Veugen *et al. EURASIP Journal on Information Security* (2017) 2017:3
DOI 10.1186/s13635-017-0054-7

EURASIP Journal on Information Security

RESEARCH Open Access

Improved privacy of dynamic group services

Thijs Veugen[1,2*], Jeroen Doumen[3], Zekeriya Erkin[1], Gaetano Pellegrino[1], Sicco Verwer[1] and Jos Weber[1]

Abstract

We consider dynamic group services, where outputs based on small samples of privacy-sensitive user inputs are repetitively computed. The leakage of user input data is analysed, caused by producing multiple outputs, resulting from inputs of frequently changing sets of users. A cryptographic technique, known as random user selection, is investigated. We show the effect of random user selection, given different types of output functions, thereby disproving earlier work. A new security measure is introduced, which provably improves the privacy-preserving effect of random user selection, irrespective of the output function. We show how this new security measure can be implemented in existing cryptographic protocols. To investigate the effectiveness of our security measure, we conducted a couple of statistical simulations with large user populations, which show that it forms a key ingredient, at least for the output function addition. Without it, an adversary is able to determine a user input, with increasing accuracy when more outputs become available. When the security measure is implemented, an adversary remains oblivious of user inputs, even when thousands of outputs are collected. Therefore, our new security measure assures that random user selection is an effective way of protecting the privacy of dynamic group services.

Keywords: Dynamic group services, Random user selection, Cryptography, User data privacy

1 Introduction

Privacy enhancing technologies are increasingly being deployed [1], as they provide secure solutions that limit information leakage of sensitive data. Among others, cryptography-based solutions have been applied in several domains, such as biometric data matching [2, 3], smart grids [4], recommendation systems [5], and genomic privacy [6].

The main idea in this line of research is to obfuscate the privacy-sensitive data prior to processing, which is supposed to be performed in an untrustworthy environment. A typical approach is to provide homomorphically *encrypted* versions of the original data and let the service provider process encrypted data to offer the service. This approach protects the privacy-sensitive data, and possibly also the algorithm being performed by the service provider, at the cost of increased computation and communication: the cryptographic privacy-preserving protocols require more computationally intensive operations on the encrypted data and more interaction with the holder of the decryption key.

In a typical privacy-sensitive system, for example, a recommendation system, inputs from several users are being aggregated and processed for a certain goal, in this example, generating recommendations for a particular user. The cryptographic protocol designed for such a system can be proven secure assuming a security model: semi-honest, covert, or malicious, meaning that to a well-defined extent, the inputs from the users are protected from the service provider and any malicious attacker in an execution, thus providing trustworthiness. However, in [7], Kononchuk et al. addressed a practical aspect of privacy-preserving protocols: repeating the protocol for a number of times with a different set of participating users reveals information on the private data of the users. Preserving data privacy of users, who repeatedly join the execution of a service with different groups, had not been considered until then. This trust problem is a valid scenario, seen for example in recommendation systems [5], aggregating data in smart grids, reputation

*Correspondence: thijs.veugen@tno.nl
[1]Cyber Security research group, Delft University of Technology, Mekelweg 2, 2628 CD, Delft, The Netherlands
[2]TNO, Technical Sciences, P.O. Box 96800, 2509 JE, The Hague, The Netherlands
Full list of author information is available at the end of the article

systems [8], unsupervised machine learning [9], collective decision-making [10], data clustering [11], and social classification [12].

To cope with leaking information after several service executions with different sets of inputs, Kononchuk et al. [7] propose the idea of selecting a random subset of users for the computation of an output function, for example, for computing suggestions in a recommendation system. To randomly select users, a random bit vector is generated, where each element indicates whether a particular user is selected or not for computing the current group service. This selection vector is kept hidden from all parties, including the server. We call this the *random user selection* approach, see Section 3.1 for a possible implementation. It is argued in [7] that this random user selection approach protects the private data of the users better, when the group service is executed several times.

1.1 Problem description

In this paper, we investigate the information leakage after a number of executions and improve the random user selection approach. We consider a server, or a group of servers, which repeatedly provide services to many users. The inputs of the users are considered sensitive, personal information, and the server computes a function F from these inputs. The output of the function is made public or is given to a particular user. The server, or any adversary, is not allowed to learn or infer the inputs.

More precisely, let N be the number of users, with each user u, $1 \leq u \leq N$, having an input x_u. To simplify the problem, we assume that the inputs of the users are independently and identically distributed according to some random distribution $\mathcal{X}$. We consider a number, say k, of consecutive executions of the *group service* F, leading to outputs $y_1, \ldots, y_k$. In each execution, the users that are involved in the group service may change, possibly because it is a service provided through Internet and not all users are continuously online. We assume the user inputs x_u do not change during the k executions.

Just as in [7], we distinguish two sets of users. The set U_S is the set of *static* users that are involved in all k computations of F. The users that participate at least once, but not each time, form the set of *dynamic* users U_D. We assume that each user participates at least once, so $U_S \cup U_D = \{1, 2, \ldots, N\}$. Let U_r be the set of users that are available for execution r, $1 \leq r \leq k$. Since static users are always available, $U_S \subseteq U_r$, for each r, $1 \leq r \leq k$.

We also assume that function F is symmetric, so the order of the inputs is not relevant and also that F can take any number of inputs. This is a valid assumption for most examples of group services mentioned earlier. We summarize our notation in Table 1.

We are concerned about the privacy leakage of user data to an adversary, given the k outputs of F. The amount of leakage naturally depends on the function F. We assume the adversary knows the function F, the user sets U_r, $1 \leq r \leq k$, as well as all the outputs y_r. This assumption seems fair in an Internet-like setting where the server can observe the connections with the users, so he knows who is currently online or not.

The random user selection approach consists of choosing for each execution r a random subset T_r of U_r, containing M user indices, and having the server compute $y_r = F(\{x_u \mid u \in T_r\})$, which we also denote as $y_r = F(T_r)$. We will use cryptographic tools, as shown in Section 3.1, to assure that neither the server, nor the users, will learn these subsets T_r. The idea is that by hiding T_r, which is the set of users whose data is actually used as an input, the k outputs will reveal less about the personal user data. The adversary is allowed to know the cardinality M, but not the elements of the sets T_r. We define the privacy of user u as the conditional entropy, which can be calculated by $H(X \mid Y) = -E \log \Pr(X \mid Y)$, of the user input given all outputs and user sets:

$$\text{Privacy}_u = H(x_u \mid y_1, \ldots, y_k, U_1, \ldots U_k) \,. \tag{1}$$

In this definition, we consider the sets U_r as random variables and assume T_r is uniformly drawn from U_r. Since the random user selection approach possibly decreases the number of inputs (from $|U_r|$ to $|T_r|$), this could lead to a reduced accuracy of the output, e.g. the result of the recommendation. Therefore, our approach is most suited for the class of functions with low sample complexity [13], where the output can be accurately approximated by looking only at a small fraction of the database.

1.2 Related work

To overcome the privacy problem, mentioned in the previous subsection, different measures including laws and organizational means have been deployed. These measures are also supported by scientific solutions, which aim to guarantee the privacy of user data, like data perturbation [12] and data anonymization [14]. A recent idea in the field is to employ secure multiparty computations techniques [15], which allow service providers to process user data through interactive protocols without disclosing their content.

Unfortunately, the existing solutions only consider a static environment, where the set 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 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 privacy-preserving

Table 1 Notation summary

Symbol	Definition	Symbol	Definition
N	Number of users	$H(\cdot)$	Binary entropy function
x_u	Input of user u	$F(\cdot)$	Output function
k	Number of runs (or rounds)	$\mathcal{X}$	Random input distribution
y_r	Output at run r	U_S	Set of static users
U_D	Set of dynamic users	U_r	Set of users available at run r
T_r	Random subset of U_r	M	Cardinality of T_r
$\mathcal{A}$	The service party	$h(\cdot)$	Hash function

protocols that are able to reduce the effect of information leakage caused by repetitively executing some group service.

Related problems exist in data publishing [16], where different anonymized data sets can be combined by an adversary. Here, the output function is a list of (anonymized) inputs, whereas we consider arithmetic computations leading to relatively small outputs. In model inversion [17], machine-learning algorithms are used to deduce sensitive personal information from various outputs. This is different from finding personal inputs to a specific output function.

Privacy of dynamic group services is also linked to differential privacy [13], where information is queried from statistical databases without identifying users. Their goal is to obfuscate the input of the user, by adding some noise to it. The main difference with our setting is that they assume the database is fixed, whereas we consider the set U_r as a random variable. Instead of adding noise to the user inputs, we randomize and hide the random sample T which is used for computing the query.

1.3 Outline

The rest of the paper is organized as follows: Section 2 provides a privacy analysis of the random user selection approach. Section 2.1 shows that the effect of the random user selection approach strongly depends on the type of output function. A new security measure is introduced in Section 2.2, which improves the privacy-preserving effect of the random user selection approach, regardless of the output function. This, and some basic properties of the random user selection approach, are proven in Section 2.3. In Section 3, we show how to incorporate our new security measure into existing random user selection protocols. The effect of this security measure on the level of privacy protection is demonstrated in Section 4. And finally, in Section 5, we summarize our contributions.

2 Analysis of random user selection approach

In this section, we analyse the security properties of the random user selection approach from [7]. In particular, we show the relation between privacy of different users, following the privacy definition, as depicted in Eq. 1 from the previous section. The small examples in the first subsection show that the actual choice of function F greatly influences the privacy aspects. In the second subsection, we introduce a way of reducing the information leakage of privacy-sensitive user inputs. This is formally shown in the final subsection, together with a number of basic propositions for random user selection, which are valid for any symmetric function F.

Let us continue by considering the privacy claims of Kononchuk et al. [7], who use the same definition of privacy by means of entropy. Their main claim is that the random user selection approach assures that the privacy of static users is protected better than the privacy of dynamic users, as long as there are at least as many static users as dynamic ones, i.e. $|U_S| \geq |U_D|$. The informal argument for this claim is that the differences between consecutive outputs will reveal less about static inputs because they are always incorporated. Only for dynamic inputs, it makes sense to look at the differences between outputs. Quite surprisingly, although all their examples use an addition function, Kononchuk et al. only assume the function F is symmetric.

Their formal argument for this claim contains two steps:

1. First, they show that if the majority of users is static, then the intersections $T_{r_1} \cap T_{r_2}$ and symmetric differences $T_{r_1} \triangle T_{r_2}$ of any two (or more) drawn subsets T_{r_1} and T_{r_2} also have a majority of static users on average.
2. Second, they state (without proof) that if all these intersections and symmetric differences contain an average majority of static users, then the privacy of static users will be better than the privacy of dynamic users.

In their proof of the first step, they erroneously omitted the restriction that at most half of the available users are drawn. For example, if $T = U$, then $T_{r_1} \triangle T_{r_2}$ will not contain any static user. More importantly, a simple counter example of the second step is the case where only singletons are drawn, i.e. $|T_r| = 1$ and $F(x) = x$. Since static

users are more likely to be chosen, their inputs are more likely revealed, and their privacy is protected less. The first example of the next subsection illustrates this.

2.1 Different types of output functions

We investigated a couple of cases with different output functions, showing that the type of function is of great importance. It even shows that the main claim of [7] certainly does not hold for all symmetric functions.

In our examples, we only have $N = 4$ users, of which the first two are static ($U_S = \{1, 2\}$) and the second two are dynamic ($U_D = \{3, 4\}$). We consider only $k = 2$ consecutive computations of the function F. During the first run, the users $U_1 = \{1, 2, 3\}$ are online, and during the second run, the users $U_2 = \{1, 2, 4\}$ are online. We assume the input x_i of user i, $1 \leq i \leq 4$, is a uniformly randomly chosen bit.

In the first example, the system uniformly randomly chooses one of the three online users and outputs its input. In other words, $|T_1| = |T_2| = 1$ and $y_j = F(T_j) = x_i$ such that $T_j = \{i\}$, $j = 1, 2$. Computing the entropies, by considering all possible outcomes, shows that in this example, $H(x_3 \mid y_1, y_2) \approx 0.91$ bits and $H(x_1 \mid y_1, y_2) \approx 0.86$ bits. This means that the input of user 3, a dynamic user, is protected better than the input of user 1, a static user. Intuitively, this can be explained: since static users are online more often, their input will be selected and revealed more often. This may be different when more user inputs are combined in the output function F.

In the second example, the system adds up the inputs of all three online users. In other words, $T_j = U_j$ and $y_j = F(T_j) = \sum_{i \in T_j} x_i, j = 1, 2$. Computing the entropies, by considering all possible outcomes, shows that in this example, $H(x_3 \mid y_1, y_2) \approx 0.34$ bits and $H(x_1 \mid y_1, y_2) \approx 0.59$ bits. This means that the input of user 3, a dynamic user, is protected worse than the input of user 1, a static user. Although formally, the random user selection part is missing ($T_j = U_j$), this example is in line with [7].

Instead of adding, we can also multiply all inputs. This is done in the third example. Computing the entropies again leads to $H(x_3 \mid y_1, y_2) \approx 0.81$ bits and $H(x_1 \mid y_1, y_2) \approx 0.78$ bits. Somewhat surprising, there is a difference with the previous addition function because this time, the inputs of dynamic users are protected better than the inputs of the static users. An informal analysis show that all inputs must be one, in case the output is one, so this instance reveals a lot of information. Since static users are involved more often, their privacy is protected worse.

As a final example, we choose an exclusive-or of the input bits. In this case, it can be easily argued that the values of y_1 and y_2 do not reveal any knowledge on each of the individual user bits, i.e. $H(x_i \mid y_1, y_2) = 1$ for all i. Hence, static and dynamic users are equally well protected.

We conclude from these examples that in the comparison of the privacy of dynamic users versus the privacy of static users, all options are possible, depending on the function F.

2.2 Reducing information leakage

To analyse the amount of information leakage, we consider as an example the addition function F and the user sets $U_S = \{1, 2, 4, 5\}$ and $U_D = \{3\}$, with inputs $x_i = i$. Set the size M of the sets T to 2. Given sufficient output runs, an adversary (who knows the online user sets) learns two groups of answers: if user 3 is offline, the six possible outcomes for F are 3, 5, 6, 6, 7, and 9. If user 3 is online, the ten possible outcomes are 3, 4, 5, 5, 6, 6, 7, 7, 8, and 9. Simple linear algebra will reveal that $\{x_1, x_2, x_4, x_5\} = \{1, 2, 4, 5\}$ and $x_3 = 3$. Although the adversary will not be able to distinguish between inputs of different static users, he will learn the values of all user inputs.

Fortunately, we are able to reduce the amount of information that an adversary can derive. In order to achieve this, we are going to fix the random choice of T_r, given the same online user set U_r. To illustrate its effect, we take the same example, where the online user set U is either $\{1, 2, 4, 5\}$ or $\{1, 2, 3, 4, 5\}$. For both options, a random subset will be generated the first time the online user set appear, say $\{2, 5\}$ and $\{1, 3\}$. The next time the same online user set appears, we will not generate a new random subset, but simply pick the one from the previous run. When the subset $T = \{2, 5\}$ is chosen, the output will be fixed to $x_2 + x_5 = 7$, and when the subset T equals $\{1, 3\}$, the output will always be $x_1 + x_3 = 4$. Given these two possible output values, the adversary will not be able to determine all user inputs. Since the adversary knows U, but not T, he will only learn that two inputs from x_1, x_2, x_4, x_5 sum up to 7, and two inputs from x_1, x_2, x_3, x_4, x_5 sum up to 4.

The effects of this new security measure are formally proven in Theorem 1 of the next subsection and are independent of the output function.

2.3 Function independent analysis

In this subsection, we prove the effect of our new security measure and show two basic properties of the random user selection approach, which are independent of the output function. Static users are selected equally likely each round, so intuitively their privacy is equally well protected. This is shown in Proposition 1.

Proposition 1 *Let* $u_1, u_2 \in U_S$, *then* $Privacy_{u_1} = Privacy_{u_2}$.

Proof For clarity, we first show the proof for a single round, so $k = 1$, $U = U_1$, $T = T_1$, and $y = F(\{x_u \mid u \in T\})$. All user inputs are drawn independently

from the same distribution, so $\Pr(x_{u_1}, U) = \Pr(x_{u_2}, U)$. Furthermore, since u_1 and u_2 are both in U, $\Pr(u_1 \in T) = \Pr(u_2 \in T)$, and $\Pr(y \mid x_{u_1}, U) = \Pr(y \mid x_{u_2}, U)$. Combining these shows that the two joint distributions $\Pr(x_{u_1}, y, U)$ and $\Pr(x_{u_2}, y, U)$ are equal. It follows that $\text{Privacy}_{u_1} = H(x_{u_1} \mid y, U) = H(x_{u_2} \mid y, U) = \text{Privacy}_{u_2}$.

This can be easily generalized to arbitrary number of rounds, since the random distributions are identical and independent over the rounds. In particular, $\Pr(x_{u_1}, y_1, \ldots, y_k, U_1, \ldots, U_k) = \Pr(x_{u_2}, y_1, \ldots, y_k, U_1, \ldots, U_k)$, and therefore $\text{Privacy}_{u_1} = H(x_{u_1} \mid y_1, \ldots, y_k, U_1, \ldots, U_k) = H(x_{u_2} \mid y_1, \ldots, y_k, U_1, \ldots, U_k) = \text{Privacy}_{u_2}$. □

This proof does not extend to dynamic users as they are not always jointly online. However, we have a similar proposition about the privacy of dynamic users.

Proposition 2 *Let $u_1, u_2 \in U_D$ be two dynamic users, participating in the exact same rounds: $u_1 \in U_r$, if and only if, $u_2 \in U_r$, for all r, $1 \le r \le k$. Then $\text{Privacy}_{u_1} = \text{Privacy}_{u_2}$.*

Proof The proof is similar to Proposition 1. We still have $\Pr(u_1 \in T_r) = \Pr(u_2 \in T_r)$ for all r. In particular, when $u_1, u_2 \notin U_r$, this probability equals 0. The rest of the proof is identical. □

In Section 2.2, a new security measure was introduced, which consisted of fixing the random subset, given the same online user set. In Theorem 1, this idea is shown to be effective for any type of output function.

Theorem 1 *Let $\text{Privacy}'_u$ be the privacy of user u, when fixing the choice of T, given U. Then, $\text{Privacy}'_u \ge \text{Privacy}_u$.*

Proof We have $\text{Privacy}'_u = H(x_u \mid y'_1, \ldots, y'_k, U_1, \ldots U_k)$, where the outputs y'_r are the result of the modified output computation, which includes fixing the random choice of T_r. Let $\mathcal{R}$ be the set of rounds r, where the choice of T_r has been fixed, because $U_r = U_{r'}$ (and thus $y'_r = y'_{r'}$) for some r', $1 \le r' < r$. Since the values of y'_r and U_r, for $r \in \mathcal{R}$, do not give any additional information, we have $\text{Privacy}'_u = H(x_u \mid (y'_r, U_r), r \notin \mathcal{R})$. For the rounds $r \notin R$, we have $y'_r = y_r$ because the choice of T_r has not been fixed. By increasing the conditional information, we decrease the entropy, so $\text{Privacy}'_u \ge H(x_u \mid (y_r, U_r), 1 \le r \le k) = \text{Privacy}_u$. □

3 Improved random user selection

The basic group service, without using random user selection, is just adding the private inputs of all users and outputting the sum. This can be securely implemented by means of an additively homomorphic threshold decryption scheme like [18], which we denote by [.]. The additively homomorphic property of the scheme enables computing with encrypted values: $[a] \cdot [b] = [a + b]$, where we ignore the modulus operator for convenience.

1. Each available user $u \in U_r$ encrypts his private input x_u and sends $[x_u]$ to the server.
2. The server computes the output $[y] = \left[\sum_{u \in U_r} x_u\right] = \prod_{u \in U_r} [x_u]$.
3. The available users jointly decrypt $[y]$ and open it to the querying user.

Since only outputs are decrypted, the server or any adversary will not learn the inputs. Although Kononchuk et al. did not use an implicit client-server model, a couple of users were assigned as service party, one of these parties in particular denoted by $\mathcal{A}$. The service party $\mathcal{A}$ could be seen as a server. For simplicity, we use the semi-honest security model to describe the protocols in this paper, but they could be extended to the malicious model [7].

In this paper, we do not describe how the basic group service could be extended to a complete privacy-friendly service, by means of homomorphic encryption. Interested readers are, as an example, referred to [5], where a secure recommender system and all its cryptographic protocols are given.

3.1 Previous work

In [7], a number of protocols have been described for randomly choosing subset T_r from the online users U_r. The challenge is to generate the subset, without revealing its elements. This is done by jointly generating $|U_r|$ random bits, each bit indicating the selection (yes or no) of a particular user. To include random user selection in the basic protocol, we need the available users to generate random bits. We illustrate a simple form of random user selection.

1. Each available user $u \in U_r$ generates a vector β^u, containing $|U_r|$ random bits $\beta^u_{u'}$, $u' \in U_r$.
2. The available users jointly compute, for example, by an unbounded fan-in XOR protocol [7], the encrypted random bits $[b_{u'}] = [\oplus_{u \in U_r} \beta^u_{u'}]$, for each $u' \in U_r$.
3. Each available user $u \in U_r$ includes his private input x_u, by computing $[b_u \cdot x_u] = [b_u]^{x_u}$, and sends the result to the server.
4. The server computes the output $[y] = \left[\sum_{u \in U_r} b_u \cdot x_u\right] = \prod_{u \in U_r} [b_u \cdot x_u]$.
5. The available users jointly decrypt $[y]$ and open it to the querying user.

In this random user selection protocol, the online users U_r jointly generated a random subset $T_r = \{u \in U_r \mid b_u = 1\}$, without revealing T_r. Furthermore, the server was able to compute $y = \sum_{u \in T_r} x_u$. In step 1, each user

had to generate random bits, which is commonly done by means of a random number generator and a random seed to initialize the generator.

3.2 Improvement

As shown in Theorem 1 (see Section 2.3), we introduced a new measure to improve user privacy. Therefore, we would like to modify these random bit generating protocols, such that the generated bit vector does not change when for a new execution the same users happen to be online. A straightforward way of assuring this is to jointly store the pair (U_r, y_r) for each run r. When an old value of U_r recurs, the corresponding stored output can be reused.

An alternative way to accomplish this, which avoids storing information from all previous runs, is to fix the seed of the random number generators, used by each user to locally generate random numbers. Let h be a proper hash function, then the following subprotocol is needed to fix these random seeds.

1. Initially, each user u generates a secret key K_u, which is kept private.
2. In the beginning of execution r, the server, party $\mathcal{A}$, learns the set U_r of online users.
3. Using U_r, the server generates a random value s_U, by computing $h(U_r, K_{\mathcal{A}})$, where the ordered elements of the set U_r are concatenated, together with the secret key of service party $\mathcal{A}$.
4. The server sends s_U to all online users, who are represented by the elements of U_r.
5. Each online user $u \in U_r$ generates a random seed $h(s_U, K_u)$, which he uses to generate his local random numbers.

This subprotocol is secure in the semi-honest model and requires all parties to follow the subprotocol. Although the random value s_U is revealed, an adversary will not be able to learn the random seed values because of the private user keys, which are privately generated random numbers. On the other hand, the random seeds are guaranteed to be the same, given the same set U_r of online users. In step 3, the key $K_{\mathcal{A}}$ of the service party is not strictly necessary because an adversary is allowed to learn U_r.

The paper by Kononchuk et al. [7] describes various protocols from the field of secure multiparty computation to jointly generate random bits. As in any secure multiparty protocol, all parties need to locally generate random bits and integers. Our subprotocol described above illustrates a way of securely fixing the seed of the local random number generators, given a certain set of online users. Therefore, when this subprotocol is used in any of the secure multiparty computation solutions from [7], all parties will locally generate the same sequence of random numbers, given a certain set of online users, thereby guaranteeing the same joint function output.

4 Simulations

In Section 2.2, we introduced a new measure to increase the effectiveness of random user selection by fixing the selection for each different online user set. We would like to investigate to what extent this new measure contributes to the protection of user input values.

Our approach is most suited for the class of functions with low sample complexity [13], where the output can be accurately approximated by looking only at a small fraction of the database. Typical examples are linear algebraic functions, with bounded inputs, and histograms [13]. Therefore, we assume the user inputs are presented by small integers and focus on the most common operation: addition. More precisely, we consider the output function that adds up all inputs of the sample.

To analyse the effectiveness of our new measure for large numbers of users and queries, we set up two types of simulations to statistically estimate its effectiveness. The first type does not fix the random selection for each different online set, the second one does.

4.1 Threat model

We have a server that computes an output function F, using the inputs from the available users U_r. As illustrated in Section 3.1, we use cryptographic techniques to draw a random subset T_r of U_r, so the server can compute $y_r = F(T_r)$. A query can be initiated by a user, who also obtains the output y_r.

The goal of an adversary is to learn user inputs, by collecting many outputs. The adversary can be a user, possibly colluding with the server or an external malicious party observing the outputs. We assume the adversary is able to initiate a query any time he likes and will always obtain the output. We also assume the adversary knows U_r (since an adverary can easily observe the online connections), the function F, and the size M of T_r, but not the elements of T_r. We investigate the obtained amount of information on the inputs, as revealed by the outputs.

When the random choice of T_r is not fixed, the adversary will initiate many queries for the same U_r, giving him highly correlated outputs. When we use our security measure, which fixes the choice of T_r, given U_r, the adversary will not learn anything by repeatedly querying the same U_r, so he will look for slightly changed online user sets, which force different choices of T_r.

4.2 Simulation setup

Inspired by a typical recommender system, we simulate in our experiments a group of one million users (N = 1,000,000), which run a group service many times. Each run, ten thousand users are online ($|U_r|$ = 10,000), and a few thousands (M=1,000, 5,000, or 9,000) of them are uniformly randomly chosen to actually participate in the computation of the output function.

The input of each user (x_u) is randomly generated by uniformly drawing an integer from the set $\{1, 2, \ldots, 16\}$. The output function $F(T_r) = \sum_{u \in T_r} x_u$ adds up all inputs. The adversary will try to deduce information on the input of a particular user, say user 1. In particular, he will try to infer whether this input is high, $x_1 > 8$, or low, $x_1 \leq 8$.

Let q be the query parameter. Then, the simulation system will start generating outputs many times, until user 1 has been online at least q times and offline at least another q times, at least $k = 2q$ runs in total. If the output sum for the online q runs exceeds the sum of outputs for the offline q runs, the adversary will guess that $x_1 > 8$; otherwise, his guess will be $x_1 \leq 8$. This is repeated 100 times, and the number of times that the adversary guessed correctly is called the accuracy, presented as a fraction between 0 and 1. Each time after the adversary made his guess, all user inputs are randomly generated anew.

We consider two ways of choosing the set U_r of online users. For each approach, we compare the accuracy of the adversary with the query parameter. The higher the query parameter, the more information is revealed on the inputs, giving the adversary the opportunity to further increase the accuracy of his estimates. Therefore, in our experiments, we generated thousands of different queries and investigated the course of the accuracy of the adversary's estimates. We compare the runs where a particular user was online, with the runs where this user was offline.

Each graph depicts the course of the accuracy of the adversary's estimates (vertical axis) for an increasing query parameter (horizontal axis). The query parameter is increased by steps of 100, and the resulting data points (small crosses) are connected. Each data point is the result of $100 \cdot 2q$ computations of the output function. The straight red line shows the average progress of the adversary's accuracy.

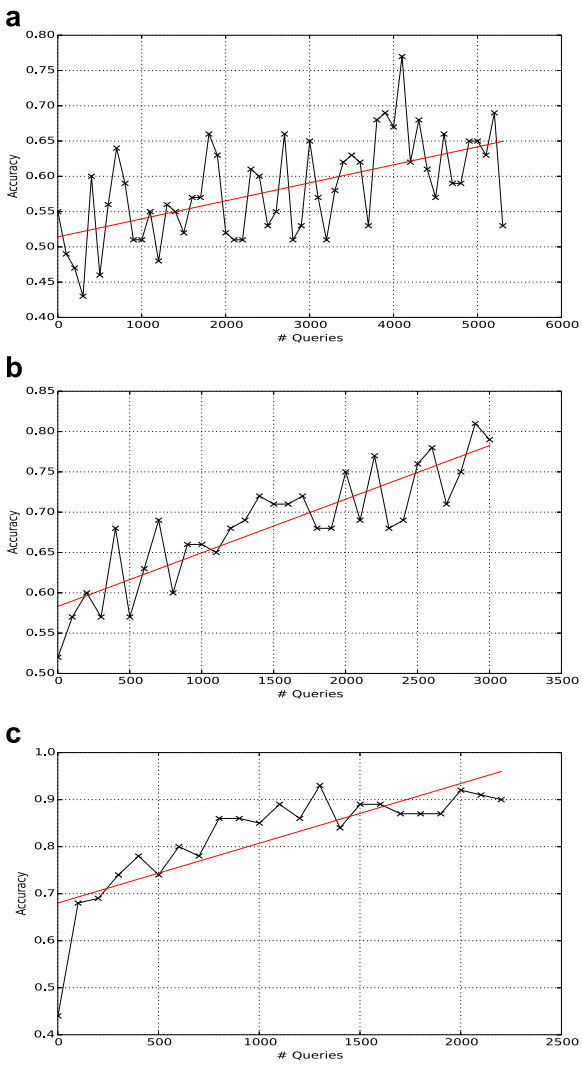

Fig. 1 Sum function. **a** $M = 1000$, **b** $M = 5000$ and **c** $M = 9000$

4.3 No fixing of random choice

In case the random subset choice (given the same online user set) is not fixed, the adversary will try to collect many outputs resulting from the same online user set (see Section 4.1). This will give him the best estimate of x_1, with minimal noise from other inputs. Therefore, in this first type of experiments, we randomly chose U_1 once, and fixed $U_r = U_1$ during all $2q$ runs. We performed this type of experiment with three different sample sizes, namely M = 1000, 5000, and 9000, as depicted in Fig. 1.

The experiments show that an adversary can produce a fair estimate of user input x_1. Of course, the more outputs from different runs are available (larger k), the better the estimate. Also, the more user inputs are sampled (increasing sample size M), the more information is revealed on x_1. This is due to the fact that the sample mean is an estimate of the population mean, and its variance decreases in M. We estimate the mean of x_u for all $u \in U_r$, with a variance of $\frac{\sigma_{\mathcal{X}}^2}{M}$.

4.4 With fixing of random choice

If the random user selection is fixed, each time the same online user set appears, an output resulting from the same online user set will give an adversary no additional information (see Section 4.1). Accordingly, the adversary will be forced to collect outputs from different online user sets. To minimize the noise from other user inputs, he will try to maximize the overlap between online user sets. Therefore, in our second type of experiments, the set U_r is chosen by slightly changing the previous set U_{r-1}. More precisely, either one randomly chosen user, which was online in run $r - 1$, is removed from the online user set

U_{r-1}, or one previously offline user is added. We performed this experiment with three different sample sizes, namely M=1000, 5000 and 9000, as depicted in Fig. 2.

It shows that an adversary is no longer able to estimate the value of x_1. His probability of guessing correctly that $x_1 > 8$ is close to 0.5, irrespective of the number of runs, or the sample size M. The extra noise produced by changing the online user set with only one user, apparently caused a sufficient impediment for the adversary to guessing x_1.

4.5 Highly skewed data

During our experiments, user inputs were drawn uniformly from the set $\{1, 2, \ldots, 16\}$. To investigate whether our results also hold for other distributions, we conducted the same experiments for highly skewed data, where user inputs were drawn by a power law distribution: $\Pr(x_u = x) = C/x^2$, $x \in \{1, 2, \ldots, 16\}$, for some positive constant C. In a power law distribution, low-valued inputs are more likely to occur than high ones. An adversary will try to guess whether a particular user input is below or above the average user input $E(x_u)$.

The results of those experiments were very similar. When the random subset choice is not fixed, the adversary is likely to guess whether the user input was high or low. On the other hand, when this random choice is fixed, his accuracy remains close to 0.5. We conclude that for guessing whether a user input is high or low, it does not matter much how frequent certain user values occur.

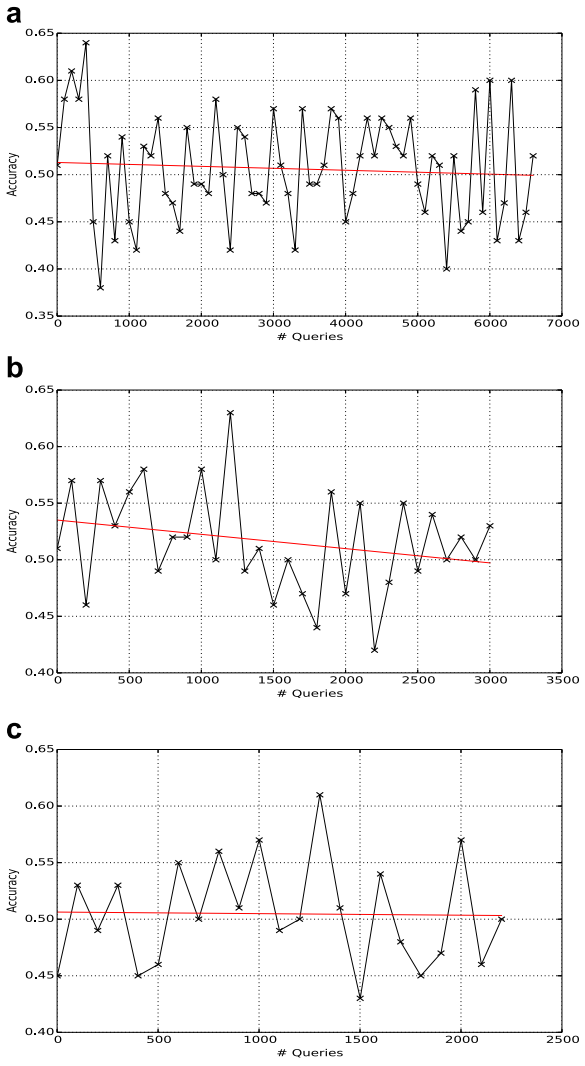

Fig. 2 Sum function, incremental changes in online users. **a** $M = 1000$, **b** $M = 5000$ and **c** $M = 9000$

4.6 Applications

The experiments show that fixing the random user selection sufficiently hides input data because the adversary is forced to collect data samples from different online user sets. Of course, within the given simulation setting (randomly selecting 10,000 users from a set of one million), running into the same online user set is unlikely. However, if user inputs can be revealed within a set of 10,000 users (see Fig. 1), then this will also be possible when fewer users are online. And on the other hand, if changing the online user set with only one person out of 10,000 is sufficient (see Fig. 2), then this will certainly be sufficient for smaller online user sets. Therefore, fixing the random user selection will be a proper solution for most group services in practice, especially when the same online user set is likely to return.

As mentioned before, we consider output functions with low sample complexity. These include unsupervised machine learning [9] and recommender systems [5], where sensitive user inputs need to be protected. For these kind of group services, users tend to return at regular moments in time. For example, people start watching television when their favourite show starts (and they receive recommendations for other interesting programmes) or they join an online game when coming home from work (which initiates a machine-learning-based face recognition algorithm). Therefore, the same online user set is likely to return for these services.

5 Conclusions

In this paper, we had a closer look at the random user selection approach from [7]. We were able to formally prove a number of basic properties of the random selection approach, assuming a symmetric output function. Kononchuk et al. [7] argued that, given a symmetric output function, the privacy of static users is at least as well protected as the privacy of dynamic users, provided the static users form a majority. We disproved this statement, and by investigating four different cases, we showed that

the effectiveness of the random user selection approach strongly depends on the type of output function.

We introduced a new security measure, by fixing the randomly chosen subset, given the same online user set, which was proven to lead to an improved protection of privacy-sensitive user inputs, irrespective of the output function. We developed a cryptographic subprotocol, which can be used to implement our new security measure, given an arbitrary random user selection protocol.

Finally, we performed a couple of statistical simulation experiments to investigate the effectiveness of our measure for the addition output function. Without the measure, an adversary was able to infer information on specific user inputs. The accuracy of the adversary's estimate increased significantly, when more outputs were revealed, or more inputs were sampled. On the other hand, when the randomly chosen subset was fixed, given the same online user set, the adversary remained ignorant of the user inputs, even when thousands of outputs were revealed, and many inputs were sampled. Therefore, our measure seems to be the key ingredient for making the random user selection approach an effective way of protecting user privacy in dynamic group services with low sample complexity.

Acknowledgements
This work is supported by the Dutch COMMIT programme. We would like to thank Sebastiaan de Hoogh for his valuable contributions while developing our paper.

Authors' contributions
TV is the main author of the paper, led the discussions, and, amongst other contributions, developed the cryptographic protocol of Section 3. JD made the examples in Section 2.2, and did part of Section 2.3. ZE wrote the introduction, reviewed the paper, and participated in the discussions. GP performed the simulation experiments from Section 4, which where designed and explained by SV. Finally, JW computed the examples in Section 2.1 and found the first counterexample. All authors read and approved the final manuscript.

Competing interests
The authors declare that they have no competing interests.

Author details
[1]Cyber Security research group, Delft University of Technology, Mekelweg 2, 2628 CD, Delft, The Netherlands. [2]TNO, Technical Sciences, P.O. Box 96800, 2509 JE, The Hague, The Netherlands. [3]Irdeto, High Tech Campus 84, 5656 AG, Eindhoven, The Netherlands.

Received: 13 July 2016 Accepted: 12 January 2017
Published online: 01 February 2017

References

1. RL Lagendijk, Z Erkin, M Barni, Encrypted signal processing for privacy protection. IEEE Signal Process. Mag. **30**(1), 82–105 (2013)
2. T Ignatenko, F Willems, Biometric systems: privacy and secrecy aspects. IEEE Trans. Inform. Forensics Secur. **4**(4), 956–973 (2009)
3. M Blanton, P Gasti, in *Computer Security* (ESORICS) Lecture Notes in Computer Science, 6879. Secure and efficient protocols for iris and fingerprint identification (Springer, Berlin, 2011)
4. Z Erkin, JR Troncoso-Pastoriza, RL Lagendijk, F Perez-Gonzalez, Privacy-preserving data aggregation in smart metering systems: an overview. IEEE Signal Process. Mag. **30**(2), 75–86 (2013)
5. Z Erkin, T Veugen, T Toft, RL Lagendijk, Generating private recommendations efficiently using homomorphic encryption and data packing. IEEE Trans. Inform. Forensics Secur. **7**(3), 1053–1066 (2012)
6. M Naveed, E Ayday, EW Clayton, J Fellay, CA Gunter, JP Hubaux, BA Malin, X Wang, Privacy in the genomic era. ACM Comput. Surv. **48**(1) (2015). doi: 10.1145/2767007
7. D Kononchuk, Z Erkin, J van der Lubbe, RL Lagendijk, in *Computer Security* (ESORICS) Lecture Notes in Computer Science, 8134, ed. by J Crampton, S Jajodia, and K Mayes. Privacy-preserving user data oriented services for groups with dynamic participation (Springer, Berlin, 2013)
8. J Kacprzyk, Group decision making with a fuzzy linguistic majority. Fuzzy Sets Syst. **18**, 105–118 (1986)
9. F Anselmi, JZ Leibo, L Rosasco, J Mutch, A Tachetti, T Poggio, Unsupervised learning of invariant representations with low sample complexity: the magic of sensory cortex or a new framework for machine learning? J. Theor. Comput. Sci. **633**(C), 112–121 (2016). CBMM, MIT, arXiv:1311.4158v5
10. A Mathes, Folksonomies—cooperative classification and communication through shared metadata. Comput. Mediated Commun. **47**, 1–28 (2004). nr. 10
11. Z Erkin, T Veugen, T Toft, RL Lagendijk, in *IEEE International Workshop on Information Forensics and Security*. Privacy-preserving user clustering in a social network, (IEEE, Washington, 2009)
12. H Kargupta, S Datta, Q Wang, K Sivakumar, in *ICDM, IEEE Computer Society*. On the privacy preserving properties of random data perturbation techniques (IEEE, Washington, 2003), pp. 99–106
13. C Dwork, F McSherry, K Nissim, A Smith, in *TCC 2006 Lecture Notes in Computer Science*. Calibrating noise to sensitivity in private data analysis, vol. 3876 (Springer, Berlin, 2006), pp. 265–284
14. B Zhou, J Pei, WS Luk, A brief survey on anonymization techniques for privacy preserving publishing of social network data. SIGKDD Explorations. **10**, 12–22 (2008)
15. AC Yao, in *Proceedings of the 23rd Annual IEEE Symposium on Foundations of Computer Science*. Protocols for secure computations (IEEE Computer Society, Washington, DC, 1982), pp. 160–164
16. BCM Fung, K Wang, A Wai-Chee Fu, J Pei, in *EDBT'08 March 25–30*. Anonymity for continuous data publishing, (Nantes, 2008)
17. M Fredrikson, S Jha, T Ristenpart, in *CCS'15*. Model inversion attacks that exploit confidence information and basic countermeasures (ACM, New York, 2015)
18. T Nishide, K Sakurai, in *WISA 2010*, ed. by Y Chung, M Yung. Distributed paillier cryptosystem without trusted dealer. LNCS, vol. 6513 (Springer, Heidelberg, 2011), pp. 44–60