WeStat: a Privacy-Preserving Mobile Data Usage Statistics System

Sébastien Canard*
Nicolas Desmoulins
Sébastien Hallay
Adel Hamdi
Dominique Le Hello
Orange Labs – Applied Crypto Group
Caen Cedex 4, France

ABSTRACT

The preponderance of smart devices, such as smartphones, has boosted the development and use of mobile applications (apps) in the recent years. This prevalence induces a large volume of mobile app usage data. The analysis of such information could lead to a better understanding of users' behaviours in using the apps they have installed, even more if these data can be coupled with a given context (location, time, date, sociological data…). However, mobile and apps usage data are very sensitive, and are today considered as personal. Their collection and use pose serious concerns associated with individuals' privacy. To reconcile harnessing of data and privacy of users, we investigate in this paper the possibility to conduct privacy-preserving mobile data usage statistics that will prevent any inference or re-identification risks. The key idea is for each user to encrypt their (private and sensitive) inputs before sending them to the data processor. The possibility to perform statistics on those data is then possible thanks to the use of functional encryption, a cryptographic building block permitting to perform some allowed operations over encrypted data. In this paper, we first show how it is possible to obtain such individuals' usage of their apps, which step is necessary for our use case, but can at the same time pose some security problems w.r.t. those apps. We then design our new encryption scheme, adding some fault tolerance property to a recent dynamic decentralized function encryption scheme. We finally show how we have implemented all that, and give some benchmarks.

CCS CONCEPTS

• **Security and privacy → Privacy-preserving protocols**.

KEYWORDS

Cryptography, functional encryption, privacy, mobile apps

IWSPA'21, April 28, 2021, Virtual Event, USA

ACM ISBN 978-1-4503-8320-2/21/04…$15.00
https://doi.org/10.1145/3445970.3451151

ACM Reference Format:
Sébastien Canard, Nicolas Desmoulins, Sébastien Hallay, Adel Hamdi, and Dominique Le Hello. 2021. WeStat: a Privacy-Preserving Mobile Data Usage Statistics System. In *Proceedings of the 2021 ACM International Workshop on Security and Privacy Analytics (IWSPA'21), April 28, 2021, Virtual Event, USA.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3445970.3451151

1 INTRODUCTION

Our daily life is nowadays widely punctuated by the use we make of our smartphones and the applications (*apps* in short) we have installed on it. Such ubiquity of mobile usage empowers users to access information, whenever they want and wherever they are, and impacts the society by modifying everyday life habits. Gaming, finance, retail, streaming, social networks, IoT… Mobile and their apps have become the "central nervous system of our connected lives", as explained and proved by the well-known App Annie's "State of the Mobile in 2020" report (https://www.appannie.com/en/go/state-of-mobile-2020/). This latter has indeed recorded in 2019 204 billions of downloaded apps, 3.7 hours per day spent in mobile by the average user, 150 sessions per app per month per user among top 25 non-gaming apps on average…

This explosion generates a volume of mobile and app usage data that is getting more and more impressive. According to the recent Ericsson Mobility Report (https://www.ericsson.com/en/mobility-report), the average smartphone user churns through 1.4 GB of data every month, and this figure is expected to reach 8.9 GB in 2021!

The knowledge and analysis of these data is at the core of many competitive sectors. In particular, this could permit to better understand users' behaviours in using their smartphone and the installed apps. Besides, if these data are coupled with a given context (location, time, date, sociological data…), the insights resulting from these analytics could be very meaningful and valuable for a wide range of stakeholders, ranging from city services to retail stores, not to mention application developers or sociologists. It can be very useful to obtain the carbon emission of e.g., social networks apps, or to know when games apps are more often used, depending on e.g., the age range. Such kind of study can also interest individuals so that they compare their own consumption with that of other individuals with a similar profile.

However, mobile and app usage data are very sensitive, and today considered as personal data. The collection and use of these data pose serious concerns associated with individuals' privacy. A few of recent examples of privacy breaches with respect to app usage

shed light on how sensitive these data can be. For instance, some of the documents provided by Edward Snowden in 2014 revealed that the NSA has been using the mobile game Angry Birds to collect users' personal data such as age, gender and location. In April 2018, the Norwegian research institute SINTEF reported that Grindr, a gay dating application, was sharing sensitive data such as HIV status (that the users disclose – or not – in their public profile) or locations to two third-party companies. In early 2018, the Facebook-Cambridge Analytica data scandal hit the headline, when it was revealed that the consulting firm has harvested Facebook's data (given by users who had signed up for an application named "This is Your Digital Life") to profile US voters. Inside Europe, the recent introduction of the General Data Protection Regulation (GDPR) has drastically changed the way the data industry can use personal data, having to explain how they proceed to protect it and reduce the re-identification risks.

In this paper, we propose the first privacy-preserving mobile data usage statistics that prevents any inference or re-identification of individuals. Let us first detail our use case.

1.1 Use Case Description

We consider for example a team of social scientists that are interested in having a better knowledge of the habits of users of mobile phones and in particular to study the impact on the population of the usage of e.g., social network applications. Traditionally, this kind of study is conducted through tools such as face-to-face interviews, diaries, questionnaires or surveys. Participants would be asked to self-report the frequency or the duration of their use of social networks apps via questions such as "How frequently do you open your social networks application? Less than a month; once a month; once a week; etc.". This approach, while being widely spread in the social science research community, suffers from several drawbacks. First, participants' answers to the questionnaire may be biased by several factors: subjectivity (people may underestimate or overestimate their use of social networks), human memory limitations (it is hard to remember when or how often they open the application) or willingness (for fear of other's eyes). Secondly, as a consequence of the first drawback, questions must be thoughtfully chosen and asked in order to avoid biases. Additionally, questionnaires and surveys only cover a small sample of the population. All these flaws of conventional tools for social science studies lead to ineffective implementation (right questions to ask), poor representativeness (small sample) and inaccurate results (biased answers).

Therefore, the aforementioned limitations influence our team of social scientists to study behaviours of social networks users at source, based on their mobile usage data, that consist of timestamps of application opening and closing, actions performed in the application (such as "like" button), configuration parameters, etc. Collection and analytics of such data present several advantages compared to traditional tools: mobile usage data analytics deal with objective data that cannot be biased, cover a bigger and more scalable sample and result in more fine-grained and more accurate analytics. Besides, the usage data can be enriched with additional data such as phone data (OS, used network, i.e. 3G, 4G or Wi-Fi, ...) and demographic data (age, size of family, ...) to obtain even more fine-grained insights. Hence mobile usage data analytics can either be used in combination with traditional tools (surveys, questionnaires) or even potentially supersede questionnaire-based methods.

The idea is then to provide a new independent service, called WeStat, for privacy-preserving analytics on mobile usage data collected from registered users. To perform such a collection, the Service Provider, called Aggregator in the sequel, provides to users a dedicated WeStat smartphone app to be installed on their mobile phones, which will take care of the collection of data coming from several other apps. Besides, users can be encouraged to participate to the study since they will be able to compare their own consumption with the ones of similar people, or by receiving some rewards (discounts, vouchers, etc.).

We moreover request to restrict the number of interactions between all actors. Indeed, we first consider that a solution based on the active participation of individuals is not enough sustainable, based on the additional fact that individuals do not know each other: the main operations should be managed by the Aggregator solely. Furthermore, we also prefer to lower the role of the Third Party in the different steps, since considering that the latter requests a service from the former, and does not necessarily want to be very active. The Third Party should then only participate, in a non-interactive way, in (i) the creation of the study and (ii) the computation of the final statistics.

In this context, the team of social scientists contacts (playing the role of a Third Party in our system) the Aggregator (as the service provider) with a specific analytics request. They will request some statistics (counting, average duration, linear regression...) on one or a set of apps are used for a given period of time. The analytics requests specify the attributes on which they would like the analytics operations to be performed (for instance, age, day of the week, month, duration of use etc.) as well as the operation to be performed on the data. In the sequel, we will consider the average carbon emission of social networks apps (Instagram, Facebook, WhatsApp...), correlated with the age and place of living. The Aggregator then performs the requested analytics on the individual's aggregated data collected during the specified observation period and sends back the analytics results to the research team. Having access to the result, the research team can derive social conclusions about social networks usage.

1.2 Privacy Constraints

As the usage data could be highly sensitive in terms of users' privacy, the WeStat system should include mechanisms for privacy protection, in particular in accordance with the GDPR (https://ec.europa.eu/info/law/law-topic/data-protection/). Consent is a first key condition to collect and process users' data (Articles 6, 7 and 8 of the GDPR). Therefore, before each new study, users are requested to give their (informed and explicit) consent on the collection and processing of their data by the Aggregator. They should agree on (i) the attributes that will be collected from them; (ii) the duration of the collection; (iii) the purpose of the processing (the requested analytics) that will be performed on the collected data and (iv) the period of storage of their data. The users are also empowered with the ability to specify privacy preferences on collection and usage of their data.

Furthermore, to fulfil the principle of privacy-by-design requested by the GDPR (Articles 25 and 32), the present use case should leverage technical solutions for data protection. The main purpose for those technical mechanisms is to reduce the risk of re-identification of any individual's data. The main constraints we need to manage are the following ones:

- the Aggregator nor the Third Party should obtain any individual's personal data;
- only the Third Party should obtain the final result (even if it can finally decide to publish such statistics, or some derivative conclusions);
- any individual can participate, but one individual does not know, at the time she is sending her data, who are the other participants. In particular, an individual may first claim that she wants to participate, and then decide not to send her data.

Pseudonymisation and anonymization techniques, such as k-anonymity [13] or differential privacy [8] are potential solutions. In fact, the former seems hard to use in our context since we need that one entity aggregates the data coming from several independent users. In this case, the value k seems hard to manage upstream. The latter seems more suitable, assuming that each individual adds some differential private noise to her input before sending it to the Aggregator. As we do not know in advance who will participate, it may be hard to precisely calibrate such noise, since the sensitivity usually depends on the whole set of data. But this is still feasible, using so called *local differential privacy*, which may need to add substantially more noise than necessary [11]. In fact, the solution we propose can quite easily be plugged to some additional pre-defined differential private mechanisms.

Our solution is to make use of encryption techniques directly on the individual's side. For the GDPR's point of view, this corresponds to a pseudonymization technique, with the difference that in case of private data compromise, the data controller does not have to inform the data subject, but only the authority. Let us now take a look on the different possibilities one can find on the literature.

1.3 Overview of our Solution

Our solution can in fact be summarized as follows: before the usage data leave the users' mobile phone, they will be encrypted using some specific encryption key defined for the study. Using a suitable encryption scheme, the Aggregator is then capable to perform the requested analytical operation on the encrypted collected data, hence without having a plain access to those data. The encrypted results of this operation are then given to the Third Party which can obtain it in clear. In this paper, we will mainly focus on specific simple analytics operations, namely simple summary statistics such as computing the mean. More precisely, we will consider the inner product operation. Our method can be generalized to more complex operations, but with less efficiency.

In a nutshell, our solution is first an adaptation of the DSum technique given in [5], which one is a special case of dynamic decentralized multi-client Functional Encryption [6, 10]. We then combine such technique with some ideas taken from fault-tolerant Private Stream Aggregation [4] to manage the fact that we do not know in advance who will participate. But this additional property obliges us to perform some adaptations in the DSum construction given in [5]. In particular, the All-or-Nothing encapsulation technique needs in our case to be used twice. We finally add a new trick in the resulting construction, which consists in adding some randomness on the Third Party side, in order to prevent the Aggregator from learning partial results.

1.4 Organization of the Paper

We now organize the paper as follows. In Section 2, we recall the main elements of the studied use case, and then give the main security requirements. Section 3 recalls the main cryptographic components that we need and details our solution, at first with the cryptographic algorithms, and then with the interactions between all users. In Section 4, before concluding, we furnish all the details of our real implementation of the whole system. In particular, we explain how we have obtained the usage statistic (how long such app has been used, how many times per day, what is the used configuration ...) related to any smartphone app that is relevant for the Third Party study. For this part of our work, which may be of independent interest, we haven't used any security flaw, but only legal and available tools.

2 USE CASE DESCRIPTION AND SECURITY REQUIREMENTS

In this section, we start by formalizing more the use case we are considering in this paper. We then give the desired requirements in terms on security and privacy. This finally permits us to detail the main requirements for our cryptographic tool.

2.1 High-Level Use Case Description

2.1.1 Actors. We consider a system, called WeStat, with three types of actors:

- a set of individuals that after having installed the WeStat app, will obtain a notification for each new study. After having giving their consent in participating in the study, the WeStat app will aggregate and send encrypted data to the Aggregator. Individuals will finally obtain the result of the study so as to be able to compare it with their own consumption, in a privacy-preserving way. They can also obtain some rewards for their participation;
- an Aggregator that can obtain individuals' encrypted data so as to perform the statistics in a privacy-preserving manner. It makes the interface with Third Parties to prepare the study and to permit them to obtain the results;
- Third Parties that can contact the Aggregator for a new study, filling some on-line form with the name of the study, the start and end dates, the targeted apps (social networks, communication, games ...), the type of measure (counting, duration, carbon emission...), the type of statistics (mean, count...), and the Requested profiles (gender, age, place of living, job...). They finally obtain the diagrams giving the results of their study.

2.1.2 Additional requirements. For efficiency reasons, we only consider statistics that can be managed using an inner product between a vector where each component is the entry of one individual and

another vector related to the study. For example, the sum (resp. mean) of all the entries can be computed with vector $(1, \cdots, 1)$ (resp. $(1, \cdots, 1)$ divided by the number of entries). More complex statistics can also be computed using inner product, such as e.g. linear regression or data classification.

Another important issue is that an individual may plan to participate but then decide not (or fail) to. This is quite natural and should not pose any problem regarding both the final result and the privacy of this individual (and the one of the others).

2.1.3 Privacy issues. This use case raises several privacy concerns. Individuals are requested to give their consent to the collection and processing of their data for a specific study, hence a specific statistical analysis. In case consent is given, as collected information is privacy-sensitive, they should be protected with regard to the current legislation. The used encryption algorithm should both provide data confidentiality and allow the Aggregator to derive the above statistics on the encrypted aggregated data. Hence, one of the privacy requirements is that the possibility to decrypt any single user's data should be infeasible. Similarly, it must be impossible for the Aggregator to compute the analytics operations on a dataset originating from a single user; otherwise, this user will straightforwardly be re-identified. This implies that analytics must not be performed before multi-source data are aggregated. Finally, as consent is given for one study or one data processing, it must be infeasible to perform any other analytics on the data that have been collected for other purposes.

2.1.4 Cryptographic requirements. We now focus on the cryptographic scheme. From all the above points, we can conclude that:

- each individual sends one message in a complete decentralized fashion without knowing in advance which other users are going to participate;
- the Aggregator should not learn any additional information about individuals;
- the Third Party is the only entity that gets the final result;
- the cryptographic scheme should be fault-tolerant, i.e. if one user fails, the final result should only be considered over the remaining users and no partial information should be given.

2.2 More Formal Description

2.2.1 Procedures. We now define a more formal view of our need, considering that the WeStat system has been initialized by the Aggregator, which outputs some parameters param.

- The registration to the service is executed by any individual i. It permits her to generate a secret key sk_i and to send some related public parameters pk_i to the Aggregator.
- The new study creation permits a Third Party and the Aggregator to create a new study, labelled as ℓ. The Third Party outputs a secret key sk_ℓ and a related public parameter pk_ℓ is generated by the interactions between the Third Party and the Aggregator.
- The data sending for study labelled ℓ is executed by each individual i, on input sk_i, pk_i and pk_ℓ. It outputs a ciphertext $c_{i,\ell}$ from the plain data $a_{i,\ell}$ (corresponding to this individual's contribution to the study).
- The result computation is seen as a protocol between the Third Party and the Aggregator. The former takes as input sk_ℓ and pk_ℓ, and the latter the public parameters pk_ℓ and the set $(c_{i_0,\ell}, c_{i_1,\ell}, \cdots)$ of all ciphertexts outputs by the participating individuals. It permits the Third Party to output the result.

2.2.2 Security. From all the above discussion, our main purpose regarding security is to provide indistinguishability w.r.t. individuals' data a_i. There are three main entities (users, Aggregator and Third Party) that we may consider as corrupted to break such security property, trying to learn more information than authorized (honest-but-curious).

It follows that we may considered compromised users, given to the adversary some auxiliary information about their own inputs and/or their secret keys. In particular, a wishful security property that we handle is to guarantee that the only leaked information to the adversary is the sum of the non-compromised users. On this aspect, we follow the main idea of the security model given by Chan et al. [4].

We can then consider a corrupted Third Party that may have corrupted some users. Having access to the final result, it is obviously able to learn some intermediate result, and in particular the sum of the contributions of non-compromised users. But we here argue that this should not be more than that. In particular, there is no possibility for such adversary to single out the entry of one non-compromised user.

A similar conclusion can be given for a corrupted Aggregator, except that in this case, the sum of non-compromised users can only be retrieved assuming that the final result is finally published by the Third Party.

Let us now consider an adversary having compromised both the Aggregator and the Third Party, which is the strongest possible security model. It seems theoretically feasible to design such a secure scheme. Assuming that non-compromised users give their consent for one specific data treatment, we can certainly find solution in which they can detect that the Aggregator and the Third Party are trying to compute something else. But in fact, we haven't seen any course of action that permits to efficiently solve our problem. More precisely, such design seems possible using some multi-party computation system in which the complete set of users participate: but this is not our setting, as we specifically assume that users cannot directly participate to the whole data treatment. Another possibility may be to use some Non-Interactive Zero-Knowledge proof of knowledge to give the possibility for Aggregator and the Third Party to prove that they have behaved correctly. This is definitely not enough efficient. As a consequence, we do not consider the case of such coalition in the sequel.

3 OUR CONSTRUCTION

We now our solution to the above problem, starting from an overview before going into more details.

3.1 Overview of our Solution

The starting point of our construction is the *binary tree* idea of Chan et al. [4] that we modify according to our needs. First, we split users in different groups. Then, each user is associated to one leaf in the tree, and is related to all the nodes from her leaf to the

root. We then run a DSum protocol for each group/node, using the construction given in [5]. If some users fail, we are then able to find a set of subgroups that can cover the participating users. This gives us the fault-tolerant property. More precisely,

- each user i considers the set $\mathcal{N}$ of nodes where it appears and generates a secret key $\mathsf{sk}_{i,\mathcal{N}}$ for each node $\mathcal{N}$, using the KeyGen algorithm of the DSum scheme;
- using several times the encryption algorithm of the DSum scheme on input her contribution a_i and the secret key $\mathsf{sk}_{i,\mathcal{N}}$, user i generates as many ciphertexts as the number of nodes in which she appears;
- after having received the contribution of all participating users, the Aggregator has to find a set of blocks that cover all of them. This next permits it to obtain first all partial sums, and then the whole result, which is finally sent to the Third Party;
- considering that the Third Party has previously (during the creation of the study) participated to the protocol with some randomness, playing as an extra user (belonging to all leaves in the tree), the resulting sum is noised with some global randomness that it can remove after having received the result from the Aggregator.

3.2 Building Blocks

As explained above, the core of our cryptographic system is the DSum functional encryption scheme given in [5]. Such construction makes use of a so-called NIKE (for Non-Interactive Key Exchange) [9] together with a new concept they have called All-or-Nothing Encapsulation. For the former, we make use of the construction in the standard model from pairings given in [9], which itself uses the chameleon signature based on discrete logarithm scheme given in [12].

3.2.1 DL Based Chameleon Hashing. In a nutshell, a chameleon hashing corresponds to a collision-resistant algebraic hash function with a trapdoor for finding collisions. In their paper [12], Krawczyk and Rabin have introduced such concept of (together with the one of chameleon signature schemes) and have proposed several constructions. We here focus on the discrete logarithm based one, as it corresponds to the setting in which we fall for our general construction.

Let p and q be prime numbers such that $p = kq+1$ for some k. Let g of order q in $\mathbb{Z}_p^*$. The secret key to find trapdoors is $\mathsf{ck} \in \mathbb{Z}_q^*$ and the corresponding hashing public key is $\mathsf{hk} = g^{\mathsf{ck}} \pmod p$. Given a message $m \in \mathbb{Z}_q^*$, the hashing procedure first chooses at random $r \in \mathbb{Z}_q^*$ and computes $h = g^m \mathsf{hk}^r \pmod p$. Using the collision secret key ck, one can find a collision, that is for a given message m', a value r' such that $g^m \mathsf{hk}^r = g^{m'} \mathsf{hk}^{r'} \pmod p$. This can be done by solving the equation $m + xr = m' + xr' \pmod q$.

Note that in the NIKE construction we use [9], the collision secret key ck is not used in the construction but rather to prove the security of the scheme. See [9] for details.

3.2.2 Non-Interactive Key Exchange. The notion of Non-Interactive Key Exchange (NIKE) has been introduced in [9]. It corresponds to a public-key cryptographic primitive which enables two parties to agree on a symmetric shared key without requiring any interaction. Each party owns a key pair $(\mathsf{sk}_i, \mathsf{pk}_i)$ and is able to compute the shared key by using her private key sk_1 (resp. sk_2) and the public key pk_2 (resp. pk_1) or the other party.

For our main construction, we make use of the pairing based construction, still given in [9]. Let $(\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, p, g_1, g_2, e)$ be a so-called bilinear environment, where $\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T$ are groups of order prime p, $g_1 \in \mathbb{G}_1$ and $g_2 \in \mathbb{G}_2$ and e is a bilinear map. Let $u, u_1, u_2 \in \mathbb{G}_1^*$ and let hk, ck be as defined above for the chameleon hash function (the key ck is only necessary for the security proof, and then not used in the below description). All those values compose the parameters of the NIKE scheme.

The key generation phase, executed by each party i then consists in choosing at random $x_i \in \mathbb{Z}_p$ and $r_i \in \mathbb{Z}_q^*$ (as in the chameleon hash function above), then computing $Z_i = g_2^{x_i}$, $t = g^{\mathcal{H}_2(Z_i)} \mathsf{hk}^{r_i} \pmod p$ (a chameleon hash), $Y_i = u_0 u_1^{t_i} u_2^{t_i^2}$ and $X_i = Y_i^{x_i}$. The public key pk_i is then (X_i, Z_i, r_i) and the private key sk_i is x_i.

Note that we have modified the description given in [9]. We have first removed the identity part and we have then replaced Z by $\mathcal{H}_2(Z)$ during the computation of t, where $\mathcal{H}_2 : \{0,1\}^* \to \mathbb{Z}_q^*$ is a hash function. This is due to the fact that when considering a practical instance for the NIKE scheme, using the above chameleon hash, Z should be in $\mathbb{G}_2$ while the value to be chameleon hashed should be in $\mathbb{Z}_q^*$, as explained in [12].

Finally, the computation of the shared key $K_{1,2}$ is done as follows: using a public key pk_1 and a private key sk_2. It computes $t_1 = g^{\mathcal{H}_2(Z_1)} \mathsf{hk}^{r_1} \pmod p$ and generates the key $K_{1,2} = e(S^{x_2}, Z_1)$ iff $e(X_1, g_2) = e(u_0 u_1^{t_1} u_2^{t_1^2}, Z_1)$.

3.2.3 All-or-Nothing Encapsulation. All-or-Nothing Encapsulation (AoNE) [5] allows several parties of a group to encapsulate individual messages, that can all be extracted by anybody if and only if all the parties of this group have sent their contributions. We here make use of the bilinear map based construction in [5], which is not repeated here, due to space restrictions.

3.3 DSum Functional Encryption Instantiation

We first give the details on the used DSum functional encryption scheme given in [5]. In fact, Chotard et al. only give a generic construction from a NIKE and their newly introduced concept of All-or-Nothing Encapsulation.

We consider a bilinear environment $(\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, p, g_1, g_2, e)$ where $\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T$ are groups of order prime p and $g_1 \in \mathbb{G}_1$ and $g_2 \in \mathbb{G}_2$. Let q a prime number and k an integer such that $p = kq + 1$ for some k. Let g of order q in $\mathbb{Z}_p^*$. Finally, let $\mathcal{H}_1 : \{0,1\}^* \to \mathbb{G}_1$ and $\mathcal{H}_2 : \{0,1\}^* \to \mathbb{Z}_q^*$ be two hash functions.

- Setup. During the setup, one has to generate $u_0, u_1, u_2, S \in \mathbb{G}_1$, generate $\mathsf{ck} \in \mathbb{Z}_q^*$ and compute $\mathsf{hk} = g^{\mathsf{ck}} \pmod p$. The parameters param is then defined as $(u_0, u_1, u_2, S, \mathsf{hk})$, together with the bilinear environment $(\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, p, g_1, g_2, e)$ and the two has functions $\mathcal{H}_1$ and $\mathcal{H}_2$.
- KeyGen. We next describe the KeyGen procedure, which contains the following steps for each user i, on input param:
  - choose at random $x_i \in \mathbb{Z}_p^*$ and $r_i \in \mathbb{Z}_q^*$;
  - compute $Z_i = g_2^{x_i}$ and $t_i = g^{H_2(Z_i)} \mathsf{hk}^{r_i} \pmod p$;
  - compute $Y_i = u_0 u_1^{t_i} u_2^{t_i^2}$ and $X_i = Y_i^{x_i}$;

– choose at random $v_i \in \mathbb{Z}_p^*$ and compute $T_i = g_2^{v_i}$.
The i-th public key is $\mathsf{pk}_i = (X_i, Z_i, r_i, T_i)$ and the corresponding private key is $\mathsf{sk}_i = (x_i, v_i)$.

- Encrypt. We now focus on the encryption procedure Encrypt, executed by a user i on input the set of public keys $\mathcal{PK} = \{\mathsf{pk}_i\}_i$, a input a_i and a label ℓ. Even if such method can be described in general, we need below to encrypt a message a_i defined as a vector of length L. We then denote by $a_i[k]$ the k-th component of vector a_i (similar notation is used for other vectors).
The encryption procedure is denoted Encrypt and proceeds as follows:
– $\forall j \in \mathcal{PK}, j \neq i$, compute
  * $t_j = g^{\mathcal{H}_2(Z_j)}\mathsf{hk}^{r_j}$;
  * $K_{i,j} = e(S^{x_i}, Z_j)$ iff $e(X_j, g_2) = e(u_0 u_1^{t_j} u_2^{t_j^2}, Z_j)$;
  * $r_{i,j}[k] = \mathsf{PRF\text{-}SHA256}(K_{i,j}, \mathcal{PK}\|\ell\|k)$ for all $k \in [0, L[$;
– compute $c_i = a_i + \sum_{j<i} r_{i,j} - \sum_{j>i} r_{i,j}$ (in the L-length vector space, c_i being then a vector of length L);
– choose $w_i \in \mathbb{Z}_p^*$ and compute $W_i = g_2^{w_i}$;
– compute $K_i = e(\mathcal{H}_1(\mathcal{PK}\|\ell), (\prod_j T_j)^{w_i})$ and the ciphertext $C_i = \mathsf{AES}(K_i, c_i)$;
– compute $S_i = \mathcal{H}_1(\mathcal{PK}\|\ell)^{w_i}$.
The ciphertext is finally $\mathsf{ct}_i = (C_i, W_i, S_i, \mathcal{PK}, \ell)$.
- Decrypt. We finally give the details of the decryption procedure Decrypt, which takes as input a set of ciphertexts $C = \{\mathsf{ct}_i\}_i$. It works as follows:
– $\forall i \in \mathcal{PK}$, compute $K_i = e(\prod_j S_j, W_i)$;
– $\forall i \in \mathcal{PK}$, compute $c_i = \mathsf{AES}^{-1}(K_i, C_i)$;
– compute the result $R = \sum_i c_i$.
As the initial encrypted message is given by a vector, this last step is performed component-wise.

3.4 The WeStat System

We are now ready to describe our main construction.

3.4.1 Main Setup. We consider that there are at most $N = 2^n$ users in the system. The idea is then to manage a binary tree of height $n+1$, for which each node is given a set of cryptographic keys. The used nomenclature is as follows (see an example in Figure 1):

- each leaf is represented by a unique number from 1 (extreme left) to N (extreme right);
- each node is represented by "$i\|j$" where i is the extreme left number of node's left son, and j is the extreme right number of node's right son. For example, a node with sons "1‖4" (on the left) and "5‖8" (on the right) is denoted "1‖8".

We then consider that each leaf of the tree represents a unique user and each user will next be related to the nodes from the root to its leaf. For example, user 1 is related to the nodes represented by the set $\{1, 1\|2, 1\|4, \cdots, 1\|N\}$.

This tree is managed and maintained by the Aggregator and is common to all studies. The Aggregator also executes the general Setup procedure of the encryption scheme (see Section 3.3), which outputs param. All the details on the tree are also put on this set of parameters.

3.4.2 Registration to the Service. We now consider that user i wants to register to the service. The Aggregator sends him the parameter param and associates this new user to a particular leaf on the tree. User i executes the KeyGen(param) = (pk, sk) for each node in the tree in which it is involved (e.g., nodes $\{1, 1\|2, 1\|4, \cdots, 1\|N\}$ for user 1), as described in Section 3.3. He then obtains $n+1$ key pairs. For example, for user 1, those keys are denoted

$$\{(\mathsf{pk}_{1,1}, \mathsf{sk}_{1,1}), (\mathsf{pk}_{1\|2,1}, \mathsf{sk}_{1\|2,1}), \cdots, (\mathsf{pk}_{1\|N,1}, \mathsf{sk}_{1\|N,1})\}.$$

There are then 1 key related to each leaf at level $n+1$, 2 keys related to each node at level n, 3 keys at level $n-1$, $\cdots$, and finally N keys at root level 1. We then define one public key set per node in the tree as follows:

- $\mathcal{PK}'_1 = \{\mathsf{pk}_{1,1}\}$, $\mathcal{PK}'_2 = \{\mathsf{pk}_{2,2}\}$, $\cdots$, $\mathcal{PK}'_N = \{\mathsf{pk}_{N,N}\}$;
- $\mathcal{PK}'_{1\|2} = \{\mathsf{pk}_{1\|2,1}, \mathsf{pk}_{1\|2,2}\}$, $\mathcal{PK}'_{3\|4} = \{\mathsf{pk}_{3\|4,3}, \mathsf{pk}_{3\|4,4}\}$, $\cdots$, $\mathcal{PK}'_{N-1\|N} = \{\mathsf{pk}_{N-1\|N,N-1}, \mathsf{pk}_{N-1\|N,N}\}$;
- $\cdots$;
- $\mathcal{PK}'_{1\|N} = \{\mathsf{pk}_{1\|N,1}, \mathsf{pk}_{1\|N,2}, \cdots, \mathsf{pk}_{1\|N,N}\}$.

Those lists are maintained by the Aggregator and added to the set of parameters param of the system. The new user outputs her secret key sk as the set of all generated secret keys (e.g. for user 1, we have $(\mathsf{sk}_{1,1}, \mathsf{sk}_{1\|2,1}, \cdots, \mathsf{sk}_{1\|N,1})$).

3.4.3 A New Study. We consider a Third Party wanting to create a new study, labelled as ℓ. For each node j in the tree, the Third Party executes KeyGen(param) = $(\mathsf{tpk}_j, \mathsf{tsk}_j)$, which permits to define the following sets:

- $\mathcal{PK}_1 = \mathcal{PK}'_1 \cup \{\mathsf{tpk}_{1,0}\}$, $\mathcal{PK}_2 = \mathcal{PK}'_2 \cup \{\mathsf{tpk}_{2,0}\}$, $\cdots$, $\mathcal{PK}_N = \mathcal{PK}'_N \cup \{\mathsf{tpk}_{N,0}\}$;
- $\mathcal{PK}_{1\|2} = \mathcal{PK}'_{1\|2} \cup \{\mathsf{tpk}_{1\|2,0}\}$, $\mathcal{PK}_{3\|4} = \mathcal{PK}'_{3\|4} \cup \{\mathsf{tpk}_{3\|4,0}\}$, $\cdots$, $\mathcal{PK}_{N-1\|N} = \mathcal{PK}'_{N-1\|N} \cup \{\mathsf{tpk}_{N-1\|N,0}\}$;
- $\cdots$;
- $\mathcal{PK}_{1\|N} = \mathcal{PK}'_{1\|N} \cup \{\mathsf{tpk}_{1\|N,N}\}$.

The Third Party then proceeds, again for each node j in the tree, as follows:

- chooses at random one r_j;
- computes $\mathsf{ct}_j = \mathsf{Encrypt}(\mathcal{PK}_j, r_j, \ell)$.

This makes available to the Aggregator the following ciphertexts:

$$\mathsf{ct}_1, \cdots, \mathsf{ct}_N, \mathsf{ct}_{1\|2}, \cdots, \mathsf{ct}_{N-1\|N}, \cdots, \mathsf{ct}_{N\|N}$$

that corresponds, together with the above set of public keys, to the public parameters pk_ℓ of the study. The secret output of the Third Party, sk_ℓ is given by the set of random numbers $\{r_j\}$.

3.4.4 Sending Data for a Study. We then consider the participation phase by user i. He first gets back all the public key sets as defined in the previous step (see above section), and more specifically the ones related to her own nodes. For example, user 1 gets back the sets $\mathcal{PK}_1, \mathcal{PK}_{1\|2}, \cdots, \mathcal{PK}_{1\|N}$. The user i takes then as input his entry a_i and the label ℓ corresponding to the study and compute the following $n+1$ ciphertexts (one for each node in which he belongs), using the encryption scheme given in Section 3.2:

- $\mathsf{Encrypt}(\mathcal{PK}_i, a_i, \ell) = \mathsf{ct}_{i,1}$, $\cdots$, $\mathsf{Encrypt}(\mathcal{PK}_{i\|N}, a_i, \ell) = \mathsf{ct}_{i,n+1}$.

All the ciphertexts are then sent by the individual (through her WeStat app) to the Aggregator as her participation to the study.

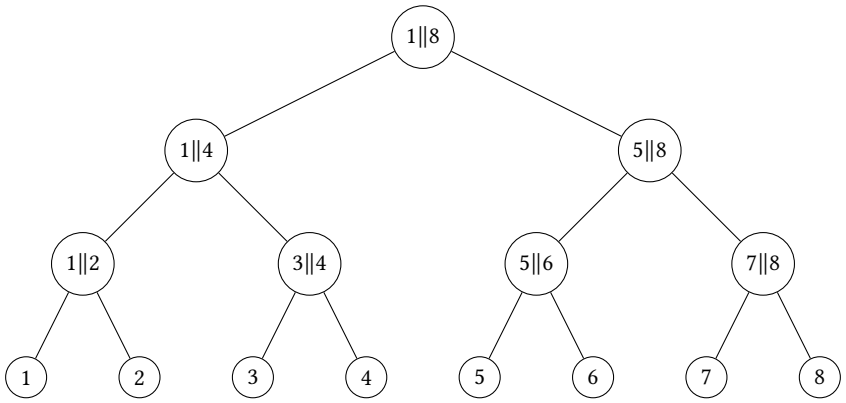

Figure 1: Tree structure

3.4.5 Obtaining the Result. Having access to all ciphertexts of all participating users (plus the ones of the Third Party), the Aggregator can start to proceed the computation of the result. For this purpose, it proceeds as follows:

- find a set of "target nodes" that uniquely covers all the leaves of participating users. For example, in the tree given in Figure 1, if user 5 haven't participated, one can obtain the final result by using nodes 1‖4, 6 and 7‖8, which ones permit to scan all true participants. Details are given in [4];
- execute the decryption procedure Decrypt (given in Section 3.3) for each given target node (always possible since we have obtained all the relevant ciphertexts), which permits the Aggregator to obtain as many partial blinded sums[1] as there are target nodes;
- compute the whole blinded sum. For example, given the above case, the Aggregator first obtains $a_1 + a_2 + a_3 + a_4 + r$ with the node 1‖4, $a_6 + r'$ with node 6 and $a_7 + a_8 + r''$ with node 7‖8, and can deduce $\sum_{i\in[1,8]\setminus\{5\}} a_i + (r + r' + r'')$;
- sends this result to the Third Party, together with the used target nodes, so that the later can remove the corresponding randomness (r, r' and r'' in our example) that it has generated during the new study creation (see Section 3.4.3). This finally gives the Third Party the unblinded sum, which corresponds to the expected result.

The way we can treat a more general inner product with this scheme is quite easy: each individual i can be associated to a scalar α_i and can easily replace, in the above computations, the value a_i by $\alpha_i \cdot a_i$.

3.5 Security Arguments

Our solution is directly verified using the same arguments of [4] with the adaptation of using the DSum functionality in [5].

First, the security of the DSum dynamic DMFE building block given in [5] provides us the guarantee, using the fact that all the parties additionally make use of the same label (which imposes a constraint on which values can be added together), that,

- for each group/node in the tree, the sum of the contributions is automatically revealed when all the parties belonging to this group/node have sent their contributions;
- the individual contributions of non-participating users, together with the sum related to groups/nodes where there are non-participating users, remain hidden for any actor, including the Aggregator that makes the computations.

As an example, if users 1, 3 and 4 have sent their contributions, but not user 2, the partial sums of nodes 1, 3, 4 and 3‖4 can be retrieved, while the ones of nodes 1‖2, 1‖4 and 1‖8 cannot, since the DSum dynamic DMFE is secure [5].

Then, from that result, the security of the fault-tolerant tree-based system given in [4] permits us to argue that

- the sum of the contributions of all the participating users (including the one by the Third Party, see below) is automatically revealed when all the said parties have sent their contribution;
- any other partial sum (except the above intermediate ones related to full groups/nodes) cannot be obtained, since the users contribution are provided for the whole set of "target nodes" of the tree that uniquely cover the participating users.

Recall that we do not consider the case of a coalition between the Aggregator and the Third Party. Then, as the non-corrupted Third Party is seen as an extra user that contributes to hide the intermediate values by adding/removing randomness to all nodes, the Aggregator only obtained noised sum, so that even if it has compromised some users, what it can get is either the sum of compromised users, or a noised sum of compromised and non-compromised users.

Regarding the indistinguishability against the Third Party, it is obvious that the only information that it could obtain is only the final sum.

4 IMPLEMENTATION AND BENCHMARKS

In this section, we give several details concerning our implementation. At first, we focus on the smartphone app, and explain the way we have proceed to obtain usage statistics related to installed apps (in our running example, social networks apps such as Facebook, Instagram ...). We then focus on the implementation of our

[1] We consider the sum as blinded since, at this step, it still includes the secret randomness coming from the Third Party.

cryptographic protocol. We finally give our benchmarks to prove that our prototype is very close to a real use.

4.1 Smartphone Implementation

Our use case is strongly related to our capacity to obtain, from the WeStat app we have implemented, several usage statistics on the way the smartphone owner is using several already installed apps, related to games, social networks, banking and so on. This is definitely possible for Google (for Android OS) and Apple (for iOS). This is also quite easy for the editor of a specific smartphone related service, for its own app (e.g., Instagram can potentially obtain those statistics for the Instagram app). But when we consider the case of an independent app editor wanting to obtain such usage statistics for other non-related apps, this may be seen as finding a security breach. But this is in fact not the case, as we will see.

We do not want to learn or modify some secrets that are embedded into those apps, but we only want to know, e.g., how long such app has been used, how many times per day, what is the used configuration, ... We show here that this is indeed possible using legal and available tools for app developers, at least for Android OS.

4.1.1 Overview. In Android (since API Level 21 that is Android 5.0), there are some APIs that can directly be used for our purpose:

- `UsageStatsManager` (in Android 5.0) which provides access to device usage history and statistics, usage data is aggregated into time intervals: days, weeks, months, and years;
- `NetworkStatsManager` (in Android 6.0) which provides access to network usage history and statistics, usage data is collected in discrete bins of time called 'Buckets'.

Any call to these APIs requires the permission

`Manifest.permission.PACKAGE_USAGE_STATS`,

even to access an app's own data usage (not that carrier-privileged apps are not included). This means that the user must explicitly authorize the application to retrieve the different statistics of her user's phone. Hence before the application can retrieve the statistics it must show a dialog box to ask the user to accept the retrieval of such kind of information. In our case, this is quite natural as this is directly related to the study, for which we need the user consent, as already explained before.

Indeed, `UsageStatsManager` and `NetworkStatsManager` are official public APIs that work on standard and official Android operating system (https://developer.android.com/reference/android/app/usage/package-summary). In particular, there is no need to use workaround like "Java reflection" to access them. Moreover, the devices do not need to be "rooted" or to tun a custom Android image. As far as we obtain the explicit permission:

```
<uses-permission android:
name="android.permission.PACKAGE_USAGE_STATS"/>,
```

we can proceed and obtain the desired statistics.

4.1.2 How does it work. A simple way to know what is possible to do is first to list the applications of a device to get the related package names. Using such information, it is then possible to query the managers (Usage or Network depending of the OS version) to get some statistics and retrieve the applications name and icon (used next for the WeStat app).

The WeStat application then run queries hourly, daily or monthly (depending on the wanted frequency) to gather information such as how long an application is running, how long an application have been brought to the foreground, since when it has been used, how much bytes have been uploaded or downloaded, what network's carrier has been used (data or wifi), ...

To illustrate what can be done in practice, the Shift project (https://theshiftproject.org/) is a good example of the implementation possibilities since it periodically measure the internet traffic of a device and compute the bytes consumed through the Wifi or data network.

Remark. *With some version of Android, the background schedulers can be paused by the system if the application is not launched regularly. And thus the hourly requests can be delayed or simply stopped.*

4.2 Cryptographic Implementation

4.2.1 Libraries. Our cryptographic prototype is implemented in Scala, a JVM based language, but using a native pairing implementation. Note that this internal pairing library, while easily portable, is not as fast as more up to date pairing libraries (see e.g. the RELIC library [1]). The use of such alternative libraries may offer substantial performance improvements. We also use the SHA-256 hash function for our implementation.

Our implementation provides a 114-bits security (and not 128-bit, as we are using BN-256 pairing-friendly elliptic curves, see [2]). One complex point we had to treat is the fact that, as shown in Section 3.2, the group for the chameleon hash should be $\mathbb{Z}_p^*$ of prime order q, and where p should itself be the prime order of the groups $\mathbb{G}_1$, $\mathbb{G}_2$ and $\mathbb{G}_T$ of the bilinear map e. As this last step is the less flexible one (finding a pairing-friendly group is not as easy [2, 7]), we need first to fix p according to the used pairing environment, and then test whether it is compatible with the chameleon hash. Using Barreto-Naehrig curves [3] with

$$p = 2523648240000001ba344d8000000007ff9f800000000010 \\ a10000000000000d$$

permits us to use the prime

$$q = 125404359535461796304386661765 9.$$

4.2.2 Configuration. We have implemented our system in both a PC (AMD Ryzen 7 3700X 8-Core) and two different Android smartphones: the Xiaomi Redmi Note 7 (ARMv8, Snapdragon 660 (SDM 660), octacore 2.2 Ghz (x4), 1.8 Ghz (x2)) and the Samsung Galaxy S10e (ARMv8, Exynos 9 octacore 2.7 Ghz (x2), 2.3 Ghz (x2), 1.9 Ghz (x4)). In the next section, we give different timings for different number (n) of participants.

4.2.3 Our tests. As we have seen in Section 3.4.4, each user has to execute $n + 1$ times the encryption procedure given in Section 3.3, the message being a vector of length $L = \ell_1 \times \ell_2$. In fact, some of the computations can be pre-computed, in the sense that they do not necessitate the knowledge of a_i (that is of d_i) to be done. The encryption phase is then done in two steps: pre-computation (before knowing a_i) and real encryption (when a_i is known).

In fact, there are two ways to consider such pre-computation. Either we can only consider public elements (and then not taking the label ℓ of the w_i random value), which decreases the number of such pre-computation, but permits to potentially reuse them for several studies. Or we consider all the computation that do not necessitates the knowledge of a_i (we can then make use of the label ℓ and the w_i random value), to better accelerate the online operation, but at the detriment of the storage capacity. More precisely, we can give the following information.

- Can be pre-computed: $\{t_j\}_{j\in\mathcal{PK},j\neq i}$, $\{K_{i,j}\}_{j\in\mathcal{PK},j\neq i}$.
- Can be pre-computed but not necessary: w_i, W_i, K_i, S_i, and $\{r_{i,j}[k]\}_{j\in\mathcal{PK},j\neq i,k\in[0,L[}$.
- Cannot be pre-computed: c_i, C_i.

The second step is the actual encryption, which with the knowledge of a_i, makes use of the said pre-computations, and do the remaining operations specific to the study, which includes the random w_i generation and operation involving w_i, and label ℓ.

The decryption is done by the Aggregator which obtains the blinded result. The final step is done by the Third Party to get the final unblinded result.

Most operations are executed in parallel, which gives faster execution times, but render the timings more fluctuate.

4.3 Tested Use Case

Let us consider the example of a study in which we would like to count the carbon emission of a smart phone when its owner uses social networks apps. We would like to correlate such carbon emission w.r.t. the age range and the residence area. More precisely, we define the following ranges for the age (with $\ell_1 = 6$ possibilities), and the following areas (with, e.g., $\ell_2 = 18$ possibilities).

- $\mathsf{age}_0 = [0, 15[$, $\mathsf{age}_1 = [15, 20[$, $\mathsf{age}_2 = [20, 30[$, $\mathsf{age}_3 = [30, 40[$, $\mathsf{age}_4 = [40, 50[$, $\mathsf{age}_5 = [50, 100[$.
- $\mathsf{area}_0, \mathsf{area}_1, \cdots, \mathsf{area}_{17}$.

Let us consider user 1, 27-years old (belonging to range age_2) and living in area_1, having a carbon emission of d_1 during the study period. She has previously registered to the service, and after having executed the KeyGen procedure, has access to the different secret keys $(\mathsf{sk}_{1,1}, \mathsf{sk}_{1\|2,1}, \mathsf{sk}_{1\|4,1}, \cdots, \mathsf{sk}_{1\|N,1})$ related to her position in the tree.

We define a vector of length 108 ($L = \ell_1 \times \ell_2 \times \cdots \ell_M$ in the general case, for the correlation with M attributes) with the following correspondences:

- component 0 is related to the $(\mathsf{age}_0, \mathsf{area}_0)$;
- component 1 is related to the $(\mathsf{age}_0, \mathsf{area}_1)$;
- $\cdots$;
- component 18 is related to the $(\mathsf{age}_0, \mathsf{area}_{17})$;
- component 19 is related to the $(\mathsf{age}_1, \mathsf{area}_0)$;
- $\cdots$;
- component 107 is related to the $(\mathsf{age}_5, \mathsf{area}_{17})$.

Since she belongs to $(\mathsf{age}_2, \mathsf{area}_1)$, user 1 is related to component 39 in such vector. She then has to consider, for her computations during encryption, the vector message a_1 given by

- $a_1[39] = d_1$;
- $\forall j \in [0, 107] \setminus \{39\}, a_1[j] = 0$.

and uses vector a_1 as the vector message to be encrypted, using the method given in Sections 3.4.4 and 3.3.

The idea is that during the result computation/decryption procedure, the contribution of the user in the groups she does not belong is equal to 0, in an unnoticeable way, since all the components are similarly encrypted.

Regarding the number of users ($N = 10, 100, 1000$ in the sequel), we may have to manage the fault tolerance property. In fact, we know from [4] that each user will affect $\log_2 N = n$ user groups if (s)he does not participate. We can then set the size of the tree to be sure to get the result if we know a good estimate on the number of non-participating users.

4.4 Benchmarks

We give in Table 1 the benchmarks we have obtained in the above PC, using this detailed use case, for different possible number of participants ($N = 10, 100, 1000$ and then $n = 4, 7, 10$). The final step, executed by the third party to unblind the result, takes less than 1 ms and is not included in the table.

Then, in Table 2, we provide timings on the two Android smartphones for the user specific operations, namely pre-computation and encryption.

The given timings have been obtained after the first execution of the operation by the freshly started mobile application. When executed several times, some system optimizations might be applied on the fly, so as to obtain better performances. But in our case, these operations will only be done once for each study. So retaining such first execution timings is more realistic and that's what we have done. Furthermore, the given timings include the serialization of the result into a json string.

Those timings are in accordance with what was expected in terms of complexity. Focusing on the Third Party, the ciphers' generation is, as expected, linear in the number of users (which corresponds to each node in the tree). Regarding the complexity of the user, each one has to execute $\log_2 N = n$ encryptions (one per node in which he belongs) but each such encryption necessitates a number of computation which is linear in the number of users in the considered node (2 at the leaf level, 4 in the above node, 8 overhead, etc.).

Strangely, the on-mobile ratio between the pre-computation and encryption times is smaller than on PC, particularly with bigger values of n. If the ratio was similar, the encryption for $N = 1000$ would be less than 800 ms. It appears that the encryption phase on mobile for big values of N triggers the garbage collector, which may explain the relatively poor performances. It is not clear if this is the only reason. The only solution may be to implement the cryptographic scheme entirely in native language. That way we would be able to finely manage the memory usage and see if there are other reasons.

We finally discuss about the communication cost. During registration, each user has to send the set of $n + 1$ public keys (s)he has computed and each public key is composed of one element in $\mathbb{G}_1$ (X_i), one element in $\mathbb{Z}_q$ (r_i) and two elements in $\mathbb{G}_2$ (Z_i and T_i). With BN-256 curves, we have $|q| = |\mathbb{G}_1| = 256$ and $|\mathbb{G}_2| = 512$, which gives each public key of size 1536-bits. Similarly, each has to send $n + 1$ ciphertexts for each participation and one ciphertext corresponds to one AES cipher C_i (128 bits), one element W_i in $\mathbb{G}_2$

Table 1: Performances (in ms) on PC (min / max / mean / median)

number of participants (N)	10	100	1000
third party ciphers generation	214	1565	19130
user pre-computation	44/61/52/53	273/420/355/362	2351/3479/2858/2856
encryption	10/24/16/16	12/43/19/18	28/73/41/38
decryption	57	191	1165

Table 2: Performances (in ms) on Android smartphones (min/max/mean/median)

number of participants (N)	10	100	1000
	Xiaomi Redmi Note 7		
user pre-computation	549/689/603/598	3846/5073/4415/4297	34465/48489/43192/43053
encryption	142/167/155/153	343/412/374/371	2150/2501/2303/2294
	Samsung Galaxy S10e		
user pre-computation	376/459/413/416	2502/2997/2637/2578	24119/34580/27137/26156
encryption	106/139/121/120	268/377/333/334	1819/2469/2018/1961

and one hash value S_i (256 bits), omitting the public key $\mathcal{PK}$ and the label ℓ. Each ciphertext has then a size of 896 bits. This finally gives us Table 3.

Table 3: Communication cost on the user side

number of participants (N)	10	100	1000
registration	2.06kB	18.94kB	187.69kB
participation	1.2kB	11.05kB	109.48kB

5 CONCLUSION

The main purpose of this paper is to make an in-depth privacy-by-design study of a real-life use case, namely mobile data usage statistics. Making use of advanced cryptographic mechanisms that have recently been published, we have shown that such kind of tool are quite close to a real deployment, for some cases such as the one we have detailed all along this paper.

It certainly remains a lot of work to do, especially regarding the obtained performances, so as to increase the number of potential participants in such kind of study.

ACKNOWLEDGEMENT

The authors would like to thank Jérémy Chotard for his suggestions during the redaction of this paper. All the authors were supported by the European Union H2020 PAPAYA Innovation Program Grant 786767.

REFERENCES

[1] D. F. Aranha, C. P. L. Gouvêa, T. Markmann, R. S. Wahby, and K. Liao. [n.d.]. RELIC is an Efficient LIbrary for Cryptography. https://github.com/relic-toolkit/relic.

[2] Razvan Barbulescu and Sylvain Duquesne. 2019. Updating Key Size Estimations for Pairings. *J. Cryptol.* 32, 4 (2019), 1298–1336.

[3] Paulo S. L. M. Barreto and Michael Naehrig. 2005. Pairing-Friendly Elliptic Curves of Prime Order. In *Selected Areas in Cryptography, 12th International Workshop, SAC 2005, Kingston, ON, Canada, August 11-12, 2005, Revised Selected Papers (Lecture Notes in Computer Science, Vol. 3897)*, Bart Preneel and Stafford E. Tavares (Eds.). Springer, 319–331. https://doi.org/10.1007/11693383_22

[4] T.-H. Hubert Chan, Elaine Shi, and Dawn Song. 2012. Privacy-Preserving Stream Aggregation with Fault Tolerance. In *Financial Cryptography and Data Security - 16th International Conference, FC 2012, Kralendijk, Bonaire, Februray 27-March 2, 2012, Revised Selected Papers (Lecture Notes in Computer Science, Vol. 7397)*, Angelos D. Keromytis (Ed.). Springer, 200–214.

[5] Jérémy Chotard, Edouard Dufour-Sans, Romain Gay, Duong Hieu Phan, and David Pointcheval. 2020. Dynamic Decentralized Functional Encryption. In *Advances in Cryptology - CRYPTO 2020 - 40th Annual International Cryptology Conference, CRYPTO 2020, Santa Barbara, CA, USA, August 17-21, 2020, Proceedings, Part I (Lecture Notes in Computer Science, Vol. 12170)*, Daniele Micciancio and Thomas Ristenpart (Eds.). Springer, 747–775.

[6] Jérémy Chotard, Edouard Dufour Sans, Romain Gay, Duong Hieu Phan, and David Pointcheval. 2018. Decentralized Multi-Client Functional Encryption for Inner Product. In *Advances in Cryptology - ASIACRYPT 2018 - 24th International Conference on the Theory and Application of Cryptology and Information Security, Brisbane, QLD, Australia, December 2-6, 2018, Proceedings, Part II (Lecture Notes in Computer Science, Vol. 11273)*, Thomas Peyrin and Steven D. Galbraith (Eds.). Springer, 703–732.

[7] Remi Clarisse, Sylvain Duquesne, and Olivier Sanders. 2020. Curves with fast computations in the first pairing group. *IACR Cryptol. ePrint Arch.* 2020 (2020), 760. https://eprint.iacr.org/2020/760

[8] Cynthia Dwork, Frank McSherry, Kobbi Nissim, and Adam D. Smith. 2016. Calibrating Noise to Sensitivity in Private Data Analysis. *J. Priv. Confidentiality* 7, 3 (2016), 17–51.

[9] Eduarda S. V. Freire, Dennis Hofheinz, Eike Kiltz, and Kenneth G. Paterson. 2013. Non-Interactive Key Exchange. In *Public-Key Cryptography - PKC 2013 - 16th International Conference on Practice and Theory in Public-Key Cryptography, Nara, Japan, February 26 - March 1, 2013. Proceedings (Lecture Notes in Computer Science, Vol. 7778)*, Kaoru Kurosawa and Goichiro Hanaoka (Eds.). Springer, 254–271.

[10] Shafi Goldwasser, S. Dov Gordon, Vipul Goyal, Abhishek Jain, Jonathan Katz, Feng-Hao Liu, Amit Sahai, Elaine Shi, and Hong-Sheng Zhou. 2014. Multi-input Functional Encryption. In *Advances in Cryptology - EUROCRYPT 2014 - 33rd Annual International Conference on the Theory and Applications of Cryptographic Techniques, Copenhagen, Denmark, May 11-15, 2014. Proceedings (Lecture Notes in Computer Science, Vol. 8441)*, Phong Q. Nguyen and Elisabeth Oswald (Eds.). Springer, 578–602.

[11] Shiva Prasad Kasiviswanathan, Homin K. Lee, Kobbi Nissim, Sofya Raskhodnikova, and Adam D. Smith. 2011. What Can We Learn Privately? *SIAM J. Comput.* 40, 3 (2011), 793–826.

[12] Hugo Krawczyk and Tal Rabin. 2000. Chameleon Signatures. In *Proceedings of the Network and Distributed System Security Symposium, NDSS 2000, San Diego, California, USA.* The Internet Society.

[13] Pierangela Samarati and Latanya Sweeney. 1998. *Protecting Privacy when Disclosing Information: k-Anonymity and Its Enforcement through Generalization and Suppression.* Technical Report.