Private Over-threshold Aggregation Protocols over Distributed Databases

Myungsun Kim[1], Abedelaziz Mohaisen[2], Jung Hee Cheon[3], and Yongdae Kim[4]

[1] The University of Suwon, Suwon, South Korea
msunkim@suwon.ac.kr
[2] VeriSign Labs, VA, USA
amohaisen@verisign.com
[3] Seoul National University, Seoul, South Korea
jhcheon@snu.ac.kr
[4] Korea Advanced Institute of Science and Technology, Daejeon, South Korea
yongdaek@ee.kaist.ac.kr

Abstract. In this paper, we revisit the private over-threshold data aggregation problem, and formally define the problem's security requirements as both data and user privacy goals. To achieve both goals, and to strike a balance between efficiency and functionality, we devise a novel cryptographic construction that comes in two schemes; a fully decentralized construction and its practical but semi-decentralized variant. Both schemes are provably secure in the semi-honest model. We analyze the computational and communication complexities of our construction, and show that it is much more efficient than the existing protocols in the literature. Finally, we show that our basic protocol is efficiently transformed into a stronger protocol secure in the presence of malicious adversaries, together with performance and security analysis.

Keywords: Network traffic distribution, data aggregation, privacy preservation, malicious security

1 Introduction

Of particular interest in many applications is the problem of *computing the over-threshold elements*, elements whose count is greater than a given value, in a *private* manner. A typical application that involves such primitive is network traffic distribution, where n network sensors need to jointly analyze the security alert broadcasted by different sources in order to find suspect sites. In such an application, and without losing generality, each of such sensors has a set of suspects and would like to collaboratively compute the most frequent elements on each of these sets (e.g., the count greater than κ, referred to as κ^+) without revealing the set of suspects to other sensors with whom she collaborates.

Let there be n users denoted by $u_i, 1 \leq i \leq n$, and each of them have a private *multiset* $\mathtt{X}_i$ of cardinality k. For simplicity, assume that the cardinality of each multiset is equal to each other.

Private κ^+ Aggregation Problem Let $\zeta, \kappa \in \mathbb{N}$, and assume κ has been implicitly predefined among all users. Then the problem at hand is defined as follows: Given n multisets of cardinality k, find a set $Z = \{\alpha_1, \ldots, \alpha_\zeta\} \subset \mathtt{U} = \bigcup_{i=1}^{n} \mathtt{X}_i$ such that (*i*) for all elements $\alpha \in \mathtt{U}$, if α has a multiplicity greater than or equal to κ, then $\alpha \in Z$, (*ii*) no polynomial-time algorithm can learn any element other than the output of a κ^+ protocol, and (*iii*) no polynomial-time algorithm should know which output of the execution belongs to which user [23]. As pointed out in [19], using a trusted third party (TTP) to achieve the end goal is impractical since it is hard to find such entity in many settings. Also, using security multiparty computations (SMC) is impractical since they are computationally expensive. A final approach is to use existing private set-operation protocols such as [20,31], especially multiset union protocols. These protocols securely compute all elements appearing in the union of input multisets at least τ times. However all of them fail to show the distribution of the resulting elements.

Our Approach—Informal Descriptions The non-trivial part of κ^+ protocols is that we should satisfy two privacy requirements simultaneously: the data- and user-privacy.

From the literature, e-voting protocols have similar requirements. In these protocols, each ballot is mixed with a shuffle scheme to remove linkability between voters and ballots. Thus when each element in multisets is encrypted and shuffled using e-voting protocols, all encrypted elements can be decrypted with hiding linkability. However, all non-κ^+ elements also are revealed, which violates privacy in our applications. Thus we demand a way to preserve privacy even when all encrypted elements were decrypted.

For that we introduce an efficient function E that commutes with an underlying public-key encryption $\mathsf{Enc}_{pk}(\cdot)$ under a public key pk. We demand that: (i) for all s and for all pk, $E_s \circ \mathsf{Enc}_{pk} = \mathsf{Enc}_{pk} \circ E_s$, and (ii) for all elements α, given $\mathsf{Dec}_{sk}(\mathsf{Enc}_{pk}(E_s(\alpha)))$ no algorithm can efficiently find α without s, where $\mathsf{Dec}(\cdot)$ is a corresponding decryption algorithm. We call this notion *double encryption*.

In conclusion, our main technique is to shuffle elements doubly encrypted. We need to remark that shuffle schemes used in e-voting systems rely on the property that their underlying homomorphic encryption is re-randomizable, while it does not change the plaintexts of input ciphertexts (e.g. [12,25,17,16]). Rather, in our protocols a double encryption scheme does not preserve the plaintexts of input ciphertexts, but it still gives a way to recover the plaintexts.

Summary of Our Results We present a formal definition of private κ^+ protocol and its security in a *fully decentralized* system among n users over non-partitioned data. We refrain from using SMC and construct an efficient private κ^+ protocol which is both data- and user-private. In its efficiency, our construction is comparable to [2], which achieves its efficiency by giving up the decentralized model, and in its security guarantees is comparable to the work in [3], which is secure but expensive. Our protocol has several desirable features as follows: (1) It has $\mathcal{O}(n^2k)$ computational complexity where n is the number of users and k is the cardinality of each user's set, assuming $\kappa \leq k$, (2) It has $\mathcal{O}(n^2k)$ communication complexity, and (3) It has a linear round complexity in the number of users. In real-world applications, n is much smaller than k, which further justifies the efficiency of our protocol. It remains an open problem to devise a protocol with round complexity that does not depend on the number of users.

Last but not least, we construct a stronger protocol for private over-threshold aggregation, where "stronger" means the protocol is secure against malicious adversaries. Unfortunately, since there is a tradeoff between security and complexity we have to pay more computation and communication costs for a stronger security.

Organization In Section 2, we discuss the related work. In Section 3, we outline the preliminaries, including double encryption, our formalism of κ^+, and cryptographic primitives used in our protocol. In Section 4, we introduce our construction. In Sections 4.2 and 4.3 we analyze the security and complexity of our work. In Section 5, we provide a full description of a κ^+ protocol which is secure in the malicious model and analyze its security in the simulation paradigm. Concluding remarks are in Section 6.

2 Related Work

The related literature work is three directions: fully centralized, fully decentralized, and semi-decentralized [23]. While the centralized schemes assume the existence of a trusted third party (TTP), making them of less interest from a practical point of view, the fully decentralized schemes utilize cryptographic primitives and protocols to replace the TTP. Finally, semi-decentralized schemes try to bridge the functional and security gap between the two other directions. As they are of particular relevance to our work, we limited our discussion to the decentralized and semi-decentralized protocols.

Decentralized solutions replace the TTP with cryptographic constructions, which come in various forms. One form is based on SMC, as in [3]. However, SMC's inefficiency is its main drawback: since it uses Yao's garbled circuits [35], the computational resources required for ordinary data sizes are overwhelmingly high. Furthermore, as the datasets become disjoint, the efficiency of such construction decreases sharply. Burkhart and Dimitropoulos devised a construction in which *the round complexity is linear to the number of bits* in the data elements [3]. However, due to using sketches to improve its efficiency, the aggregate results are *probabilistic*. While computationally efficient, the protocol has also a high round complexity. Kissner and Song [20] devised an over-threshold set union protocol—where *a threshold value τ is given in advance*—to find all elements appearing in the union of input multisets at least τ times. The protocol requires apriori knowledge of τ, although operates in a decentralized manner. We compare it to our work in §4.3.

Semi-decentralized constructions, represented by the work of Applebaum et al. in [2], aim to enhance the efficiency of fully-decentralized instantiations by adding new entities: proxy and database (DB). However the proxy and the DB are assumed to be semi-honest restricting the possibility of coalition between proxy and DB. This allows obtaining a constant round protocol. While the authors claim that both proxy and DB are expected to act as semi-honest, it might be a strong assumption both theoretically and practically. Furthermore, their scheme extensively relies on oblivious transfer (OT) [24], which is a very expensive public-key primitive since it may require two modular exponentiations per invocation, and run for each bit of the user's data element.

To sum up, Table 1 summarizes properties and the efficiency features of existing solutions compared with our proposed protocol. Computational complexity is expressed in terms of the number of multiplications over modulo p, and assuming that multisets have values less than the prime p. Note that Applebaum et al.'s protocol requires both ElGamal encryption [9] and Goldwasser-Micali encryption [14], but we assume that both encryption systems use the same size modulus.

Table 1. Summary and Comparison

	Comm. Model	Round Cpx	Comp. Cpx	Comm. Cpx
Ours	Fully decentralized	$\mathcal{O}(n)$	$\mathcal{O}(n^2k\log p)$	$\mathcal{O}(n^2k\log p)$
[3]	Fully decentralized	$\mathcal{O}(n(n+k\log k)\log p)$	$\mathcal{O}(n^2k)$	$\mathcal{O}(n^2k\log p)$
[2]	Semi-decentralized	$\mathcal{O}(1)$	$\mathcal{O}(nk\log^2 p)$	$\mathcal{O}(nk\log p)$

Data Aggregation Data aggregation is one of important technologies in distributed systems, enabling efficient utilization of resources. Thus many researchers in wireless and smart grid networks have focused on data aggregation schemes [18,10,33,15,28,34,11,21,22,30]. However, data aggregation is used in this paper differently. We consider data aggregation as a part of data and information mining process where data is searched, gathered and presented in a report-based, summarized format to achieve specific business objectives. In particular, we are interested in keeping privacy during the whole process of data aggregation.

3 Preliminaries

Let us denote $F(\alpha)$ for the *multiplicity* (also known as frequency) of an element α in a multiset $\mathtt{X}$ and $F(\mathtt{X})$ for the collection of multiplicities for all elements in the multiset $\mathtt{X}$—here the multiplicity

$F(\alpha)$ of an element α refers to how many times the element appears in $\mathtt{X}$. For $n \in \mathbb{N}$, $[1, n]$ denotes the set $\{1, \ldots, n\}$.

If A is a probabilistic polynomial-time (PPT) machine, we use $a \leftarrow A$ to denote A which produces output according to its internal randomness. If X is a set, then $x \overset{\$}{\leftarrow} X$ is used to denote sampling from the uniform distribution on X. We shall write

$$\Pr[x_1 \overset{\$}{\leftarrow} X_1, x_2 \overset{\$}{\leftarrow} X_2(x_1), \ldots, x_n \overset{\$}{\leftarrow} X_n(x_1, \ldots, x_{n-1}) : \varphi(x_1, \ldots, x_n)]$$

to denote the probability that when x_1 is drawn from a certain distribution X_1, and x_2 is drawn from a certain distribution $X_2(x_1)$, possibly depending on the particular choice of x_1, and so on, all the way to x_n, the predicate $\varphi(x_1, \ldots, x_n)$ is true.

A function $\mu : \mathbb{N} \to \mathbb{R}$ is *negligible* if for every positive polynomial $p(\cdot)$ there exists an integer L such that $\mu(\lambda) < 1/p(\lambda)$ for all $\lambda > L$. Let $X = \{X(a, \lambda)\}_{a \in \{0,1\}^*, \lambda \in \mathbb{N}}$ and $Y = \{Y(a, \lambda)\}_{a \in \{0,1\}^*, \lambda \in \mathbb{N}}$ be distribution ensembles. We say that X and Y are *computationally indistinguishable*, which is denoted by $X \overset{c}{\approx} Y$, if for every polynomial-time algorithm D there exists a negligible function $\mu(\cdot)$ such that (for every $a \in \{0,1\}^*$ and $\lambda \in \mathbb{N}$)

$$|\Pr[D(X(a, \lambda)) = 1] - \Pr[D(Y(a, \lambda)) = 1]| < \mu(\lambda)$$

3.1 Definitions

We begin by reviewing the definitions of public-key encryption (PKE) [14].

Definition 1 (Public-key Encryption Scheme) $\mathcal{E} = (\mathsf{KG}, \mathsf{Enc}, \mathsf{Dec})$ *is a* public-key encryption scheme *if* KG, Enc, *and* Dec *are polynomial-time (PT) algorithms defined as follows:*

- KG, *given a security parameter λ, outputs a pair of keys (pk, sk), where pk is a public key and sk is a secret key. We denote this by $(pk, sk) \leftarrow \mathsf{KG}(1^\lambda)$. The key pk also describes the plaintext and ciphertext message spaces $\mathbf{M}_{pk}, \mathbf{C}_{pk}$ respectively.*
- Enc, *given the public key pk and a plaintext m, outputs a ciphertext c encrypting m, which is denoted by $c \leftarrow \mathsf{Enc}_{pk}(m)$; and when we sometimes need to emphasize the randomness r used for encryption, we denote this by $c \leftarrow \mathsf{Enc}_{pk}(m; r)$.*
- Dec, *given the public key pk, secret key sk and a ciphertext message c, outputs a plaintext m such that there exists randomness r for which $c = \mathsf{Enc}_{pk}(m; r)$; otherwise outputs $\perp$. We denote this by $m \leftarrow \mathsf{Dec}_{sk}(c)$. We omit pk for simplicity.*

$\mathcal{E}$ is said to be *correct* if $m = \mathsf{Dec}_{sk}(\mathsf{Enc}_{pk}(m))$, for any key-pair $(pk, sk) \leftarrow \mathsf{KG}(\lambda)$ and any $m \in \mathbf{M}_{pk}$. Furthermore, for $\mathcal{E} = (\mathsf{KG}, \mathsf{Enc}, \mathsf{Dec})$ and a PT adversary $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$, we consider the semantic security game defined as follows.

Definition 2 (Semantic Security) *A public-key cryptosystem $\mathcal{E} = (\mathsf{KG}, \mathsf{Enc}, \mathsf{Dec})$ with a security parameter λ is called* semantically secure (or IND-CPA secure) *if after the standard CPA game being played with any PPT adversary $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$, the advantage $\mathsf{Adv}^{\mathsf{cpa}}_{\mathcal{E},\mathcal{A}}(\lambda)$ is negligible in λ for all sufficiently large λ, where:*

$$\mathsf{Adv}^{\mathsf{cpa}}_{\mathcal{E},\mathcal{A}}(\lambda) = \left| \Pr_{b,r} \begin{bmatrix} (pk, sk) \leftarrow \mathsf{KG}(\lambda), (\mathsf{state}, m_0, m_1) \leftarrow \mathcal{A}_1(pk), \\ c = \mathsf{Enc}_{pk}(m_b; r) : b \leftarrow \mathcal{A}_2(\mathsf{state}, m_0, m_1, c) \end{bmatrix} - \frac{1}{2} \right|.$$

We say that a public-key encryption scheme $\mathcal{E} = (\mathsf{KG}, \mathsf{Enc}, \mathsf{Dec})$ is *homomorphic* if the plaintext message and ciphertext message spaces are groups $\mathbf{M}_{pk}$ and $\mathbf{C}_{pk}$, respectively, and for any plaintext messages $m_1, m_2 \in \mathbf{M}_{pk}$,

$$(pk, \mathsf{Enc}_{pk}(m_1), \mathsf{Enc}_{pk}(m_1) \cdot \mathsf{Enc}_{pk}(m_2)) \equiv (pk, \mathsf{Enc}_{pk}(m_1), \mathsf{Enc}_{pk}(m_1 \cdot m_2)),$$

where the binary operator $(\cdot)$ is carried out in the groups $\mathbf{M}_{pk}$ and $\mathbf{C}_{pk}$, respectively, and the encryptions of m_1, m_2 and $(m_1 \cdot m_2)$ are independently random.

Relying on the PKE scheme does not achieve our security, so we propose a double encryption scheme to aid that. A double encryption scheme is a pair of encryption schemes $\mathcal{E} = (\mathsf{KG}, \mathsf{Enc}, \mathsf{Dec})$ and $\mathsf{E} = (G, E, D)$ such that $\mathsf{Enc} \circ E = E \circ \mathsf{Enc}$. Here E need to be *deterministic* to know a complete distribution of multisets without revealing their elements.

Definition 3 (Double Encryption) *Let $\mathcal{E} = (\mathsf{KG}, \mathsf{Enc}, \mathsf{Dec})$ be a PKE scheme defined as above with $(pk, sk) \leftarrow \mathsf{KG}(1^\lambda)$ and a plaintext message space (resp., ciphertext message space) $\mathbf{M}_{pk}$ (resp., $\mathbf{C}_{pk}$). A pair $(\mathcal{E}, \mathsf{E})$ is called* double encryption *if there exists a triple of PT computable functions, $\mathsf{E} = (G, E, D)$, that satisfies the following properties:*

- *A probabilistic function $G(1^\lambda)$ takes as input a parameter λ, and outputs (s, s') s.t. $\forall s, s'$ and $\forall\, m \in \mathbf{M}_{pk}$, $m = D_{s'}(E_s(m))$, E and D are deterministic.*
- *$\forall pk, s$, $\forall m \in \mathbf{M}_{pk}$, $\mathsf{Enc}_{pk}(E_s(m)) = E_s(\mathsf{Enc}_{pk}(m))$ up to the randomness of $\mathsf{Enc}_{pk}(\cdot)$.*
- *$\forall\, c \in \mathsf{Enc}(m)$, $E_s(m) = \mathsf{Dec}_{sk}(E_s(c))$.*

We observed that a double encryption scheme is necessary but not sufficient for constructing a secure over-threshold protocol. For that, we consider a shuffle scheme. We argue that double encryption and shuffle are a necessary and sufficient condition for a *secure* over-threshold aggregation protocol.

A shuffle of a list of ciphertexts $e_1, \ldots, e_n$ is a newly updated set of a list of ciphertexts $e'_1, \ldots, e'_n$ with the same plaintexts in permuted order. We require that a shuffle scheme be verifiable to prove the correctness of the shuffle. We follow the definition given by Nguyen et al. [26].

Definition 4 (Shuffle) *Let $\mathcal{E} = (\mathsf{KG}, \mathsf{Enc}, \mathsf{Dec})$ be a public-key encryption scheme defined as in Definition 1. A* shuffle *is a pair of PT algorithms $(\mathcal{E}, \mathsf{Shuffle})$ defined as:*

- $\mathsf{Shuffle}$*: given a public key pk, a list of ciphertexts $(e_1, \ldots, e_n)$ and a random permutation π on $[1, n]$, the shuffle outputs a list of ciphertexts $(e'_1, \ldots, e'_n)$.*

The $\mathsf{Shuffle}$ is said to be correct *if $\forall i \in [1, n]$ there exists i such that $\mathsf{Dec}_{sk}(e_i) = \mathsf{Dec}_{sk}\left(e_{\pi(i)}\right)$.*

Then we say that a shuffle $(\mathcal{E}, \mathsf{Shuffle})$ is *verifiable* if there exists a proof system (P, V) such that, given a public key pk, a list of input ciphertexts $(e_1, \ldots, e_n)$ and a list of output ciphertexts $(e'_1, \ldots, e'_n)$, it proves that $\mathsf{Shuffle}$ is correct.

Now we are ready to define an *over-threshold aggregation protocol and sometimes abbreviate it by κ^+ protocol*. Throughout the paper, we use Σ_n to denote the set of all permutations on $[1, n]$. A κ^+ protocol consists of five computable (in PT) algorithms, (Setup, $\mathsf{DEncrypt}$, $\mathsf{Shuffle}$, $\mathsf{Aggregate}$, Reveal), over a double encryption $(\mathcal{E}, \mathsf{E})$, which are explained as follows:

- $(pk, sk, s, s') \leftarrow \mathsf{Setup}(1^\lambda)$**:** The setup algorithm takes as input a security parameter λ, and outputs the public and secret parameters by invoking $(pk, sk) \leftarrow \mathsf{KG}(1^\lambda)$ and $(s, s') \leftarrow G(1^\lambda)$.
- $(E_s(c_1), \ldots, E_s(c_n)) \leftarrow \mathsf{DEncrypt}(pk, s, c_1, \ldots, c_n)$**:** $\mathsf{DEncrypt}$ takes as input system parameters (pk, s) and a list of ciphertexts $(c_1, \ldots, c_n)$, and produces a list of doubly encrypted ciphertexts $(E_s(c_1), \ldots, E_s(c_n))$.

- $\left(E_s(c_{\pi(1)}),\ldots,E_s(c_{\pi(n)})\right) \leftarrow \mathsf{Shuffle}(\pi, E_s(c_1),\ldots,E_s(c_n))$: Shuffle chooses a random permutation $\pi \in \Sigma_n$ and shuffles the doubly encrypted ciphertexts $(E_s(c_1), \ldots, E_s(c_n))$, and then outputs the mixed list.
- $(E_s(\alpha_1),\ldots,E_s(\alpha_\zeta)) \leftarrow \mathsf{Aggregate}\left(pk, sk, E_s\left(c_{\pi(1)}\right),\ldots,E_s\left(c_{\pi(n)}\right)\right)$: Aggregate takes as input a pair of keys (pk, sk) and a list of permuted, doubly encrypted ciphertexts, and computes $\mathsf{Dec}_{sk}\left(E_s\left(c_{\pi(i)}\right)\right) = E_s\left(\alpha_{\pi(i)}\right)$ for all $i \in [1,n]$. Then it computes $F\left(E_s\left(\alpha_{\pi(i)}\right)\right)$ for each $i \in [1,n]$, and outputs only the elements whose multiplicity is greater than or equal to κ.
- $(\alpha_1,\ldots,\alpha_\zeta) \leftarrow \mathsf{Reveal}\left(pk, s', E_s(\alpha_1),\ldots,E_s(\alpha_\zeta)\right)$: Reveal outputs the most frequent κ elements by computing $D_{s'}\left(E_s(\alpha_j)\right)$ for all $j \in [1,\zeta]$.

3.2 Security Definition

We need to prove that our protocol is secure in the presence of *malicious adversaries*; adversaries that may arbitrarily deviate from the protocol specification. Following the standard technique, the security of our protocol is analyzed by comparing what an adversary can do in a *real-world* execution to what he can do in an *ideal world*. In the ideal world, the computation is carried out by an incorruptible trusted party to which the users send their inputs over secure channel. Then the trusted party performs a functionality on the inputs and sends to each user its output. The protocol is secure if any adversary in the real world can do no more harm than what he could do in the ideal world. We only consider a static adversary who can corrupt a fixed number of users before executing the protocol. Thus we cannot expect to always succeed in achieving fairness or guaranteed output delivery.

Execution in the Ideal Model In the ideal world execution, each user sends his input to a trusted party who computes the output. If the user is honest, he sends directly his input to the trusted party, whereas a corrupted user may replace his input with any other value of the same length. After computing the output, the trusted party first sends the output of the corrupted users to the adversary, and the adversary then decides whether each honest user receives the output or aborts the protocol. Let f be an n-party functionality, let $\mathcal{A}$ be a PT algorithm, and $\Upsilon \subsetneq \{u_1,\ldots,u_n\}$ be a set of corrupted users. Then the ideal execution of f on inputs $(x_1,\ldots,x_n)$, auxiliary input z to $\mathcal{A}$, and security parameter λ is defined as the outputs of the honest users and the adversary from the above execution. We denote this by $\mathtt{IDEAL}_{f,\mathcal{A}(z),\Upsilon}(x_1,\ldots,x_n;\lambda)$.

Execution in the Real Model In the real-world model there is no trusted party and the users exchange rounds of communication with each other. The adversary $\mathcal{A}$ has a full control over the corrupted users and thus can send all messages on their behalf. The adversary does not have to send messages according to the protocol, and may follow any PT strategy. However, an honest user follows the instructions described in the protocol.

Let f be as above, let $\wp$ be an n-party protocol for computing f, let $\mathcal{A}$ be a PPT algorithm and let Υ be a set of corrupted users. The real-world execution of $\wp$ on inputs $(x_1,\ldots,x_n)$, auxiliary input z to $\mathcal{A}$, and security parameter λ is defined as the outputs of the honest users and the adversary $\mathcal{A}$ from executing $\wp$. We denote this by $\mathtt{REAL}_{\wp,\mathcal{A}(z),\Upsilon}(x_1,\ldots,x_n;\lambda)$.

Security as Simulation of a Real-world Execution in the Ideal Model Using the definitions of the real and ideal models, we can then define the security of protocols. Informally, simulating all behaviors of a real-world adversary $\mathcal{A}$ in the ideal model with a trusted party implies that none of his activities causes damage to the real-world protocol.

Definition 5 (Secure Protocol) *Let f and $\wp$ be as described above. Then, we say that protocol $\wp$ securely computes f in the malicious model if for all PPT adversaries $\mathcal{A}$ in the real model, there exists a PPT adversary $\mathcal{S}$ in the ideal model, such that for all $\varUpsilon \subsetneq \{u_1, \ldots, u_n\}$,*

$$\left\{\mathtt{IDEAL}_{f,\mathcal{A}(z),\varUpsilon}(x_1, \ldots, x_n; \lambda)\right\} \overset{c}{\approx} \left\{\mathtt{REAL}_{\wp,\mathcal{A}(z),\varUpsilon}(x_1, \ldots, x_n; \lambda)\right\}.$$

We would like to note that for designing a secure κ^+ protocol against malicious adversaries, we begin by constructing a basic protocol secure in the semi-honest model where even the adversary must follow the instructions specified in the protocol. We then transform it into a maliciously secure protocol, which is an effective technique widely used in this field. Here we briefly presented the standard definition for security of protocols. Readers may refer to [13, Chap. 7] for more details and motivating theories.

Data Privacy and User Privacy *Data privacy* requires that no users, or coalition of users, should learn anything about honest users' inputs except what can be trivially derived from the output itself. In this field "privacy" in a protocol usually implies data privacy.

In the following we explain and motivate for user privacy. Let $Z = \{\alpha_1, \ldots, \alpha_\zeta\}$ for some $\zeta \in \mathbb{N}$ be the output of a κ^+ protocol. We say that a κ^+ protocol is *user-private* if no user or coalition of users can gain a non-negligible advantage in distinguishing an honest user u_i for all $\alpha \in Z$ such that $\alpha \in \mathtt{X}_i$. In case of data privacy its necessity and notion are obvious, but comparatively user privacy may not appeal to readers. For better understanding, we thus give an alternative description of user privacy in Appendix A.

3.3 Cryptographic Assumptions and Tools

Next we outline the cryptographic tools and assumptions our protocols rely on.

Let $\mathbb{G}$ be a finite cyclic group of large prime order q, and let $g \in \mathbb{G}$ be a generator. Given $h \in \mathbb{G}$, the discrete logarithm (DL) problem requires us to compute $x \in \mathbb{Z}_q$ such that $g^x = h$.

Definition 6 (DL Assumption) *We say that the* discrete logarithm problem is intractable *with respect to $\mathbb{G}$ if for every PPT algorithm $\mathcal{A}$ there exists a negligible function μ in λ such that*

$$\Pr_{g,x}\left[\mathcal{A}\left(\mathbb{G}, q, g, g^x\right)\right] \leq \mu(\lambda).$$

A stronger assumption is the Decisional Diffie-Hellman (DDH) assumption. The DDH problem says that given $\mathbb{G}$ with a generator g, and three elements $a, b, c \in \mathbb{G}$, we are asked to decide whether there exist $x, y \in \mathbb{Z}_q$ such that $a = g^x, b = g^y$, and $c = g^{xy}$. More formally, the DDH assumption can be state as follows.

Definition 7 (DDH Assumption) *We say that the* decisional Diffie-Hellman problem is intractable *with respect to $\mathbb{G}$ if for every PPT algorithm $\mathcal{A}$ there exists a negligible function μ in λ such that*

$$\left|\Pr\left[\mathcal{A}\left(\mathbb{G}, q, g, g^x, g^y, g^z\right) = 1\right] - \Pr\left[\mathcal{A}\left(\mathbb{G}, q, g, g^x, g^y, g^{xy}\right) = 1\right]\right| \leq \mu(\lambda),$$

where the probabilities are taken over the choices of g and $x, y, z \in \mathbb{Z}_q$.

The ElGamal Public Key Encryption Scheme We extensively use the ElGamal encryption scheme [9] in this work. Throughout this paper, we work in the group $\mathbb{Z}_p^\times$ where $p = 2q + 1$ is also prime, and set $\mathbb{G}$ be a cyclic subgroup of $\mathbb{Z}_p^\times$ of quadratic residues modulo p.

Let g be a random generator of $\mathbb{G}$. Then the public and secret keys are $\langle\mathbb{G}, q, g, h\rangle$ and $\langle\mathbb{G}, q, g, x\rangle$, respectively, where $x \overset{\$}{\leftarrow} \mathbb{Z}_q$ and $h = g^x \mod p$. A plaintext message $m \in \mathbb{G}$ is encrypted by choosing $r \xleftarrow{\$} \mathbb{Z}_q$ and the ciphertext message is $(g^r, m \cdot h^r)$. A ciphertext message $c = (u, v)$ is decrypted as $m = v/u^x$. Later we use the property that if one knows $r = \log_g u$ he can reconstruct $m = v/h^r$ and thus a user who encrypted m can prove knowledge of plaintext m by proving knowledge of randomness r. The semantic security of the ElGamal encryption scheme relies on the DDH assumption in $\mathbb{G}$. In reality, we use its threshold version since our system consists of multiple users $(n > 2)$. See Appendix B.1 for the threshold ElGamal encryption scheme.

Instantiating a Double Encryption Scheme using the ElGamal PKE Scheme We give a concrete instantiation of a double encryption scheme using the standard ElGamal encryption scheme. Let a standard CPA-secure ElGamal encryption $\mathcal{E} = (\mathsf{KG}, \mathsf{Enc}, \mathsf{Dec})$ be defined as above. Now $\mathsf{E} = (G, E, D)$ is defined as follows:

- A probabilistic function $G(1^\lambda)$ outputs $(s, s') \in (\mathbb{Z}_q)^2$ such that $ss' = 1 \pmod q$.
- Given $\alpha \in \mathbb{G}$, $E : \mathbb{G} \to \mathbb{G}$ is given by $\alpha \mapsto \alpha^s \mod p$.
- A deterministic function $D_{s'}(\beta)$ computes $\beta^{s'} \mod p$.

The following lemma shows that $(\mathcal{E}, \mathsf{E})$ is an example of double encryption.

Lemma 1 *Let $\mathcal{E}$ and E be defined as above. Then, $(\mathcal{E}, \mathsf{E})$ is a double encryption scheme.*

Proof. We can easily verify that $(\mathcal{E}, \mathsf{E})$ satisfies the conditions of double encryption. First we see that for all values $m \in \mathbb{G}_q$, $m = (m^s)^{s'} \pmod p$. For any message $m \in \mathbb{G}_q$, there exists $r' = rs$ s.t. $\left(g^{r'}, (m^s) \cdot y^{r'}\right) = ((g^r)^s, (my^r)^s)$. Lastly, for any ElGamal ciphertext $c = (u, v) \in (\mathbb{G}_q)^2$, where $u = g^r$ and $v = my^r$, $m^s = v^s \cdot (u^s)^{-x} \pmod p$. □

3.4 Statistically Hiding Commitment Schemes

We say a triple of PPT algorithms $(\mathsf{KG}_{\mathsf{com}}, \mathsf{com}, \mathsf{decom})$ is a *commitment scheme* if the algorithms are defined as follows:

- $\mathsf{KG}_{\mathsf{com}}$, given a security parameter λ, outputs a public commitment key ck which specifies a message space $\mathbf{M}_{ck}$, a randomizer space $\mathbf{R}_{ck}$ and a commitment space $\mathbf{C}_{ck}$.
- com, given a message m, outputs a commitment c to m, which is denoted by $c \leftarrow \mathsf{com}_{ck}(m)$. If we want to emphasize the randomizer r use for committing, we write $c \leftarrow \mathsf{com}_{ck}(m; r)$.
- decom, given a commitment c, outputs a message m and randomizer r for which $c = \mathsf{com}_{ck}(m, r)$ or $\perp$ if no such message exists. We denote this by $(m, r) \leftarrow \mathsf{decom}_{ck}(c)$.

Loosely speaking, statistically hiding commitment schemes allow us to bind, in the computational sense, between a user and a value chosen by her, without revealing this committed value in the informational theoretical sense. A commitment scheme with this statistical hiding property is required to achieve our security goal.

We say the commitment scheme is *statistically hiding* if it satisfies the following:

- *Unconditional Hiding.* Given two same-sized messages $m_0, m_1 \in \mathbf{M}_{ck}$,
$$\mathsf{com}_{ck}(m_0) \approx \mathsf{com}_{ck}(m_1).$$
- *Computational Binding.* For all PPT algorithm $\mathcal{A}$, there exists a negligible function μ in λ such that
$$\Pr_{r,r'}\left[(m, r, m', r') \leftarrow \mathcal{A}(c) \middle| c = \mathsf{com}_{ck}(m; r) \wedge c = \mathsf{com}_{ck}(m'; r')\right] \le \mu(\lambda).$$

Instantiations In this paper, we instantiate the commitment scheme with Pedersen's commitment [29], using same group $\mathbb{G}$ used in Section 3.3. More specifically, let $p = 2q+1$ where p, q are large primes and g, h be random generators of $\mathbb{G}$ of quadratic residues modulo p. A commitment to $m \in \mathbb{Z}_q$ is given as $\mathsf{com}_{ck}(m; r) := \mathsf{g}^m \mathsf{h}^r$ with $r \xleftarrow{\$} \mathbb{Z}_q$. The scheme is proven to be statistically hiding, as for all m, r, m' there exists a r' such that $\mathsf{g}^m \mathsf{h}^r = \mathsf{g}^{m'} \mathsf{h}^{r'}$ as well as biding under the DL assumption. However, given $\log_{\mathsf{g}} \mathsf{h}$ it is possible to efficiently decommit any commitment c to m.

In particular, since the Pedersen commitment scheme has an equivocal property, in our protocol secure against malicious players the simulator can open the commitment to an arbitrary value without being detected by the adversary.

3.5 Zero-knowledge Proofs of Knowledge

In this paper we utilize several zero-knowledge proofs (ZKP) of knowledge in our protocol for the malicious adversary model. All these proofs of knowledge play a crucial role of enforcing all players to correctly behave but all of them used in our protocol have been studied elsewhere. Due to lack of space, we delegate the basics of ZKP to the Appendix B.2.

All the proofs of knowledge described in this section have been proved zero-knowledge in a statistical or computational sense within the random oracle model, under the DL assumption and the DDH assumption explained above. On any DDH group $\mathbb{G}$ we can efficiently construct these ZKPs using standard constructions [32,5,6,16]. For better readability, we summarize all of them in Table 2.

Details of ZKPs used in our scheme are as follows, where $(pk, p, q, g, \mathbb{G})$ and ck are defined as above.

- $\mathsf{PoK}(h)$ denotes a ZKP that given $(p, q, \mathbb{G}, g, h)$, a prover *knows the discrete logarithm* to the base g of h.
- $\mathsf{PoK}(g_2, c)$ denotes a ZKP of *correct double encryption*, that given (pk, ck, c, g_2), a prover knows the discrete logarithm x of g_2 to the base g_1 to which $c = \mathsf{com}_{ck}(x; r)$ is the committed value, where g_1, g_2 are random elements in $\mathbb{G}$.
- $\mathsf{PoK}(e)$ allows a prover in a zero-knowledge manner to prove for a given ElGamal encryption $e \in \mathsf{Enc}_{pk}(m)$ that she *knows the corresponding plaintext* m.
- $\mathsf{PoK}(L, L')$ for *correct shuffle* is used to prove the relation

$$\begin{aligned}\mathcal{R} = \ & \Big\{ \big\langle \big(pk, e_1, \ldots, e_k, e'_1, \ldots, e'_k\big), (\pi, r_1, \ldots, r_k) \big\rangle \ \big| \\ & \qquad \pi \in \Sigma_k \wedge (r_1, \ldots, r_k) \xleftarrow{\$} (\mathbb{Z}_q)^k \wedge e'_i = e_{\pi(i)} \cdot \mathsf{Enc}_{pk}(1; r_i) \Big\} .\end{aligned}$$

The relation $\mathcal{R}_{\mathsf{CS}}$ is shorthand notation of the above relation $\mathcal{R}$ in this paper.

Table 2. Summary of Zero-knowledge Protocols

Protocol	Relation	References
$\mathsf{PoK}(h)$	$\mathcal{R}_{\mathsf{DL}} = \{\langle p, q, \mathbb{G}, g, h\rangle \mid \exists x \in \mathbb{Z}_q \text{ s.t. } h = g^x\}$	[32]
$\mathsf{PoK}(g_2, c)$	$\mathcal{R}_{\mathsf{DE}} = \big\{\langle ck, c, g_1, g_2\rangle \mid \exists (x, s) \in (\mathbb{Z}_q)^2 \text{ s.t. } g_2 = g_1^s \wedge c = \mathsf{com}_{ck}(s; r)\big\}$	[5,6]
$\mathsf{PoK}(e)$	$\mathcal{R}_{\mathsf{PK}} = \big\{\langle pk, e\rangle \mid \exists m \in \mathbb{Z}_q \text{ s.t. } e \in \mathsf{Enc}_{pk}(m)\big\}$	§ 3.3
$\mathsf{PoK}(L, L')$	Let $L = (e_1, \ldots, e_k)$ and $L' = (e'_1, \ldots, e'_k)$ where $e_{i \in [1,k]}, e'_{i \in [1,k]}$ are legal ElGamal ciphertext messages under the public key pk. $\mathcal{R}_{\mathsf{CS}} = \big\{\langle pk, L, L'\rangle \mid \exists \pi \in \Sigma_k, \exists r_i \in \mathbb{Z}_q \text{ s.t. } \forall i \in [1, k], e'_{\pi(i)} = e_i \cdot \mathsf{Enc}_{pk}(1; r_i)\big\}$	[16]

For implementation considerations, we assume that according to its input parameters one can invoke a correct ZKP protocol among those mentioned above as function overriding in the `C++` programming language.

4 A κ^+ Protocol for the Semi-Honest Model

In this section, we describe our construction for computing κ^+ elements privately. We begin by considering a basic setting of n users, denoted by $u_1, \ldots, u_n$. Let $\mathtt{X}_i = \{\alpha_{i,1}, \ldots, \alpha_{i,k}\}$ for each $i \in [1, n]$. Each user u_i has his private multiset $\mathtt{X}_i$, and the users wish to jointly compute

$$Z = \left\{ \alpha \in \bigcup_{i=1}^{n} \mathtt{X}_i \middle| F(\alpha) \geq \kappa \right\}.$$

For simplicity, assume that all elements are in a proper domain $\mathbf{M}_{pk} = \mathbb{G} \subset \mathbb{Z}_p^\times$ of an ElGamal encryption scheme.

4.1 Description

Let λ be a security parameter. As above, let p be a λ-bit prime such that for some prime q, $p = 2q + 1$, $\mathbb{G}$ be a finite cyclic subgroup of $\mathbb{Z}_p^\times$ whose order is q, and g be a generator of $\mathbb{G}$. We use $(\mathcal{E}, \mathsf{E})$ as a double encryption scheme given in §3.3 by a global parameter $\mathsf{params} = (\mathbb{G}, p, q, g)$. For a set $\mathtt{X} = \{\alpha_1, \ldots, \alpha_k\}$, we denote $\mathtt{X}^s$ as $\{\alpha_1^s, \ldots, \alpha_k^s\}$ for some $s \in \mathbb{Z}_q$.

With such notation, we proceed to describe our construction.

Setup(1^λ) For $i \in [1, n]$, each user u_i

1. selects a value $x_i \xleftarrow{\$} \mathbb{Z}_q$, computes $h_i = g^{x_i}$, and sets $sk = (\mathsf{params}, x_i)$ and $pk = \left(\mathsf{params}, h = \prod_{i=1}^n h_i = g^{\sum_{i \in [1,n]} x_i} \pmod{p}\right)$, for a threshold ElGamal encryption $\mathcal{E}$ with a public/private key pair (pk, sk).
2. distributes a share $(s_i, s_i') : s = \prod_{i=1}^n s_i$, $s' = \prod_{i=1}^n s_i'$, and $s \cdot s' = 1 \pmod{q}$.

DEncrypt Using the double encryption scheme,

1. Every user u_i first encrypts his multiset $\mathtt{X}_i$ as follows:

$$\mathsf{Enc}_{pk}(\mathtt{X}_i) = \{\mathsf{Enc}_{pk}(\alpha_{i,1}), \ldots, \mathsf{Enc}_{pk}(\alpha_{i,k})\}.$$

Then he sends $\mathsf{Enc}_{pk}(\mathtt{X}_i)$ to u_1.

2. u_1 computes $\{E_{s_1}(\mathsf{Enc}_{pk}(\mathtt{X}_1)), \ldots, E_{s_1}(\mathsf{Enc}_{pk}(\mathtt{X}_n))\}$. We denote this Y_0.

Shuffle & DEncrypt For $i \in [1, n]$, u_i receives a vector Y_{i-1}, and then computes its permuted and doubly encrypted version Y_i as follows:

1. $u_{i \neq 1}$ computes

$$\begin{aligned} E_{s_i}(Y_{i-1}) &= \{c_1, \ldots, c_{nk}\} \\ &= \left\{E_{s_i}\left(E_{s_{i-1}}\left(\cdots E_{s_1}\left(\alpha_{\pi_{i-1}(1)}\right)\cdots\right)\right), \ldots, E_{s_i}\left(E_{s_{i-1}}\left(\cdots E_{s_1}\left(\alpha_{\pi_{i-1}(nk)}\right)\cdots\right)\right)\right\} \end{aligned}$$

where $\alpha_{\pi_{i-1}(\ell)} = \alpha_{\pi_{i-1} \circ \cdots \circ \pi_1(\ell)}$ for all $\ell \in [1, nk]$.

2. u_i chooses a random permutation $\pi_i \in \Sigma_{nk}$, and applies it to a vector of c_ℓ; we denote the result Y_i.

3. u_i sends Y_i to u_{i+1}; the last user u_n sends Y_n to all users.

Aggregate Let $\mathtt{U} = \bigcup_{i=1}^{n} \mathtt{X}_i$. After every user has received $E_s(\mathsf{Enc}_{pk}(\mathtt{U}))$,

1. Each user participates in a group decryption and obtains

$$E_s(\mathtt{U}) = \left\{ E_s\left(\alpha_{\pi(1)}\right), \ldots, E_s\left(\alpha_{\pi(nk)}\right) \right\}$$

   where $\pi = \pi_n \circ \cdots \circ \pi_1$.

2. Every user computes $Z = \{E_s(\alpha) \in E_s(\mathtt{U}) \big| F(E_s(\alpha)) \geq \kappa\}$.

Reveal For all $i \in [1, n]$:

1. For every $E_s(\alpha) \in Z$, user u_i sends $D_{s'_i}(E_s(\alpha))$ to his neighbor u_{i+1}.
2. After receiving the outputs from u_n, user u_1 broadcasts $\alpha = D_{s'}(E_s(\alpha))$ to all other users.

Finally, each user obtains a κ^+ set including all elements $\alpha \in \mathtt{U}$ such that $F(\alpha) \geq \kappa$.

The advantage of this protocol is as follows. Compared to Kissner and Song's protocol [20], our scheme allows not only to find a threshold value itself but also to compute the "over threshold" set at the same computation and communication cost—whereas they incur different and higher costs in [20]. Compared to that described in [3], our protocol has much better computational complexity; see details in Section 4.3. In order to present a fair comparison between our protocol and Applebaum et al.'s protocol [2] we devise our protocol for a semi-decentralized model in Appendix C. The other purpose of our modification is to reduce the round complexity to a constant.

In threshold encryption schemes, during decryption process each user should output his share of the decrypted element, but if a user sends a uniformly generated random share instead of a valid share, then the decrypted element is uniformly random as well. If the decrypted element is uniform, the resulting decryption reveals no information to the users.

4.2 Security Analysis

In this section we prove that our protocol is secure in the semi-honest model. We then continue analyzing the efficiency of our protocol in the next section.

Theorem 1 (Correctness) *In the private κ^+ protocol in §4.1, every honest user learns the joint set distribution of all users' private inputs, i.e., each element $E_s(\alpha)$ such that $\alpha \in \bigcup_{i=1}^{n} \mathtt{X}_i$ and the number of times it appears, with overwhelming probability.*

Proof. Let $|\mathtt{U}| = nk$. After completing the algorithm Aggregate, each player learns a randomly permuted joint multiset $E_s(\mathtt{U}) = \left\{ E_s\left(\alpha_{\pi(1)}\right), \ldots, E_s\left(\alpha_{\pi(nk)}\right) \right\}$. Since π is a permutation, for each $E_s\left(\alpha_{\pi(\ell)}\right)$ and for all $\ell \in [1, nk]$, there exists a pair of the unique index ℓ^* such that

$$\begin{aligned} \ell^* &= \pi^{-1}(\ell) \\ &= \pi_n^{-1}(\ell) \circ \cdots \circ \pi_1^{-1}(\ell). \end{aligned}$$

Namely, $E_s\left(\alpha_{\pi(\ell)}\right)$ is a unique blinded version of $\alpha_{\ell^*} \in \bigcup_{i=1}^{n} \mathtt{X}_i$. Moreover, $\forall \ell, \ell^* \in [1, nk]$, $\alpha_\ell = \alpha_{\ell^*}$ if and only if $E_s\left(\alpha_\ell\right) = E_s\left(\alpha_{\ell^*}\right)$ with overwhelming probability. □

Corollary 1 *In the private κ^+ protocol in §4.1, every honest user learns all κ^+ elements in the union of private multisets with overwhelming probability.*

Now we show that our protocol satisfies the privacy requirements in the semi-honest model. Let $\mathcal{T}$ be a trusted party in the ideal world which receives the private input multiset $\mathtt{X}_i$ of size k from user u_i for $i \in [1, n]$, and then returns to every user the joint multiset distribution $\{F(\alpha)\}$ for all $\alpha \in \bigcup_{i=1}^{n} \mathtt{X}_i$.

Lemma 2 (Data Privacy) *Assume that the threshold ElGamal encryption scheme* $\mathcal{E} = (\mathsf{KG}, \mathsf{Enc}, \mathsf{Dec})$ *is semantically secure. In the private* κ^+ *protocol in §4.1, any coalition of less than* n *semi-honest users learn no more information than would be given by using the same private inputs in the ideal model with* $\mathcal{T}$.

Proof. We assume that the ElGamal encryption scheme is semantically secure, and so each user learns only

$$
\begin{aligned}
&\mathsf{Enc}_{pk}(\mathtt{X}_1), \ldots, \mathsf{Enc}_{pk}(\mathtt{X}_n);\\
&E_{s_1}(\mathsf{Enc}_{pk}(\mathtt{X}_1)), \ldots, E_{s_1}(\mathsf{Enc}_{pk}(\mathtt{X}_1)), \ldots, E_{s_{i-1}}(\mathsf{Enc}_{pk}(\mathtt{X}_1)), \ldots, E_{s_{i-1}}(\mathsf{Enc}_{pk}(\mathtt{X}_n));\\
&\qquad\vdots\\
&E_{s_{i-1}}(\cdots E_{s_1}(\mathsf{Enc}_{pk}(\mathtt{X}_1))\cdots), \ldots, E_{s_{i-1}}(\cdots E_{s_1}(\mathsf{Enc}_{pk}(\mathtt{X}_n))\cdots)
\end{aligned}
$$

during an execution. At the end of the protocol all users further know $E_s(\mathsf{Enc}_{pk}(\mathtt{U}))$ where $\mathtt{U} = \bigcup_{i=1}^{n} \mathtt{X}_i$, and for some $\gamma_{\ell\in[1,nk]} \in \mathbb{Z}_q$

$$
\begin{aligned}
E_s(\mathsf{Enc}_{pk}(\mathtt{U})) &= \{E_s(\mathsf{Enc}_{pk}(\mathtt{X}_1)), \ldots, E_s(\mathsf{Enc}_{pk}(\mathtt{X}_n))\}\\
&= \left(g^{\gamma_1}, \left(\alpha_{\pi(1)}\right)^s \cdot y^{\gamma_1}\right), \ldots, \left(g^{\gamma_{nk}}, \left(\alpha_{\pi(nk)}\right)^s \cdot y^{\gamma_{nk}}\right).
\end{aligned}
$$

As s is uniformly distributed, a group decryption of ElGamal encryptions reveals no more than

$$\{E_s(\alpha_\ell)\}_{\ell\in[1,nk]} = E_s\left(\bigcup_{i=1}^{n} \mathtt{X}_i\right) = E_s(\mathtt{U}).$$

We know the fact that $F(\alpha) = F(\alpha^s)$ for two multisets $\mathtt{X}$ and $E_s(\mathtt{X}) \in (\mathbb{G}_q)^k$, for all $s \in \mathbb{Z}_q$ and for all $\alpha \in \mathtt{X}$. Hence we see that

$$F(E_s(\mathtt{U})) = F\left(E_s\left(\bigcup_{i=1}^{n} \mathtt{X}_i\right)\right) = F\left(\bigcup_{i=1}^{n} \mathtt{X}_i\right) = F(\mathtt{U}),$$

which can be derived from the output returned by $\mathcal{T}$ in the ideal-world model. □

Lemma 3 (User Privacy) *Assume that the threshold ElGamal encryption* Enc *is semantically secure. The protocol in §4.1 is user-private against any coalition of less than* n *semi-honest users.*

Proof. Assume that there is at least an honest user in the system, and that the threshold ElGamal encryption $\mathcal{E} = (\mathsf{KG}, \mathsf{Enc}, \mathsf{Dec})$ is CPA-secure. After performing $\mathsf{DEncrypt}$ and $\mathsf{Shuffle}$ algorithms, every user obtains a collection of ElGamal encryptions $\{c_1, \ldots, c_{nk}\}$. By the assumption, the adversary cannot learn any further information except that which encryptions have been sent from which users. Running these algorithms, each user should raise the power of a vector of encryptions to his secret s_i. Namely, at the end of this step each user holds $\{E_{s_i}(c_1), E_{s_i}(c_2), \ldots, E_{s_i}(c_{nk})\}$. Next, user u_i should apply a random permutation π_i to the vector so that he obtains a new vector in form of $\left\{E_{s_i}\left(c_{\pi_i(1)}\right), E_{s_i}\left(c_{\pi_i(2)}\right), \ldots, E_{s_i}\left(c_{\pi_i(nk)}\right)\right\}$. Finishing the algorithms, all users get a permuted and doubly encrypted list

$$\left\{E_s\left(c_{\pi(1)}\right), E_s\left(c_{\pi(2)}\right), \ldots, E_s\left(c_{\pi(nk)}\right)\right\},$$

where the permutation $\pi = \pi_n \circ \cdots \circ \pi_1$ and $s = \prod_{i=1}^{n} s_i$. As there exists at least an honest user, even when $n-1$ users collude, s is uniformly distributed and unknown to all users and π is a random permutation. This completes the proof of the claim. □

Theorem 2 *Assuming that the threshold ElGamal encryption is semantically secure and the DL assumption holds, the proposed* κ^+ *protocol is secure in the semi-honest model.*

Proof. We complete the proof of security by Lemma 2 and Lemma 3. □

4.3 Efficiency Analysis

In the following we give a detailed analysis of the running time and the space requirements of the protocol. on ElGamal encryption and the power function with primes $|p| = 1024, |q| = 160$. To measure users' overhead, we count the number of exponentiations using a 1024-bit modulus.

In Table 3 we show a summary of the complexity result for our proposed protocol. The total computational complexity is dominated by DEncrypt and Shuffle algorithms. Putting the computational complexities together shows that the total computation complexity is $\mathcal{O}(n^2k)$ in $\mathcal{O}(n)$ rounds. The proposed protocol has $\mathcal{O}(n^2k \log p)$ bits in total. It is impossible to directly compare our protocol with Applebaum et al.'s protocol, since it runs in the semi-decentralized model, so we just present the computational complexity.

Detailed Comparison. We consider three protocols: Kissner and Song (KS) protocol [20], Burkhart and Dimitropoulos (BD) protocol [3], and Applebaum et al. protocol.

- **KS protocol.** Since KS does not provide a way for finding a threshold value τ that separates the κ-th from the $(\kappa + 1)$-th item, we do not know the complexity in computing τ. Assuming τ is given, their protocol has $\mathcal{O}(n^2k)$ computation complexity in $\mathcal{O}(n)$ rounds.
- **BD protocol.** In turn, we give a comparison with BD protocol. To our knowledge, it is the only fully decentralized κ^+ protocol that does not use Yao's garbled circuit evaluation. Their protocol utilizes two special-purpose sub-protocols–`equality` and `lessthan` (see [8,27]), but in [4] as the authors pointed out, comparing two shared secrets is very expensive and computational intensive. Thus, they use a computationally efficient version of the basic sub-protocols as follows: `equality` requires $\log p$ rounds and `lessthan` requires $(2 \log p + 10)$ rounds. Their protocol need to run two key algorithms as follows:
  - Finding a correct threshold value τ:
    This algorithm has $(\log k(2 \log p + 10) + \log p + 2 \log p + 10)\, nk$ rounds.
  - Resolving collisions:
    This algorithm requires $\frac{n(n-1)}{2} \log p + 2(n-1) \log p + 10(n-1)$ rounds.

  Thus, the total round complexity is $\mathcal{O}(n(n + k \log k) \log p)$ for hash tables of size $\log k$ and `U` of cardinality nk. Their protocol requires $4\left(\frac{n(n-1)}{2} k + k(n-1)\right)$ multiplications in $\mathbb{Z}_p^\times$.
- **Applebaum et al. protocol.** Let us use $\mathcal{O}_p(\cdot)$ to denote the complexity using modulus prime p and $\mathcal{O}_N(\cdot)$ complexity using modulus composite N. Assume all elements are integers less than p and the maximum multiplicity is less than $\log \log p$.
  Their major computation-intensive parts are as follows:
  - Interactive computation between Users and Proxy: First, users should run a protocol for oblivious evaluation of pseudorandom function by communicating with proxies, then encrypt the result with ElGamal encryption. This requires $n(k(2 \log p + 2) + 2k)$ exponentiations over $\mathbb{Z}_p^\times$. Also, users should encrypt the multiplicity of each element with GM encryption, requiring $nk \log \log p$ multiplications over $\mathbb{Z}_N^\times$. Finally each user encrypts his

Table 3. Complexity Analysis

	Comp. Cpx (expo.)	Comm. Cpx (bits)	Rounds Cpx
Setup	n	$n \log p$	1
DEncrypt & Shuffle	$4nk + 2n^2k$	$2(n-1)k \log p + 2n^2k \log p$	n
Aggregate	n^2k	$2n^2k \log p$	1
Reveal	$n\kappa$	$n\kappa \log p$	$n-1$

ElGamal ciphertexts using ElGamal encryption once more, which requires $2nk$ exponentiations over $\mathbb{Z}_p^\times$.

- Aggregation by Database: The most computationally-intensive part is ElGamal and GM decryption. Since database receives two types of ElGamal ciphertexts, it has to perform $2nk$ exponentiations over $\mathbb{Z}_p^\times$. GM decryption requires $2nk \log\log p$ exponentiations over $\mathbb{Z}_N^\times$.

Thus, its complexity is $\mathcal{O}_p(nk \log p) + \mathcal{O}_N(nk \log\log p)$ exponentiations.

5 A κ^+ Protocol with Malicious Adversaries

We begin by defining an ideal functionality for over-threshold aggregation. Then the trusted party computes a set whose elements are in the union of each user's multiset and has a multiplicity greater than or equal to κ, and outputs the set. We state this more formally in the following section.

Definition 8 (Ideal Functionality $\mathcal{F}_{\mathsf{topk}}$) *There are a set of n users, $U = \{u_i\}_{i=1}^n$, a trusted party $\mathcal{T}$, and an ideal adversary $\mathcal{S}$ controlling a set of corrupted users $\tilde{u}_t = \{u_{i_j}\}_{j=1}^t$ for some $0 \le t < n$. Again let $\mathtt{X}_i = \{\alpha_{i,j}\}_{j=1}^{k_i}$ be a multiset of user $u_{i\in[1,n]}$.*

1. *Each user u_i sends $\mathtt{X}_i$ to $\mathcal{T}$.*
2. *$\mathcal{T}$ computes the following functionality, and returns the output Z_l to each $u_{l\in[1,n]}$:*

$$Z_l = \left\{ \alpha_{i,j} \in \bigcup\nolimits_{i\in[1,n]} \mathtt{X}_i \middle| F(\alpha_{i,j}) \ge \kappa \right\}.$$

In the rest of this section we present in details our construction for a protocol realizing the ideal functionality $\mathcal{F}_{\mathsf{topk}}$. To this end, we first describe several issues that we have to address in the construction. We then provide a full description of a κ^+ protocol secure against malicious adversaries and end with the security analysis.

5.1 Constructing a Protocol Secure in the Malicious Model

Before describing the details of the protocols, we first need to outline several issues that should be resolved in transforming the basic protocol in §4.1 into a protocol with malicious adversaries. As above we denote by $U = \{u_i\}_{i=1}^n$ a set of all players and by $\Upsilon \subsetneq U$ a set of corrupted players.

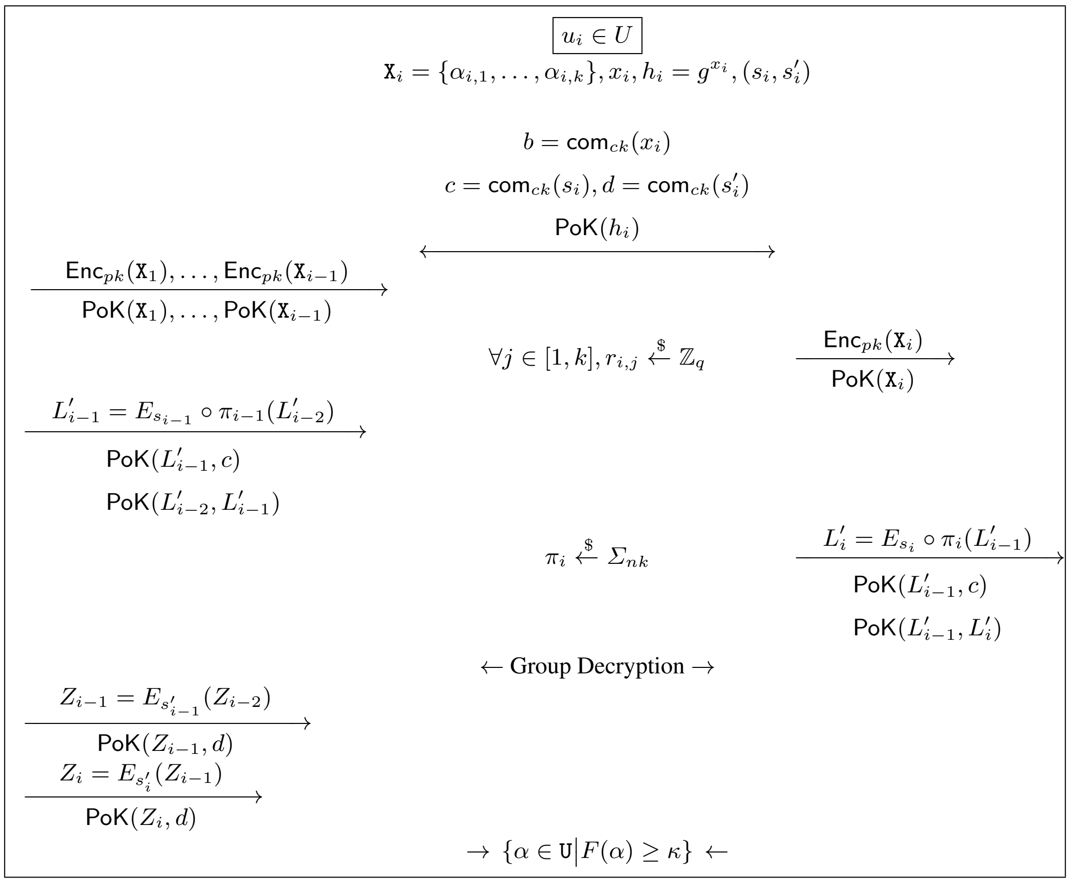
- It is clearly easy for a corrupted player to construct his multiset by copying an honest player's multiset. For example, a user $u \in \Upsilon$ obtains a multiset $\mathtt{X}_i$ of an honest user $u_i \in U \setminus \Upsilon$ through a public channel. Since ElGamal encryption is homomorphic, she can re-randomize the encryptions of u_i so that she can submit the re-randomized encryptions as the encryptions of his input multiset without detection.

  To address these problems, we introduce a zero-knowledge protocol for verifying that the prover knows the corresponding plaintext message m to an ElGamal ciphertext message $c = \mathsf{Enc}_{pk}(m)$.
- Corrupted players may deviate from the protocol instructions by computing their outputs using an incorrect permutation or using a value which is different from a secret share s_i determined in the setup of the protocol.

  These problems can be also resolved by using zero-knowledge protocols mentioned in §3.5. First—when given an ElGamal ciphertext $e = \mathsf{Enc}_{pk}(m)$ a player u_i computes $e' = e^{s_i} = E_{s_i}(e)$, and then need to prove that he raised e exactly to the power of s_i. In addition, given a list of ElGamal ciphertexts $L = (e_1, \ldots, e_\ell)$, he must produce a different list of ElGamal ciphertexts $L' = (e'_1, \ldots, e'_\ell)$ with a proof that there exists a permutation $\pi \in \Sigma_k$ such that for all $j \in [1, \ell]$ we have $\mathsf{Dec}_{sk}\left(e'_{\pi(j)}\right) = \mathsf{Dec}_{sk}(e_j)$.

5.2 Over-threshold Aggregation Protocol with Malicious Adversaries

We are now ready to describe a protocol that securely computes $\mathcal{F}_{\mathsf{topk}}$ with allowing an adversary to maliciously behave, in the standard model. We graphically shows a high-level description without details in Fig. 1. We then give a full description.



$\boxed{u_i \in U}$
$\mathtt{X}_i = \{\alpha_{i,1}, \ldots, \alpha_{i,k}\}, x_i, h_i = g^{x_i}, (s_i, s_i')$
$b = \mathsf{com}_{ck}(x_i)$
$c = \mathsf{com}_{ck}(s_i), d = \mathsf{com}_{ck}(s_i')$
$\mathsf{PoK}(h_i)$
$\mathsf{Enc}_{pk}(\mathtt{X}_1), \ldots, \mathsf{Enc}_{pk}(\mathtt{X}_{i-1})$
$\mathsf{PoK}(\mathtt{X}_1), \ldots, \mathsf{PoK}(\mathtt{X}_{i-1})$
$\forall j \in [1,k], r_{i,j} \xleftarrow{\$} \mathbb{Z}_q$
$\mathsf{Enc}_{pk}(\mathtt{X}_i)$
$\mathsf{PoK}(\mathtt{X}_i)$
$L_{i-1}' = E_{s_{i-1}} \circ \pi_{i-1}(L_{i-2}')$
$\mathsf{PoK}(L_{i-1}', c)$
$\mathsf{PoK}(L_{i-2}', L_{i-1}')$
$\pi_i \xleftarrow{\$} \Sigma_{nk}$
$L_i' = E_{s_i} \circ \pi_i(L_{i-1}')$
$\mathsf{PoK}(L_{i-1}', c)$
$\mathsf{PoK}(L_{i-1}', L_i')$
$\leftarrow$ Group Decryption $\rightarrow$
$Z_{i-1} = E_{s_{i-1}'}(Z_{i-2})$
$\mathsf{PoK}(Z_{i-1}, d)$
$Z_i = E_{s_i'}(Z_{i-1})$
$\mathsf{PoK}(Z_i, d)$
$\rightarrow \{\alpha \in \mathtt{U} | F(\alpha) \geq \kappa\} \leftarrow$

Fig. 1. A High-level Description of $\wp_{\mathsf{topk}}$

The main component is a careful combination of the basic κ^+ protocol explained in §4 and several ZKPs in §3.5. However, introducing these zero-knowledge protocols demands to modify several parts of the basic protocol. While each player computes his secret share, he need to commit to his secret using the com algorithm. Moreover, using a zero-knowledge protocol for correct shuffle requires us to commit to a list of ElGamal ciphertexts raised to the power of each player's secret share s_i.

We continue with a full description of the protocol $\wp_{\mathsf{topk}}$. Especially, in this section we use $\wp_{\mathsf{topk}}$ to denote the real-world protocol corresponding to the ideal functionality $\mathcal{F}_{\mathsf{topk}}$. Further, a step followed by a star symbol ($\star$) means that the step is newly added to the protocol $\wp_{\mathsf{topk}}$ and a step followed by a dagger symbol ($\dagger$) means that the step is modified from the corresponding step in the basic protocol.

Protocol $\wp_{\mathsf{topk}}$

- *Inputs*. The input of each user is a multiset $\mathtt{X}_i = \{\alpha_{i,1}, \ldots, \alpha_{i,k}\}$. (For notational convenience we assume $|\mathtt{X}_i| = k$ for all $i \in [1, n]$.)

– *Selection of global parameters*. Each user has a security parameter λ and a large prime p such that $p = 2q + 1$ for a prime q. The group $\mathbb{G}$ is the cyclic subgroup of $\mathbb{Z}_p^\times$ in which the DDH assumption holds. Set $\mathsf{params} = (\mathbb{G}, p, q)$.

– *Protocol actions*.

Setup(1^λ) Let $g, \mathsf{g}, \mathsf{h}$ be random generators of $\mathbb{G}$.

*1. Each user gets a commitment key $ck = (\mathsf{params}, \mathsf{g}, \mathsf{h})$ by running the key generation algorithm $\mathsf{KG}_{\mathsf{com}}$ of Pederson's commitment scheme.

2. Each user agrees to a threshold ElGamal encryption $\mathcal{E}$ with a public/private key pair (pk, sk), which are computed as follows.
   - selects a value $x_i \xleftarrow{\$} \mathbb{Z}_q$, computes $h_i = g^{x_i}$, and sets $sk = (\mathsf{params}, g, x_i)$ and $pk = \left(\mathsf{params}, g, h = \prod_{i=1}^n h_i = g^{\sum_{i \in [1,n]} x_i}\right)$.

*3. Each user u_i proves knowledge of $\log_g h_i$ using zero-knowledge proofs of knowledge for $\mathcal{R}_{\mathsf{DL}}$, that is, by invoking $\mathsf{PoK}(h_i)$.

4. All users are distributed a share (s_i, s_i') such that $s = \prod_{i=1}^n s_i$, $s' = \prod_{i=1}^n s_i'$, and $s \cdot s' = 1 \pmod q$.

*5. Finally, each user computes two commitments $c = \mathsf{com}_{ck}(s_i), d = \mathsf{com}_{ck}(s_i')$ to his secret shares respectively.

DEncrypt For all $i \in [1, n]$ and $j \in [1, k]$:

†1. Every user u_i encrypts his multiset $\mathtt{X}_i$ as follows:

$$\mathsf{Enc}_{pk}(\mathtt{X}_i) = \{\mathsf{Enc}_{pk}(\alpha_{i,1}), \ldots, \mathsf{Enc}_{pk}(\alpha_{i,k})\}$$

where $\mathsf{Enc}_{pk}(\alpha_{i,j}) = (g^{r_{i,j}}, \alpha_{i,j} \cdot h^{r_{i,j}})$ for some randomizer $r_{i,j} \in \mathbb{Z}_q$, and sends $\mathsf{Enc}_{pk}(\mathtt{X}_i)$ to u_1. Let $e_{i,j} = \mathsf{Enc}_{pk}(\alpha_{i,j}; r_{i,j})$. Each user then proves the knowledge of $\alpha_{i,j}$ by invoking the zero-knowledge protocol $\mathsf{PoK}(e_{i,j})$.

†2. User u_1 computes $\{E_{s_1}(\mathsf{Enc}_{pk}(\mathtt{X}_1)), \ldots, E_{s_1}(\mathsf{Enc}_{pk}(\mathtt{X}_n))\}$, which is denoted by Y_0. Let $\beta_{(1,i)} = E_{s_1}(\mathsf{Enc}_{pk}(\mathtt{X}_i)) = \{E_{s_1}(\mathsf{Enc}_{pk}(\alpha_{i,1})), \ldots, E_{s_1}(\mathsf{Enc}_{pk}(\alpha_{i,k}))\}$. The user u_1 and other users then engage in the zero-knowledge protocol $\mathsf{PoK}\left(\beta_{(1,i)}, c\right)$ for which he proves that $\beta_{(1,i)} = E_{s_1}(\mathsf{Enc}_{pk}(\mathtt{X}_i))$ were correctly computed.

Shuffle & DEncrypt For all $i \in [1, n]$, u_i receives vector Y_{i-1} and computes a permuted, doubly encrypted version Y_i as follows:

1. $u_{i \geq 2}$ computes

$$\begin{aligned} E_{s_i}(Y_{i-1}) &= \{c_1, \ldots, c_{nk}\} \\ &= \left\{E_{s_i}\left(E_{s_{i-1}}\left(\cdots E_{s_1}\left(\alpha_{\pi_{i-1}(1)}\right)\cdots\right)\right), \ldots, \right. \\ &\qquad \left. E_{s_i}\left(E_{s_{i-1}}\left(\cdots E_{s_1}\left(\alpha_{\pi_{i-1}(nk)}\right)\cdots\right)\right)\right\}, \end{aligned}$$

where $\alpha_{\pi_{i-1}(\ell)} = \alpha_{\pi_{i-1} \circ \cdots \circ \pi_1(\ell)}$ for all $\ell \in [1, nk]$.

*2. The user then executes a zero-knowledge protocol $\mathsf{PoK}(c_\ell, c)$ to prove that he correctly computed c_ℓ for all $\ell \in [1, nk]$.

†3. u_i chooses a random permutation $\pi_i \in \Sigma_{nk}$, applies π_i to the list of $c_{\ell \in [1,nk]}$, and computes Y_i as follows.

$$Y_i = \left\{\mathsf{Enc}_{pk}\left(c_{\pi_i(1)}; \gamma_1\right), \ldots, \mathsf{Enc}_{pk}\left(c_{\pi_i(nk)}; \gamma_{nk}\right)\right\},$$

where $\forall \ell \in [1, nk] : \gamma_\ell \xleftarrow{\$} \mathbb{Z}_q$. Since the ElGamal encryption scheme is multiplicatively homomorphic, users can perform these re-randomizations efficiently.

[†]4. u_i sends Y_i to u_{i+1} with a zero-knowledge proof of knowledge for $\mathcal{R}_{\mathsf{CS}}$ by executing $\mathsf{PoK}\left(E_{s_i}(Y_{i-1}), Y_i\right)$; the last user u_n sends Y_n to all users.

Aggregate Let $\mathtt{U} = \bigcup_{i=1}^{n} \mathtt{X}_i$. After receiving $E_s\left(\mathsf{Enc}_{pk}(\mathtt{U})\right)$.

1. Every user participates in a group decryption and obtains

$$E_s(\mathtt{U}) = \left\{E_s\left(\alpha_{\pi(1)}\right), \ldots, E_s\left(\alpha_{\pi(nk)}\right)\right\}$$

where $\pi = \pi_n \circ \cdots \circ \pi_1$.

2. Every user computes $Z = \{E_s(\alpha) \in E_s(\mathtt{U}) \big| F(E_s(\alpha)) \geq \kappa\}$.

Reveal For every $i \in [1, n]$:

[†]1. For every $\tilde{\alpha} \in Z$, user u_i computes $D_{s'_i}(\tilde{\alpha})$, sends it to all other users $u_{i' \in [1,n] \setminus \{i\}}$, and then proves that he correctly computed with his committed secret s'_i by invoking a zero-knowledge protocol $\mathsf{PoK}\left(D_{s'_i}(\tilde{\alpha}), d\right)$.

[†]2. Finally if user u_1 receives from u_n

$$\alpha = D_{s'}(\tilde{\alpha}) = D_{s'}\left(E_s\left(\alpha\right)\right) = \alpha^{ss'},$$

and succeeds in verifying zero-knowledge proofs of knowledge for $\mathcal{R}_{\mathsf{DE}}$, he broadcasts all α's.

In conclusion, each user u_i gets a κ^+ set, $\{\alpha \in \mathtt{U} | F(\alpha) \geq \kappa\}$.

Efficiency Compared to the basic κ^+ protocol, the additional cost for the protocol $\wp_{\mathsf{topk}}$ is only needed for performing the zero-knowledge protocols. Table 4 summarizes the complexities of all zero-knowledge protocols. We evaluated our scheme using ElGamal encryption and Pedersen commitments with primes p, q where $q | p-1, |q| = 160, |p| = 1024$. In particular we considered a ZKP protocol for correct shuffle by Groth [16]. Since all of them are a special honest verifier zero-knowledge agreement of knowledge, we need to transform the used protocol into a standard ZKP protocol, which requires additional computation and communication cost. However, because this transformation does not increase the complexities in the sense of big-$\mathcal{O}$, we ignored this cost in our evaluation. Moreover, we did not consider some optimized variants for other three zero-knowledge protocols. Notice that while n^2 shows up as a factor in the complexity analysis, n is usually small in practice, and the overhead is mostly dominated by the constant factor and the value of k.

Table 4. Complexities of Zero-knowledge Protocols (ZKP)

	ZKP for $\mathcal{R}_{\mathsf{DL}}$	ZKP for $\mathcal{R}_{\mathsf{DE}}$	ZKP for $\mathcal{R}_{\mathsf{PK}}$	ZKP for $\mathcal{R}_{\mathsf{CS}}$
Prover	n	$2n^2k + n\kappa$	n^2k	$0.4nk$
Verifier	$2n$	$4n^2k + 2n\kappa$	$2n^2k$	$0.5nk$
Prover's communications (bits)	$1184n$	$2368n^2k + 1184n\kappa$	$1184n^2k$	$720nk$

Security We proceed to proving that the protocol $\wp_{\mathsf{topk}}$ is secure in the presence of malicious adversaries. The following is our main theorem.

Theorem 3 *Assuming the threshold ElGamal encryption $\mathcal{E}$ is semantically secure, hardness of the DDH and DL problems, and all the specified ZKPs cannot be forged, then the protocol $\wp_{\mathsf{topk}}$ described above securely computes $\mathcal{F}_{\mathsf{topk}}$ in the presence of malicious adversaries. That is, for any* *coalition Υ of corrupted users (at most $t < n$ such corrupted users), there is a simulator $\mathcal{S}$ executing in the ideal model, such that the views of the users in the ideal model are computationally indistinguishable from those of the honest users and Υ in the real model.*

Proof Sketch The full proof of this theorem is shown in Appendix **??**. Instead we provide an intuitive explanation how to achieve our security in the malicious model.

By the security definition, we need to build a polynomial-time simulator $\mathcal{S}$ that outputs a list of fake transcripts indistinguishable from a list of real transcripts created by executing $\wp_{\mathsf{topk}}$. Recall that the ideal world also has an adversary, but by assumption the adversary cannot compromise its security. The key observation is that if there exists an adversary in the ideal world such that it can mimic all activities of an adversary in the real world where our protocol runs, then two adversaries in the different worlds have the equivalent capability from security's point of view. Thus we can say the protocol $\wp_{\mathsf{topk}}$ is secure against corresponding adversaries in the real world.

Now we only have to focus on showing that there exists such an adversary living in the ideal world. In the literature the adversary is called a simulator. A typical way of constructing a simulator is to give a polynomial-time algorithm for a simulator $\mathcal{S}$. This simulator communicates with the malicious players Υ, pretending to be one or more honest players in a way that Υ cannot distinguish that it is not in the real world. The trusted third party takes the input from $\mathcal{S}$ and the honest parties, and gives the result to both $\mathcal{S}$ and the honest parties. $\mathcal{S}$ then communicates with the malicious players Υ, so they also learn the result.

The most challenging part in constructing the simulator is how to extract secrets of the malicious players Υ. Fortunately, knowledge extractor defined in ZKPs enables us to learn all secret of Υ. See the following for the full proof.

Proof. Following the standard simulation proof technique, we will give an PPT algorithm for a simulator $\mathcal{S}$ which is a malicious player in the ideal model. This player interacts with the corrupted players Υ, which pretends to be one or more honest players in a way that Υ cannot detect that the simulator does not live in the real world as they do. All corrupted players are allowed to collude.

First we take a look at the ideal world. The trusted party $\mathcal{T}$ in the ideal world takes the input from $\mathcal{S}$ and the honest players, and outputs the κ^+ set to $\mathcal{S}$ and the honest players. Then the ideal adversary $\mathcal{S}$ communicates with the corrupted players Υ and thus all corrupted players in the real world also will know the κ^+ set.

Now we describe how the simulator $\mathcal{S}$ operates in the real world in a computationally indistinguishable manner. We denote by J a set of indices for the corrupted players, whereas we denote by I a set of indices that are assigned to all honest players.

1. For each simulated honest player $u_{i\in I}$, the simulator $\mathcal{S}$
   (a) chooses a secret key $x_i \in \mathbb{Z}_q$ randomly and computes $h_i = g^{x_i}$.
   (b) chooses a random generator $\mathsf{g} \in \mathbb{G}$ and computes $\mathsf{h} = \mathsf{g}^\gamma$ where $\gamma \overset{\$}{\leftarrow} \mathbb{Z}_q$ which plays a role of a trapdoor later. Then $\mathcal{S}$ computes a pair of commitments (c, d) to (s_i, s'_i) respectively, where $c = \mathsf{com}_{ck}(s_i; \gamma_1)$ and $d = \mathsf{com}_{ck}(s'_i, \gamma_2)$ where γ_1, γ_2 are randomly chosen randomizers.
   (c) constructs a multiset $\mathtt{X}_i = \{\alpha_{i,1}, \ldots, \alpha_{i,k}\}$ where $\alpha_{i,j} \overset{\$}{\leftarrow} \mathbb{G}$ for all $j \in [1, k]$
2. The simulator $\mathcal{S}$ carries out **Setup** in the protocol $\wp_{\mathsf{topk}}$ as follows:
   (a) publishes $ck = (\mathsf{g}, \mathsf{h})$ and h_i along with zero-knowledge proofs of the discrete logarithm to the base g of h_i.
   (b) receives from each corrupted player $u_{\omega\in J}$, a share of public key h_ω along with zero-knowledge proofs for h_ω.

3. For all corrupted players in $\varUpsilon$, the simulator $\mathcal{S}$ extracts the secret key $s_{\omega\in J}$ that each corrupted player u_ω has chosen, from the proofs in $\mathsf{PoK}(h_\omega)$.
4. The simulator then performs **DEncrypt** in the protocol:
   (a) sends the encryption of $\mathtt{X}_i$ on behalf of the honest players $u_{i\in I}$ to all the malicious players $\varUpsilon$, along with proofs of plaintext knowledge.
   (b) receives from each corrupted player $u_{\omega\in J}$ the encryptions of a multiset $\mathtt{X}_\omega$ and proofs of plaintext knowledge for its elements $\alpha_{\omega,j} \in \mathtt{X}_\omega$ for all $j \in [1,k]$.
5. The simulator $\mathcal{S}$ extracts from the proofs of plaintext knowledge, the multiset $\mathtt{X}_{\omega\in J}$ that the corrupted player u_ω has held.
6. The simulator playing a role of an adversary in the ideal world submits all multisets $\mathtt{X}_{\omega\in J}$ to the trusted party $\mathcal{T}$ on behalf of all corrupted players living in the ideal world. At the same time, each honest player also sends his multiset to the trusted party. The trusted party $\mathcal{T}$ then computes a κ^+ set

$$Z_{\kappa^+} = \left\{ \alpha \in \bigcup_{i=1}^{n} \mathtt{X}_i \,\middle|\, F(\alpha) \geq \kappa \right\},$$

   and returns the κ^+ set to both the simulator $\mathcal{S}$ and the honest players.
7. In turn, the simulator $\mathcal{S}$ prepares to return the same κ^+ set to the corrupted players $\varUpsilon$. Our proof technique heavily relies on the trapdoor commitment scheme. We need to change the plaintexts in the encryption of multisets that $\mathcal{S}$ sent to $\varUpsilon$ on behalf of the honest players so that the output of the protocol becomes the same set as Z_{κ^+}. However, since we cannot change the encryptions of $\mathtt{X}_{i\in I}$, we have to modify those plaintexts during performing the following algorithms in the protocol. The modification is possible because the Pedersen commitment scheme is equivocal. A more detailed description follows.

   NOTATION For convenience when describing the simulation in this step, we add several notations. Let $\delta \in \mathbb{N}$ and $\delta \leq nk$. We denote by ζ the cardinality of the set Z_{κ^+} revealed by $\mathcal{S}$ in Step 6. In addition, we assume that $1 \in I$ and that the sets of honest players in the ideal world have no intersection with the sets of random elements chosen by $\mathcal{S}$ for the honest players in the real world.
   (a) If the set Z_{κ^+} obtained in Step 6 can be constructed only using $\mathtt{X}_{\omega\in J}$, the simulator follows the rest of the protocol interacting with the corrupted players $u_{\omega\in J}$ as specified. Otherwise, go to the next step.
   (b) The simulator $\mathcal{S}$ finds how many elements need to be changed and then determines which elements in $\mathtt{X}_{i\in I}$ should be changed into the elements in Z_{κ^+}. Let $\{\tilde{\alpha}_1, \ldots, \tilde{\alpha}_\delta\}$ denote a set of such elements to be changed. Then, we see that

$$Z_{\kappa^+} \subset \left(\bigcup_{\omega\in J} \mathtt{X}_\omega \right) \bigcup \{\tilde{\alpha}_i\}_{i=1}^{\delta}.$$

   (c) The simulator computes $\tilde{s}_{i\in I}, \tilde{s}'_{i\in I}$ such that for some $i \in I, j \in [1,k]$,

$$(\alpha_{i,j})^{\prod_{i\in I}(\tilde{s}_i \cdot \tilde{s}'_i)\cdot\prod_{\omega\in J}(s_\omega \cdot s'_\omega)} \in \{\tilde{\alpha}_1, \ldots, \tilde{\alpha}_\delta\} \text{ and } \prod_{i\in I}\left(\tilde{s}_i \cdot \tilde{s}'_i\right) \cdot \prod_{\omega\in J}\left(s_\omega \cdot s'_\omega\right) = 1 \bmod q,$$

   while storing in a set $\varDelta$ such a pair of indices $(i,j) \in I \times [1,k]$, in a sequential order.
   (d) The simulator $\mathcal{S}$ performs **Shuffle & DEncrypt** of the protocol:
      i. For all $(i,j) \in \varDelta$, computes $E_{\tilde{s}_i}(Y_{i-1})$ with $\tilde{s}_i$ instead of s_i together with zero-knowledge proofs of correct double encryption. Even though $\tilde{s}_i \neq s_i$ with high probability, the simulator can persuade a verifier to accept the proof of correct double encryption. The reason is why $\mathcal{S}$ has trapdoors including $\gamma = \log_{\mathsf{g}} \mathsf{h}$ and a randomizer γ_1 used in the commitment c of Step 1.

ii. chooses a random permutation $\pi_i \in \Sigma_{nk}$ and re-randomizes all doubly encrypted El-Gamal ciphertexts in a randomly permuted order by using π_i with proofs of correct shuffle.
iii. sends to the corrupted players $u_{\omega \in J}$ all the computations with corresponding zero-knowledge proofs.
iv. receives from each corrupted players $u_{\omega \in J}$ randomly permuted double encryptions and proofs of correct double encryption and correct shuffle.

8. The simulator $\mathcal{S}$ extracts from the proofs of correct double encryption and correct shuffle, $s_{\omega \in J}$ and $\pi_{\omega \in J}$ that the corrupted players Υ have chosen. Then $\mathcal{S}$ compares all s_ω's in Step 3 with s_ω's in this step for all $\omega \in J$. If there were different values, terminate the simulation with failure. Otherwise, the simulator keeps all permutations $\pi_{\omega \in J}$. In the later step, the simulator can find which elements it should raise to the power of $\tilde{s}_{i \in I}$ by using all permutations $\pi_{i \in I}, \pi_{\omega \in J}$.
9. The simulator $\mathcal{S}$ engages in carrying out **Aggregate** in the protocol with the corrupted players Υ.
10. The simulator and the corrupted players commonly hold $\{E_{\tilde{s}}(\alpha_1), \ldots, E_{\tilde{s}}(\alpha_\zeta)\}$ where $\tilde{s} = \prod_{i \in I} \tilde{s}_i \cdot \prod_{\omega \in J} s_\omega$. Now the simulator $\mathcal{S}$ computes

$$\left\{D_{\prod_{i \in I} \tilde{s}'_i}\left(E_{\tilde{s}}(\alpha_1)\right), \ldots, D_{\prod_{i \in I} \tilde{s}'_i}\left(E_{\tilde{s}}(\alpha_\zeta)\right)\right\}$$

with proofs of correct double encryption. $\mathcal{S}$ then sends the computations and proofs of correct double encryption to the corrupted players—**Reveal** in the protocol. After all, the corrupted players learn the κ^+ set with simple calculations.

A word of explanation can help to clear understand the simulation done in Step 7. At the step, the simulator decommits the trapdoor commitment $\tilde{s}_{i \in I}$ for the new chosen randomness $\tilde{\gamma}_1$ such that

$$c = \mathsf{g}^{s_i}\mathsf{h}^{\gamma_1} = \mathsf{g}^{\tilde{s}_i}\mathsf{h}^{\tilde{\gamma}_1}.$$

It is clear that the simulator runs in polynomial time. Recall that we compare the simulated execution to a hybrid execution where a trusted party $\mathcal{T}$ is used to compute the ideal functionality $\mathcal{F}_{\mathsf{topk}}$ and the zero-knowledge proofs of knowledge for $\mathcal{R}_{\mathsf{DL}}, \mathcal{R}_{\mathsf{DE}}, \mathcal{R}_{\mathsf{PK}}$, and $\mathcal{R}_{\mathsf{CS}}$. It is straightforward to prove that $\mathcal{A}$'s output in the hybrid and simulated executions described above are computationally indistinguishable. Note that the corrupted players cannot distinguish that they are interacting with $\mathcal{S}$ which is in fact working in the ideal world instead of the honest players (which are in the real world), and the correct answer is learned by all players, in both the real and ideal world models. This completes the proof of Theorem 3. □

6 Conclusion

In this paper we have looked at the problem of finding the κ^+ elements securely, and formally defined what it means for a protocol to be a secure κ^+ protocol. We developed two protocols, with varying operation overhead, analyzed their security, and demonstrated their practicality. That is while developing κ^+ protocols, we analyze its precisely computational and communicational cost that our protocol requires to run properly. Moreover, we provide a full proof showing that our protocol is secure in the presence of semi-honest adversaries.

Since semi-honest protocols commonly have critical restrictions in the security sense–for example, even adversary should follow the instructions specified in the protocol, we transformed our basic protocol (given in Section 4) into a stronger κ^+ protocol which is also secure in the presence of malicious adversaries. In addition to a full description of our protocol with malicious adversaries, we proved the protocol to be secure within the simulation paradigm.

References

1. J. Algesheimer, J. Camenisch, and V. Shoup. Efficient computation modulo a shared secret with application to the generation of shared safe-prime products. In M. Yung, editor, *Advances in Cryptology-Crypto*, LNCS 2442, pages 417–432, 2002. 22
2. B. Applebaum, H. Ringberg, M. Freedman, M. Caesar, and J. Rexford. Collaborative, privacy-preserving sata aggregation at scale. In M. Atallah and N. Hopper, editors, *PETS*, LNCS 6205, pages 56–74, 2010. 2, 3, 11, 24
3. M. Burkhart and X. Dimitropoulos. Fast privacy-preserving top-k queries using secret sharing. In *IEEE ICCCN*, 2010. 2, 3, 11, 13
4. M. Burkhart, M. Strasser, D. Many, and X. Dimitropoulos. SEPIA: Privacy-preserving aggregation of multi-domain network events and statistics. In *USENIX Security*, 2010. 13
5. J. Camenisch. Proof systems for general statements about discrete logarithms. Technical Report TR 260, Dept. of Computer Science, ETH Zurich, 1997. 9
6. J. Camenisch and M. Stadler. Efficient group signature schemes for large groups. In B. Kaliski Jr., editor, *Advances in Cryptology-Crypto*, LNCS 1294, pages 410–424, 1997. 9
7. R. Cramer, I. Damgård, and B. Schoenmakers. Proofs of partial knowledge and simplified design of witness hiding protocols. In Y. Desmedt, editor, *Advances in Cryptology-Crypto*, LNCS 839, pages 174–187, 1994. 23
8. I. Damgård, M. Fitzi, E. Kiltz, J. B. Nielsen, and T. Toft. Unconditionally secure constant-rounds multi-party computation for equality, comparison, bits and exponentiation. In S. Halevi and T. Rabin, editors, *TCC*, LNCS 3876, pages 285–304, 2006. 13
9. T. El Gamal. A public key cryptosystem and a signature scheme based on discrete logarithms. In G. R. Blakley and D. Chaum, editors, *Advances in Cryptology-Crypto*, LNCS 196, pages 10–18, 1984. 3, 7
10. Y.-C. Fan and A. L. P. Chen. Efficient and robust schemes for sensor data aggregation based on linear counting. *IEEE Trans. Parallel Distrib. Syst.*, 21(11):1675–1691, 2010. 3
11. Y.-C. Fan and A. L. P. Chen. Energy efficient schemes for accuracy-guaranteed sensor data aggregation using scalable counting. *IEEE Trans. Knowl. Data Eng.*, 24(8):1463–1477, 2012. 3
12. J. Furukawa and K. Sako. An efficient scheme for proving a shuffle. In J. Kilian, editor, *Advances in Cryptology-Crypto*, LNCS 2139, pages 368–387, 2001. 2
13. O. Goldreich. *The foundations of cryptography: Volume 2–Basic Applications*. Cambridge University Press, 2004. 7
14. S. Goldwasser and S. Micali. Probabilistic encryption. *J. Comput. Syst. Sci.*, 1984. 3, 4
15. M. Groat, W. He, and S. Forrest. KIPDA: k-indistinguishable privacy-preserving data aggregation in wireless sensor networks. In *INFOCOM*, pages 2024–2032, 2011. 3
16. J. Groth. A verifiable secret shuffle of homomorphic encryptions. *J. of Cryptology*, 23:546–579, 2010. 2, 9, 17
17. J. Groth and S. Lu. Verifiable shuffle of large size ciphertexts. In T. Okamoto and X. Wang, editors, *PKC*, LNCS 4450, pages 377–392, 2007. 2
18. W. He, X. Liu, H. Nguyen, K. Nahrstedt, and T. Abdelzaher. PDA: privacy-preserving data aggregation in wireless sensor networks. In *INFOCOM*, pages 2045–2053, 2007. 3
19. M. Kim, A. Mohaisen, J. H. Cheon, and Y. Kim. Private over-threshold aggregation protocols. In T. Kwon, M.-K. Lee, and D. Kwon, editors, *ICISC 2012*, LNCS 7839, pages 472–486, 2012. 1
20. L. Kissner and D. Song. Privacy-preserving set operations. In V. Shoup, editor, *Advances in Cryptology-Crypto*, LNCS 3621, pages 241–257, 2005. 1, 3, 11, 13
21. Q. Li and G. Cao. Efficient and privacy-preserving data aggregation in mobile sensing. In *ICNP*, pages 1–10, 2012. 3
22. Y.-H. Lin, S.-Y. Chang, and H.-M. Sun. CDAMA: concealed data aggregation scheme for multiple applications in wireless sensor networks. *IEEE Trans. Knowl. Data Eng.*, 25(7):1471–1483, 2013. 3
23. A. Mohaisen, D. Hong, and D. Nyang. Privacy in location based services: Primitives toward the solution. In *NCM*, 2008. 1, 2
24. M. Naor and B. Pinkas. Oblivious transfer with adaptive queries. In M. Wiener, editor, *Advances in Cryptology-Crypto*, LNCS 1666, pages 573–590, 1999. 3
25. C. Neff. A verifiable secret shuffle and its application to e-voting. In *ACM Conference on Computer and Communications Security*, pages 116–125, 2001. 2
26. L. Nguyen, R. Safavi-Naini, and K. Kurosawa. Verifiable shuffles: A formal model and a Paillier-based efficient construction with provable security. In M. Jakobsson, M. Yung, and J. Zhou, editors, *ACNS*, LNCS 3089, pages 61–75, 2004. 5
27. T. Nishide and K. Ohta. Multiparty computation for interval, equality, and comparison without bit-decomposition protocol. In T. Okamoto and X. Wang, editors, *PKC*, LNCS 4450, pages 343–360, 2007. 13
28. H. Özgür Tan, I. Korpeoglu, and I. Stojmenovic. Computing localized power-efficient data aggregation trees for sensor networks. *IEEE Trans. Parallel Distrib. Syst.*, 22(3):489–500, 2011. 3
29. T. Pedersen. Non-interactive and information-theoretic secure verifiable secret sharing. In J. Feigenbaum, editor, *Advances in Cryptology-Crypto*, LNCS 576, pages 129–140, 1991. 9

30. C. Rottondi, G. Verticale, and C. Krauß. Distributed privacy-preserving aggregation of metering data in smart grids. *IEEE JSAC*, 31(7):1342–1354, 2013. 3
31. Y. Sang and H. Shen. Efficient and secure protocols for privacy-preserving set operations. *ACM Transactions on Information and System Security (TISSEC)*, 13(1):9:1–9:35, 2009. 1
32. C.-P. Schnorr. Efficient identification and signatures for smart cards. In G. Brassard, editor, *Advances in Cryptology-Crypto*, LNCS 435, pages 239–252, 1989. 9
33. J. Shi, R. Zhang, Y. Liu, and Y. Zhang. PriSense: privacy-preserving data aggregation in people-centric urban sensing systems. In *INFOCOM*, pages 758–766, 2010. 3
34. X. Xu, X.-Y. Li, X. Mao, S. Tang, and S. Wang. A delay-efficient algorithm for data aggregation in multihop wireless sensor networks. *IEEE Trans. Parallel Distrib. Syst.*, 22(1):163–175, 2011. 3
35. A. Yao. Protocols for secure computations. In *FOCS*, pages 160–164, 1982. 3

A An Alternative View of User Privacy

The following description may help to get a clearer view on user privacy. Let $\Pi_{\kappa,\mathcal{E},\mathsf{E}}$ be a κ^+ protocol defined as in Section 3.1 over a double encryption scheme $(\mathcal{E},\mathsf{E})$ and $\mathcal{A}=(\mathcal{A}_1,\mathcal{A}_2)$ be a PPT adversary.

$(pk, sk, s, s') \leftarrow \mathsf{Setup}(\lambda)$;
$(\mathsf{state}, \Upsilon_t, m_1, \ldots, m_{n-t}) \leftarrow \mathcal{A}_1(pk, n, t)$ s.t. Υ_t is a set of corrupted t users;
$\sigma \xleftarrow{\$} \Sigma_n$ and assign $m_{\sigma(i)}$ to each honest i-th user $u_i \notin \Upsilon_t$;
$(\alpha_1, \ldots, \alpha_\kappa) \leftarrow \Pi_{\kappa,\mathcal{E},\mathsf{E}}$, where $\mathcal{A}_1$ interacts with the $n-t$ honest users;
$(i, j) \leftarrow \mathcal{A}_2(pk, m_1, \ldots, m_{n-t}, \mathsf{state})$;

We define the advantage of an adversary $\mathcal{A}$ as $\mathsf{Adv}_{\mathcal{A}}^{\kappa^+}(\Pi_{\kappa,\mathcal{E},\mathsf{E}},\lambda) = \left|\Pr[\sigma(i)=j] - \frac{1}{n-t}\right|$. Then the κ^+ protocol is *user-private* if the advantage $\mathsf{Adv}_{\mathcal{A}}^{\kappa^+}(\Pi_{\kappa,\mathcal{E},\mathsf{E}},\lambda)$ is negligible in security parameter λ. Notice that the security game above is described for illustration and not for real-world use in κ^+.

B Cryptographic Details for the Constructions

B.1 Threshold Schemes

In order to apply our protocol to an environment consisting of multiple players (> 2) we also need to consider a threshold version of the ElGamal encryption scheme. Let x_i be a secret key and h_i be a corresponding public key of a player u_i. The public key $h = \prod_{i=1}^n h_i = g^{\sum_{i=1}^n x_i} = g^x$ is public and known to all players, and encryption is as in the standard ElGamal encryption scheme above. For decryption, a player u_j sends a request for decryption containing $c = (u, v)$ to all other players. On receiving the request, all other players compute a decryption share $d_i = u^{x_i}$ and send it to the player u_j. Upon receiving all decryption shares, the player computes the message as $m = \frac{v}{\prod_{i=1}^n d_i} = \frac{v}{u^{\sum_{i=1}^n x_i}} = v/u^x$.

In addition, we need an efficient scheme which works as follows: When each user holds a shared secret key s_i such that $s = \prod_{i=1}^n s_i$, the scheme allows each user to have a share s'_i satisfying $s' = \prod_{i=1}^n s'_i$ and $s' = s^{-1} \pmod q$ for a public modulus q. In this paper, we use the scheme studied by Algesheimer et al. [1, §5].

B.2 Definitional Details of Zero-knowledge Proofs

Definition 9 (Interactive Proof Systems) *We say that a pair of PPT algorithms (P, V) is an* interactive proof system *for a language $\mathcal{L}$ if there exists a negligible function μ such that the following conditions hold:*

1. Completeness. *For all $x \in \mathcal{L}$,*

$$\Pr\left[(P, V)(x) = 1\right] \geq 1 - \mu(|x|).$$

2. Soundness. *For all $x \notin \mathcal{L}$ and PPT algorithm P^*,*

$$\Pr\left[(P^*, V)(x) = 1\right] \leq \mu(|x|).$$

Definition 10 (Zero-knowledge) *Let (P, V) be an interactive proof system for a language $\mathcal{L}$. We say (P, V) is computationally zero-knowledge if for all PPT algorithms V^*, there exists a PPT algorithm $\mathcal{S}$ such that*

$$\{(P, V^*)(x)\}_{x \in \mathcal{L}} \overset{c}{\approx} \{\mathcal{S}(x)\}_{x \in \mathcal{L}},$$

where the left term means the output of V^* after it interacts with P on common input x and the right term means the output of $\mathcal{S}$ on x.

Let $\mathcal{R}$ be a binary relation and $\varepsilon : \mathbb{N} \to [0, 1]$. We say that a PPT interactive algorithm V is a *knowledge verifier for the relation $\mathcal{R}$ with knowledge error ε* if the following two conditions hold:

– *Non-triviality*. There exists a PPT interactive algorithm P such that for all $(x, y) \in \mathcal{R}$, every interaction of V with P on common input $x \in \mathcal{L}_{\mathcal{R}}$ and auxiliary input y is accepting.
– *Validity with knowledge error ε*. There exists a polynomial $\phi(\cdot)$ and a probabilistic oracle machine M such that for every PPT interactive algorithm P, for all $x \in \mathcal{L}_{\mathcal{R}}$, and for every $y, r \in \{0, 1\}^*$, the machine M satisfies the following condition:
  Denote by $\varphi(x, y, r)$ the probability that the PPT interactive algorithm V accepts on common input x, when interacting with the prover specified by $P_{x,y,r}$. If $\varphi(x, y, r) > \varepsilon(|x|)$, then on input x and with access to oracle $P_{x,y,r}$ the machine M outputs a solution $w \in \mathcal{R}(x)$ with the expected number of steps bounded by

$$\frac{\phi(|x|)}{\varphi(x, y, r) - \varepsilon(|x|)}.$$

Definition 11 (Knowledge extractor) *The oracle machine defined above is called a* knowledge extractor.

Definition 12 (Proof of Knowledge) *Let (P, V) and $\varepsilon(\cdot)$ be defined as above. If $\varepsilon(\cdot) = 0$, then V is simply called a knowledge verifier for the relation $\mathcal{R}$. Further, we say that a pair of PPT interactive algorithms (P, V) such that V is a knowledge verifier for a relation $\mathcal{R}$ and P is an algorithm satisfying the non-triviality condition is a system for* proofs of knowledge *for the relation $\mathcal{R}$.*

Last we need to define special honest verifier zero-knowledge [7] with the ability to simulate the transcript for any set of challenges without access to the witness.

Definition 13 (Special Honest Verifier Zero-knowledge) *We say that a protocol is a special honest verifier zero-knowledge protocol for a binary relation $\mathcal{R}$ if it is a 3-round public-coin protocol satisfying the following conditions:*

– Completeness. *If P and V runs the protocol on input x and private input w to P where $(x, w) \in \mathcal{R}$, then V always accepts.*
– Special soundness. *There exists a PT algorithm $\mathcal{A}$ such that, given any x and any pair of accepting transcripts $(a, e, z), (a, e', z')$ on input x with $e \neq e'$, it outputs w such that $(x, w) \in \mathcal{R}$.*
– Special honest verifier zero-knowledge. *There exists a PPT algorithm $\mathcal{S}$ such that*

$$\{(P(x, w), V(x, e))\}_{x \in \mathcal{L}_{\mathcal{R}}} \overset{c}{\approx} \{\mathcal{S}(x, e)\}_{x \in \mathcal{L}_{\mathcal{R}}},$$

  where $\mathcal{S}(x, e)$ denotes the output of $\mathcal{S}$ on input x and e, and $(P(x, w), V(x, e))$ denotes the transcript of an execution between P and V where P takes as input (x, w) and V takes as input x and its random challenge e.

C Our Semi-Decentralized Construction

The most crucial drawback of the above protocol is its $\mathcal{O}(n)$ round complexity. To avoid this problem, Applebaum et al. introduced two semi-honest users: a *proxy* which shuffles a list of input ciphertexts, and a *database* which aggregates κ^+ elements. Applying the same technique to our protocol, we can also obtain a constant-round κ^+ protocol with the same security with modifications as follows:

- Assume that there are n_1 proxies and n_2 databases described as in [2].
- Each database engages in setting up a threshold ElGamal encryption and publishes a public key. Instead of users, all proxies are distributed secret shares $(s_l, s_l')_{l\in[1,n_1]}$.
- Each user computes a list of ElGamal ciphertexts and sends it to a proxy.
- Each proxy runs $\mathsf{DEncrypt}$ and $\mathsf{Shuffle}$, and returns the result to all databases.
- Databases perform group decryption, and get the list of encrypted κ^+ elements
- Finally, all proxies decrypt the encrypted κ^+ list and return the κ^+ to all users.

Compared to [2], our protocol does not require OT operations, nor an extra encryption scheme. Recall that Applebaum et al.'s protocol requires to use both the ElGamal encryption scheme and the Goldwasser-Micali (GM) encryption scheme: the ElGamal encryption scheme is used to encrypt elements in multisets, but the GM encryption scheme is used to encrypt their multiplicity.