# Private Over-threshold Aggregation Protocols over Distributed Darabases

Myungsun Kim, Abedelaziz Mohaisen, Jung Hee Cheon, and Yongdae Kim

**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. Further, we show that our basic protocol is efficiently transformed into a stronger protocol secure in the presence of malicious adversaries.

**Index Terms**

Network traffic distribution, data aggregation, privacy, 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 on each of these sets (e.g., the count greater than $\kappa$, referred to as $\kappa^+$) without revealing the set of suspects to other sensors with whom she collaborates.

### 1.1 Problem Definition

Let there be $n$ users denoted by $u_i, 1 \leq i \leq n$, and each of them has a private (multi-)set $\mathtt{X}_i$ of cardinality $k$. For simplicity, assume that the cardinality of each multiset is equal to each other. There may exist one or more elements such that $\alpha_{i,j} = \alpha_{i,j'}$ for some $j \neq j'$.

**Private $\kappa^+$ Aggregation problem.** By the multiplicity of an element of a multiset we mean the number of times it appears in the multiset. Let $\zeta, \kappa \in \mathbb{N}$, and assume $\kappa$ 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 $\alpha$ has a multiplicity greater than or equal to $\kappa$, then $\alpha \in Z$, (*ii*) no polynomial-time algorithm can learn any element other than the output of a $\kappa^+$ protocol,

- *M. Kims is with the University of Suwon, Suwon, South Korea. Email: msunkim@suwon.ac.kr*
- *A. Mohaisen is with VeriSign Labs, VA, USA. EmaiL: amohaisen@verisign.com*
- *J. H. Cheon is with Seoul National University, Seoul, South Korea. Email: jhcheon@snu.ac.kr*
- *Y. Kim is with Korea Advanced Institute of Science and Technology, Daejeon, South Korea. Email: yongdaek@ee.kaist.ac.kr*

and (*iii*) no polynomial-time algorithm should know which output of the execution belongs to which user [28].

One straightforward technique to solve the problem is to use a trusted third party (TTP), where each user sends his private set to such TTP which performs the $\kappa^+$ aggregation task and reports the result back to each user. However, finding such TTP is not always possible in many applications. Moreover, compromising the TTP could lead to a complete privacy loss for all participating users.

Another approach is to use secure multiparty computation (SMC), which allows to securely compute a function over private data where users only learn the result of the function and nothing else. However, despite recent advances in their efficiency, SMC methods still require substantial computation and communication costs, making them impractical for real-world applications that mainly operate on large datasets.

A final approach is to use existing private set-operation protocols such as [23], [40], [21], especially multiset union protocols. These protocols securely compute all elements appearing in the union of input multisets at least $\tau$ times. Here all of them demand a priori-knowledge of the threshold value $\tau$.

*Remark 1:* There have been many results [43], [44], [47], [48] in the literature with titles containing the term "top-$\kappa$". We stress that these protocols produce the greatest $\kappa$ elements in the union of the given sets in a private manner, and thus are different in their end results from our protocol. Our protocols discussed in the rest of this paper produce the set of elements greater than a given value as a parameter to those protocols. For example, there is a secure method for finding the $\kappa$-th ranked element in multiple multisets by Aggarwal et al. [1]. Repeatedly applying this protocol we can efficiently find the biggest $\kappa$ elements in a distributed list.

### 1.2 Our Approach—Informal Descriptions

The most non-trivial part of $\kappa^+$ protocols is that we should satisfy two privacy requirements, namely the data privacy and user-privacy, at the same time.

First let us take a closer look at e-voting protocols, which have similar security requirements. In e-voting protocols, each ballot is mixed with a shuffle scheme, which plays a crucial role in removing linkability between voters and ballots, which would hint user privacy. In fact, in e-voting literature this notion is called unlinkability. In order to emphasize the difference with e-voting protocols, we use the term user privacy. Now assuming that each element in multisets is encrypted and shuffled as in e-voting protocols, all encrypted elements can be decrypted, especially in a threshold manner, while satisfying user privacy. However, all non-$\kappa^+$ elements also are revealed, which violates data privacy in our application. Thus we need a way to keep data privacy even when all encrypted elements were decrypted.

For this purpose we introduce an efficient function $E$ that commutes with an underlying public-key encryption. More specifically, let $\mathsf{Enc}$ be a public-key encryption algorithm and $\mathsf{Dec}$ be the corresponding decryption algorithm. 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 $\alpha$, given $\mathsf{Dec}_{sk}(\mathsf{Enc}_{pk}(E_s(\alpha)))$ no algorithm can efficiently find $\alpha$ without $s$. We call this notion *double encryption*.

In conclusion, our main technique is to shuffle doubly encrypted elements by each user. We should notice that all shuffle algorithms used in e-voting protocols rely on the re-randomizable property of underlying homomorphic encryption (e.g. see [14], [30], [31], [19], [18]), but its re-randomization algorithm does not change the plaintexts of input ciphertexts. However, in our protocol a double encryption scheme will change the plaintexts of input ciphertexts, which is one of subtle differences from existing shuffle algorithms.

### 1.3 Summary of Our Results

In this paper, we present a formal definition of private $\kappa^+$ protocol and its security. Our operation setting is *fully decentralized* among $n$ users over non-partitioned data. For the efficiency of our protocol, we refrain from using secure multiparty computation and construct an efficient private $\kappa^+$ protocol which is both data-private and user-private. Our construction strikes a real balance in the consumed resources and achieved security, and satisfies both privacy requirements. In particular, in its efficiency, our construction is comparable to the work in [3], which achieves its efficiency by giving up decentralized communication model, and in its security guarantees it is comparable to the work [4], which is secure but resources exhaustive. Our protocol on the other hand overcomes the limitations and shortcomings of those protocols.

More specifically, our scheme does not require a trusted party to set up public-key cryptosystems. Note that using Paillier cryptosystem [34] in a threshold manner demands some trusted setup. Moreover, our proposed 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 general, real-world applications have $n$ much smaller than $k$, which further justifies the efficiency of our protocol. This is, our protocol is beneficial in such real-world applications where the number of participating users is small but the size of their multisets is large. It remains an important open problem to devise a protocol whose round complexity does not depend on the number of users. Then we could make our protocol have $\mathcal{O}(nk)$ computation and communication complexity.

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

**On the Importance of Formal Proofs.** We would like to note why formal proofs in the cryptographic protocol construction are so important. A recurring tale in the history of cryptographic protocol design is the proposal of a cryptographic protocol followed by the discovery of security flaws. For example, following Chaum's seminal paper [8], Pfitzmann ad Pfitzmann [38] pointed out that Chaumian mixnets are vulnerable to attacks. The mix-net scheme of Park et al. [35] was also shown to be vulnerable to similar attacks [37]. Jakobsson's scheme of [22] was broken by Mitomo and Kurosawa [27], who also suggested a fix.

While a formal proof of security is of course not an ironclad guarantee that no vulnerabilities will ever be found because proofs may have subtle errors and hardness assumptions may be proven to be wrong, they do significantly improve the trust in the security of a cryptographic scheme. This is the reason why we elaborated on formal proofs of our schemes in this work, as it is the practice in the similar literature.

**Organization.** The rest of this paper is organized as follows. In Section 2, we discuss the related work with extensive analysis and comparison to our work. In Section 3, we outline the preliminaries required for understanding the rest of the paper, including double encryption, our formalism of $\kappa^+$ protocol and its security, and cryptographic primitives used in the context of computing the $\kappa^+$. In Section 4, we introduce our construction that comes into two forms with different requirements and guarantees and meets data privacy and user privacy. In Sections 4.3 and 4.4 we analyze the security and complexity of our work, by proving its security and showing its resources consumption requirements. In Section 5, we provide a full description of a $\kappa^+$ protocol which is secure in the malicious model and

analyze its security in the simulation paradigm. Finally, we draw concluding remarks and point future work in Section 6.

## 2 Related Work

There has been plenty of work in the literature to solve the problem of private data aggregation. Such schemes can be classified under three schools of thoughts: fully centralized, fully decentralized, and semi-decentralized [28]. While the centralized schemes assume the existence of a trusted third party (TTP), which makes them of less interest from the cryptographic and practical point of views, the fully decentralized schemes utilize cryptographic primitives and protocols to replace the centralized 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 to the problem try to replace the centralized TTP with cryptographic constructions, which come in various forms leading to several directions of research. One direction is based on SMC, as it is the case in [4]. However, at the core of that protocol's drawbacks is its inefficiency: since it uses Yao's garbled circuits [46], the computational resources required for ordinary data sizes are overwhelmingly high. Furthermore, as the datasets become disjoint, the efficiency of such construction decreases sharply. In [4], Burkhart and Dimitropoulos—in what we consider to be the most relevant work to ours—have devised a construction in which *the round complexity is linear to the number of bits* in the data elements. However, due to using sketches to improve the efficiency of subprotocols, the aggregate results are *probabilistic*. Furthermore, while the work in [4] is efficient in terms of its computational complexity, this efficiency comes at the expense of high round complexity. Kissner and Song [23] devised an over-threshold set union protocol—where *a threshold value $\tau$ should be given in advance*—to find all elements appearing in the union of input multisets at least $\tau$ times. The protocol requires a priori knowledge of the threshold, although operates in a decentralized manner. We compare it to our work in Section 4.4.

Finally, *semi-decentralized* constructions, represented by the work of Applebaum et al. in [3], 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) [29], 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 $p$. For simplicity, we assume that multisets have values less than the prime $p$. Note that Applebaum et al.'s protocol requires both ElGamal encryption [11] and Goldwasser-Micali encryption [16], but we assume that both encryption systems use the same size modulus.

Related Work on Data Aggregation: Data aggregation is one of important technologies in distributed systems since it allows to efficiently utilize resources like bandwidth and power. The reason is why so many researchers in wireless and smart grid networks have focused on data aggregation schemes [20], [12], [42], [17], [33], [45], [13], [24], [25], [39]. Most of the literature in that domain computes on encrypted data for achieving data privacy using homomorphic encryption. However, these schemes are restricted to additive

TABLE 1
Summary and Comparison

| | Comm. Model | Round Cpx | Comp. Cpx | Comm. Cpx |
|---|---|---|---|---|
| Ours | Fully decentralized | $\mathcal{O}(n)$ | $\mathcal{O}(n^2k\log p)$ | $\mathcal{O}(n^2k\log p)$ |
| [4] | Fully decentralized | $\mathcal{O}(n(n+k\log k)\log p)$ | $\mathcal{O}(n^2k)$ | $\mathcal{O}(n^2k\log p)$ |
| [3] | Semi-decentralized | $\mathcal{O}(1)$ | $\mathcal{O}(nk\log^2 p)$ | $\mathcal{O}(nk\log p)$ |

and multiplicative aggregation functions over numeric data except few examples like [17] due to functional limitations of underlying encryption schemes. In particular, none of them present rigorous security analysis in the cryptographic sense because they do not consider malicious behaviors of nodes and a collusion of malicious nodes. In other words, possible attacks are first enumerated and then explanations for their security in ad-hoc manner are provided without cryptographic proofs.

On the other hand, our schemes are a generic schemes that enable to find all elements over the union of users' sets which appear more than a prefixed threshold. Moreover, we prove that an advanced version of our scheme is secure in the presence of malicious adversaries.

## 3 PRELIMINARIES

NOTATION: Let us denote $F(\alpha)$ for the *multiplicity* (also known as frequency) of an element $\alpha$ 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 $\alpha$ 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. In particular, if $X$ is a set, then $x \xleftarrow{\$} X$ is used to denote sampling from the uniform distribution on $X$. We shall write

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

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

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

for every $a \in \{0,1\}^*$ and $\lambda \in \mathbb{N}$.

### 3.1 Definitions

We begin by reviewing the definitions of public-key encryption and its security [16]. Then we proceed to specify a sequence of definitions required for designing and constructing our scheme.

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

- $\mathsf{KG}$, given a security parameter $\lambda$, 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 public key also describes the plaintext and ciphertext message spaces $\mathbf{M}_{pk}, \mathbf{C}_{pk}$ respectively.
- $\mathsf{Enc}$, given the public key $pk$ and a plaintext message $m$, outputs a ciphertext message $c$ 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 $c \leftarrow \mathsf{Enc}_{pk}(m; r)$.
- $\mathsf{Dec}$, given the public key $pk$, secret key $sk$ and a ciphertext message $c$, outputs a plaintext message $m$ such that there exists randomness $r$ for which $c = \mathsf{Enc}_{pk}(m; r)$; otherwise outputs $\perp$. We denote this by $m \leftarrow \mathsf{Dec}_{sk}(c)$, where we usually omit the public parameter $pk$.

We say that a public-key cryptosystem $\mathcal{E}$ is *correct* if $m = \mathsf{Dec}_{sk}(\mathsf{Enc}_{pk}(m))$, for any key-pair $(pk, sk) \leftarrow \mathsf{KG}(\lambda)$ and any $m \in \mathbf{M}_{pk}$.

For a public-key encryption scheme $\mathcal{E} = (\mathsf{KG}, \mathsf{Enc}, \mathsf{Dec})$ and a polynomial-time 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 $\lambda$ 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 $\mathsf{Adv}^{\mathsf{cpa}}_{\mathcal{E},\mathcal{A}}(\lambda)$, formally defined as

$$\left| \Pr_{b,r} \left[ \begin{array}{l} (pk, sk) \leftarrow \mathsf{KG}(\lambda), (\mathsf{state}, m_0, m_1) \leftarrow \mathcal{A}_1(pk), \\ c = \mathsf{Enc}_{pk}(m_b; r) : b \leftarrow \mathcal{A}_2(\mathsf{state}, m_0, m_1, c) \end{array} \right] - \frac{1}{2} \right|,$$

is negligible in $\lambda$ for all sufficiently large $\lambda$.

Now let us define a public-key homomorphic encryption scheme.

*Definition 3 (Homomorphic PKE):* 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.

We observed that merely relying on a public-key encryption scheme does not allow to achieve our final goal. One of main challenges is that the security goal we have to gain is so complicated that some advanced cryptographic tools are required. To this end, we propose a double encryption scheme. Informally, 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 $\mathsf{Enc}(E(a)) = E(\mathsf{Enc}(a))$. We demand that an encryption scheme $\mathsf{E}$ be *deterministic*. The reason is that we need to know a complete distribution of multisets while hiding their elements.

*Definition 4 (Double Encryption):* Let $\mathcal{E} = (\mathsf{KG}, \mathsf{Enc}, \mathsf{Dec})$ be a public-key encryption scheme defined as in Definition 1 with a pair of keys $(pk, sk) \leftarrow \mathsf{KG}(1^\lambda)$ 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 polynomial-time computable functions, $\mathsf{E} = (G, E, D)$, that satisfies the following properties:

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

A double encryption scheme is obviously a necessary ingredient for constructing a secure over-threshold protocol, but not a sufficient one. Thus in addition, we need to consider a

shuffle scheme. In fact 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 [14], [30], [31], [18]. In particular, we require that a shuffle scheme be verifiable in order to prove the correctness of the shuffle. We follow the definition given by Nguyen et al. [31].

*Definition 5 (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 polynomial-time algorithms $(\mathcal{E}, \mathsf{Shuffle})$ defined as:

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

We say that Shuffle is *correct* if for all $i \in [1, n]$ there exists an index $i$ such that $\mathsf{Dec}_{sk}(e_i) = \mathsf{Dec}_{sk}\left(e_{\pi(i)}\right)$.

*Definition 6 (Verifiable Shuffle):* We say that a shuffle $(\mathcal{E}, \mathsf{Shuffle})$ as defined above is verifiable if there exists a proof system $(P, V)$ such that the proof system $(P, V)$, given a public key $pk$, a list of input ciphertexts $(e_1, \ldots, e_n)$ and a list of output ciphertexts $(e'_1, \ldots, e'_n)$, proves that Shuffle is correct.

Now we are ready to define an *over-threshold aggregation protocol and sometimes abbreviate it by $\kappa^+$ protocol*, and so give its algorithmic description. Throughout the paper, we use $\Sigma_n$ to denote the set of all permutations on $[1, n]$. A private $\kappa^+$ protocol consists of five computable (in polynomial time) algorithms, (Setup, DEncrypt, Shuffle, 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 an input the security parameter $\lambda$, and outputs the public and secret parameters for doubly encrypting input ciphertexts, by invoking $(pk, sk) \leftarrow \mathsf{KG}(1^\lambda)$ and $(s, s') \leftarrow G(1^\lambda)$.
- $(E_s(c_1), \ldots, E_s(c_n)) \leftarrow \mathsf{DEncrypt}(pk, s, c_1, \ldots, c_n)$: The algorithm 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 $(E_s(c_1), \ldots, E_s(c_n))$.
- $\left(E_s(c_{\pi(1)}), \ldots, E_s(c_{\pi(n)})\right) \leftarrow \mathsf{Shuffle}(\pi, E_s(c_1), \ldots, E_s(c_n))$: The algorithm 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.
- $(E_s(\alpha_1), \ldots, E_s(\alpha_\zeta)) \leftarrow \mathsf{Aggregate}\left(pk, sk, E_s\left(c_{\pi(1)}\right), \ldots, E_s\left(c_{\pi(n)}\right)\right)$: The algorithm Aggregate takes as input a pair of keys $(pk, sk)$ and a list of permuted, doubly encrypted ciphertexts, and for all $i \in [1, n]$ computes $\mathsf{Dec}_{sk}\left(E_s\left(c_{\pi(i)}\right)\right) = E_s\left(\alpha_{\pi(i)}\right)$. Finally it computes $F\left(E_s\left(\alpha_{\pi(i)}\right)\right), i \in [1, n]$ and outputs only the elements whose multiplicity is greater than or equal to $\kappa$. Here for some $\zeta \in \mathbb{N}$

$$\{E_s(\alpha_1), \ldots, E_s(\alpha_\zeta)\} = \left\{E_s\left(\alpha_{\pi(i)}\right) \middle| F\left(E_s\left(\alpha_{\pi(i)}\right)\right) \geq \kappa\right\}.$$

- $(\alpha_1, \ldots, \alpha_\zeta) \leftarrow \mathsf{Reveal}\left(pk, s', E_s(\alpha_1), \ldots, E_s(\alpha_\zeta)\right)$: The algorithm Reveal outputs the most frequent $\kappa^+$ elements by computing $D_{s'}\left(E_s(\alpha_j)\right)$ for all $j \in [1, \zeta]$.

In the next section, we will define the meaning of *secure* $\kappa^+$ protocol. Then we describe cryptographic building blocks for constructing a secure $\kappa^+$ protocol under proper cryptographic assumptions.

### 3.2 Security Definition

From the theoretical point of view, one important aspect we try to achieve is to formally 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* protocol 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 players send their inputs over secure channel. Then the trusted party performs the functionality on the inputs and sends to each player its output. Roughly speaking, a protocol is secure if any adversary interacting with in the real-world protocol 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 players before executing the protocol. Thus we cannot expect to always succeed in achieving fairness or guaranteed output delivery.

**Execution in the Ideal-world Model.** In the ideal world execution, each player sends his input to a trusted party who computes the output. If the player is honest, he sends directly his input to the trusted party, whereas a corrupted player 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 players to the adversary, and the adversary then decides whether each honest player receives the output or aborts the protocol. Let $f$ be an $n$-party functionality, let $\mathcal{A}$ be a polynomial-time algorithm, and $\Upsilon \subsetneq \{u_1, \ldots, u_n\}$ be a set of corrupted players. Then the ideal execution of $f$ on inputs $(x_1, \ldots, x_n)$, auxiliary input $z$ to $\mathcal{A}$ and security parameter $\lambda$ is defined as the outputs of the honest players and the adversary from the above execution. We denote this by $\texttt{IDEAL}_{f,\mathcal{A}(z),\Upsilon}(x_1, \ldots, x_n; \lambda)$.

**Execution in the Real-world Model.** In the real-world model there is no trusted party and the players exchange rounds of communication with each other. The adversary $\mathcal{A}$ has a full control over the corrupted players 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 polynomial-time strategy. However, an honest player follows the specifications 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 $\Upsilon$ be a set of corrupted players. The real-world execution of $\wp$ on inputs $(x_1, \ldots, x_n)$, auxiliary input $z$ to $\mathcal{A}$, and security parameter $\lambda$ is defined as the outputs of the honest players and the adversary $\mathcal{A}$ from executing $\wp$. We denote this by $\texttt{REAL}_{\wp,\mathcal{A}(z),\Upsilon}(x_1, \ldots, x_n; \lambda)$.

**Security as Simulation of a Real-world Execution in the Ideal-world Model.** Using the definitions of the real-world and ideal-world 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 7 (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-world model, there exists a PPT adversary $\mathcal{S}$ in the ideal-world model, such that for all $\Upsilon \subsetneq \{u_1, \ldots, u_n\}$,

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

We note that even though our final goal is to construct an over-threshold aggregation protocol secure in the presence of malicious adversaries, we begin by designing a basic protocol secure in the semi-honest model where even the adversary must follow the instructions specified in the protocol. We then transform this 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 [15, Chap. 7] for more details and motivating theories.

### 3.2.1 Security of Over-threshold Aggregation Protocols

Implementing this notion of security in the simulation paradigm requires us to firstly describe a target ideal-world functionality. In Section 5, however we more formally give an ideal functionality for over-threshold aggregation. Rather here we explain what we obtain by realizing the ideal functionality in the real world–data privacy and user privacy.

**Data Privacy and User Privacy.** Due to technical reasons, it is more convenient to deal with the security of $\kappa^+$ protocol by separating it into data privacy and user privacy. In fact, this dividing strategy may help to give the reader a clearer understanding of our scheme.

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

While one can easily adopt the notion of data privacy for $\kappa^+$ protocols following the standard privacy definition, he may experience difficulties in immediately capturing the necessity and notion of user privacy.

In the following we explain and motivate for the user privacy. Let $Z = \{\alpha_1, \ldots, \alpha_\zeta\}$ for some $\zeta \in \mathbb{N}$ be an output of a $\kappa^+$ protocol. We say that a $\kappa^+$ 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 $\alpha \in \mathtt{X}_i$.

The following description may help to get a clearer view on user privacy. Let $\Pi_{\kappa,\mathcal{E},\mathsf{E}}$ be a $\kappa^+$ 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. $\Upsilon_t$ 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 $\mathsf{Adv}^{\kappa^+}_{\mathcal{A}}(\Pi_{\kappa,\mathcal{E},\mathsf{E}}, \lambda) = \left|\Pr[\sigma(i) = j] - \frac{1}{n-t}\right|$. Then the $\kappa^+$ protocol is *user-private* if the advantage $\mathsf{Adv}^{\kappa^+}_{\mathcal{A}}(\Pi_{\kappa,\mathcal{E},\mathsf{E}}, \lambda)$ is negligible in security parameter $\lambda$.

The reason why the above security game is described is not for direct use in our security analysis but for better understanding of the notion of privacy and its definition. As mentioned before, these two notions of privacy for a $\kappa^+$ protocol are formalized by considering an ideal world where a trusted party $\mathcal{T}$ receives the inputs of the players and outputs the result of the defined function. We demand that in the real world application of the protocol–that is, one without the $\mathcal{T}$–no user should learn more information than in the ideal world.

## 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 8 (DL Assumption):* We say that the *discrete logarithm problem is intractable* with respect to $\mathbb{G} = \{\mathbb{G}_\lambda\}$ if for every PPT algorithm $\mathcal{A} = \{\mathcal{A}_\lambda\}$ there exists a negligible function $\mu$ in $\lambda$ 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 $\mathbb{G}$, a generator $\langle g \rangle = \mathbb{G}$, 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 9 (DDH Assumption):* We say that the *decisional Diffie-Hellman problem is intractable* with respect to $\mathbb{G} = \{\mathbb{G}_\lambda\}$ if for every PPT algorithm $\mathcal{A} = \{\mathcal{A}_\lambda\}$ there exists a negligible function $\mu$ in $\lambda$ such that

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

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

### 3.3.1 The ElGamal Public Key Encryption Scheme

We extensively use the ElGamal encryption scheme [11] in constructing our $\kappa^+$ protocol, which usually works on a cyclic group $\mathbb{G}$ of prime order $q$. 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 $\mathbb{G}$. We denote this by $\langle g \rangle = \mathbb{G}$. Then the public and secret keys are $\langle \mathbb{G}, q, g, h \rangle$ and $\langle \mathbb{G}, q, g, x \rangle$, respectively, where $x \xleftarrow{\$} \mathbb{Z}_q$ 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 player 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 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. [2, §5].

### 3.3.2 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 $p$ be a large prime of the form $p = 2q+1$, where $q$ is also prime. Let $\mathbb{G}_q$ be a subgroup of $\mathbb{Z}_p^\times$ of order $q$ with a generator $g$. Then a standard CPA-secure ElGamal encryption $\mathcal{E} = (\mathsf{KG}, \mathsf{Enc}, \mathsf{Dec})$ is 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 $\alpha \in \mathbb{G}_q$, $E : \mathbb{G}_q \to \mathbb{G}_q$ is given by $\alpha \mapsto \alpha^s \pmod p$.
- A deterministic function $D_{s'}(\beta)$ computes $\beta^{s'} \pmod p$.

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

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

*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 Hinding Commitment Schemes with a Trapdoor

Loosely speaking, statistically hiding commitment schemes allow us to bind, in the computational sense, between a player and a value chosen by him, without revealing this committed value in the informational theoretical sense.

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 $\lambda$, outputs a public commitment key $ck$ which specifies a message space $\mathbf{M}_{ck}$, a randomizer space $\mathbf{R}_{ck}$ and a commitment space $\mathbf{C}_{ck}$.
- $\mathsf{com}$, given a message $m$, outputs a commitment $c$ to $m$. We denote this by $c \leftarrow \mathsf{com}_{ck}(m)$; and if we want to emphasize the randomizer $r$ use for committing, we denote this by $c \leftarrow \mathsf{com}_{ck}(m; r)$.
- $\mathsf{decom}$, 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 $(m, r) \leftarrow \mathsf{decom}_{ck}(c)$.

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 $\mu$ in $\lambda$ such that

$$\Pr_{r,r'}\left[(m, r, m', r') \leftarrow \mathcal{A}(c) \middle| c = \mathsf{com}_{ck}(m; r) \wedge c = \mathsf{com}_{ck}(m'; r')\right] \leq \mu(\lambda).$$

3.4.0.1 INSTANTIATIONS.: In this paper, we instantiate the commitment scheme with Pedersen's commitment [36], using same group $\mathbb{G}$ used in Section 3.3. More specifically, let $p = 2q + 1$ where $p, q$ are large primes and $\mathsf{g}, \mathsf{h}$ 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 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. In order to make the paper self-contained we reproduce these various results.

*Definition 10 (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 $\mu$ 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 11 (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 $\varepsilon$* 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 $\varepsilon$*. 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 12 (Knowledge extractor):* The oracle machine defined above is called a *knowledge extractor*.

*Definition 13 (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 [9] with the ability to simulate the transcript for any set of challenges without access to the witness.

*Definition 14 (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 polynomial-time 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 $e$.

| Protocol | Relation | References |
|---|---|---|
| $\mathsf{PoK}(h)$ | $\mathcal{R}_{\mathsf{DL}} = \{\langle p, q, \mathbb{G}, g, h\rangle \vert \exists x \in \mathbb{Z}_q \text{ s.t. } h = g^x\}$ | [41] |
| $\mathsf{PoK}(g_2, c)$ | $\mathcal{R}_{\mathsf{DE}} = \{\langle ck, c, g_1, g_2\rangle \vert \exists (x, s) \in (\mathbb{Z}_q)^2 \text{ s.t. } g_2 = g_1^s \wedge c = \mathsf{com}_{ck}(s; r)\}$ | [6], [7] |
| $\mathsf{PoK}(e)$ | $\mathcal{R}_{\mathsf{PK}} = \{\langle pk, e\rangle \vert \exists m \in \mathbb{Z}_q \text{ s.t. } e \in \mathsf{Enc}_{pk}(m)\}$ | § 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 $pk$. $\mathcal{R}_{\mathsf{CS}} = \{\langle pk, L, L'\rangle \vert \exists \pi \in \Sigma_k, \exists r_i \in \mathbb{Z}_q \text{ s.t. } \forall i \in [1, k], e'_{\pi(i)} = e_i \cdot \mathsf{Enc}_{pk}(1; r_i)\}$ | [18] |

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. To this end, we introduce the notation for these zero-knowledge proofs below; with public key cryptosystems working on a DDH group $\mathbb{G}$ such as the ElGamal encryption scheme we can efficiently construct these zero-knowledge proofs using standard constructions [41], [6], [7], [18].

Let $(pk, p, q, g, \mathbb{G})$ be defined as in Section 3.3 and let $ck$ be defined as in Section 3.4. The first protocol $\mathsf{PoK}(h)$ denotes a zero-knowledge proof that given $(p, q, \mathbb{G}, g, h)$, a prover *knows the discrete logarithm* to the base $g$ of $h$. The second protocol $\mathsf{PoK}(g_2, c)$ denotes a zero-knowledge proof 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 $g_1, g_2$ and random elements in $\mathbb{G}$. In fact, this protocol is used for zero-knowledge proofs of *correct double encryption*. The third one enables a prover in a zero-knowledge manner to prove for a given public ElGamal encryption $\mathsf{Enc}_{pk}(m)$ that the prover *knows* the corresponding *plaintext message* $m$. The last protocol $\mathsf{PoK}(L, L')$ for *correct shuffle* is used to prove the relation

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

The relation $\mathcal{R}_{\mathsf{CS}}$ is shorthand notation of the above relation $\mathcal{R}$ in this paper. For more details of zero-knowledge proofs of proving that a player does shuffle correctly, refer to [18] and references mentioned therein. For implementation considerations, we assume that according to its input parameters one can invoke the proper zero-knowledge protocol among those mentioned above as function overriding in the C++ programming language.

## 4 A $\kappa^+$ Protocol for the Semi-Honest Model

In this section, we describe our construction for computing the $\kappa^+$ 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 all $i \in [1, n]$. Each user $u_i$ has its private multiset $\mathtt{X}_i$, and the users wish to jointly compute

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

For simplicity, assume that all elements are in the proper message domain $\mathrm{M}_{pk}$ of an ElGamal encryption scheme, e.g., a finite cyclic subgroup $\mathbb{G}_q$ of $\mathbb{Z}_p^\times$ in which the DDH assumption holds. For a multiset $\mathtt{X} = \{\alpha_1, \ldots, \alpha_k\}$, we denote $\mathtt{X}^s$ as $\{\alpha_1^s, \ldots, \alpha_k^s\}$ for some $s \in \mathbb{Z}_q$.

### 4.1 Description

Let $\lambda$ be a security parameter, $p$ be a $\lambda$-bit prime such that for some prime $q$, $p = 2q + 1$, and $\mathbb{G}_q$ be a finite cyclic subgroup of $\mathbb{Z}_p^\times$ whose order is $q$, and $g$ be a generator of $\mathbb{G}_q$. We notice that in threshold decryption schemes, users generally produce shares of the decrypted element, and during the operation of the schemes if one user sends a uniformly generated share instead of a valid one the decrypted element is uniform as well. Also, if the decrypted element is uniform, the resulting decryption reveals no information to the users. With such notation in mind, we proceed to describe our construction.

- **Setup**$(1^\lambda)$ Each user $u_i$ for $i \in [1, n]$ with a system parameter $\mathsf{params} = (p, q, g, \mathbb{G}_q)$,
  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 $(s_i, s_i')$ such that $s = \prod_{i=1}^n s_i$, $s' = \prod_{i=1}^n s_i'$, and $s \cdot s' = 1 \pmod q$.
- **DEncrypt** As above, let the deterministic power function be $E_{s_i}(\alpha) = \alpha^{s_i} \pmod p$.
  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 $u_1$.
  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 $Y_0$.
- **Shuffle & DEncrypt** For $i \in [1, n]$, $u_i$ receives vector $Y_{i-1}$ and computes a permuted, doubly encrypted version $Y_i$ as follows:
  1. $u_{i\neq 1}$ 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 $\pi_i$ to the list of $c_{\ell\in[1,nk]}$ computed above; denote the result by $Y_i$.
  3. $u_i$ sends $Y_i$ to $u_{i+1}$; the last user $u_n$ sends $Y_n$ to all users.
- **Aggregate** Let $\mathtt{U} = \bigcup_{i=1}^n \mathtt{X}_i$. Suppose that every user has received $E_s(\mathsf{Enc}_{pk}(\mathtt{U}))$.
  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 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, every user obtains a $\kappa^+$ set, $\{\alpha \in \mathtt{U} | F(\alpha) \geq \kappa\}$.

The advantage of the above protocol is multifold. First, compared to Kissner and Song's protocol [23], this protocol provides the functionality of finding a threshold value and computing the "over threshold" at the same computation and communication cost—whereas they incur different and higher costs in [23]. Second, compared to the $\kappa^+$ protocol described

in [4], this protocol has much better computational complexity; see details in Section 4.4. In order to present a fair comparison between this proposed protocol and Applebaum et al.'s protocol [3] we devise our protocol for a semi-decentralized model in the next section. The other purpose of our modification is to reduce the round complexity to a constant.

### 4.2 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 $\kappa^+$ elements. Applying the same technique to our protocol, we can also obtain a constant-round $\kappa^+$ protocol with the same security with modifications as follows:

- Assume that there are $n_1$ proxies and $n_2$ databases described as in [3].
- 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 DEncrypt and Shuffle, and returns the result to all databases.
- Databases perform group decryption, and get the list of encrypted $\kappa^+$ elements
- Finally, all proxies decrypt the encrypted $\kappa^+$ list and return the $\kappa^+$ to all users.

Compared to [3], 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.

### 4.3 Security Analysis

In this section we prove that our protocol is secure in the semi-honest security model. Then we will analyze the efficiency of our protocol in the next section.

*Theorem 1 (Correctness):* In the private $\kappa^+$ 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:* Each player learns a randomly permuted joint multiset $E_s(\mathtt{U}) = \left\{E_s\left(\alpha_{\pi(1)}\right), \ldots, E_s\left(\alpha_{\pi(nk)}\right)\right\}$. We know that $|\mathtt{U}^s| = nk$. Since $\pi$ 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 $\ell^*$ 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 $E_s(\alpha_\ell) = E_s(\alpha_{\ell^*})$ with overwhelming probability. □

*Corollary 1:* In the private $\kappa^+$ protocol in §4.1, every honest user learns the $\kappa^+$ 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 $\alpha \in \bigcup_{i=1}^{n} \mathtt{X}_i$.

*Theorem 2 (Data Privacy):* Assume that the threshold ElGamal encryption $\mathcal{E} = (\mathsf{KG}, \mathsf{Enc}, \mathsf{Dec})$ is secure against CPA. In the private $\kappa^+$ 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-world model with $\mathcal{T}$.

*Proof:* We assume that the ElGamal encryption scheme is CPA-secure, and so each user learns only

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

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 $\gamma_{\ell\in[1,nk]} \in \mathbb{Z}_q$

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

Note that $\pi$ is a composition of random permutations and is unknown to all users, as the maximum coalition size is smaller than $n$. That is, if there exists at least an honest user, then a composition of random permutations $\pi = \pi_n \circ \cdots \circ \pi_1$ is a random permutation because at least a permutation $\pi_{i\in[1,n]}$ is secure. What is more, note that $s$ is uniformly distributed and unknown to all users for the same reason. As $s$ is uniformly distributed for all user inputs and $\pi$ is random, no user or coalition of them can learn more than a set of re-randomized ElGamal encryptions. 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. □

*Theorem 3 (User Privacy):* Assume that the threshold ElGamal encryption $\mathsf{Enc}$ is CPA-secure. The private $\kappa^+$ 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 second 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 the received encryptions with his shared secret $s_i$. Namely, each user holds the modified list of the encryptions,

$$\{E_{s_i}(c_1), E_{s_i}(c_2), \ldots, E_{s_i}(c_{nk})\}.$$

Next the user should apply his private permutation $\pi_i$ to the list to transform it to

$$\left\{E_{s_i}\left(c_{\pi(1)}\right), E_{s_i}\left(c_{\pi(2)}\right), \ldots, E_{s_i}\left(c_{\pi(nk)}\right)\right\}.$$

At the end of running 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 $\pi$ is a random permutation. This completes the proof of the claim. □

*Theorem 4:* Assuming that the threshold ElGamal encryption is CPA-secure and the DL assumption holds, the proposed $\kappa^+$ protocol is secure in the semi-honest model.

*Proof:* We complete the proof of security by Theorem 2 and Theorem 3. □

### 4.4 Efficiency Analysis

In the following we give a detailed analysis of the running time and the space requirements of the $\kappa^+$, expressing each evaluation criterion as a number defined by the security and system parameters. We base our 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.

TABLE 2
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 | $n\kappa$ | $n\kappa \log p$ | $n-1$ |

In Table 2 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.

COMPARISON: For comparison, we consider three protocols: Kissner and Song (KS) protocol [23], Burkhart and Dimitropoulos (BD) protocol [4], and Applebaum et al. protocol.

– **Based on KS protocol** We first compare our work with a KS-based $\kappa^+$ protocol. As mentioned earlier, since it does not provide a way for finding $\tau$, we do not know computational and communication complexity in computing $\tau$. Assuming $\tau$ 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 $\kappa^+$ protocol that does not use Yao's garbled circuit evaluation. Their protocol utilizes two special-purpose sub-protocols–`equality` and `lessthan` (see [10], [32]), but in [5] as the authors pointed out, the comparison of 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 consists of two key ingredients as follows:
  - Finding the correct $\tau$: takes $(\log k(2 \log p + 10) + \log p + 2 \log p + 10)\, nk$ rounds.
  - Resolving collisions: requires $\frac{n(n-1)}{2} \log p + 2(n-1) \log p + 10(n-1)$ rounds.

  Note that BD protocol also should know $\tau$ as in KS protocol. Hence, the total round complexity is $\mathcal{O}(n(n + k \log k) \log p)$ for hash tables of size $\log k$ and `U` of $nk$.

We find their protocol takes $4\left(\frac{n(n-1)}{2}k + k(n-1)\right)$ 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 doubly encrypts his elements using ElGamal encryption. This 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 performs $2nk$ exponentiations over $\mathbb{Z}_p^\times$. GM decryption requires $2nk\log\log p$ exponentiations over $\mathbb{Z}_N^\times$.

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

## 5 A $\kappa^+$ Protocol with Malicious Adversaries

We begin by defining an ideal functionality of over-threshold aggregation protocols in the malicious model. In the ideal-world model, each player's input consists of a multiset as described in § 4.1. 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 a pre-specified threshold value $\kappa$, and outputs the set. We state this more formally in the following section.

*Definition 15 (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 $0 \le t < n$. 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 $\kappa^+$ protocol secure against malicious adversaries and conclude 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 above protocol for the semi-honest adversaries into a protocol for malicious adversaries. Let's 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. Using the homomorphic property of the underlying public ket encryption scheme, he can re-randomize the obtained encryptions of $u_i$ and then 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 $s_i$ fixed 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 needs 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 $\mathsf{Dec}_{sk}\left(e'_{\pi(j)}\right) = \mathsf{Dec}_{sk}\left(e_j\right)$.

### 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. Fig. 1 graphically shows a high-level description without details. We then give a full description of the protocol.

The main component is a careful combination of the basic $\kappa^+$ protocol explained in §4 and various zero-knowledge proofs of knowledge. However, in order to apply these zero-knowledge protocols we need to modify several parts of the basic protocol. While each player computes his secret share, he needs 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[1] of the protocol $\wp_{\mathsf{topk}}$. We remark that in §4 we did not use a specific symbol for the $\kappa^+$ protocol. However in this section we are going to 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 for the protocol from the previous description in the original 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 $\lambda$ and a large prime $p$ such that $p = 2q + 1$ for a prime $q$. The group $\mathbb{G}_q$ is the cyclic subgroup of $\mathbb{Z}_p^\times$ in which the DDH assumption holds. Set $\mathsf{params} = (p, q, \mathbb{G})$.
- *Protocol actions*.
  - **Setup**$(1^\lambda)$ Let $g, \mathsf{g}, \mathsf{h}$ be random generators of $\mathbb{G}$.
    - $^\star$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 $pk = \left(\mathsf{params}, g, h = \prod_{i=1}^n h_i = g^{\sum_{i \in [1,n]} x_i}\right)$.
    - $^\star$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 $\mathsf{PoK}(h_i)$.

1. We use a slight variant of the document conventions for protocols used in [26].

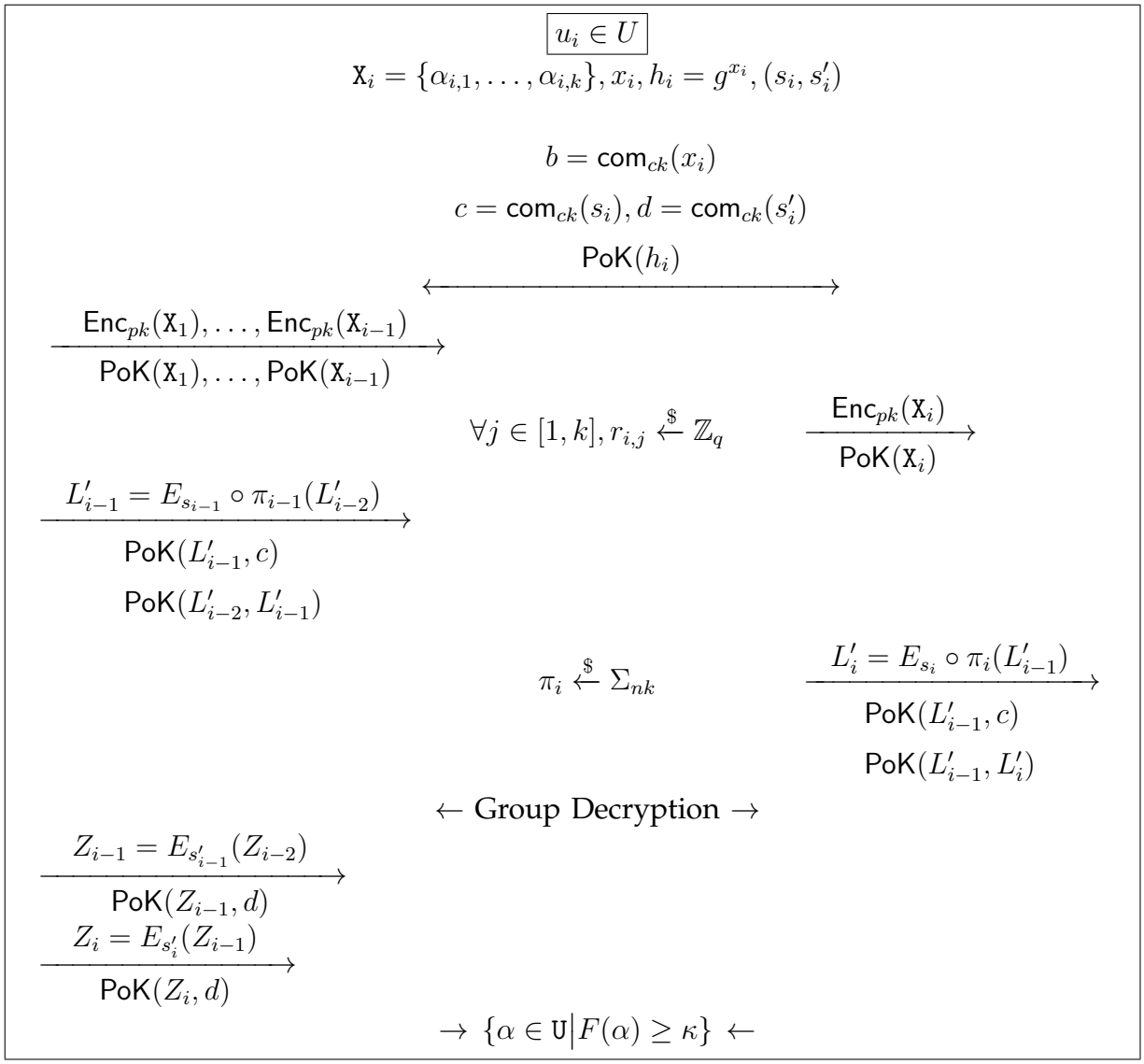


Fig. 1. A High-level Description of $\wp_{\mathsf{topk}}$

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 $\mathsf{PoK}(\beta_{(1,i)}, c)$ 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 $c_\ell$ for all $\ell \in [1,nk]$.

†3. $u_i$ chooses a random permutation $\pi_i \in \Sigma_{nk}$, applies $\pi_i$ 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 $\forall \ell \in [1,nk] : \gamma_\ell \overset{\$}{\leftarrow} \mathbb{Z}_q$. 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 $\mathsf{PoK}\,(E_{s_i}(Y_{i-1}), Y_i)$; the last user $u_n$ sends $Y_n$ to all users.

– **Aggregate** Let $\mathtt{U} = \bigcup_{i=1}^{n} \mathtt{X}_i$. Suppose that every user has received $E_s\left(\mathsf{Enc}_{pk}(\mathtt{U})\right)$.

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

### 5.2.1 Efficiency

Compared to the basic $\kappa^+$ protocol, the additional cost for the protocol $\wp_{\mathsf{topk}}$ is only needed for performing the zero-knowledge protocols. Table 3 summaries the complexities of all zero-knowledge protocols. We evaluated our scheme using ElGamal encryption and Pedersen commitments with primes $p, q$ where $q|p-1, |q| = 160, |p| = 1024$. Especially we assume that our scheme makes use of a zero-knowledge proof protocol for correct shuffle studied by Groth [18]. Since all of them are a special honest verifier zero-knowledge agreement of knowledge, we need to transform the used protocol into a standard zero-knowledge proof of knowledge, 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 3
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) | $1184n$ | $2368n^2k + 1184n\kappa$ | $1184n^2k$ | $720nk$ |

### 5.2.2 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 5:* Assuming the threshold ElGamal encryption $\mathcal{E}$ is semantically secure, hardness of the DDH and DL problems, and all the specified zero-knowledge proofs 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 $\Upsilon$ of corrupted players (at most $t < n$ such corrupted players), there is a simulator $\mathcal{S}$ executing in the ideal-world model, such that the views of the players in the ideal-world model are computationally indistinguishable from those of the honest players and $\Upsilon$ in the real-world model.

*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 $\Upsilon$, which pretends to be one or more honest players in a way that $\Upsilon$ 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 $\kappa^+$ set to $\mathcal{S}$ and the honest players. Then the ideal adversary $\mathcal{S}$ communicates with the corrupted players $\Upsilon$ and thus all corrupted players in the real world also will know the $\kappa^+$ 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 $\gamma \xleftarrow{\$} \mathbb{Z}_q$ 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 $\gamma_1, \gamma_2$ are randomly chosen randomizers.
   c) constructs a multiset $\mathtt{X}_i = \{\alpha_{i,1}, \ldots, \alpha_{i,k}\}$ where $\alpha_{i,j} \xleftarrow{\$} \mathbb{G}$ 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 $h_\omega$ along with zero-knowledge proofs for $h_\omega$.
3) For all corrupted players in $\Upsilon$, the simulator $\mathcal{S}$ extracts the secret key $s_{\omega\in J}$ that each corrupted player $u_\omega$ 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 $\Upsilon$, 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 $u_\omega$ 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 $\kappa^+$ set

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

and returns the $\kappa^+$ set to both the simulator $\mathcal{S}$ and the honest players.
7) In turn, the simulator $\mathcal{S}$ prepares to return the same $\kappa^+$ set to the corrupted players $\Upsilon$. 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 $\Upsilon$ on behalf of the honest players so that the output of the protocol becomes the same set as $Z_{\kappa^+}$. 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 $\zeta$ the cardinality of the set $Z_{\kappa^+}$ 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 $Z_{\kappa^+}$ 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 $Z_{\kappa^+}$. 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 $i \in I, j \in [1,k]$,

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

   while storing in a set $\Delta$ 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 $(i,j) \in \Delta$, 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 $\gamma_1$ used in the commitment $c$ of Step 1.
      ii) chooses a random permutation $\pi_i \in \Sigma_{nk}$ and re-randomizes all doubly encrypted ElGamal ciphertexts in a randomly permuted order by using $\pi_i$ 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 $\Upsilon$ have chosen. Then $\mathcal{S}$ compares all $s_\omega$'s in Step 3 with $s_\omega$'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 $\Upsilon$.
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

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

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 $\kappa^+$ 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 5.[2] □

## 6 Conclusion

In this paper we have looked at the problem of finding the $\kappa^+$ elements securely, and formally defined what it means for a protocol to be a secure $\kappa^+$ protocol. We developed two protocols, with varying operation overhead, analyzed their security, and demonstrated their practicality. That is while developing $\kappa^+$ 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 $\kappa^+$ 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.

2. To complete the proof of Theorem 5 one may require a long, tedious description of a sequence of hybrid games. Rather, such a long description may hinder readers' understanding and thus we believe that this proof is enough to show that our protocol is secure in the presence of malicious adversaries.

## ACKNOWLEDGEMENT

We thank Burt Kaliski for his feedback on an earlier version of this work. The work of A. Mohaisen, Y. Kim, M. Kim, and J. H. Cheon was partly done while they were all at the University of Minnesota.

## REFERENCES

[1] G. Aggarwal, N. Mishra, and B. Pinkas. Secure computation of the $k^{th}$-ranked element. In C. Cachin and J. Camenisch, editors, *Advances in Cryptology-EuroCrypt*, LNCS 3027, pages 40–55, 2004. 2

[2] 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. 10

[3] 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. 3, 4, 5, 15

[4] M. Burkhart and X. Dimitropoulos. Fast privacy-preserving top-$k$ queries using secret sharing. In *IEEE ICCCN*, 2010. 3, 4, 5, 15, 17

[5] M. Burkhart, M. Strasser, D. Many, and X. Dimitropoulos. SEPIA: Privacy-preserving aggregation of multi-domain network events and statistics. In *USENIX Security*, 2010. 17

[6] J. Camenisch. Proof systems for general statements about discrete logarithms. Technical Report TR 260, Dept. of Computer Science, ETH Zurich, 1997. 13

[7] 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. 13

[8] D. Chaum. Untraceable electronic mail, return addresses, and digital pseudonyms. *Communications of the ACM*, 24(2), 1981. 3

[9] 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. 12

[10] 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. 17

[11] 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. 4, 10

[12] 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. 4

[13] 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. 4

[14] 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, 7

[15] O. Goldreich. *The foundations of cryptography: Volume 2–Basic Applications*. Cambridge University Press, 2004. 8

[16] S. Goldwasser and S. Micali. Probabilistic encryption. *J. Comput. Syst. Sci.*, 1984. 4, 5

[17] M. Groat, W. He, and S. Forrest. KIPDA: $k$-indistinguishable privacy-preserving data aggregation in wireless sensor networks. In *INFOCOM*, pages 2024–2032, 2011. 4, 5

[18] J. Groth. A verifiable secret shuffle of homomorphic encryptions. *J. of Cryptology*, 23:546–579, 2010. 2, 7, 13, 21

[19] 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

[20] 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. 4

[21] J. Hong, J. W. Kim, J. Kim, K. Park, and J. H. Cheon. Constant-round privacy preserving multiset union. In *Cryptology ePrint Archive*, 2011/138, 2011. 2

[22] M. Jakobsson. Flash mixing. In B. Coan and J. Welch, editors, *PODC*, pages 83–89, 1999. 3

[23] L. Kissner and D. Song. Privacy-preserving set operations. In V. Shoup, editor, *Advances in Cryptology-Crypto*, LNCS 3621, pages 241–257, 2005. 2, 4, 14, 17

[24] Q. Li and G. Cao. Efficient and privacy-preserving data aggregation in mobile sensing. In *ICNP*, pages 1–10, 2012. 4

[25] 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. 4

[26] A. Menezes, P. van Oorschot, and S. Vanstone. *Handbook of applied cryptography*. CRC Press, 1996. 19

[27] M. Mitomo and K. Kurosawa. Attack for flash MIX. In T. Okamoto, editor, *Advances in Cryptology-AsiaCrypt*, LNCS 1976, pages 192–204, 2000. 3

[28] A. Mohaisen, D. Hong, and D. Nyang. Privacy in location based services: Primitives toward the solution. In *NCM*, 2008. 2, 4

[29] M. Naor and B. Pinkas. Oblivious transfer with adaptive queries. In M. Wiener, editor, *Advances in Cryptology-Crypto*, LNCS 1666, pages 573–590, 1999. 4

[30] 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, 7

[31] 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. 2, 7

[32] 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. 17

[33] 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. 4

[34] P. Paillier. Public-key cryptosystems based on composite degree residuosity classes. In J. Stern, editor, *Advances in Cryptology-EuroCrypt*, LNCS 1592, pages 223–238, 1999. 3

[35] C. Park, K. Itoh, and K. Kurosawa. Efficient anonymous channel and all/nothing election scheme. In T. Helleseth, editor, *Advances in Cryptology-EuroCrypt*, LNCS 765, pages 248–259, 1993. 3

[36] 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. 11

[37] B. Pfitzmann. Breaking efficient anonymous channel. In A. De Santis, editor, *Advances in Cryptology-EuroCrypt*, LNCS 950, pages 332–340, 1994. 3

[38] B. Pfitzmann and A. Pfitzmann. How to break the direct RSA-implementation of mixes. In J.-J. Quisquater and J. Vandewalle, editors, *Advances in Cryptology-EuroCrypt*, LNCS 434, pages 373–381, 1989. 3

[39] C. Rottondi, G. Verticale, and C. Krauß. Distributed privacy-preserving aggregation of metering data in smart grids. *IEEE JSAC*, 31(7):1342–1354, 2013. 4

[40] 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. 2

[41] C.-P. Schnorr. Efficient identification and signatures for smart cards. In G. Brassard, editor, *Advances in Cryptology-Crypto*, LNCS 435, pages 239–252, 1989. 13

[42] 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. 4

[43] J. Vaidya and C. Clifton. Privacy-preserving top-$k$ queries. In *ICDE*, 2005. 2

[44] L. Xiong, S. Chitti, and L. Liu. Top$k$ queries across multiple private databases. In *International Conference on Distributed Computing Systems (ICDCS)*, pages 145–154, 2005. 2

[45] 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. 4

[46] A. Yao. Protocols for secure computations. In *FOCS*, pages 160–164, 1982. 4

[47] R. Zhang, J. Shi, Y. Liu, and Y. Zhang. Verifiable fine-grained top-$k$ queries in tiered sensor networks. In *INFOCOM*, pages 2633–2641, 2010. 2

[48] R. Zhang, Y. Zhang, and C. Zhang. Secure top-$k$ query processing via untrusted location-based service providers. In *INFOCOM*, pages 1170–1178, 2012. 2