Privacy-Preserving User Data Oriented Services for Groups with Dynamic Participation

Dmitry Kononchuk, Zekeriya Erkin,
Jan C.A. van der Lubbe, and Reginald L. Lagendijk

Information Security and Privacy Lab
Department of Intelligent Systems
Delft University of Technology
2628 CD, Delft, The Netherlands
{d.kononchuk,z.erkin,j.c.a.vanderlubbe,r.l.lagendijk}@tudelft.nl

Abstract. In recent years, services that process user-generated data have become increasingly popular due to the spreading of social technologies in online applications. The data being processed by these services are mostly considered sensitive personal information, which raises privacy concerns. Hence, privacy related problems have been addressed by the research community and privacy-preserving solutions based on cryptography, like [1–5], have been proposed. Unfortunately, the existing solutions consider static settings, where the computation is executed only once for a fixed number of users, while in practice applications have a dynamic environment, where users come and leave between the executions. In this work we show that user-data oriented services, which are privacy-preserving in static settings, leak information in dynamic environments. We then present building blocks to be used in the design of privacy-preserving cryptographic protocols for dynamic settings. We also present realizations of our ideas in two different attacker models, namely semi-honest and malicious.

Keywords: Privacy, user-data oriented services, secure multi-party computation, threshold homomorphic encryption.

1 Introduction

In the past decade, online social networks and personalized e-commerce applications have become very popular as they offer customized services to people. To provide customization and personalization the data collected from many users need to be processed by a service. One of the typical example of such user-data oriented services are so-called recommender systems [6], which aim to generate personal recommendations for a particular person from the likings of other similar users by computing similarity scores based on profile information or user preferences. Other examples of user-data oriented service that can be named here are reputation systems [7], collective decision making [8] and social classification [9].

Although user-data oriented services proved themselves to be very useful in online services, as they increase the user satisfaction and business profit at the same time. But the benefits come with a number of privacy risks since such services heavily depend on the data collected from the users, which is considered private in many cases, for example in case of services for medical domain. The user data can be re-purposed, transferred to third parties, sold or lost by the service provider. In either case, the privacy of the users will be damaged and the consequences will be unpredictable.

To overcome the above mentioned privacy problem, different measures including laws and organizational means have been deployed. These measures are also supported by the scientific solutions, which aim to guarantee the privacy of user data, like data perturbation [10] and data anonymization [11]. The recent idea in the field is to employ secure multiparty computations techniques [12], which allow service providers to process user data through interactive protocols without disclosing their content. This approach has been applied to a wide range of applications including recommender systems [1–3], collaborative filtering [4] and data clustering [5].

Unfortunately, the existing solutions only consider a *static* environment, where the number of users involved in the group service does not change in time. Even though these solutions provide provable privacy protection in static settings, their sequential invocation with changing number of users leaks information, damaging the purpose of the privacy-preserving protocol. As almost all of the popular online services have a *dynamic* setting with constantly joining and leaving users, we consider the privacy-preserving protocols that do not cope with the threats of dynamic execution limited to be used in practice. Therefore, in this paper we aim to provide a solution for privacy-preserving group services in a dynamic setting based on cryptographic tools.

The groups with dynamic participation have drawn attention in the cryptographic community, especially to solve the problem of key management [13,14]. There is also prior work in data publishing to protect the privacy of users in case of continuous publishing of data of dynamic user groups [15–17]. Nevertheless, to the best of our knowledge, there has been no previous work addressing dynamic settings for user-data oriented services.

In this paper we focus on dynamic settings for user-data oriented services: we define the notion of privacy in this setting and propose novel tools to provide privacy protection to the users of such services. To achieve this, we propose to select a random sub-group of users and compute the services based on the data from this random group, while keeping the group secret. We introduce three different strategies to select this random sub-group, each suitable for a different group service scenario, and present the protocols implementing each strategy in two attacker models, namely semi-honest and malicious. For each protocol we sketch a proof of its correctness and analyze the protocol with respect to number of rounds, communication and computational complexities. Our protocols use homomorphic encryption and zero-knowledge proofs, and are designed to be executed in a constant number of interactive rounds and to be efficient in terms of computational complexity.

The rest of this paper is structured as follows. In Section 2 we formalize the notion of user-data oriented services privacy-preserving in dynamic environment and introduce a method for providing privacy in dynamic setting we use further. In Section 4 we describe the cryptographic protocols in two different security models, while Section 3 contains the cryptographic primitives used for these protocols. In Section 5 we provide an analysis on the complexity of the proposed protocols and some discussion on possible optimizations, and we conclude this paper in Section 6.

2 Proposed Solution

In this section we define user-data oriented services (from now on group services), the notion of privacy in a dynamic setting and propose a method to provide privacy protection to the users of such services.

2.1 Definitions

In our settings, a *user* represents a party that holds a private input — value selected from predefined field $\mathbb{F}$. All users are numbered and denoted as U_i, their private data is denoted as $d_i \in \mathbb{F}$. We assume that the the upper bound of the number of users in the system is N. All other parties that provide computation resources for a group service are called *service parties*. We denote one of such parties as $\mathcal{A}$.

Definition 1 (Group service). *A group service is the system that consists of:*

- *a set of users $\{U_{i \leq N}\}$, each of them holding corresponding private input $d_i \in \mathbb{F}$;*
- *a predefined number of service parties, including $\mathcal{A}$;*
- *a predefined function $f\colon \bigcup_{k=1}^{N} \mathbb{F}^k \to \mathbb{G}$, which is symmetric, i.e. for any permutation π and values $a_1, \ldots, a_M \in \mathbb{F}, M \leq N\colon f(a_1, \ldots, a_M) = f(\pi(a_1, \ldots, a_M))$.*

A group service run (execution) is an invocation of the predefined multiparty computation protocol (MPC) that involves a subset of M users $\{U_{i_j}\} \subseteq \{U_{i \leq N}\}$, named participating (or involved) users, and all service parties. During an execution of MPC the result $r = f(d_{i_1}, \ldots, d_{i_M}) \in \mathbb{G}$ is computed and outputted to $\mathcal{A}$.

Described group service is called *privacy-preserving in a static setting*, if after its execution party $\mathcal{A}$ learns only the value of r, and other parties do not learn any information about r and d_i. The notion of dynamic settings is formalized as follows:

Definition 2 (Group service (t, M)-dynamic execution). *A group service with users $\{U_{i \leq N}\}$ is executed (t, M)-dynamically when:*

– *there exists a fixed subset of users* $\mathcal{U}^D \subset \{U_{i \le N}\}$ $\left|\mathcal{U}^D\right| = M$*, named dynamic users, remaining users* $\overline{\mathcal{U}^D} = \{U_{i \le N}\} \setminus \mathcal{U}^D$ *are named static ones;*
– *there exists subsets* $\mathcal{U}_1, \ldots, \mathcal{U}_t \subset \mathcal{U}^D$ *defining dynamic user participation, such that* $\bigcup_{j=1}^{t} \mathcal{U}_j = \mathcal{U}^D$ *and* $\forall U_i \in \mathcal{U}^D\, \exists k \in [1, t] \colon U_i \notin \mathcal{U}_k$*;*
– *the group service is executed* t *times, computing results* $r_1, \ldots, r_t$*;*
– *on a* k*-th group service execution only the users from* $\overline{\mathcal{U}^D} \bigcup \mathcal{U}_k$ *are participating.*

Clearly, (t, M)-dynamic execution of the privacy-preserving protocols designed for a static setting reveals $\mathcal{A}$ information on the private data of the dynamic users, as this information can be inferred from $\mathcal{A}$'s observations $(\mathcal{U}_1, r_1), \ldots, (\mathcal{U}_t, r_t)$. We define the scenario, when $\mathcal{A}$ exploit such information leakage, as the *new group attack*. This attack can be illustrated on the following example: assume that $\mathcal{U}_1 \setminus \mathcal{U}_2 = U_3$, then by comparing r_1 and r_2 $\mathcal{A}$ can reveal information on d_3 (or even disclose the value of d_3).

To eliminate such an information leakage, we define a upper-bound on the amount of information on dynamic users' private values that $\mathcal{A}$ can infer from $r_1, \ldots, r_t$ and $\mathcal{U}_1, \ldots, \mathcal{U}_t$ available to it. More formally, for each $U_i \in \mathcal{U}^D$ we give a lower-bound for the value of entropy $H_i = H(d_i \mid (\mathcal{U}_1, r_1), \ldots, (\mathcal{U}_t, r_t))$.

Values of H_i cannot be restricted with absolute values, independent from f, as the quantity of information, which $\mathcal{A}$ can deduce from received computation results, strongly depends on properties of f used in a specific group service. For example, if f computes an average of its arguments, then $\mathcal{A}$ can learn the exact values of private input of dynamic users (for some configuration of $\mathcal{U}_1, \ldots, \mathcal{U}_t$). In case f computes just the number of its arguments, $\mathcal{A}$ cannot infer any information about d_i from received r_k.

To restrict the values of H_i in general cases, but using the properties of a specific f, two strategies can be used:

1. using the entropy of $\mathcal{A}$'s prediction of private users' input in a static case;
2. using the entropy of $\mathcal{A}$'s prediction of private data of non-dynamic users in a dynamic case.

The first approach is too strict for practice, therefore in this work we use the second one and define the privacy in the context of a dynamic group service execution as follows:

Definition 3 (Privacy-preserving (t, M)-dynamic execution). *A* (t, M)*-dynamic execution of a group service with users* $\{U_{i \le N}\}$ *and dynamic users* $\mathcal{U}^D \subset \{U_{i \le N}\}$ *is called privacy-preserving, when after this execution the following conditions hold:*

– *Party* $\mathcal{A}$ *learns only the values of* r_k *and other parties do not learn any information about* r_k *and* d_i*.*
– *Party* $\mathcal{A}$ *can deduce less (or equal) amount of information on the private inputs of dynamic users as on static ones:* $\forall U_i \in \mathcal{U}^D\, \exists U_j \in \overline{\mathcal{U}^D} \colon H_i \le H_j$*.*

Group service is called *privacy-preserving in a* (t, M)*-dynamic setting*, if any (t, M)-dynamic execution of a such group service is privacy-preserving.

2.2 Group Masking Method

Based on the previous definitions, we propose a general method, in which a group service is made privacy-preserving in (t, M)-dynamic setting and secured against the new group attack. The method is based on the idea of blurring the difference between dynamic and static users. We achieve this by adding similar random behaviour to both types of users. More formally the method is as follows:

Method (Group masking). Assume, that the group service with users $\{U_{i\leq N}\}$ is executed (t, M)-dynamically and sets $\mathcal{U}_1, \ldots, \mathcal{U}_t \subset \mathcal{U}^D \subset \{U_{i\leq N}\}$ define the dynamic user participation in group service runs. On the k-th run, the set $\widetilde{\mathcal{U}}_k \subseteq \overline{\mathcal{U}^D} \bigcup \mathcal{U}_k$, named *included users*, is randomly selected in a way that it is kept hidden from $\mathcal{A}$. The result r_k is computed as $r_k = f(\{d_i \mid U_i \in \widetilde{\mathcal{U}}_k\})$.

It is clear that group masking reduces the amount of information that $\mathcal{A}$ can deduce about users' private input during the dynamic execution, as it hides from $\mathcal{A}$ which users are participating in the computations. Next, we check whether and in what conditions this method hides enough information to guarantee the dynamic privacy-protection.

Note that as the function f is symmetric, then for any $x, y, z \in \mathbb{F}$ the following two observations hold:

$$H(x \mid r = f(x, y)) = H(y \mid r = f(x, y)) \ , \tag{1}$$

$$H(x \mid r_1 = f(x, y, z), \, r_2 = f(z)) = H(y \mid r_1 = f(x, y, z), \, r_2 = f(z)) \ . \tag{2}$$

Consequently, if $\widetilde{\mathcal{U}}_1, \ldots, \widetilde{\mathcal{U}}_t$ are generated in a such way that an intersection and a symmetric difference of any number of $\widetilde{\mathcal{U}}_k$ contain at least as many static users as dynamic ones, then the method above guarantees that a group service execution is privacy-preserving.

As far as the specific values of $\widetilde{\mathcal{U}}_k$ are hidden from $\mathcal{A}$, we can relax the condition above and state the following criteria: a group service dynamic execution is privacy-preserving, when an intersection and a symmetric difference of any number of $\widetilde{\mathcal{U}}_k$ contains on average more static users than dynamic ones. Next, we check what conditions should be met to satisfy this criteria.

Let us consider the case, when all $\widetilde{\mathcal{U}}_k$ generated by the group masking method are independent and uniform, in the sense of included users, i.e. on k-th group service execution all involved users (both static and dynamic) have a same probability to be included in $\widetilde{\mathcal{U}}_k$.

Let $\widetilde{\mathcal{U}}_k$ and $\widetilde{\mathcal{U}}_l$ be two sets of included users, both uniform in the sense above. As the sets are uniform, then the probabilities $p = P(U_i \in \widetilde{\mathcal{U}}_k \mid U_i \in \overline{\mathcal{U}^D} \bigcup \mathcal{U}_k)$ and $q = P(U_i \in \widetilde{\mathcal{U}}_l \mid U_i \in \overline{\mathcal{U}^D} \bigcup \mathcal{U}_l)$ are defined. Without loss of generality we assume that $p \geq q$.

Consider the intersection and the symmetric difference of $\widetilde{\mathcal{U}}_k$ and $\widetilde{\mathcal{U}}_l$. It is clear that on average the following statements hold:

$$\left|\left(\widetilde{\mathcal{U}}_k \bigcap \widetilde{\mathcal{U}}_l\right) \bigcap \overline{\mathcal{U}^D}\right| - \left|\left(\widetilde{\mathcal{U}}_k \bigcap \widetilde{\mathcal{U}}_l\right) \bigcap \mathcal{U}^D\right| = pq\left(N - M - \left|\mathcal{U}_k \bigcap \mathcal{U}_l\right|\right) \ , \tag{3}$$

$$\left|\left(\widetilde{\mathcal{U}}_k \triangle \widetilde{\mathcal{U}}_l\right) \bigcap \overline{\mathcal{U}^D}\right| - \left|\left(\widetilde{\mathcal{U}}_k \triangle \widetilde{\mathcal{U}}_l\right) \bigcap \mathcal{U}^D\right| = (p + q - 2pq)\left(N - M - \left|\mathcal{U}_k \bigcap \mathcal{U}_l\right|\right) - p\left|\mathcal{U}_k \setminus \mathcal{U}_l\right| - q\left|\mathcal{U}_l \setminus \mathcal{U}_k\right| \ . \tag{4}$$

Note that as $|\mathcal{U}_k \bigcap \mathcal{U}_l| \leq M$, $|\mathcal{U}_k \bigcap \mathcal{U}_l| + |\mathcal{U}_k \setminus \mathcal{U}_l| + |\mathcal{U}_l \setminus \mathcal{U}_k| \leq M$ and $q \leq p$, then the minimums of Equations (3) and (4) are reached when $|\mathcal{U}_k \bigcap \mathcal{U}_l| = M$. And these minimums are non-negative iff $N \geq 2M$, i.e. when the majority of the users are static.

Consequently, when the majority of the users are static Equations (3) and (4) are non-negative. That is, the intersection and the symmetric difference of any two randomly selected (independent and uniform in the sense of included users) $\widetilde{\mathcal{U}}_k$ and $\widetilde{\mathcal{U}}_l$ contains on average more (or equal) static users than dynamic ones. It is clear that then the same property holds for any number of sets $\widetilde{\mathcal{U}}_k$.

To sum up, if the majority of the users are static, then the group service dynamic execution protected using uniformly and independently selected group masks is privacy-preserving. Next in this work we will target only the settings with majority of the static users.

Note that in real-world group services, for example in aforementioned recommender systems, the utility of computed results depends on the number of users involved in the computation. Hence, applying the group masking method in practice may cause the quality degradation of group service results, due to decreasing the number of users involved in each group service execution. To eliminate such quality fall-of we additionally restrict generated $\widetilde{\mathcal{U}}_k$ by introducing lower-bound of the number of included users:

$$\forall k \in [1, t]\colon \ \left|\widetilde{\mathcal{U}}_k\right| \geq Q\left(\left|\overline{\mathcal{U}^D} \bigcup \mathcal{U}_k\right|\right) \ , \tag{5}$$

where function $Q(x)$ specifies the minimum number of parties (from x available), which should participate in a group service execution to compute a result with the level of utility sufficient compared to what can be achieved by involving all x parties. We assume that Q is publicly known, but we will not specify it, because its exact value is defined by a concrete group service and a concrete application.

2.3 Approaches to Select Included Users

As it was stated in Section 2.2, in the settings, where the majority of the users are static, all subsets $\widetilde{\mathcal{U}}_k$ can be selected independently and uniformly, in the sense of included users. Hence, generating of $\widetilde{\mathcal{U}}_k$ can be done without knowing which users are static and which are dynamic, and which users, except involved in k-th execution, exist in the system. So, for the sake of simplicity, we can assume that a group service is executed only once, and that all existing N users are employed during that execution.

A group service execution using group masking method processes as follows: a random subgroup $\mathcal{U} \subset \{U_{i \leq N}\}$ is selected, and then result r is computed as $r = f(\{d_i \mid U_i \in \mathcal{U}\})$ and outputted privately to $\mathcal{A}$. Generating the subgroup $\mathcal{U}$ is equivalent to generating a vector $e \in \{0,1\}^N$, such that $e_i = 1$ iff $U_i \in \mathcal{U}$. Such a vector e is named a *group mask*.

To apply the group masking method in practice we propose three different approaches for generating a group mask:

Approach 1. Vector e is generated uniformly randomly, such that it contains exactly m ones, where $m \geq Q(N)$ is publicly known.

Approach 2. Each value e_i is generated independently, such that $P(e_i{=}1) = p$, where p is publicly known and satisfies $P(\sum_{i=1}^{N} e_i \geq Q(N)) \approx 1$[1].

Approach 3. Vector e is generated in two steps: (i) uniformly random $m \in_R [Q(N), N]$ is generated; (ii) e is selected uniformly randomly, such that it contains exactly m ones. In this approach, not only the value of e but also the value of m should be hidden from $\mathcal{A}$.

Note that for e generated according to Approach 1, the probability $P(e_i{=}1) = m/N$ and the value $\sum_{i=1}^{N} e_i = m$ are known, while for e generated according to Approach 2, only the probability $P(e_i{=}1) = p$ is known. For a vector generated according to Approach 3 only the lower bound of $P(e_i{=}1)$ is known: $P(e_i{=}1) \geq Q(N)/N$, which is exactly equal to what we can be estimated based on limitation from Equation (5). Hence, we can claim that Approach 3 generates group masks e, such that a priori knowledge about e is minimum. Approach 2 leaks more information on e than Approach 3, and Approach 1 leaks more than Approach 2.

As Approaches 1 and 2 generate group masks with the higher a priori knowledge about the result, this approaches provide more information to potential attackers than Approach 3 and thus they are less secure. Nevertheless, Approaches 1 and 2 have their own advantages, which make them preferable in certain scenarios: Approach 2 can be implemented much more efficiently then Approaches 1 and 3, and thus it introduces a tradeoff between complexity and privacy. Approach 1 has one advantage over other two approaches — it generate group masks with pre-defined number of involved users, which is important for a certain applications, where the values of parameters, say threshold, depend on the amount of data processed or the amount of user participated.

3 Preliminaries

In this section we briefly introduce the cryptographic primitives employed throughout the paper, namely threshold homomorphic encryption and non-interactive zero-knowledge proofs.

[1] For example, due to de Moivre–Laplace theorem and the fact that $\Phi(-4) \approx 0$, p satisfying the following inequation is suitable: $Np + 1 - Q(N) - 4\sqrt{Np(1-p)} \geq 0$.

3.1 Threshold Homomorphic Encryption

For our protocols we rely on homomorphic encryption that allows users to process private data without disclosing them. We use its threshold version to make the processing secure even in the case when all except one users are colluding.

A cryptosystem is called additively homomorphic when there exist an operation $\otimes$ such that applying $\otimes$ to encryptions of two messages, say x and y, produces the cyphertext which decryption yields the sum of these messages:

$$D(E(x) \otimes E(y)) = x + y \ , \tag{6}$$

where E and D represent the encryption and decryption functions. The public-key cryptosystem is called K-out-of-N threshold, when contributions from any K (from in total N) users are required to compute the decryption of a given cyphertext.

In this paper we use the threshold Paillier cryptosystem either with a trusted dealer [18] or without it [19]. Both cryptosystems have the same properties, so we will not distinguish them further.

In the threshold Paillier cryptosystem the public key of the form $pk = (n, g, \theta)$ is used. Here n is the RSA modulus, computed as a product of two random safe primes $2p' + 1$ and $2q' + 1$; g is the generator of the field $\mathbb{Z}^*_{n^2}$ such that $g = (n+1)^a \cdot b^n \bmod n^2$ for random $a, b \in \mathbb{Z}^*_n$; and $\theta = a\beta\eta \bmod n$, where β is randomly chosen from $\mathbb{Z}^*_n$ and $\eta = p'q'$. The corresponding private key is $sk = \beta\eta$. This key is shared between all users using the Shamir's K-out-of-N secret sharing scheme [20]: each U_i receive $sk_i = f(i) \bmod n\eta$, where f is a random polynomial in $\mathbb{Z}_{n\eta}$ of degree $K - 1$, whose first coefficient is sk.

To encrypt a message $x \in \mathbb{Z}_n$ with the public key $pk = (n, g, \theta)$, $E(x, r)$ is computed with a randomly chosen $r \in \mathbb{Z}^*_n$:

$$E(x, r) = g^x r^n \bmod n^2 \ , \tag{7}$$

where $\varDelta = N!$ is publicly known and precomputed.

To perform a threshold K-out-of-N decryption of a cyphertext $c \in \mathbb{Z}^*_{n^2}$ contribution of the users from the set $\mathbb{S}$ of size K is necessary. Each contributing $U_i \in \mathbb{S}$ computes a partial decryption of c:

$$D_i(c) = c^{2\varDelta sk_i} \bmod n^2 \ . \tag{8}$$

Partial decryptions are then passed to a party (parties), which would like to receive a decrypted plaintext, and combined as follows:

$$\begin{aligned} &D(c) = L\left(\textstyle\prod_{U_i \in \mathbb{S}} D_i(c)^{2\mu_i} \bmod n^2\right) \cdot \frac{1}{4\varDelta^2\theta} \bmod n \ , \\ &\text{where } L(x) = \frac{x-1}{n} \ , \quad \mu_i = \varDelta \cdot \prod_{\substack{U_j \in \mathbb{S} \\ j \neq i}} \frac{j}{j-i} \ . \end{aligned} \tag{9}$$

To secure the decryption protocol against malicious private key share holders, the zero-knowledge proofs are used: each user, submitting its partial decryption

$D_i(c)$ should also submit a proof to show that this value was computed correctly, i.e. a proof of correct decryption. These proofs rely on the verification keys VK and $VK_{i=1,\dots,N}$, which should be generated and distributed together with the public key during the initialization stage. The key VK is a randomly chosen quadratic residue in $\mathbb{Z}_{n^2}^*$ and each VK_i is computed as $VK_i = VK^{\Delta sk_i} \bmod n^2$. Afterwards a user can prove that $D_i(c)$ was computed correctly by proving that $\log_{VK^\Delta} VK_i = \log_{c^{4\Delta}} D_i(c)^2$. We refer the reader to the work [18] for further details.

Note that if all users behave in a semi-honest fashion, then the decryption protocol can be simplified to achieve lower computational complexity: each partial decryption is computed as $D_i(c) = c^{4\mu_i \Delta sk_i} \bmod n^2$ and the combining function is computed as $D(c) = L(\prod_{U_i \in \mathbb{S}} D_i(c) \bmod n^2)/(4\Delta^2\theta) \bmod n$.

It is clear that this encryption function E has the following properties:

$$E(x, r_x) \cdot E(y, r_y) = E(x + y, r_x r_y) \quad \bmod n^2 \;, \tag{10}$$

$$E(x, r)^c = E(xc, r^c) \quad \bmod n^2 \;, \tag{11}$$

$$E(x, r_1) \cdot r_2^n = E(x, r_1 r_2) \quad \bmod n^2 \;. \tag{12}$$

Hence E is homomorphic with respect to addition and multiplication by a constant. Moreover, due to the property (12), any party that knows the public key can build a cyphertext equivalent to the given one. This operation is denoted as rerandomization: $\text{Rand}\,(c) = c \cdot r^n \bmod n^2$.

The threshold Paillier encryption is semantically secure under the decisional composite residuosity assumption in the random oracle model. We refer the reader to the works [18, 21] for further information on encryption properties.

In this work all operations over plaintext values will be performed over $\mathbb{Z}_n$, for encrypted values over $\mathbb{Z}_{n^2}^*$, and for randomness over $\mathbb{Z}_n^*$. That is, the field used for each operation can be easily determined by the context, and thus we will omit writing mod n and mod n^2 for simplifying the notation. Also we use the notation $[\![x]\!]$ to denote an encryption $E(x, r)$. omitting the randomness for simplicity.

3.2 Zero-Knowledge Proofs

Zero-knowledge (ZK) proofs are the protocols between two parties: the prover and the verifier, during which the prover tries to convince the verifier that a given statement is true, without leaking any information other than the veracity of that statement. A lot of ZK proofs have been proposed recently, an overview of the current development can be found in [22].

In case when a common random string is available to prover and verifier, the existing ZK proof protocol can be made non-interactive using the method by [23]. One of the advantages of resulting non-interactive zero-knowledge (NIZK) proofs is that they can be used not only in a two-party settings, but in a multiparty settings with one prover and many verifiers.

For our protocols, we employ a few NIZK proofs: (i) proof of correct decryption $\Pi_{CD}([\![x]\!], d, i)$, which shows that $d = D_i([\![x]\!])$, based on ZK proof introduced

[18]; (ii) proof of correct multiplication $\Pi_{CM}(\llbracket x \rrbracket, \llbracket y \rrbracket, \llbracket z \rrbracket)$, which shows that $z = xy$, based on the proof introduced in [24]; (iii) and NIZK proof of knowledge of a plaintext, which is chosen from in a given set, $\Pi_{PK}(\llbracket x \rrbracket, \mathbb{S})$, which shows that $x \in \mathbb{S}$, based on the general technique presented in [22].

4 Protocols

In this section we describe how the Approaches 1–3 introduced in Section 2 can be employed to protect an existing group service against the new group attack and to provide a privacy of user data in the dynamic settings.

The group service we consider computes the sum of the users' private data. More precisely, for users $U_1, \ldots, U_N$, each holding corresponding private value d_i, this group service evaluates

$$r = f(d_1, \ldots, d_N) = \sum_{i=1}^{N} d_i \tag{13}$$

and outputs the result privately to $\mathcal{A}$. The group masking method is applied to this group service by modifying its function f to the following:

$$r = f(d_1, \ldots, d_N) = \sum_{i=1}^{N} d_i e_i \; , \tag{14}$$

where e is a group mask, which is generated according to Approaches 1–3 and is kept hidden from $\mathcal{A}$ and all parties that can collude with $\mathcal{A}$.

We describe the protocols implementing the considered group service and its versions that provide protection against the new group attack in the two security settings:

A. Semi-honest settings, where all parties follow the protocol steps correctly, but can collect the observation during the protocol execution in attempt to obtain any information about private values of other parties.
B. Malicious settings, where all parties can additionally deviate from the protocol.

In both settings parties can collude either to disclose the private data of other (non-colluding with them) parties or to corrupt the computation result. We assume that the number of users participating in each coalition is upper bounded, i.e. that there exists a predefined number $K \leq N$ such, that each coalition involve at most $K - 1$ users and any number of service parties.

For each of these settings we provide four protocols: reference implementation of the considered group service without using the group masking method and three protocols for the group service protected with a group mask generated according to Approaches 1–3. Presented protocols are referenced as **Protocol** $\mathcal{P}_{\mathbf{1}}^{\mathbf{A}}$, where "A" denotes the target security settings and "1" denotes used approach for generating a group mask (0 denotes the protocol without group masking).

In the following protocols the first K users carry the major part of the computations, therefore for simplifying further notation we assume, that in each operation by default only that users are involved, i.e. "all users" should be read as "users $U_{i \leq K}$". Note that at least one of these users is not colluding with others.

We assume that the Paillier K-out-of-N threshold encryption scheme has already been set up: its private key has been shared between all users (each user receives share sk_i) and its public key pk and verification keys VK and $VK_{i=1,\dots,N}$ are known to all parties. In the following protocols all encryptions are done using pk.

The described protocols work over channels of two types: broadcasted to $U_{i \leq N}$ and point-to-point between $\mathcal{A}$ and U_i. Protocols are designed under the assumption that each party has access to the random oracle and to the common random string. The protocols widely use the well-known subprotocols for threshold Paillier cryptosystem, like secure multiplication [25] and unbound fan-in multiplication subprotocols [25, 26].

4.1 Protocols for Semi-honest Setting

In this section we describe the protocols, which are secure in the semi-honest settings, where parties can form a coalitions involving at most $K - 1$ users and any number of service parties. Following protocols preserve the users' privacy by hiding their data from all other users and party $\mathcal{A}$, and $\mathcal{A}$'s privacy by hiding the computation result from all users. We protect the protocols from the new group attack by using group masking, where a group mask should be kept hidden from all parties (as all parties can collude with $\mathcal{A}$).

Reference Protocol. First, we describe the protocol for reference group service implementation, which just outputs the sum of all users data to $\mathcal{A}$ without using group masking. The protocol is described in Protocol $\mathcal{P}_0^A$.

Input: Each $U_{i \leq N}$ holds his private value d_i.
Output: Party $\mathcal{A}$ receives $r = \sum_{i=1}^{N} d_i$.

1. Each $U_{i \leq N}$ broadcasts $[\![d_i]\!]$.
2. Each user computes $[\![r]\!] = \left[\!\!\left[\sum_{i=1}^{N} d_i\right]\!\!\right] = \prod_{i=1}^{N} [\![d_i]\!]$.
3. All users jointly run decryption of $[\![r]\!]$ and open r to $\mathcal{A}$.

Protocol $\mathcal{P}_0^A$. *GS without group masking, semi-honest setting.*

Security and privacy properties of Protocol $\mathcal{P}_0^A$ can be verified as follows. On Steps 1–2 users receive and process only encrypted data. No information can be extracted from it as the K-out-of-N threshold Paillier cryptosystem is known to be semantically secure against $K-1$ colluding private key holders. On Step 3 all users do partial decryption of $[\![r]\!]$ and $\mathcal{A}$ (and all colluding users) gets access to both computation result r and its partial decryptions $D_i([\![r]\!])$. Values $D_i([\![r]\!])$ do not leak any information about sk_i as it was shown in [18], and the value r is a private output of $\mathcal{A}$ and thus is allowed to be learned by $\mathcal{A}$ and colluding users. Hence, no information about private values is leaked during one protocol execution. However, as the protocol does not use the group masking, it fails to protect the privacy in the dynamic settings.

Protocol with Group Masking Using Approach 1. Next, we present the protocol, which can cope with the new group attack. This protocol uses the proposed group masking method, where the mask is generated according to Approach 1.

Approach 1 requires the group mask e to have the following property: $e \in_R \{0,1\}^N$ and exactly m its components are equal to 1. To generate such vector we use the multiple-try method, which process as follows: (i) users generate t vectors $\beta_j \in_R \{0,1\}^N$ in parallel such, that $\forall l \in [1,N]$: $P(\beta_{\cdot,l}{=}1) = m/N$; (ii) users select as e the first β_j, such that its elements sum $b_j = \sum_{l=1}^{N} \beta_{j,l}$ is equal to m.

Multiple-try method is applicable for generating a vector e as the rate of suitable candidates β_j, i.e. vectors satisfying $\sum_{l=1}^{N} \beta_{j,l} = m$, is fixed:

$$S = P\left(\sum_{l=1}^{N} \beta_{j,l}{=}m\right) = \binom{N}{m} \frac{m^m (N-m)^{N-m}}{N^N} \approx \sqrt{\frac{N}{2\pi m (N-m)}} \,. \qquad (15)$$

Consequently, by executing the sufficient numbers of tries t, we can guarantee that the method will fail, i.e. will not generate e, only with negligible probability $2^{-\kappa}$, where κ denotes the statistical security parameter (usually is chosen around 80).

In practice, we can note that $S \geq 1/\sqrt{2\pi m}$, and though we can use the following estimation of the value t: $t = \lceil \kappa \sqrt{2\pi m} \ln 2 \rceil \approx \lceil 1.74 \kappa \sqrt{m} \rceil$.

To perform described multiple-try approach in privacy-preserving manner, we should generate each vector β_j jointly random, i.e. in a such way that K users contribute to it and any subgroup of users together cannot infer, which of candidates for β_j are more likely.

Jointly-random generation of vector $\beta \in_R \{0,1\}^N$ is performed in the following way: (i) each U_i independently generate vector $\alpha_i \in_R \{0,1\}^N$; (ii) generated vectors are composed into β using exclusive OR (XOR) as $\beta = \alpha_1 \oplus \ldots \oplus \alpha_K$, where XOR combination is computed bitwise by employing the *unbounded fan-in XOR subprotocol* [25]. Obviously, if XOR subprotocol is secure, then even $K-1$ colluding users can not extract any information about β.

The following formula holds for β_l as for a XOR-combination of K equally distributed random values $\alpha_{i,l}$ (see [27] for construction):

$$P(\beta_l{=}1) = \frac{1}{2} - \frac{1}{2}(1 - 2P(\alpha_{\cdot,l}{=}1))^K \,. \qquad (16)$$

Consequently, to satisfy the property $\forall l \in [1,N]\colon P(\beta_l{=}1) = m/N$, each α_i should be generated following the next element distribution:

$$q = P(\alpha_{i,\cdot}{=}1) = \frac{1}{2} - \frac{1}{2}\sqrt[K]{1 - 2m/N}\;. \tag{17}$$

Note that the formula above is inapplicable in cases when $m > N/2$ and K is even. For that cases we suggest to increment the value of K, i.e. to involve one more user into the procedure of generation of β.

Protocol for Approach 2, which is based on the described tools, primitives from Section 3 and aforementioned subprotocols, is given in Protocol $\mathcal{P}_1^A$. Security and privacy of the protocol rely on the security properties of the underlying cryptographic primitives. Note that opening of values b_j on Step 3 does not leak any information about e, as $m = \sum_{i=1}^N e_i$ is a priori knowledge.

Input: Each $U_{i\le N}$ holds his private value d_i.
Output: Party $\mathcal{A}$ receives $r = \sum_{i=1}^N d_i e_i$.

1. Each U_i generates t random $\alpha_{i,j} \in_R \{0,1\}^N$ such, that: $\forall l \in [1,N], \forall j \in [1,t]\colon P(\alpha_{i,j,l}{=}1) = q$.
2. Users jointly run the unbounded fan-in XOR subprotocol tN times in parallel, computing for each $j \in [1,t]$, $l \in [1,N]$ value $[\![\beta_{j,l}]\!] = [\![\oplus_{i=1}^K \alpha_{i,j,l}]\!]$.
3. Users locally compute $[\![b_j]\!] = \left[\!\!\left[\sum_{l=1}^N \beta_{j,l}\right]\!\!\right] = \prod_{l=1}^N [\![\beta_{j,l}]\!]$, and jointly run t decryptions in parallel to open values b_j to U_1.
4. U_1 selects minimum j, such that $b_j = m$, and broadcasts $[\![e]\!] = [\![\beta_j]\!]$. This step fails with probability $2^{-\kappa}$.
5. Each $U_{i\le N}$ computes $[\![d_i e_i]\!] = [\![e_i]\!]^{d_i}$ and broadcasts the result.
6. Users locally compute $[\![r]\!] = \left[\!\!\left[\sum_{i=1}^N d_i e_i\right]\!\!\right] = \prod_{i=1}^N [\![d_i e_i]\!]$, and jointly run decryption to open r to $\mathcal{A}$.

Protocol $\mathcal{P}_1^A$. *GS with 1-st group masking, semi-honest setting.*

Protocol with Group Masking Using Approach 2. Protocol $\mathcal{P}_1^A$ can be simplified to achieve relaxed requirements of Approach 2, where only the probability of user participation is fixed, but not the total amount of participating users.

Resulting protocol is presented in Protocol $\mathcal{P}_2^A$. This protocol utilizes less computational resources and discloses less information about generated e than Protocol $\mathcal{P}_1^A$, but is not applicable for several kinds of group services (see Section 2 for examples).

The security of Protocol $\mathcal{P}_2^A$ can be verified in a same way as for the previous protocol. The correctness of generated e, i.e. the fact that it follows the distribution stated in Approach 2 $P(e_i{=}1) = p$, can be easily verified using Equation (16).

Input: Each $U_{i \leq N}$ holds his private value d_i.
Output: Party $\mathcal{A}$ receives $r = \sum_{i=1}^{N} d_i e_i$.

1. Each U_i generates $\alpha_i \in_R \{0,1\}^N$ such, that:
$$\forall j \in [1, N] : P(\alpha_{i,j}{=}1) = \frac{1}{2} - \frac{1}{2}\sqrt[K]{1-2p} .$$
2. Users jointly run the unbounded fan-in XOR subprotocol N times in parallel, computing for each $j \in [1, N]$ value $[\![e_j]\!] = [\![\oplus_{i=1}^{K} \alpha_{i,j}]\!]$.
3. Each $U_{i \leq N}$ computes $[\![d_i e_i]\!] = [\![e_i]\!]^{d_i}$ and broadcasts the result.
4. Users locally compute $[\![r]\!] = \left[\!\!\left[\sum_{i=1}^{N} d_i e_i\right]\!\!\right] = \prod_{i=1}^{N} [\![d_i e_i]\!]$, and jointly run decryption to open r to $\mathcal{A}$.

Protocol $\mathcal{P}_2^A$. *GS with 2-nd group masking, semi-honest setting.*

Protocol with Group Masking Using Approach 3. Now we present the protocol that uses Approach 3 for generating a group mask. Approach 3, compared to Approaches 1 and 2, generates masks with minimum constraints, and hence discloses less information to potential attackers.

Approach 3 requires the group mask e to have the following property: $e \in_R \{0,1\}^N$ and $\sum_{i=1}^{N} e_i$ is uniformly random in $[Q(N), N]$. To generate such e, three steps are executed: (i) uniformly random $r \in_R [0, N - Q(N)]$ is jointly generated (next $N - Q(N)$ is denoted as σ for notation simplicity); (ii) r is converted to $v \in \{0,1\}^\sigma$, which contains exactly r ones; (iii) v is supplemented by $Q(N)$ ones and permuted to produce e.

To implement the described protocol steps we need to present two additional protocols: *secure unary conversion* and *jointly random shuffling.*

Secure unary conversion. This protocol transforms encrypted integer $[\![r]\!]$, which is from the interval $[0, \sigma]$, to encrypted vector $[\![v]\!] : v \in \{0,1\}^\sigma$ of the following form:

$$v = (\underbrace{1, \ldots, 1}_{r}, \overbrace{0, \ldots, 0}^{\sigma - r}) .$$

We implement this protocol using Lagrange polynomial interpolation. Indeed, each vector element v_i can be computed using the function $v_i(x) = (x \overset{?}{\leq} i)$ as $v_i = v_i(r)$. Admitted region of $v_i(x)$ is $\mathbb{Z}_{\sigma+1}$, and hence, $v_i(x)$ can be evaluated in all possible $\sigma + 1$ points and then represented as a Lagrange polynomial:

$$v_i(x) = \sum_{j=0}^{\sigma} v_i(j) \prod_{\substack{l=0 \\ l \neq j}}^{\sigma} \frac{x - l}{j - l} = \sum_{j=0}^{\sigma} \alpha_{i,j} x^j . \tag{18}$$

Using the observation above, we can describe the secure unary conversion protocol. First, all users jointly run the *prefix multiplication subprotocol* [25, 26] to compute $([\![r^2]\!], \ldots, [\![r^\sigma]\!])$ from $[\![r]\!]$. And then, each user locally computes for all $i \in [1, \sigma]$ $[\![v_i]\!] = [\![v_i(r)]\!] = [\![\alpha_{i,0}]\!] \prod_{j=1}^{\sigma} [\![r^j]\!]^{\alpha_{i,j}}$ using the same randomness for encrypting $[\![\alpha_{i,0}]\!]$. The common random string can be used as the source of such randomness.

The protocol is, obviously, secure, as all computations are done over encrypted data and the prefix multiplication subprotocol is secure.

Jointly random shuffling. This protocol generates a jointly random permutation π, applies it to a given vector and rerandomizes the result. This subprotocol is based on matrix representations of permutations.

Matrix form of permutation $\pi = \begin{pmatrix} 1\,2\,3\,4\,5 \\ 3\,5\,2\,4\,1 \end{pmatrix}$ is the following full-range matrix $P(\pi)$:

$$P(\pi) = \begin{pmatrix} 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 & 0 \end{pmatrix} . \tag{19}$$

Applying a permutation to a vector and composing two permutations in the matrix form are performed using left-multiplication: $\pi(v)^T = P(\pi) \cdot v^T$ and $P(\pi_2 \circ \pi_1) = P(\pi_1) \cdot P(\pi_2)$. Moreover, if a permutation matrix and a source vector are both encrypted, then applying the *secure matrix multiplication subprotocol* [25] to them produces rerandomized shuffle, i.e. $[\![P(\pi) \cdot v^T]\!] = \text{Rand}\left(\pi([\![v]\!])\right)^T$.

Now we can describe the jointly random shuffling protocol. This protocol uses t-speedup method, which aims to decrease computational complexity of the protocol in t times by executing t additional rounds. For simplicity we assume $t \mid K$. The protocol is as follows:

1. Each user generates random permutation π_i.
2. Each group of t users builds an aggregate permutation by sequential combining permutation matrices $P(\pi_i), \ldots, P(\pi_{i+t})$: U_j receives $[\![P_{i,j-1}]\!]$ from the previous user in a group (U_i sets $[\![P_{i,i-1}]\!] = [\![E]\!]$), permutes matrix elements according to $P(\pi_j)$, rerandomizes the resulting $[\![P_{i,j}]\!]$ and broadcasts it.
3. All $[\![P_i]\!]$ are combined using the *unbound fan-in matrix multiplication subprotocol* [25,26]. Resulting permutation matrix $P(\pi)$ then applied to the given vector using the secure matrix multiplication.

We can use unbounded fan-in matrices multiplication subprotocol in the protocol above only in case, when the matrix size/field size ratio is negligible: $N/n \leq 2^{-\kappa}$. As in practice $\kappa = 80$ and n is at least 1024 bit long (according to recommendation of [28]), the protocol is applicable for the settings with $N \leq 2^{944}$ users, which is quite mild restriction.

The protocol is secure, because all underlying cryptographic primitives are known to be secure and the following observations hold. For any coalition $\mathbb{A}$ there is at least one group of users $U_i, \ldots, U_{i+t}$, which involves U_{i+k} such that $U_{i+k} \notin \mathbb{A}$. It is clear, that due to the rerandomization used by U_{i+k} on Step 2 of the protocol, coalition cannot learn the value of $P_{i,i+k}$ even in case it knows the value $P_{i,i+k-1}$. Hence, $P_{i,i+k}$ is uniformly random and unknown to users from $\mathbb{A}$, and consequently these users cannot select $\pi_{i+k+1},\ldots,\pi_{i+t}$ in a such that resulting $P_i = P_{i,i+k} \cdot P(\pi_{i+k+1}) \cdot \ldots \cdot P(\pi_{i+t})$ will not be random. And thus, coalition $\mathbb{A}$ (and any other) cannot reveal a combined permutation of at least one group, and consequently, the combined permutation π. Also, this permutation cannot be traced by comparing the source and the permuted vectors, because the permuted vector is rerandomized due to the properties of encrypted permutation matrix.

In the following protocol we use $t = 6$ as a tradeoff between round and computational complexities: doubling the number of rounds gives a six-time gain in the amount of computations. For simplicity we assume $K = 0 \bmod 6$.

By combining two described protocols we can obtain the protocol, which implements Approach 3 for group mask generation. We present the protocol in Protocol $\mathcal{P}_3^A$. The protocol is secure, because all underlying subprotocols and cryptographic primitives are secure. The protocols' Steps 1–3 and Steps 4–7 should be executed in parallel to reduce an overall round complexity and execution time.

Input: Each $U_{i\le N}$ holds his private value d_i.
Output: Party $\mathcal{A}$ receives $r = \sum_{i=1}^{N} d_i e_i$.

1. Users jointly run the *bounded random number generation subprotocol* [29] and produce $[\![r]\!] : r \in \mathbb{Z}_{\sigma+1}$.
2. Users jointly run the prefix multiplication subprotocol and compute $([\![r^2]\!],\ldots,[\![r^\sigma]\!])$ from $[\![r]\!]$.
3. U_1 builds and broadcasts vector $[\![v]\!]$:
$$[\![v_i]\!] = \begin{cases} [\![\alpha_{i,0}]\!] \cdot \prod_{j=1}^{\sigma} [\![r^j]\!]^{\alpha_{i,j}} & \text{if } i \in [1,\sigma], \\ [\![1]\!] & \text{if } i \in [\sigma+1, N]. \end{cases}$$
4. Each U_i generates random N-dimensional permutation π_i, and builds $P(\pi_i)$.
5. Each $U_{i=1 \bmod 6}$ sends $[\![P_i]\!] = [\![P(\pi_i)]\!]$ to U_{i+1}.
6. For $k \in [2,6]$ one by one: $U_{i=k \bmod 6}$ receives $[\![P_{i-1}]\!]$ from U_{i-1}, computes $[\![P_i]\!] = [\![P_{i-1} \cdot P(\pi_i)]\!] = [\![P_{i-1}]\!]^{P(\pi_i)}$, rerandomizes the result and broadcasts it.
7. Users jointly run the unbound fan-in matrix multiplication subprotocol, computing combined permutation matrix $[\![P(\pi)]\!] = [\![\prod_{i=K}^{1} P_i]\!]$.
8. Users jointly run the secure matrix multiplication subprotocol and compute $[\![e]\!] = [\![P(\pi) \cdot v^T]\!]^T$.
9. Each $U_{i\le N}$ computes $[\![d_i e_i]\!] = [\![e_i]\!]^{d_i}$ and broadcasts the result.
10. Users locally compute $[\![r]\!] = [\![\sum_{i=1}^{N} d_i e_i]\!] = \prod_{i=1}^{N} [\![d_i e_i]\!]$, and jointly run decryption to open r to $\mathcal{A}$.

Protocol $\mathcal{P}_3^A$. *GS with 3-rd group masking, semi-honest setting.*

4.2 Protocols for Malicious Setting

In this section we describe the protocols, which are secure in the malicious setting, where each party can violate the protocol and can participate in a coalition involving at most $K-1$ user. Hence, in this settings we have one additional security requirement to thus stated in Section 4.1: the correctness of all computations done locally by parties should be publicly verified.

We use non-interactive zero-knowledge proofs for verifying correctness of local operations. When one party sends a proof, all receivers should verify its correctness. We also suppose that on each step of the protocols each party performs basic consistency checks for each processed values: validating that plaintext indeed lays in $\mathbb{Z}_n$, randomness in $\mathbb{Z}_n^*$, and cypertext in $\mathbb{Z}_{n^2}^*$. We omit these checks in the protocol descriptions.

We assume that in the case of detecting a protocol violation, the party aborts. Note that in the following protocols all data received by the users is sent through the broadcast channel. Hence, malicious parties cannot cause two honest users to receive different data. And consequently, if any honest user aborts due to a protocol violation, others will abort simultaneously. Note that if $\mathcal{A}$ is malicious, it can refuse to check the validity of data received from users, and thus an undetected protocol violation can occur. But as $\mathcal{A}$ receives data only on the last step of Protocols $\mathcal{P}_0^B$–$\mathcal{P}_3^B$, such violation cannot harm the privacy of the users.

Reference Protocol. Similarly to the previous settings, in the malicious settings we first present the protocol for reference group service implementation without using the group masking. The protocol is described in Protocol $\mathcal{P}_0^B$.

Input: Each $U_{i\leq N}$ holds his private value d_i.
Output: Party $\mathcal{A}$ receives $r = \sum_{i=1}^{N} d_i$.

1. Each $U_{i\leq N}$ broadcasts $[\![d_i]\!]$ together with $\varPi_{PK}([\![d_i]\!], \mathbb{F})$.
2. Each user computes $[\![r]\!] = \left[\!\left[\sum_{i=1}^{N} d_i\right]\!\right] = \prod_{i=1}^{N} [\![d_i]\!]$.
3. Each U_i runs partial decryption of $[\![r]\!]$ and sends resulting $D_i([\![r]\!])$ together with $\varPi_{CD}([\![r]\!], D_i([\![r]\!]), i)$ to $\mathcal{A}$. User U_1 additionally sends the value $[\![r]\!]$ to $\mathcal{A}$.

Protocol $\mathcal{P}_0^B$. *GS without group masking, malicious setting.*

Note that the additional transmission by U_1 on Step 2 is unavoidable, as for the verification of the proof $\varPi_{CD}([\![r]\!], D_i([\![r]\!]), i)$ submitted by U_i, $\mathcal{A}$ should known the value of all parameters, including $[\![r]\!]$.

Security, privacy and correctness of Protocol $\mathcal{P}_0^B$ can be verified as follows. On Step 1 no information about d_i can leak, because K-out-of-N threshold Paillier cryptosystem is semantically secure even against $K-1$ colluding private key holders, and $\varPi_{PK}$ is zero-knowledge. Broadcasted encryption $[\![d_i]\!]$ is proven to be formed correctly by $\varPi_{PK}([\![d_i]\!], \mathbb{F})$. On Step 2, all computations are done over encrypted data and thus are secure. Correctness of these computations are checked by $\mathcal{A}$ on Step 3: if value $[\![r]\!]$ computed by U_i is not equal to $[\![r]\!]$ available to $\mathcal{A}$, then $\mathcal{A}$ detects incorrectness of $\Pi_{CD}([\![r]\!], D_i([\![r]\!]), i)$. On Step 3, $\mathcal{A}$ learns $D_i([\![r]\!])$ and $\Pi_{CD}([\![r]\!], D_i([\![r]\!]), i)$, which reveals nothing about U_i's secret key due to security of threshold Paillier encryption and zero-knowledge of Π_{CD}. Correctness of the executed partial decryptions are verified by Π_{CD}. Hence, the protocol is secure and privacy-preserving in the malicious static setting.

Protocol with Group Masking Using Approach 1. Approach 1 requires the value e to have the following properties: $e \in_R \{0,1\}^N$ and exactly m its components are equal to 1.

To generate such e, we use an approach based on permutations. Users take the predefined vector

$$v = (\underbrace{1, \ldots, 1}_{m}, \overbrace{0, \ldots, 0}^{N-m})$$

and randomly permute it to produce e: $e = \pi(v)$. The permutation is done using the jointly random shuffling protocol introduced in Section 4.1.

The jointly random shuffling protocol should be adjusted to remain being secure in the malicious settings. We require each U_i publishing $[\![P(\pi_i)]\!]$ to provide NIZK proof of $P(\pi_i)$ correctness, named Π_{PMC}, which is described below. Furthermore, we prefer not to use t-speedup method, as it requires to employ complicated and computational intensive NIZK proofs of correctness of local rerandomized permutations of $N \times N$ matrices.

Recalling the example of permutation matrix in Equation (19), one can note that a permutation matrix is a zero-one matrix containing exactly one 1 in each column and row. As in practice $N < n$, this condition can be formalized for any permutation matrix $P(\pi_i)$, which elements are denoted by p_{kl}, as follows: $\forall k, l \in [1, N] p_{kl} \in \{0,1\}$ and $\forall k \in [1, N] \sum_{l=1}^{N} p_{kl} = \sum_{l=1}^{N} p_{lk} = 1$. The feasibility of the first of these properties can be proved using existing technique introduced in [30] and denoted $\Pi_{BZO}([\![p_{11}]\!], \ldots, [\![p_{NN}]\!])$. To construct the proof for the second property note: if $\sum_{l=1}^{N} p_{kl} = 1$, then $\prod_{l=1}^{N} [\![p_{kl}]\!] = \left[\!\!\left[\sum_{l=1}^{N} p_{kl}\right]\!\!\right] = E(1, r_k) = g r_k^N \bmod N^2 = c_k$ and only the user, who built an encryptions $[\![p_{kl}]\!] = E(p_{kl}, r_{kl})$, knows the value of $r_k = \prod_{l=1}^{N} r_{kl}$. This user can prove his knowledge by proving the knowledge of N-th root of $(c_k/g) = r_k^N \bmod N^2$ using $\Pi_{RK}(\prod_{l=1}^{N} [\![p_{kl}]\!], N)$ introduced in [31].

To sum up, U_i can prove that $[\![P(\pi_i)]\!]$ is formed correctly using one invocation of Π_{BZO} and $2N$ invocations of Π_{RK}, all of which can be done in parallel and non-interactively. We denote this proof as $\Pi_{PMC}([\![P(\pi_i)]\!])$. It is straightforward that Π_{PMC} is zero-knowledge. The resulting protocol using this primitive is described in Protocol $\mathcal{P}_1^B$.

Security, privacy and correctness of Protocol $\mathcal{P}_1^B$ can be verified as follows. Correctness of $P(\pi_i)$ broadcasted on Step 1 is verified by Π_{PMC}, its privacy is

Input: Each $U_{i\leq N}$ holds his private value d_i.
Output: Party $\mathcal{A}$ receives $r = \sum_{i=1}^{N} d_i$.

1. Each U_i generates random N-dimensional permutation π_i, builds $P(\pi_i)$ and broadcasts its encryption $[\![P(\pi_i)]\!]$ together with $\Pi_{PMC}([\![P(\pi_i)]\!])$.
2. Users jointly run the unbound fan-in matrix multiplication subprotocol, computing combined permutation matrix $[\![P(\pi)]\!] = [\![\prod_{i=K}^{1} P(\pi_i)]\!]$.
3. Each $U_{i\leq N}$ locally multiplies $[\![P(\pi)]\!]$ with plaintext $v^T = (\underbrace{1,\ldots,1}_{m},\overbrace{0,\ldots,0}^{N-m})^T$ and obtains $[\![e]\!]^T$.
4. Each $U_{i\leq N}$ computes $[\![d_ie_i]\!] = [\![e_i]\!]^{d_i}$ and broadcasts $[\![d_i]\!], [\![d_ie_i]\!], \Pi_{PK}([\![d_i]\!], \mathbb{F})$ and $\Pi_{CM}([\![d_i]\!], [\![e_i]\!], [\![d_ie_i]\!])$.
5. Each user computes $[\![r]\!] = \left[\!\!\left[\sum_{i=1}^{N} d_ie_i\right]\!\!\right] = \prod_{i=1}^{N} [\![d_ie_i]\!]$.
6. Each U_i runs partial decryption of $[\![r]\!]$ and sends resulting $D_i([\![r]\!])$ together with $\Pi_{CD}([\![r]\!], D_i([\![r]\!]), i)$ to $\mathcal{A}$. U_1 additionally sends the value $[\![r]\!]$ to $\mathcal{A}$.

Protocol $\mathcal{P}_1^B$. *GS with 1-st group masking, malicious setting.*

preserved as Π_{PMC} is zero-knowledge and threshold Paillier encryption is semantically secure. Computations on Step 2 are secure, privacy-preserving and correct due to the corresponding properties of the unbound fan-in matrix multiplication subprotocol. Computations on Step 3 obviously do not leak any data. Their correctness is verified on Step 4: if U_i computes the different value of $[\![e_i]\!]$ than honest U_j, then U_j will not accept $\Pi_{CM}([\![d_i]\!], [\![e_i]\!], [\![d_ie_i]\!])$ as valid. Correctness of $[\![d_ie_i]\!]$ computed on Step 4 is verified using $\Pi_{CM}([\![d_i]\!], [\![e_i]\!], [\![d_ie_i]\!])$, the fact that used $[\![d_i]\!]$ is well-formed — by $\Pi_{PK}([\![d_i]\!], \mathbb{F})$.

Step-by-step verification of security, privacy and correctness of the other steps of the protocol is skipped here, as it can be done similarity to Protocol $\mathcal{P}_0^B$.

Protocol with Group Masking Using Approach 2. Approach 2 requires value e to have the following properties: $e \in_R \{0,1\}^N$ and for all $i \in [1,N]$ $P(e_i{=}1) = p$.

To generate e_i with respect to the distribution above, we use the following technique: (i) users jointly generate N uniformly random $r_i \in [0, 2^k-1]$; (ii) each e_i is computed as $e_i = r_i \overset{?}{<} \lceil 2^k p\rceil$. The value of k is publicly known and should be selected in a such way that relative error $\lceil 2^k p\rceil/(2^k p) - 1$ is negligible.

As r_i is a random number from $[0, 2^k-1]$, it can be generated as k independent bits by employing the *random bit generation subprotocol* [29]. The comparison $(r_i \overset{?}{<} \lceil 2^k p\rceil)$ can be done using the *bitwise less-than subprotocol* from [32], which compares bit-decomposed number with publicly known constant. To use this subprotocol we should additionally restrict the value of k: $k < \log n - \log K - \kappa - 1$. In practice $\kappa = 80$ and n is 1024 bit length, and thus this restriction is satisfied when $k \leq 943 - \log N$.

The protocol based on the described technique and aforementioned subprotocols is given in Protocol $\mathcal{P}_2^B$. Its security, privacy and correctness are based on the corresponding properties of underlying cryptographic primitives and can be verified in the same way as for the previous protocols.

Input: Each $U_{i\leq N}$ holds his private value d_i.
Output: Party $\mathcal{A}$ receives $r = \sum_{i=1}^{N} d_i$.

1. Users jointly run the random bit generation subprotocol kN times in parallel and produce $[\![b_{1,0}]\!], \ldots, [\![b_{N,k-1}]\!]$, where each $b_{i,j} \in \{0,1\}$.
2. Users jointly run the bitwise less-than subprotocol N times in parallel, computing for each $i \in [1, N]$ value $[\![e_i]\!] = \left[\!\!\left[b_{i,k-1} \ldots b_{i,0} \overset{?}{<} \lceil 2^k p \rceil \right]\!\!\right]$.
3. Each $U_{i\leq N}$ computes $[\![d_i e_i]\!] = [\![e_i]\!]^{d_i}$ and broadcasts $[\![d_i]\!], [\![d_i e_i]\!], \Pi_{PK}([\![d_i]\!], \mathbb{F})$ and $\Pi_{CM}([\![d_i]\!], [\![e_i]\!], [\![d_i e_i]\!])$.
4. Each user computes $[\![r]\!] = \left[\!\!\left[\sum_{i=1}^{N} d_i e_i \right]\!\!\right] = \prod_{i=1}^{N} [\![d_i e_i]\!]$.
5. Each U_i runs partial decryption of $[\![r]\!]$ and sends resulting $D_i([\![r]\!])$ together with $\Pi_{CD}([\![r]\!], D_i([\![r]\!]), i)$ to $\mathcal{A}$. U_1 additionally sends the value $[\![r]\!]$ to $\mathcal{A}$.

Protocol $\mathcal{P}_2^B$. *GS with 2-nd group masking, malicious setting.*

Protocol with Group Masking Using Approach 3. Approach 3 requires value e to have the following properties: $e \in_R \{0,1\}^N$ and $\sum_{i=1}^{N} e_i$ is uniformly random in $[P(N), N]$.

To generate such e we use the same approach as in Protocol $\mathcal{P}_3^A$: (i) generate vector v with uniformly random number (greater or equal to $Q(N)$) of ones, using secure unary conversion subprotocol; (ii) shuffle v to produce e.

The protocol based on the described technique is stated in Protocol $\mathcal{P}_3^B$. Note that Steps 1–4 and Steps 5–6 can be executed in parallel to reduce an overall round complexity and execution time. We leave it to the reader to verify the security, privacy and correctness of the protocol.

5 Complexity Analysis

In this section we give the complexity of the protocols introduced in Section 4. We focus on three aspects of the performance of the protocols: number of interactive rounds executed, amount of data transferred through the network and the computational complexity of local operations executed by the parties.

The most computational intensive local operations are exponentiations in the cyphertext domain. In practice, the complexity of other operations can be considered as negligible comparing to exponentiation of a cyphertext, and thus, can be omitted from consideration while estimating the total local workload.

For the sake of simplicity, here we give only the asymptotic approximation for the number of executed exponentiations and transferred bits. Also we do not

Input: Each $U_{i \leq N}$ holds his private value d_i.
Output: Party $\mathcal{A}$ receives $r = \sum_{i=1}^{N} d_i$.

1. Users jointly run the bounded random value generation subprotocol and produce $[\![r]\!] : r \in \mathbb{Z}_{\sigma+1}$.
2. Users jointly run the prefix multiplication subprotocol, computing $([\![r^2]\!], \ldots, [\![r^\sigma]\!])$ from $[\![r]\!]$.
3. Each user computes vector $[\![w]\!]$:
$$\forall i \in [1, \sigma]\colon\ [\![w_i]\!] = [\![\alpha_{i,0}]\!] \cdot \prod_{j=1}^{\sigma} [\![r^j]\!]^{\alpha_{i,j}}$$
using common randomness for encrypting $[\![\alpha_{i,0}]\!]$.
4. Users set value $[\![v]\!]$:
$$[\![v]\!] = (\underbrace{[\![1]\!], \ldots, [\![1]\!]}_{P(N)}) \| [\![w]\!]$$
using common randomness for encrypting $[\![1]\!]$.
5. Each U_i generates random N-dimensional permutation π_i, builds $P(\pi_i)$ and broadcasts its encryption $[\![P(\pi_i)]\!]$ together with $\Pi_{PMC}([\![P(\pi_i)]\!])$.
6. Users jointly run the unbound fan-in matrix multiplication subprotocol, computing combined permutation matrix $[\![P(\pi)]\!] = [\![\prod_{i=K}^{1} P(\pi_1)]\!]$.
7. Users jointly run the secure matrix multiplication subprotocol and compute $[\![e]\!] = [\![P(\pi) \cdot v^T]\!]^T$.
8. Each $U_{i \leq N}$ computes $[\![d_i e_i]\!] = [\![e_i]\!]^{d_i}$ and broadcasts $[\![d_i]\!], [\![d_i e_i]\!], \Pi_{PK}([\![d_i]\!], \mathbb{F})$ and $\Pi_{CM}([\![d_i]\!], [\![e_i]\!], [\![d_i e_i]\!])$.
9. Each user computes $[\![r]\!] = [\![\sum_{i=1}^{N} d_i e_i]\!] = \prod_{i=1}^{N} [\![d_i e_i]\!]$.
10. Each U_i runs partial decryption of $[\![r]\!]$ and sends resulting $D_i([\![r]\!])$ together with $\Pi_{CD}([\![r]\!], D_i([\![r]\!]), i)$ to $\mathcal{A}$. U_1 additionally sends the value $[\![r]\!]$ to $\mathcal{A}$.

Protocol $\mathcal{P}_3^B$. *GS with 3-rd group masking, malicious setting.*

consider the workload and bandwidth that is required to create, transfer and verify the NIZK proofs employed through the protocols for the malicious setting, but consider only the number of invocations of these proofs.

The number of interactive rounds, executed exponentiations in the field $\mathbb{Z}_{n^2}^*$ and number of bits transferred through the network during an execution of Protocols $\mathcal{P}_0^A$–$\mathcal{P}_3^B$ are presented in Table 1. The number of invocations of different NIZK proofs during an execution of Protocols $\mathcal{P}_0^B$–$\mathcal{P}_3^B$ are presented in Table 2.

5.1 Possible Optimizations

The presented protocols are constant-round, i.e. the number of interactive rounds executed during each protocol run is constant and does not depend neither on system configuration nor on input data. This property is significant as in practice the round complexity affect on the overall system performance is crucial,

Table 1. Complexity of the protocols

	Number of rounds	Number of exponentiations	Number of bits transferred
Protocol $\mathcal{P}_0^A$	2	$O(N)$	$O(N)$
Protocol $\mathcal{P}_1^A$	10	$O\left(K^2N\kappa\sqrt{m}\right)$	$O\left(K^2N\kappa\sqrt{m}\right)$
Protocol $\mathcal{P}_2^A$	8	$O\left(K^2N\right)$	$O\left(K^2N\right)$
Protocol $\mathcal{P}_3^A$	22	$O\left(K^2N^3\right)$	$O\left(K^2N^2\right)$
Protocol $\mathcal{P}_0^B$	2	$O(N)$	$O(N)$
Protocol $\mathcal{P}_1^B$	9	$O\left(K^3N^3\right)$	$O\left(K^2N^2\right)$
Protocol $\mathcal{P}_2^B$	12	$O\left(K^2Nk\right)$	$O(KNk)$
Protocol $\mathcal{P}_3^B$	21	$O\left(K^3N^3\right)$	$O\left(K^2N^2\right)$

Table 2. Number of invocations of NIZK proofs in the protocols

	Π_{CD}	Π_{CM}	Π_{PK}	Π_{PMC}
Protocol $\mathcal{P}_0^B$	$O(K)$	0	$O(N)$	0
Protocol $\mathcal{P}_1^B$	$O\left(K^2N^2\right)$	$O\left(K^2N^3\right)$	$O(N)$	$O(K)$
Protocol $\mathcal{P}_2^B$	$O(KNk)$	$O(KNk)$	$O(KN)$	0
Protocol $\mathcal{P}_3^B$	$O\left(K^2N^2\right)$	$O\left(K^2N^3\right)$	$O(N+K\kappa)$	$O(K)$

and thus, proposed protocols can be used for settings with many users, with higher value of K and other parameters. Nevertheless, for the settings where only few users are involved, or where the security settings are relaxed (in terms that maximum number of user involved in each coalition $K-1$ is smaller), protocols with lower communicational and computational complexities can be used: linear or logarithmic-round protocols.

For example, unbounded fan-in XOR subprotocol can be executed in $O(\log K)$ rounds, using the logarithmic-depth arithmetic circuits. Shuffling subprotocol can be executed in $O(K)$ rounds using mixnets [33].

Also note that presented protocols are designed for the general case of $K \leq N$, while for the settings with higher restrictions on maximum number of colluding users the more efficient solutions can be proposed. For example, when $K \leq N/2$, i.e. when there is no coalition involving the majority of the users, protocols based on the Shamir secret sharing [20] can be used.

Other possible approach for optimisations, is to reduce the users' workload by passing their duties to the separate service parties. It can be done, for example, by introducing service parties $\mathcal{B}_1, \dots, \mathcal{B}_K$ such, that no more than $K-1$ of them are colluding. It is clear, that in this settings Protocols $\mathcal{P}_0^A$–$\mathcal{P}_3^B$ can be carried by these service parties, while users only need to once pass their encrypted private data.

6 Conclusion

In this paper we propose a method to provide protection of user data processed by a group service in dynamic scenarios, which are more realistic than static ones for a wide range of applications. This method is realized using a set of cryptographic protocols, which are designed with performance in mind, offering the powerful privacy-protection tool for group services in two mostly addressed security settings. The protocols are shown to be correct, secure and privacy-preserving. The complexity analysis with respect to the versions in two attacker models clearly shows the advantages and disadvantages of the protocols in terms of computational and communication costs, and the level of privacy protection. Our protocols can be further used as building blocks for implementing privacy-preserving group services in a dynamic setting.

Acknowledgements. We would like to thank Sebastiaan de Hoogh for suggesting the t-speedup method for the shuffling subprotocol in Protocol $\mathcal{P}_3^A$. This publication was supported by the Dutch national program COMMIT.

References

1. McSherry, F., Mironov, I.: Differentially Private Recommender Systems: Building Privacy into the Netflix Prize Contenders. In: Elder IV, J.F., Fogelman-Soulié, F., Flach, P.A., Zaki, M.J. (eds.) KDD, pp. 627–636. ACM (2009)
2. Beye, M., Erkin, Z., Lagendijk, R.: Efficient privacy preserving K-means clustering in a three-party setting. In: IEEE International Workshop on Information Forensics and Security, WIFS, pp. 1–6. IEEE (2011)
3. Erkin, Z., Veugen, T., Toft, T., Lagendijk, R.: Generating Private Recommendations Efficiently Using Homomorphic Encryption and Data Packing. IEEE Transactions on Information Forensics and Security 7, 1053–1066 (2012)
4. Canny, J.: Collaborative filtering with privacy. In: IEEE Symposium on Security and Privacy, pp. 45–57. IEEE, IEEE Computer Society (2002)
5. Jagannathan, G., Pillaipakkamnatt, K., Wright, R.: A new privacy-preserving distributed k-clustering algorithm. In: Proceedings of the Sixth SIAM International Conference on Data Mining, pp. 492–496 (2006)
6. Resnick, P., Varian, H.R.: Recommender Systems - Introduction to the Special Section. Commun. ACM 40, 56–58 (1997)
7. Resnick, P., Kuwabara, K., Zeckhauser, R., Friedman, E.: Reputation systems. Commun. ACM 43, 45–48 (2000)
8. Kacprzyk, J.: Group decision making with a fuzzy linguistic majority. Fuzzy Sets and Systems 18, 105–118 (1986)
9. Mathes, A.: Folksonomies – cooperative classification and communication through shared metadata. Computer Mediated Communication 47 (2004)
10. Kargupta, H., Datta, S., Wang, Q., Sivakumar, K.: On the Privacy Preserving Properties of Random Data Perturbation Techniques. In: ICDM, pp. 99–106. IEEE Computer Society (2003)

11. Zhou, B., Pei, J., Luk, W.S.: A brief survey on anonymization techniques for privacy preserving publishing of social network data. SIGKDD Explorations 10, 12–22 (2008)
12. Yao, A.C.: Protocols for secure computations. In: Proceedings of the 23rd Annual IEEE Symposium on Foundations of Computer Science, pp. 160–164. IEEE Computer Society, Washington, DC (1982)
13. Steiner, M., Tsudik, G., Waidner, M.: CLIQUES: a new approach to group key agreement. In: Papazoglou, M.P., Takizawa, M., Kramer, B., Chanson, S. (eds.) ICDCS, pp. 380–387 (1998)
14. Wu, S., Chen, K.: An Efficient Key-Management Scheme for Hierarchical Access Control in E-Medicine System. Journal of Medical Systems 36, 2325–2337 (2012)
15. Xiao, X., Tao, Y.: M-invariance: towards privacy preserving re-publication of dynamic datasets. In: Proceedings of the 2007 ACM SIGMOD International Conference on Management of Data, pp. 689–700. ACM (2007)
16. Bu, Y., Fu, A.W.C., Wong, R.C.W., Chen, L., Li, J.: Privacy preserving serial data publishing by role composition. PVLDB 1, 845–856 (2008)
17. Bhagat, S., Cormode, G., Krishnamurthy, B., Divesh, S.: Prediction promotes privacy in dynamic social networks. In: Proceedings of the 3rd Conference on Online Social Networks, WOSN 2010, p. 6. USENIX Association, Berkeley (2010)
18. Fouque, P.-A., Poupard, G., Stern, J.: Sharing decryption in the context of voting or lotteries. In: Frankel, Y. (ed.) FC 2000. LNCS, vol. 1962, pp. 90–104. Springer, Heidelberg (2001)
19. Nishide, T., Sakurai, K.: Distributed Paillier Cryptosystem without Trusted Dealer. In: Chung, Y., Yung, M. (eds.) WISA 2010. LNCS, vol. 6513, pp. 44–60. Springer, Heidelberg (2011)
20. Shamir, A.: How to Share a Secret. Commun. ACM 22, 612–613 (1979)
21. Paillier, P.: Public-Key Cryptosystems Based on Composite Degree Residuosity Classes. In: Stern, J. (ed.) EUROCRYPT 1999. LNCS, vol. 1592, pp. 223–238. Springer, Heidelberg (1999)
22. Hazay, C., Lindell, Y.: Sigma Protocols and Efficient Zero-Knowledge. In: Efficient Secure Two-Party Protocols. Information Security and Cryptography, pp. 147–175. Springer, Heidelberg (2010)
23. Blum, M., Feldman, P., Micali, S.: Non-Interactive Zero-Knowledge and Its Applications (Extended Abstract). In: Simon, J. (ed.) STOC, pp. 103–112. ACM (1988)
24. Cramer, R., Damgård, I., Nielsen, J.B.: Multiparty Computation from Threshold Homomorphic Encryption. Cryptology ePrint Archive, Report 2000/055 (2000), http://eprint.iacr.org/2000/055
25. Cramer, R., Damgård, I., Nielsen, J.B.: Multiparty Computation from Threshold Homomorphic Encryption. In: Pfitzmann, B. (ed.) EUROCRYPT 2001. LNCS, vol. 2045, pp. 280–299. Springer, Heidelberg (2001)
26. Bar-Ilan, J., Beaver, D.: Non-cryptographic fault-tolerant computing in constant number of rounds of interaction. In: Proceedings of the Eighth Annual ACM Symposium on Principles of Distributed Computing, pp. 201–209. ACM (1989)
27. Davies, R.: Exclusive OR (XOR) and hardware random number generators (2002), http://www.robertnz.net/
28. Barker, E., Barker, W., Burr, W., Polk, W., Smid, M.: Recommendation for key management–part 1: General (revision 3). NIST special publication 800-57 (2012)
29. Damgård, I., Fitzi, M., Kiltz, E., Nielsen, J.B., Toft, T.: Unconditionally Secure Constant-Rounds Multi-party Computation for Equality, Comparison, Bits and Exponentiation. In: Halevi, S., Rabin, T. (eds.) TCC 2006. LNCS, vol. 3876, pp. 285–304. Springer, Heidelberg (2006)

30. Peng, K., Bao, F.: Efficient Vote Validity Check in Homomorphic Electronic Voting. In: Lee, P.J., Cheon, J.H. (eds.) ICISC 2008. LNCS, vol. 5461, pp. 202–217. Springer, Heidelberg (2009)
31. Guillou, L.C., Quisquater, J.-J.: A "Paradoxical" identity-based signature scheme resulting from zero-knowledge. In: Goldwasser, S. (ed.) CRYPTO 1988. LNCS, vol. 403, pp. 216–231. Springer, Heidelberg (1990)
32. Catrina, O., de Hoogh, S.: Improved Primitives for Secure Multiparty Integer Computation. In: Garay, J.A., De Prisco, R. (eds.) SCN 2010. LNCS, vol. 6280, pp. 182–199. Springer, Heidelberg (2010)
33. Chaum, D.: Untraceable Electronic Mail, Return Addresses, and Digital Pseudonyms. Commun. ACM 24, 84–88 (1981)