_1, vk_2, \alpha_1)$ as the output of P_1.
    Select $\alpha_2 \overset{\$}{\leftarrow} \mathbb{Z}_q$.
    Set $sk_2 = \alpha_2$.
    Use $(pk, vk, vk_1, vk_2, sk_2)$ as the output of P_2.

Fig. 12: Simulator $\mathcal{S}_{DKG}$ for protocol $\mathit{\Pi}_{DKG}$ for the case $I = P_2$.

party sends $abort_{P_1}$ to P_2 and the honest P_2 aborts. In that case, P_1 fails in the real world and honest P_2 aborts in the real world.
If P_1 does not fail in any proof, then the following thing happen.

1. If P_1 is honest, then $\mathcal{S}_{DKG}$ honestly performs step 1 on behalf of P_1. The distribution of (b_1, c_1) in two worlds are identical.
2. If P_1 is corrupted, then $\mathcal{S}_{DKG}$ acts as an honest verifier in the proofs. If P_1 passes the proofs, then $\mathcal{S}_{DKG}$ extracts h_1 and j_1 by using the knowledge extractor of the zero-knowledge arguments.

   If P_1 is honest, then $\mathcal{S}_{DKG}$ acts as an honest prover using (h_1, β_1) and (j_1, θ_1) selected in step 1 as the witness. The proof transcript in two worlds are identically distributed.
3. $\mathcal{S}_{DKG}$ computes $\hat{h}_2 = \frac{h}{h_1}$, and $\hat{j}_2 = \frac{j}{j_1}$.

   If P_1 is corrupted, then $\mathcal{S}_{DKG}$ extracts the values of h_1 and j_1 in step 2. Since h_1 and j_1 are sent by $\mathcal{A}$ in step 1 in both worlds, the distribution of (h_1, j_1) in two worlds are identical. Since the value of (h, j) are fixed, the distribution of (h_2, j_2) in two worlds are identical.

   If P_1 is honest, then $\mathcal{S}_{DKG}$ selected (h_1, j_1) honestly in step 1 on behalf of P_1. In the real world, honest P_1 selects (h_1, j_1) similarly. Then, the distribution of (h_1, j_1) in two worlds are identical. Since the value of (h, j) are fixed, the distribution of (h_2, j_2) in two worlds are identical.
4. $\mathcal{S}_{DKG}$ generates proof transcripts using the simulator of the zero-knowledge argument and trapdoor δ. In the real world, honest P_2 acts as an honest prover based on witness α_2 and γ_2. By definition of zero-knowledge arguments, the proof transcript in two worlds are computationally indistinguishable.
5. If P_1 is honest, then $\mathcal{S}_{DKG}$ uses $((h_1, \beta_1), (j_1, \theta_1))$ selected in step 1 as the message from P_1 for this step. Since $\mathcal{S}_{DKG}$ performed step 1 honestly, the distribution of $(h_1, \beta_1, j_1, \theta_1)$ in two worlds are identical.
6. If P_1 is corrupted, then $\mathcal{S}_{DKG}$ honestly verifies the equalities on behalf of honest P_2. If any of them does not hold, then $\mathcal{S}_{DKG}$ sends $abort_{P_1}$ to the trusted party and halts. The trusted party sends $abort_{P_1}$ to P_2 and halts. Honest P_2 halts. In this case, in the ideal world, the honest P_2 finds out that some of the equalities do not hold. So, honest P_2 aborts in the ideal world.

   If P_1 is honest, then $\mathcal{S}_{DKG}$ proceeds to next step. As $\mathcal{S}_{DKG}$ performed step 1 honestly, P_1 is supposed to pass the tests in the ideal world. In the real world, honest P_1 passes the tests.
7. $\mathcal{S}_{DKG}$ sets $vk = g, vk_1 = h_1, vk_2 = \hat{h}_2$. $\mathcal{S}_{DKG}$ uses (h, j) of the input. As argued in step 3, the distribution of h_1 and h_2 in two worlds are identical. $\mathcal{S}_{DKG}$ uses (h, j) of the input. These values are identical in two worlds.
8. If P_1 is honest, then $\mathcal{S}_{DKG}$ computes ℓ_1 honestly. The distribution of ℓ_1 in two worlds are identical.
9. If P_1 is corrupted and $mode = 1$, then $\mathcal{S}_{DKG}$ acts as an honest verifier in the proof. If P_1 fails in some proof, then $\mathcal{S}_{DKG}$ sends $abort_{P_1}$ to the trusted party and halts. Proof argument is similar to step 2 of stage 1.

If P_1 is honest, then $\mathcal{S}_{DKG}$ acts as an honest prover using α_1 selected in step 1 as the witness. The proof transcript in two worlds are identically distributed.

10. $\mathcal{S}_{DKG}$ computes $\hat{\ell}_2 = \frac{\ell}{\ell_1}$. The value of ℓ is fixed as it is part of the input. If $\mathcal{A}$ corrupts before step 1 of stage 2, then $\mathcal{A}$ sends ℓ_1 in step 1 of stage 2.
If $mode = 1$, then $\ell_1 = j^{\alpha_1}$. The reason is that if $\ell_1 \neq j^{\alpha_1}$ then corrupted P_1 would fail in step 2 of stage 2, honest P_2 would abort and the execution of Π_{DKG} would stop at that point. Since the full key is generated and given as input to the simulator, it holds that $\ell_1 = j^{\alpha_1}$ in the real world. Since the value seen by $\mathcal{A}$ up to step 7 of stage 1 in two worlds are identically distributed, ℓ_1 supplied by $\mathcal{A}$ in step 1 of stage 2 in two worlds are identically distributed. The value of ℓ is fixed in two worlds as it is party of the input. Then, $\hat{\ell}_2$ and ℓ_2 are identically distributed.
If $mode = 0$, then ℓ_1 supplied by $\mathcal{A}$ in step 1 of stage 2 in two worlds are identically distributed, since the value seen by $\mathcal{A}$ up to step 7 of stage 1 in two worlds are identically distributed. Then, $\hat{\ell}_2$ and ℓ_2 are identically distributed.
If P_1 is honest up to step 1 of stage 2, then $\mathcal{S}_{DKG}$ selects α_1 honestly in step 1 and computes $\ell_1 = j^{\alpha_1}$ in step 1 of stage 2. The distribution of ℓ_1 is identical in two worlds and the value of ℓ is fixed. So, the distribution of ℓ_2 in two worlds are identical.
11. $\mathcal{S}_{DKG}$ generates a proof transcript using the simulator of the zero-knowledge argument and trapdoor δ. In the real world, honest P_2 acts as an honest prover based on witness α_2. By definition of zero-knowledge arguments, the proof transcript in two worlds are computationally indistinguishable.
12. $\mathcal{S}_{DKG}$ uses (ℓ, pk) of input. These values are identical in two worlds. If P_1 is honest, then $\mathcal{S}_{DKG}$ sets $(pk, vk, vk_1, vk_2, \alpha_1)$ as the output of P_1. pk and vk are identical in both worlds. As argued in step 7 of stage 1, the distribution of vk_1 and vk_2 in two worlds are identical. $\mathcal{S}_{DKG}$ computed $sk_1 = \alpha_1$ honestly on behalf of P_1 in step 1 of stage 1, so the distribution of sk_1 in two worlds are identical.
The output of honest P_2 is $(pk, vk, vk_1, vk_2, \alpha_2)$. Only the last part α_2 of the output of P_2 is different from the last part α_1 of the output of P_1. $\mathcal{S}_{DKG}$ selects $\alpha_2 \overset{\$}{\leftarrow} \mathbb{Z}_q$. $\mathcal{S}_{DKG}$ sets $sk_2 = \alpha_2$. So, the distribution of sk_2 in two worlds are identical. Therefore, the output of the honest party P_1 in two worlds are identically distributed. Note that $\mathcal{A}$ does not see the output of honest P_2.

D Proof of Lemma 3

Proof. The threshold semantic security is proved by reduction, following the idea in [18]. The lossy encryption properties of *EncLossy* are proved in [7]. Since any lossy PKE scheme is semantically secure [6], *EncLossy* is semantically secure. Assume that there exists a probabilistic polynomial-time one-sided active adaptive adversary $\mathcal{A}_1$ that can break the semantic security of the two-party lossy threshold encryption scheme $ELTA2E$. It is described how to construct a probabilistic polynomial-time one-sided active adaptive adversary $\mathcal{A}_2$, using $\mathcal{A}_1$, that can break the semantic security of the non-threshold lossy encryption scheme $EncLossy$. As $EncLossy$ is semantically secure, a contradiction is reached. In this way, it will demonstrate that $ELTA2E$ is also semantically secure.

To convert $\mathcal{A}_1$ to $\mathcal{A}_2$, it is necessary to simulate the extra information that are not available in the non-threshold lossy cryptosystem. In step G2, the verification key and the secret key share of the corrupted party have to be simulated. In steps G3 and G5, the decryption shares and the validity proofs of the honest parties are simulated. The simulator is designed using the SIP technique. The inputs of the simulator are the public key $pk = (q, g, j, h, \ell)$, the mode parameter $mode$, and the identity I of the SIP.

In step G1, if $\mathcal{A}_1$ corrupts a party P_i, then $\mathcal{A}_2$ corrupts P_i. $\mathcal{A}_2$ receives the history of P_i from $\mathcal{Z}$.

In step G2, $\mathcal{A}_2$ simulates the verification key and the secret key share of the corrupted party as follows. When P_1 is the SIP, $\mathcal{A}_2$ works as follows. $\mathcal{A}_2$ selects $\alpha_1, \alpha_2 \xleftarrow{\$} \mathbb{Z}_q$. $\mathcal{A}_2$ sets $sk_1 = \alpha_1, sk_2 = \alpha_2, vk = g, vk_2 = g^{\alpha_2}, vk_1 = \frac{h}{vk_2}$. $\mathcal{A}_2$ sends $((pk, vk, vk_1, vk_2, sk_1), (pk, vk, vk_1, vk_2, sk_2))$ to $\mathcal{A}_1$ in step G2. The distribution of sk_1, sk_2 are identical in two worlds. $\mathcal{A}_2$ sets $vk_1 = \frac{h}{vk_2}$. The value of h is fixed and the distribution of vk_2 in two worlds are identical. Therefore, the distribution of vk_1 in two worlds are identical. Here $h = vk_1 \cdot vk_2$, so it is consistent. As P_1 is the SIP, the adversary does not corrupt P_1. So the adversary never learns the inconsistency that $vk_1 \neq g^{sk_1}$. If the adversary corrupts P_2, then it sees that $vk_2 = g^{sk_2}$ so everything is consistent for P_2.

When P_2 is the SIP, $\mathcal{A}_2$ selects $\alpha_1, \alpha_2 \xleftarrow{\$} \mathbb{Z}_q$. $\mathcal{A}_2$ sets $sk_1 = \alpha_1, sk_2 = \alpha_2, vk = g, vk_1 = g^{\alpha_1}, vk_2 = \frac{h}{vk_1}$. Proof argument is similar to case 1.

In step G3(1), if $\mathcal{A}_1$ corrupts a party P_i, then $\mathcal{A}_2$ corrupts P_i. $\mathcal{A}_2$ receives the secret key share and the history of P_i from $\mathcal{Z}$. As argued in step G2, the secret key share and history of the corrupted party is consistent.

In step G3(2), $\mathcal{A}_1$ selects a message $m \in M_{pk}$ and sends m to $\mathcal{A}_2$. $\mathcal{A}_2$ computes $c_m = (y_m, z_m) = (g^s j^t, h^s \ell^t g^m)$. c_m is a valid encryption of m. $\mathcal{A}_2$ simulates the decryption shares and the validity proofs of the honest parties as follows.

If P_1 is the SIP, then $\mathcal{A}_2$ simulates the steps of protocol Π_{DEC} as follows. In step 1, $\mathcal{A}_2$ computes $ds_1 = (y_m)^{sk_1}$ where sk_1 is the secret key share of P_1 computed by $\mathcal{A}_2$ in step G2 of game G. As argued in step G2, the distribution of sk_1 in two worlds are identical. Then, the distribution of ds_1 in two worlds are identical. In step 2, $\mathcal{A}_2$ acts as an honest prover using sk_1 as witness. The proof transcript in two worlds are identically distributed. In step 3, if P_2 is honest, then $\mathcal{A}_2$ computes $ds_2 = (y_m)^{sk_2}$ where sk_2 is the secret key share of P_2 computed by $\mathcal{A}_2$ in step G2 of game G. Proof argument is similar to step 1. In step 4, if P_2 is honest, then $\mathcal{A}_2$ acts as an honest prover using sk_2 as witness. Proof argument is similar to step 2. If P_2 is corrupted, then $\mathcal{A}_2$ acts as an honest verifier in the proof. If P_2 fails, then $\mathcal{A}_2$ sends $abort_{P_2}$ to the trusted party and halt. Then, the trusted party sends $abort_{P_2}$ to P_1 and honest P_1 halts. Honest P_1 aborts in the real world. In step 5, $\mathcal{A}_2$ computes $w = g^m$. The value of w is identical in two

worlds. In step 6, $\mathcal{A}_2$ uses m. The simulation of step G3(2) when P_2 is the SIP is similar. So, it is not given separately.
In step G4, $\mathcal{A}_1$ chooses two plaintexts $m_0, m_1 \in M_{pk}$ and sends them to $\mathcal{A}_2$. $\mathcal{A}_2$ sends (m_0, m_1) to the challenger of the non-threshold lossy encryption scheme *EncLossy*. Then, the challenger of *EncLossy* selects a random bit b, computes an encryption c of m_b and returns c to $\mathcal{A}_2$. $\mathcal{A}_2$ sends c to $\mathcal{A}_1$. Step G5 is similar to step G3. In step G6, $\mathcal{A}_1$ returns a guess b_1. $\mathcal{A}_2$ returns b_1.

E Proof of Theorem 3

The security of Π_{OTAA} is proved following the simulation based security definition by Canetti [9]. As the adversary is one-sided, it can corrupt at most one party. The security is proved using the persistently inconsistent party technique of Jarecki and Lysanskaya [22].
Let $\mathcal{A}$ be a one-sided active adaptive adversary and $\mathcal{Z}$ be the environment. Let $\mathcal{S}_{OT}$ be the simulator for protocol Π_{OTAA} for adversary $\mathcal{A}$ and environment $\mathcal{Z}$. The security argument is described for two cases below. In both cases, at start, $\mathcal{S}_{OT}$ selects $\delta \xleftarrow{\$} \mathbb{Z}_q$, and sets the common reference string to $\mu = g^{\delta}$. The common reference string μ is used as the commitment key in the adaptive zero-knowledge arguments. $\mathcal{S}_{OT}$ stores δ as the trapdoor of the commitment key μ. This trapdoor δ is necessary for generating a proof transcript for the zero-knowledge arguments.
At start of the simulation of Π_{OTAA}, $\mathcal{S}_{OT}$ selects I, the identity of the SIP, uniformly at random from $\{S, R\}$. The identity of I remains fixed in any subprotocol called within Π_{OTAA}. One possible subprotocol is the distributed key generation protocol Π_{DKG} of $ELTA2E$.
In the simulation, after each step, a special step is denoted as the "action upon corruption". This step describes how the simulator performs if an honest party gets corrupted after completing the corresponding step. This tagging is used to clarify the description of the simulation.

E.1 Security for the case where the Sender S is the SIP

The simulator $\mathcal{S}_{OT}$ for the case $I = S$ is given in Figure 13.
It is proved below why the global output produced by simulator $\mathcal{S}_{OT}$ in the hybrid world is computationally indistinguishable from the global output produced in the real world. The following things happen during the steps of protocol Π_{OTAA}.

1. $\mathcal{S}_{OT}$ generates a lossy key pair of $ELTA2E$ as follows. $\mathcal{S}_{OT}$ selects $\alpha_1, \alpha_2 \xleftarrow{\$} \mathbb{Z}_q$. $\mathcal{S}_{OT}$ sets $\alpha = (\alpha_1 + \alpha_2) \bmod q, h = g^{\alpha}$. $\mathcal{S}_{OT}$ selects $\gamma \xleftarrow{\$} \mathbb{Z}_q$. $\mathcal{S}_{OT}$ sets $j = g^{\gamma}$. $\mathcal{S}_{OT}$ selects $\rho \xleftarrow{\$} \mathbb{Z}_q \setminus \{\alpha\}$, and sets $\ell = g^{\gamma\rho}$. $\mathcal{S}_{OT}$ sets $pk = (q, g, j, h, \ell)$. $\mathcal{S}_{OT}$ stores the corresponding secret key $sk = (\alpha, \gamma, \rho)$.
   $\mathcal{S}_{OT}$ generates the lossy key pair in a similar way to the way the key generation algorithm KG of $ELTA2E$ generates a lossy key pair. That means,

Simulator $\mathcal{S}_{OT}$ for protocol Π_{OTAA} for the case $I = S$.

1. Generate a lossy key pair of $ELTA2E$.
   Simulate using the simulator $\mathcal{S}_{DKG}$ on the generated key.
   Action upon corruption: If $\mathcal{A}$ corrupts R after this step, then corrupt R in the hybrid world, and receive the input σ of R from $\mathcal{Z}$.
2. If R is honest, then compute c_0, c_1 based on $\sigma = 0$.
   Action upon corruption: If $\mathcal{A}$ corrupts R after this step, then perform the same steps listed as the action upon corruption after step 1.
3. If R is corrupted, then act as an honest verifier in the proof.
   If R fails, then send $abort_R$ to the trusted party and halt.
   If R passes, then extract the plaintexts of c_0 and c_1.
   From these values, learn the possibly modified input σ_1 of corrupted R.
   Send σ_1 to the trusted party, and receive back its output x_{σ_1}.
   Set $\sigma = \sigma_1$ and the output of R to x_{σ_1}.
   If R is honest, then generate a proof transcript.
   Action upon corruption: If $\mathcal{A}$ corrupts R after this step, then perform as follows.
   If $\sigma_1 = 1$, then, using the efficient *Opener* algorithm, compute randomness $(\hat{s}_0, \hat{t}_0)$ and $(\hat{s}_1, \hat{t}_1)$ that are consistent with ciphertexts c_0, c_1 and the receiver input $\sigma = 1$. Use $\hat{s_0}, \hat{t_0}, \hat{s_1}, \hat{t_1}$ as the randomness of R for step 2.
4. (a) For each $i \in \{0, 1\}$, select $\hat{x}_i \xleftarrow{\$} \{0, 1\}$ and compute

   $$d_i = \hat{x}_i \times_h c_i, \hat{v}_i = Blind(pk, d_i).$$

   Use $(hatv_0, \hat{v}_1)$ as the message from S.
   (b) Generate two proof transcripts.
   Action upon corruption: If $\mathcal{A}$ corrupts R after this step, then perform the same steps listed as the action upon corruption after step 3.
5. Set $w_\sigma = x_\sigma, w_{1-\sigma} = 0$.
   For each $i \in \{0, 1\}$, perform the following steps.
   (a) Compute

   $$\theta_i = g^{w_i}, ds_{2,i} = (vy_i)^{sk_2}, \widehat{ds}_{1,i} = \frac{vz_i}{\theta_i \cdot ds_{1,i}}.$$

   Use $\widehat{ds}_{1,i}$ as the message from S.
   (b) Generates a proof transcript.
   (c) If R is honest, then perform the computation honestly.
   Action upon corruption: If $\mathcal{A}$ corrupts R after any substep of this step, then perform the same steps listed as the action upon corruption after step 3.
6. Do nothing.

Post-Execution Corruption : If $\mathcal{A}$ corrupts R after the execution of Π_{OTAA} is finished, then perform the same steps listed as the action upon corruption after step 3.

Fig. 13: Simulator $\mathcal{S}_{OT}$ for protocol Π_{OTAA} for the case $I = S$.

the distribution of the key pair (pk, sk) is identically distributed to a lossy key pair generated by algorithm KG.

The reason for generating the components of the keys, without using algorithm KG is as follows. When $\mathcal{S}_{OT}$ generates the values, it can obtain the values of α, γ and ρ. These three values constitute the secret key of the lossy key pair. The secret key is necessary to use the efficient *Opener* algorithm of $ELTA2E$. $\mathcal{S}_{OT}$ uses the efficient *Opener* algorithm in later step of the simulation. If $\mathcal{S}_{OT}$ used algorithm KG to generate a lossy key pair, then $\mathcal{S}_{OT}$ would not obtain the values of α, γ and ρ. In that case, $\mathcal{S}_{OT}$ could not use the efficient *Opener* algorithm.

If protocol Π_{DKG} is used to implement step 1, then $\mathcal{S}_{OT}$ uses the simulator $\mathcal{S}_{DKG}$ of protocol Π_{DKG} on input $(pk, 0, P_1)$. That means $\mathcal{S}_{OT}$ invokes simulator $\mathcal{S}_{DKG}$ on input public key pk, mode parameter set to zero to denote lossy mode, and the identity I of the SIP set to P_1. S acts as P_1 in the execution of protocol Π_{DKG} in step 1. So, I is set to S.

In the real world, S and R generate an injective key by using protocol Π_{DKG}. Since S is the SIP, S is honest and $\mathcal{A}$ does not learn the secret key share sk_1 of S. Without the knowledge of the secret key, by the "indistinguishability of keys" property of $ELTA2E$, the public key in the hybrid world is computationally indistinguishable from the public key in the real world.

By Lemma 5, the message that $\mathcal{S}_{DKG}$ generates in the hybrid world is computationally indistinguishable from the message that $\mathcal{A}$ views during the execution of protocol Π_{DKG} in the real world.

If $\mathcal{A}$ corrupts R after this step, then $\mathcal{S}_{OT}$ corrupts R in the hybrid world and receives the input σ of R from $\mathcal{Z}$. In this case, the newly corrupted R may change its input by supplying the ciphertexts in step 2 according to the input of its choice. For this reason, $\mathcal{S}_{OT}$ does not send σ to the trusted party of $\mathcal{F}_{OT}$ yet. $\mathcal{S}_{OT}$ will extract the modified input of corrupted R in step 3 and send that modified input to the trusted party of $\mathcal{F}_{OT}$ in that step.

2. If R is honest, then $\mathcal{S}_{OT}$ computes c_0, c_1 based on $\sigma = 0$, as follows. $\mathcal{S}_{OT}$ selects $s_0, t_0, s_1, t_1 \overset{\$}{\leftarrow} \mathbb{Z}_q$. $\mathcal{S}_{OT}$ computes $c_0 = E_{pk}\left(1, (s_0, t_0)\right), c_1 = E_{pk}\left(0, (s_1, t_1)\right).$
   In the real world, honest R computes c_0 and c_1 in such a way that c_σ encrypts one and $c_{1-\sigma}$ encrypts zero. By threshold semantic security of $ELTA2E$, the distribution of c_0, c_1 in two worlds are computationally indistinguishable.
   If $\mathcal{A}$ corrupts R after step 2, then $\mathcal{A}$ cannot replace the input σ as the value of σ is already fixed by the message supplied up to step 2. $\mathcal{S}_{OT}$ corrupts R in the hybrid world and receives its input σ from $\mathcal{Z}$. $\mathcal{S}_{OT}$ sends σ to the trusted party of $\mathcal{F}_{OT}$, and receives back its output x_σ.
3. If R is corrupted, then $\mathcal{S}_{OT}$ acts as an honest verifier in step 3.
   If R fails, then $\mathcal{S}_{OT}$ sends $abort_R$ to the trusted party and halts. The trusted party sends $abort_R$ to S and S aborts. In this case, honest S aborts in the real world.
   If R passes, then $\mathcal{S}_{OT}$ extracts the plaintexts of c_0 and c_1 by using the knowledge extractor of the zero-knowledge arguments. From these plaintexts, $\mathcal{S}_{OT}$ learns the possibly modified input σ_1 of corrupted R. $\mathcal{S}_{OT}$ sends σ_1 to

the trusted party of $\mathcal{F}_{OT}$, and receives back its output x_{σ_1}. $\mathcal{S}_{OT}$ sets $\sigma = \sigma_1$ and the output of R to x_{σ_1}.
In the real world, the generated key pair is injective, so $\mathcal{A}$ cannot open a ciphertext encrypting one to be a ciphertext encrypting zero. In the hybrid world, $\mathcal{S}_{OT}$ generates a lossy key pair. Since $\mathcal{A}$ corrupts at most one party, $\mathcal{A}$ cannot learn the secret key. Without the knowledge of the secret key, $\mathcal{A}$ cannot use the efficient *Opener* algorithm as the efficient *Opener* algorithm requires the secret key as one of its inputs. That means, in the hybrid world, $\mathcal{A}$ cannot open a ciphertext encrypting one to be a ciphertext encrypting zero in polynomial time. That means the result of the zero-knowledge argument will be identical in both worlds.
If R is honest, then $\mathcal{S}_{OT}$ generates a proof transcript using the simulator of the adaptive zero-knowledge argument and trapdoor δ. $\mathcal{S}_{OT}$ sets the key to a lossy key in step 1. The proof of this step does not work for a lossy key for $ELTA2E$. For this reason, $\mathcal{S}_{OT}$ generates a proof transcript. In the real world, honest R acts as an honest prover based on its witness for the ciphertext encrypting zero. By the definition of zero-knowledge arguments, the proof transcript in two worlds are computationally indistinguishable.
If $\mathcal{A}$ corrupts R after this step, then $\mathcal{S}_{OT}$ proceeds as follows.
If $\sigma_1 = 0$, then $\mathcal{S}_{OT}$ performs no additional updates since $\mathcal{S}_{OT}$ calculated c_0, c_1 based on $\sigma = 0$.
If $\sigma_1 = 1$, then $\mathcal{S}_{OT}$ computes randomness $\hat{s}_0, \hat{t}_0, \hat{s}_1, \hat{t}_1$ using the efficient *Opener* algorithm, such that $c_0 = E_{pk}(0, (\hat{s}_0, \hat{t}_0))$ and $c_1 = E_{pk}(1, (\hat{s}_1, \hat{t}_1))$. $\mathcal{S}_{OT}$ supplies $\hat{s_0}, \hat{t_0}, \hat{s_1}, \hat{t_1}$ as the randomness for step 2. Since $\mathcal{S}_{OT}$ knows the secret key of the lossy key, algorithm *Opener* produces output in polynomial time. By the "openability" property of $ELTA2E$, the generated randomness is consistent. The randomness in two worlds are identically distributed. After this update it holds that $E_{pk}(0, (\hat{s}_0, \hat{t}_0) = c_0$, and $E_{pk}(1, (\hat{s}_1, \hat{t}_1) = c_1$. Then, c_0 is an encryption of zero and c_1 is an encryption of one, which is consistent with the actual value of $\sigma_1 = 1$.

4. (a) For each $i \in \{0, 1\}$, the following holds.
   In the real world, the ciphertext v_i is based on the actual input x_i of S. In the hybrid world, $\mathcal{S}_{OT}$ selects $\hat{x}_i \overset{\$}{\leftarrow} \{0, 1\}$ and computes $d_i = \hat{x}_i \times_h c_i, \hat{v}_i = Blind(pk, d_i)$.
   By the threshold semantic security of $ELTA2E$, the distribution of the ciphertexts v_i in two worlds are computationally indistinguishable.
   (b) $\mathcal{S}_{OT}$ sets the key to a lossy key in step 1. The proof for correctness of multiplication does not work for a lossy key for $ELTA2E$. So, $\mathcal{S}_{OT}$ generates two proof transcripts for this step using the simulator of the zero-knowledge argument and trapdoor δ. In the real world, honest S acts as an honest prover, using x_0, x_1 and the randomness used in the execution of *Blind* function as the witness. By definition of zero-knowledge argument, the proof transcript in two worlds are computationally indistinguishable.

If $\mathcal{A}$ corrupts R after this step, then $\mathcal{S}_{OT}$ performs the same steps listed as action after corruption after step 3.

5. If R is corrupted after step 2, then $\mathcal{A}$ cannot replace the input of R as the input of R is already fixed by the message sent by honest R in step 2. In this case, $\mathcal{S}_{OT}$ sends σ as input of R to the trusted party of $\mathcal{F}_{OT}$ and receives back output x_σ. $\mathcal{S}_{OT}$ sets the output of R to x_σ.
   If R is corrupted before step 2, then $\mathcal{S}_{OT}$ extracts the possibly replaced input σ_1 of R in step 3. $\mathcal{S}_{OT}$ sends σ_1 as input of R to the trusted party of $\mathcal{F}_{OT}$ and receives back output x_{σ_1}. $\mathcal{S}_{OT}$ sets the output of R to x_{σ_1}. $\mathcal{S}_{OT}$ sets $\sigma = \sigma_1$. This modification is necessary for the uniform treatment of step 5 described below.
   $\mathcal{S}_{OT}$ sets $w_\sigma = x_\sigma, w_{1-\sigma} = 0$.
   For each $i \in \{0,1\}, \mathcal{S}_{OT}$ performs the following steps.
   (a) $\mathcal{S}_{OT}$ computes

$$\begin{aligned} \theta_i &= g^{w_i}, \\ ds_{2,i} &= (vy_i)^{sk_2}, \\ \widehat{ds}_{1,i} &= \frac{vz_i}{\theta_i \cdot ds_{1,i}}. \end{aligned}$$

   $\mathcal{S}_{OT}$ uses $\widehat{ds}_{1,i}$ as the message from S.
   The reason for this type of adjusting by $\mathcal{S}_{OT}$ is that correctness of decryption does not hold for a lossy key for $ELTA2E$. $\mathcal{S}_{OT}$ sets the key to a lossy key, so normal calculation will not yield the decryption correctly. But $\mathcal{S}_{OT}$ knows the output of decryption. $\mathcal{S}_{OT}$ adjusts the value of $ds_{1,i}$ to $\widehat{ds}_{1,i}$ so that in step 5(c), the required decryption result is obtained.
   In the real world, honest S sends $ds_{1,i} = (y_i)^{sk_1}$.
   Since S is the SIP, S is honest and $\mathcal{A}$ does not know sk_1. By the decisional Diffie-Hellman assumption, the distribution of $ds_{1,i}$ in two worlds are computationally indistinguishable.
   (b) $\mathcal{S}_{OT}$ generates a proof transcript for this step using the simulator of the zero-knowledge argument and trapdoor δ. The proof of step 5(b) does not work for a lossy key for $ELTA2E$. In the real world, honest S acts as an honest prover using witness sk_1. By definition of zero-knowledge argument, the proof transcripts in two worlds are computationally indistinguishable.
   (c) If R is honest, then $\mathcal{S}_{OT}$ honestly performs step 5(c).
   If R is corrupted, then, in the hybrid world, $\mathcal{A}$ obtains w_i. In the real world, $\mathcal{A}$ obtains w_i due to the "correctness on injective keys" property of $ELTA2E$.
   If $\mathcal{A}$ corrupts R after any substep of this step, then $\mathcal{S}_{OT}$ performs the same steps listed as modifications after corruption after step 3.
6. $\mathcal{S}_{OT}$ produces nothing as this is a step where R produces its output.
   If R is corrupted before step 2, then $\mathcal{S}_{OT}$ extracts the possibly replaced input σ_1 of R and sets the output $x_\sigma$of R to x_{σ_1} where x_{σ_1} is the output of R that $\mathcal{S}_{OT}$ obtains by sending σ_1 as input of R to the trusted party of $\mathcal{F}_{OT}$.
   If R is corrupted after step 2, then $\mathcal{S}_{OT}$ sets the output of R to x_σ where x_σ is the output of R that $\mathcal{S}_{OT}$ obtains by sending σ as input of R to the

trusted party of $\mathcal{F}_{OT}$. Note that the adversary cannot replace the input of R if it corrupts R after step 2.
If R is corrupted, then $\mathcal{A}$ will obtain the same value x_σ in step 6 in the hybrid world that it obtains in the real world.
In an OT protocol, S has no output. So, trivially, the output of the honest party S is identical (an empty string) in both worlds.

Post-Execution Corruption : If $\mathcal{A}$ corrupts R after the execution of the protocol is finished, then $\mathcal{S}_{OT}$ performs the same steps listed as action upon corruption after step 4. Indistinguishability follows due to the reasoning stated above.

E.2 Security for the case where the receiver *R* is the SIP

The simulator for the case $I = R$ is presented in Figure 14.

Simulator $\mathcal{S}_{OT}$ for protocol $\mathit{\Pi}_{OTAA}$ for the case $I = R$.

1. Generate a lossy key pair of $ELTA2E$.
   Simulate using the simulator $\mathcal{S}_{DKG}$ on the generated key.
   Action upon corruption: If $\mathcal{A}$ corrupts S after this step, then corrupt S in the hybrid world, and receive the input (x_0, x_1) of S from $\mathcal{Z}$.
2. Set $\sigma = 0$.
   Compute c_0, c_1 based on $\sigma = 0$.
   Action upon corruption: If $\mathcal{A}$ corrupts S after this step, then perform the same steps listed as the action upon corruption after step 1.
3. Generate a proof transcript using the simulator of the adaptive zero-knowledge argument and trapdoor δ.
   Action upon corruption: If $\mathcal{A}$ corrupts S after this step, then perform the same steps listed as the action upon corruption after step 1.
4. (a) If S is honest, then perform the following steps for each $i \in \{0, 1\}$. Select $\hat{x}_i \xleftarrow{\$} \{0, 1\}$.
   Compute
   $$\hat{d}_i = \hat{x}_i \times_h c_i, \hat{v}_i = Blind(\hat{d}_i).$$
   Use $(\hat{v}_0, \hat{v}_1)$ as the message from S.
   Action upon corruption: If $\mathcal{A}$ corrupts S after this step, then corrupt S in the hybrid world, and receive the input (x_0, x_1) of S from $\mathcal{Z}$.
   Set $\tilde{x}_0 = x_0, \tilde{x}_1 = x_1$.

   (b) If S is corrupted, then act as an honest verifier in the proofs.
   If S fails in any proof, then send $abort_S$ to the trusted party of $\mathcal{F}_{OT}$ and halt.
   If S passes the proofs, then extract the possibly replaced input $(\tilde{x}_0, \tilde{x}_1)$ of corrupted S.
   If S is honest, then generate two proof transcripts using the simulator of the zero-knowledge argument and trapdoor δ.
   Action upon corruption: If $\mathcal{A}$ corrupts S after this step, then perform the same steps listed as action upon corruption after step 4(a).

Fig. 14: Simulator $\mathcal{S}_{OT}$ for protocol $\mathit{\Pi}_{OTAA}$ for the case $I = R$.

Simulator $\mathcal{S}_{OT}$ for protocol Π_{OTAA} for the case $I = R$.

5. Set $w_\sigma = \tilde{x}_\sigma, w_{1-\sigma} = 0$.
   For each $i \in \{0, 1\}$, perform the following steps.
   (a) If S is honest, then compute
   $$\theta_i = g^{w_i}, ds_{2,i} = (vy_i)^{sk_2}, \widehat{ds}_{1,i} = \frac{vz_i}{\theta_i \cdot ds_{1,i}}.$$
   Use $\widehat{ds}_{1,i}$ as the message from S.
   (b) If S is corrupted, then act as an honest verifier in the proof.
   If S fails, then send $aborts_S$ to the trusted party and halt.
   If S is honest, then generate a proof transcript using the simulator of the zero-knowledge argument and trapdoor δ.
   (c) Perform the computations honestly.
   Action upon corruption: If $\mathcal{A}$ corrupts S after any substep of this step, then perform the same steps listed as the action upon corruption after step 4(a).
6. Send $(\tilde{x}_0, \tilde{x}_1)$ to the trusted party of $\mathcal{F}_{OT}$.

Post-Execution Corruption : If $\mathcal{A}$ corrupts S after the execution of Π_{OTAA} is finished, then perform the same steps listed as the action upon corruption after step 4(a).

Fig. 14: Simulator $\mathcal{S}_{OT}$ for protocol Π_{OTAA} for the case $I = R$ (continued).

It is proved below why the global output produced by simulator $\mathcal{S}_{OT}$ in the hybrid world is computationally indistinguishable from the global output produced in the real world. The following things happen during the steps of protocol Π_{OTAA}.

1. $\mathcal{S}_{OT}$ generates a lossy key pair of $ELTA2E$ in a similar way as step 1 of simulation where $I = S$. $\mathcal{S}_{OT}$ stores the corresponding secret key $sk = (\alpha, \gamma, \rho)$. Let pk be the public key generated by $\mathcal{S}_{OT}$. If protocol Π_{DKG} is used to implement step 1, then $\mathcal{S}_{OT}$ uses the simulator $\mathcal{S}_{DKG}$ of protocol Π_{DKG} on input $(pk, 0, P_2)$. That means $\mathcal{S}_{OT}$ invokes simulator $\mathcal{S}_{DKG}$ on input public key pk, mode parameter set to zero to denote lossy mode, and the identity I of the SIP set to P_2. R acts as P_2 in the execution of protocol Π_{DKG} in step 1. So, I is set to R.
   Indistinguishability can be proven similar to step 1 of the simulation where $I = S$.
2. $\mathcal{S}_{OT}$ computes c_0, c_1 based on $\sigma = 0$. In the real world, honest R computes c_0 and c_1 in such a way that c_σ encrypts one and $c_{1-\sigma}$ encrypts zero. By threshold semantic security of $ELTA2E$, the distribution of c_0, c_1 in two worlds are computationally indistinguishable.
3. $\mathcal{S}_{OT}$ generates a proof transcript using the simulator of the zero-knowledge argument and trapdoor δ. The indistinguishability argument is similar to step 1 of the simulation when $I = S$ and R is honest.
4. (a) If S is honest, then $\mathcal{S}_{OT}$ performs the following steps for each $i \in \{0, 1\}$.
   $\mathcal{S}_{OT}$ selects $\hat{x}_i \xleftarrow{\$} \{0, 1\}$.

$\mathcal{S}_{OT}$ computes $\hat{d}_i = \hat{x}_i \times_h c_i, \hat{v}_i = Blind(\hat{d}_i)$. $\mathcal{S}_{OT}$ uses $(\hat{v}_0, \hat{v}_1)$ as the message from S.

In the real world, the ciphertexts v_0 and v_1 are based on the actual input (x_0, x_1) of honest S.

By the threshold semantic security of $ELTA2E$, the distribution of the ciphertexts v_0, v_1 in two worlds are computationally indistinguishable.

If $\mathcal{A}$ corrupts S after this step, then $\mathcal{S}_{OT}$ corrupts S in the hybrid world, and receives the input (x_0, x_1) of S from $\mathcal{Z}$. In this case, $\mathcal{A}$ cannot modify the input of S as the input of S is already fixed by the message sent by honest S up to step 4(a). So, $\mathcal{S}_{OT}$ sets $\tilde{x}_0 = x_0, \tilde{x_1} = x_1$. This modification is necessary for the uniform treatment for two cases – the first case where $\mathcal{A}$ corrupts S before step 4(a), and the second case where $\mathcal{A}$ corrupts S after step 4(a).

(b) If S is corrupted, then $\mathcal{S}_{OT}$ acts as an honest verifier in the proofs.

If S fails in any proof, then $\mathcal{S}_{OT}$ sends $abort_S$ to the trusted party of $\mathcal{F}_{OT}$ and halts. The trusted party sends $abort_S$ to R and R aborts. In this case, honest R aborts in the real world.

If S passes the proofs, then $\mathcal{S}_{OT}$ extracts the possibly replaced input $(\tilde{x}_0, \tilde{x}_1)$ of corrupted S by using the knowledge extractor of the zero-knowledge arguments.

If S is honest, then $\mathcal{S}_{OT}$ generates two proof transcripts using the simulator of the zero-knowledge argument and trapdoor δ. Indistinguishability argument is similar to step 4(b) of simulation where $I = S$.

If $\mathcal{A}$ corrupts S after this step, then $\mathcal{S}_{OT}$ performs the same steps listed as action upon corruption after step 4(a).

5. $\mathcal{S}_{OT}$ sets $w_\sigma = \tilde{x}_\sigma, w_{1-\sigma} = 0$.

For each $i \in \{0,1\}$, the following holds.

(a) If S is honest, then $\mathcal{S}_{OT}$ computes

$$\begin{aligned} \theta_i &= g^{w_i}, \\ ds_{2,i} &= (y_i)^{sk_2}, \\ \widehat{ds}_{1,i} &= \frac{z_i}{\theta_i \cdot ds_{2,i}}. \end{aligned}$$

$\mathcal{S}_{OT}$ uses the $\widehat{ds}_{1,i}$ as the message from S. Indistinguishability argument is similar to step 5(a) of simulation where $I = S$.

(b) If S is corrupted, then $\mathcal{S}_{OT}$ acts as an honest verifier in the proof.

If S fails, then $\mathcal{S}_{OT}$ sends $abort_S$ to the trusted party and halts. The trusted party sends $abort_S$ to R and R aborts. In this case, honest R aborts in the real world.

If S is honest, then $\mathcal{S}_{OT}$ generates a proof transcript using the simulator of the zero-knowledge argument and trapdoor δ. Indistinguishability argument is similar to step 4(b) of simulation where $I = S$.

(c) $\mathcal{S}_{OT}$ performs the computations honestly on behalf of R. This is a step where R computes w_i locally. Since R is honest, so $\mathcal{A}$ obtains nothing in this step.

If $\mathcal{A}$ corrupts S after any substep of this step, then $\mathcal{S}_{OT}$ performs the same steps listed as the action upon corruption after step 4(a).

6. $\mathcal{S}_{OT}$ sends $(\tilde{x}_0, \tilde{x}_1)$ to the trusted party. Let σ be the input of honest R. Then, the trusted party sends the output $\tilde{x}_\sigma$ to the honest R.
   In the real world, honest R outputs the value $\tilde{x}_\sigma$ that it obtains, based on the calculations in step 5.
   Then, it follows that the output of the honest party R is identical in two worlds.

Post-Execution Corruption : If $\mathcal{A}$ corrupts S after the execution of the protocol is finished, then $\mathcal{S}_{OT}$ performs the same steps listed as action after corruption after step 4(a). Indistinguishability follows due to the reasoning stated above.

Efficiency:
In the encryption scheme $ELTA2E$, the size of a ciphertext is $2n$. It is possible to use a distributed key generation protocol Π_{DKG} for implementing $\mathcal{F}_{KG}$ in step 1 of Π_{OTAA}. The communication complexity of protocol Pi_{DKG} is $51n$.
The communication cost of the adaptive zero-knowledge arguments for Π_{OTAA} are presented in Table 1.

Relation	**Communication Cost**
R_{EQ}	$7n$
R_{MULT}	$7n$
$R_{OR-ZERO}$	$15n$

Table 1: Communication cost of the adaptive zero-knowledge arguments for Π_{OTAA}.

The communication complexity of protocol Π_{OTAA}, including the communication complexity of protocol Π_{DKG}, is $101n \in O(n)$.
In step 2, R performs two encryption operations of $ELTA2E$. In step 4, S performs two homomorphic multiplication by constant and two *Blind* function evaluations. One homomorphic multiplication by constant and one *Blind* function together is similar in computational complexity to one encryption operation of $ELTA2E$. So, the total number of PKE operation of Π_{OTAA} is four, in the worst case.
Next, the efficiency of the extension of Π_{OTAA} for string OT is described. If the strings x_0, x_1 are of length k, then the worst case communication complexity is $20n^2 + 81n \in O(n^2)$. In that case, the number of PKE operations is $(2n + 2)$, in the worst case.

F Proof of Lemma 4

Proof. The properties of a non-erasure Σ-protocol of Definition 5 for protocol $\Pi_{\Sigma NEQ}$ are proved below.

Completeness: If P, V follow the protocol on common input (x_1, x_2, y_1, y_2) and private input (w_1, w_2) where $((x_1, x_2, y_1, y_2), (w_1, w_2)) \in R_{EQ}$, then the followings hold.
$(y_1)^{z_1} = (y_1)^{r+ew_1} \bmod p = (y_1)^r \cdot (y_1)^{ew_1} \bmod p = a_1(x_1)^e \bmod p$.
$(y_2)^{z_2} = (y_2)^{r+ew_2} \bmod p = (y_2)^r \cdot (y_2)^{ew_2} \bmod p = a_2(x_2)^e \bmod p$.
$(y_1)^{z_2} = (y_1)^{r+ew_2} \bmod p = (y_1)^r \cdot (y_1)^{ew_2} \bmod p = a_1 \cdot (y_1)^{ew_2} \bmod p$
$\neq a_1(x_1)^e \bmod p$ since $w_1 \neq w_2$.
$(y_2)^{z_1} = (y_2)^{r+ew_1} \bmod p = (y_2)^r \cdot (y_2)^{ew_1} \bmod p = a_2(y_2)^{ew_1} \bmod p$
$\neq a_2(x_2)^e \bmod p$ since $w_1 \neq w_2$.
Then, V accepts.

Special Soundness: Let $((a_1, a_2), e, (z_1, z_2))$ and $((a_1, a_2), e', (z_1', z_2'))$ be two accepting conversations such that $e \neq e'$. Then $(y_1)^{z_1} = a_1(x_1)^e \bmod p$, $(y_2)^{z_2} = a_2(x_2)^e \bmod p$, $(y_1)^{z_1'} = a_1(x_1)^{e'} \bmod p$, $(y_2)^{z_2'} = a_2(x_2)^{e'} \bmod p$. Then, $(y_1)^{z_1 - z_1'} = (x_1)^{e-e'} \bmod p$ and $(y_2)^{z_2 - z_2'} = (x_2)^{e-e'} \bmod p$. Since $e \neq e' \bmod q$, there exists a multiplicative inverse of $(e-e')$ modulo q. Raising both sides of the first equation with power $(e-e')^{-1} \bmod q$ gives $x_1 = (y_1)^{(z_1 - z_1')(e-e')^{-1} \bmod q} \bmod p$, implying $w_1 = (z_1 - z_1')(e-e')^{-1} \bmod q$. Similarly, raising both sides of the second equation with power $(e-e')^{-1} \bmod q$ gives
$x_2 = (y_2)^{(z_2 - z_2')(e-e')^{-1} \bmod q} \bmod p$, implying $w_2 = (z_2 - z_2')(e-e')^{-1} \bmod q$.

Special Non-Erasure Honest Verifier Zero-Knowledge: For a given input (x_1, x_2, y_1, y_2) and challenge e, the honest verifier simulator hvs selects $z_1, z_2 \xleftarrow{\$} \mathbb{Z}_q$ and computes $a_1 = (y_1)^{z_1}(x_1)^{-e} \bmod p$ and $a_2 = (y_2)^{z_2}(x_2)^{-e} \bmod p$. Then $((x_1, x_2, y_1, y_2), e, (z_1, z_2))$ has the same distribution of a real conversation between an honest prover and an honest verifier.
On input $((x, w) \in R_{EQ}, e, (z_1, z_2))$ where $r_{hvs} = (z_1, z_2)$ is the randomness used by hvs, the random bits simulator rbs computes $r = z_1 - ew_1 \bmod q$. In the real execution, $r \xleftarrow{\$} \mathbb{Z}_q$. In the simulation, $z_1 \xleftarrow{\$} \mathbb{Z}_q$ and $r = z_1 - ew \bmod q$, so $r \xleftarrow{\$} \mathbb{Z}_q$.

G Proof of Lemma 5

The properties of a non-erasure Σ-protocol of Definition 5 are proved below.

Completeness: If P, V follow the protocol on common input (u_1, v_1, u_3, v_3) and private input m_2, s_3, t_3 where $((u_1, v_1, u_3, v_3), (m_2, s_3, t_3)) \in R_{MULT}$ for $ELTA2E$, then the followings hold.
$(u_1)^{z_1} g^{z_2} j^{z_3} = (u_1)^{r_1 + em_2} g^{r_2 + es_3} j^{r_3 + et_3} = (u_1)^{r_1} (u_1)^{em_2} g^{r_2} g^{es_3} j^{r_3} j^{et_3}$
$= (u_1)^{r_1} g^{r_2} j^{r_3} \cdot \left((u_1)^{m_2} g^{s_3} \cdot j^{t_3}\right)^e = a_1(u_3)^e$.
$(v_1)^{z_1} h^{z_2} \ell^{z_3} = (v_1)^{r_1 + em_2} h^{r_2 + es_3} \ell^{r_3 + et_3} = (v_1)^{r_1} (v_1)^{em_2} h^{r_2} h^{es_3} \ell^{r_3} \ell^{et_3}$
$= (v_1)^{r_1} h^{r_2} \ell^{r_3} \cdot \left((v_1)^{m_2} h^{s_3} \cdot \ell^{t_3}\right)^e = a_2(v_3)^e$. Then, it follows that V accepts.

Special Soundness: Let $((a_1, a_2), e, (z_1, z_2, z_3))$ and $((a_1, a_2), e', (z_1', z_2', z_3'))$ be two accepting conversations such that $e \neq e'$. Then,

$$(u_1)^{z_1} g^{z_2} j^{z_3} = a_1(u_3)^e \bmod p,$$

and

$$(u_1)^{z_1'} g^{z_2'} j^{z_3'} = a_1 (u_3)^{e'} \bmod p.$$

Then,

$$(u_1)^{z_1 - z_1'} g^{z_2 - z_2'} j^{z_3 - z_3'} = (u_3)^{e-e'} \bmod p.$$

Since $e \neq e' \bmod q$, there exists a multiplicative inverse of $(e - e')$ modulo q. Raising both sides with power $(e - e')^{-1} \bmod q$ gives

$$u_3 = (u_1)^{(z_1 - z_1')(e-e')^{-1} \bmod q} g^{(z_2 - z_2')(e-e')^{-1} \bmod q} j^{(z_3 - z_3')(e-e')^{-1} \bmod q} \bmod p,$$

implying $m_2 = (z_1 - z_1')(e - e')^{-1} \bmod q$, $s_3 = (z_2 - z_2')(e - e')^{-1} \bmod q$, $t_3 = (z_3 - z_3')(e - e')^{-1} \bmod q$.

Special Non-Erasure Honest Verifier Zero-Knowledge: For a given input (u_1, v_1, u_3, v_3) and challenge e, the simulator selects $z_1, z_2, z_3 \overset{\$}{\leftarrow} \mathbb{Z}_q$ and computes $a_1 = (u_1)^{z_1} g^{z_2} j^{z_3} (u_3)^{-e} \bmod p$ and $a_2 = (v_1)^{z_1} h^{z_2} \ell^{z_3} (v_3)^{-e} \bmod p$. Then $((a_1, a_2), e, (z_1, z_2, z_3))$ has the same distribution of a real conversation between an honest prover and an honest verifier.

On input $((u_1, v_1, u_3, v_3), (m_2, s_3, t_3)) \in R_{DL}, e, z_1, z_2, z_3)$ where $r_{hvs} = (z_1, z_2, z_3)$ is the randomness used by hvs, the random bits simulator rbs computes $r_1 = z_1 - em_2 \bmod q, r_2 = z_2 - es_3 \bmod q, r_3 = z_3 - et_3 \bmod q$.

H Efficiency of the OT Protocol by Hazay and Patra [21]

In this section, the efficiency of the OT protocol by Hazay and Patra [21] is described. They have different efficiency for polynomial-size message space and exponential-size message space, with respect to the security parameter n. Some details on the calculation of the efficiency of bit OT is provided below.

The OT protocol of [21] uses the following tools.

1. A non-committing encryption scheme secure against one-sided active adaptive adversaries. They designed a protocol Π_{OSC} that for this purpose.
2. The dual-mode PKE scheme by Peikert et al. [29].
3. Witness-equivocal zero-knowledge proofs of knowledge.

The efficiency of protocol Π_{OSC} of [21] for polynomial-size message space is described below. Π_{OSC} for polynomial-size message space is based on the decisional Diffie-Hellman assumption. The protocol Π_{OSC} uses the following tools.

1. The sender non-committing encryption scheme by Bellare et al. [6]. For the scheme of [6], the size of a ciphertext is $3(n + 1)$.
2. The receiver non-committing encryption scheme by Jarecki and Lysyanskaya [22] and Canetti et al. [11]. For the scheme of [22,11], the size of a ciphertext is $2(n + 1)$.
3. The somewhat non-committing encryption by Garay et al. [20], with equivocality parameter $\ell = 2$.

Next, the efficiency of the somewhat non-committing encryption protocol of [20] for equivocality parameter $\ell = 2$ is analyzed. This scheme uses the following tools.

1. A simulatable PKE scheme. ElGamal encryption scheme [17] is a simulatable PKE scheme.
2. A secret key encryption scheme (SKE). One possible choice for the SKE is AES in CBC mode.
3. The non-committing encryption scheme by Damgård and Nielsen [16].

The non-committing encryption scheme by Damgård and Nielsen [16] uses a subroutine named *attempt*. The sender and the receiver jointly execute the subroutine *attempt* sequentially until one call of *attempt* becomes successful. The subroutine *attempt* has a success probability of $\frac{1}{2}$. It follows that the expected number of repeats of *attempt* is two. In [Theorem 2, [16]], it is mentioned that $4n$ calls of *attempt* gives n successful ones except with probability $exp\{-n/2\}$, which is negligible in the security parameter n. This follows from the Markov inequality. In order to ensure that the probability of failure of subroutine *attempt* remains negligible in n, the non-committing encryption scheme of [16] has to repeat $4n$ calls of *attempt*. That means, the worst case number of repeats of *attempt* is $4n$ to ensure that the probability of failure of *attempt* remains negligible in the security parameter.

The function *attempt* uses a simulatable PKE scheme. ElGamal cryptosystem [17] is a simulatable encryption scheme. For ElGamal cryptosystem [17], the size of a plaintext is n and the size of a ciphertext is $2n$. Each call of *attempt* has communication cost $(12n + 1)$. The non-committing encryption scheme of [16] needs communication complexity of $(48n^2 + 4n + 1)$ for message size of one bit. Each call of *attempt* uses one encryption operation of a simulatable PKE scheme, so the number of PKE for *attempt* is 1. Then the non-committing encryption scheme of [16] needs $4n$ PKE operations in the worst case.

The communication complexity of somewhat non-committing encryption of [20], with equivocality parameter $\ell = 2$, for transmitting message of size $q_1(n)$ is $\big(48n^2 + 14n + 1 + 2q_1(n)\big)$.

The somewhat non-committing encryption protocol of [20] uses the non-committing encryption protocol of [16] for sending an index $i \in \{1, \ldots, \ell\}$. As mentioned in [20], the expected number of PKE operations for this step is $O(\log \ell)$. In the worst case, this step requires $4n$ PKE operations. The somewhat non-committing encryption protocol of [20] uses one more encryption operation of a simulatable encryption scheme. So, the worst case number of PKE operations of the somewhat non-committing encryption protocol of [20] is $(4n + 1)$.

Next, the efficiency of protocol Π_{OSC} of [21] for sending message of length $q_2(n)$ is analyzed. Here, $q_2(n)$ is a polynomial of n. In protocol Π_{OSC}, in step 1, the receiver sends a message of size $q_1(n) = 4n$, using the somewhat non-committing encryption of [20] with equivocality parameter $\ell = 2$. The communication cost for step 1 is $48n^2 + 14n + 1 + 2 \times 4n = 48n^2 + 22n + 1$. The number of PKE operations for step 1 is $(4n{+}1)$. In step 2 of Π_{OSC}, the sender sends two messages of size $\big(q_2(n) \cdot \frac{1}{n} \cdot (3n + 3)\big)$ and $\big(q_2(n) \cdot \frac{1}{n} \cdot (2n + 2)\big)$ using two instances of the somewhat non-committing encryption of [20] with equivocality parameter $\ell = 2$. The communication cost for step 2 is $96n^2 + 28n + 2 + q_2(n) \cdot \frac{1}{n} \cdot (5n + 5)$. The number of PKE operations for step 2 is $(4n + 1)$.

Total communication cost of Π_{OSC} for sending message of size $q_2(n)$ is $\left(144n^2 + 50n + 3 + q_2(n) \cdot \frac{1}{n} \cdot (5n+5)\right)$. Total number of PKE operations of Π_{OSC} is $4n + 1 + 4n + 1 = 8n + 2$.

The OT protocol by Hazay and Patra [21] uses the dual-mode PKE scheme Π_{DUAL} by Peikert et al. [29]. The size of ciphertext for Π_{DUAL} is $2n$. The bit OT protocol performs two encryptions of Π_{DUAL}. The bit OT protocol sends two ciphertexts of Π_{DUAL} using two instances of Π_{OSC}.

Total communication cost of the bit OT protocol of [21] (excluding the zero-knowledge proofs) is $2 \times (144n^2 + 50n + 8) = 288n^2 + 100n + 16 \in O(n^2)$.

The total number of PKE operations of the bit OT of [21], in the worst case, is $2 + 2 \times (8n + 2) = 16n + 6$.

For string OT of size n by the protocol of [21], the communication complexity is $(288n^2 + 110n + 16)$ and the number of PKE operations in the worst case is $(16n + 6)$.