IN, LCA1: Decentralized Data Sharing System based on Secure Multiparty Computation

Due on Autumn 2017

D.Froelicher, J.Troncoso-Pastoriza

Max Premi

January 4, 2018

Abstract

Unlynx and Prio are two privacy-preserving data sharing systems, with each it's way to encode, decode and aggregate datas. While Unlynx uses homomorphic encryption based on Elliptic Curve ElGamal and zero knowledge proofs, Prio uses Secret-sharing encoding and *secret-shared non-interactive proofs*(SNIP's) to validate the data, which is supposed to perform much better than classic zero knowlege proof in term of computation time, by doing a trade-off execution time/bandwidth.
We consider m servers that consitute the collective authority whose goal is to verifiably compute aggregation functions over data send by n data providers.
Several problems arise when you want to compare Unlynx and Prio. First, server are static in the first one but not in the second, and the threat model are not exactly the same, as you need to trust at least one server in Unlynx, but all in Prio for correctness. Privacy is assured if at least one server is trusted for both sytem. Then for Unlynx, data providers have all their data encrypted, while Prio do not use homomorphic encryption, and do not need to store data at all. Indeed Unlynx requires the data to be stored encrypted with a collective key, whereas Prio can proceed data decomposed in shares, sent by clients directly.
Prio also extends classic private aggregation techniques to enable collection of different class of statistics such as least-square regression, unlike Unlynx System, that can only compute sum and count querry over data.
The goal of this project is to, first implement Prio in the Unlynx framework, second to implement input validation for Unlynx, and then modify both protocol to be ran with the least significant difference in term of assumption and model.
Eventually, if possible, we want to design a System that implement the best of both sytems. The idea would be to use Prio encoding to do more than just summing and counting in Unlynx, by removing the encryption at the data provider, and doing the computation locally and then transfer to server in an encrypted way.

Contents

Introduction

Nowadays, tons of data are generated around us about what we do, and are used to compute statistics, by different parties. Even if these statistics are colleted with the goal of learning usefull aggregate informations about the users/population, for example health or work statistics for a country [7]. However it might end in collecting and storing private data from data provider, which poses a serious privacy and security problem. We can for example illustrate this with the problem of Cloud leaking that happened several times in the past years. [5], or sensitive health data such as suceptibility to contract diseases, that can be used against one individual. They might even be discloded, sold for profit [11] or be used by agencies for targeting and mass surveillance goals. The need to collect data and to share them in a privacy-preserving way has become crucial in this context, and lot of research has been done on this topic.
A lot of techniques have been developed through the years, by major technology companies such as Apple [6], but also researcher in Universities [1, 2].
First, sytems that use "Randomize response for differential privacy" [10], which change a value with probability $p < 0.5$, and by summing a large number of noisy value, the sum is still a good estimation of the real values.This technique is well scalable and perform nicely, but assure only *weak privacy*.
So encryption system were developped to solve this problem. However, by gaining *privacy*, these protocols sacrifice *robustness* and *scalability*, which are two important aspect to keep in mind while designing a decentralized system. Privacy is necessary so that no leak in the sensitive data happend, and robustness characterise the correctness of the computation. The trade-off between both should be reasonable, as we do not want any data to be leaked, and the server has to correctly compute under given circunstunces.
This difficulties lead to the use of legal agreements rather than technical solutions, as only a few of the systems have been deployed in the real-world. This agreement is not a robust solution for several reasons.
One of them is centralization. Centralized system are still widely used [12, 13] because they are way simpler and use a trusted third party. But these trusted parties still are a single point of failure in the system.
Another reason is that data provider have begun to realize the importance and value of their own data. This is why decentralized sytems are becoming more popular, and desirable.
In this paper, we present the implementation of Prio into Unlynx, an implementation of Unlynx's input validation and a theoritical comparison and discussion for future possible new sytems that combines, if possible, the best part of each paper.

Contributions

In this paper, the following contributions are made:
- An implementation of Prio SNIPs system in Unlynx, represented as two new protocols, and a new service. It includes the validation and the aggregation. It also contains the different data types supported by Prio.

- The implementation of a proof for input range validation for Elliptic Curve ElGamal, using biparing on a specific Elliptic Curve. This allows the server to exclude faulty data sent by possible malicious clients. The range checked is $[0, u^l]$ with u, l taking arbitrary values.

- An evaluation of both this protocol in terms of privacy and efficiency, with a comparison with the most similar settings. Then a conclusion on which system suits better the the needs of large scale real world scenario.

Background

Collective Authority

Nowadays, applications and systems rely on third party authorities to provide security service. For example the creation of certificate to prove ownership of public key. A collective authority is a set of server that are deployed in a decentralized and distributed way, to support a given number of protocols.
Each of them possesses a private-public key pair (k_i, K_i), where $K_i = k_i B$ with k_i is a scalar and K_i a point. This authority construct a public key $K = \sum_{i=1}^{m} K_m$ which is the sum of all servers public key. So to decrypt a message, each server partially decrypt a message encrypted using K, thus the collective authority key provides strongest link security.

ElGamal

For Unlynx, data are encrypted by using Elliptic Curve ElGamal, more precisely, P is a public key, x is a message mapped to a point and B is a bassepoint on the curve γ. The encryption is the following, with r a random nonce:
$E_P(x) = (rB, x + rP)$. The homomorphic properties states that $\alpha E_P(x_1) + \beta E_P(x_2) = E_P(\alpha x_1 + \beta x_2)$
To decrypt, the owner of the private key $P = pB$ multiplies rB and p to get rP and substract from $x + rP$.

Arithmetic Circuits

An arithmetic circuit C over a finite field $\mathbb{F}$ takes as input a vector $x = (x^{(1)}, ...x^{(L)}) \in \mathbb{F}^L$. It is represented as an acyclic graph with each vertex either be an *input*, *output* or a *gate*.
There are only two types of gates, addition and multiplication ones, all in finite field $\mathbb{F}$.
A citcuit C is just a mapping $\mathbb{F}^L \to \mathbb{F}$, as evaluatig is a walk thourght the circuit from inputs to outputs.

Beaver's MPC

A Beaver triple is defined as follow:
$(a, b, c) \in \mathbb{F}^3$, chosen at random with the constraint that $a.b = c$ As we use it in a multiparty computation context, each server i holds a share $([a]_i, [b]_i, [c]_i) \in \mathbb{F}^3$.
Using these shares, the goal is to multiply two number x and y without using them directly. In Prio the goal is to multiply shares $[x]_i$ and $[y]_i$.
To do so the following values are computed:

$$[d]_i = [x]_i - [a]_i \quad ; \quad [e]_i = [y]_i - [b]_i$$

Then from this shares, each server can compute d and e and compute this formula:

$$\sigma_i = de/m + d[b]_i + e[a]_i + [c]_i$$

The sum of these shares yields:

$$\sum_i \sigma_i = \sum_i (de/m + d[b]_i + e[a]_i + [c]_i)$$

Affine-aggregatable encodings (AFEs) functions

Given the fact that each data provider i holds a value $x_i \in D$, and the server have an aggregation function $f : D^n \to A$, whose range is a set of aggregates A. An AFE gives an efficient way to encode data such that it is possible to compute the value of the aggregation function $f(x_1, ..., x_n)$ given only the *sum of the encodings*

$x_1, ..., x_n$.
It consist of three Algorithm (Encode, Valid, Decode) defined in a field $\mathbb{F}$ and two integers $k' \leq k$

- **Encode(x)**: maps an input $x \in D$ to its encoding in $\mathbb{F}^k$
- **Valid(y)**: returns true if and only if y is a valid encoding of some item in D
- **Decode(σ)**: $\sigma = \sum_{i=1}^{n} Trunc_{k'}(Encode(x_i)) \in \mathbb{F}^{k'}$ is the input (Trunc takes the $k' \leq k$ components of the encoding), and it outputs the aggregation result $f(x_1, ..., x_n)$.

Bilinear parings over Elliptic Curve

Let G_1 and G_T be additive group of points of an elliptic curve G over a field $\mathbb{F}$ of order n and with identity O. Then the mapping $e : G_1 \times G_1 \to G_T$ satisfies the following conditions:
For all $R, S \in G_1$ and $x \in \mathbb{F}$, $e(R, S)(x) = e(Rx, S) = e(R, Sx)$
For B the base point, $e(B, B) \neq O$, and the mapping is efficiently computable.

1 Unlynx System

1.1 Model

Unlynx [1] is a privacy-preserving data sharing system developed by LCA1 [8] in collaboration with DeDiS [9].
It consists of a collective authority (CA) formed by a number m of server $S_1, ..., S_m$, and n data providers $DP_1, ...DP_n$ containing sensitive data, encrypted using EC ElGamal, following the key scheme describe in the **Background** Section. These DPs combined represent a distributed database that is used to awnser queries made by a querier Q. The querier and DPs choose one server of the CA to communicate with and can change this choice at any given time.
Functionality: Unlynx should permit SQL queries of the form SELECT SUM(*)/COUNT(*) FROM DP,.... WHERE * AND/OR GROUP BY *, with any number of * clauses.
Privacy and Robustness: Both are assured if at least one server is trusted, as we use the fact that we are allowed to publish ciphertext and their aggregation to show that the computation at the server are actually correct. It leaks nothing as all data are encrypted.

1.2 Threats

Collective authority servers It is assumed an Anytrust model [14]. It does not requires any particular server to be trusted or to be honest-but-curious. The moment it exists one server that is not malicious, functionality, security and privacy are guaranteed.
Data providers are assumed to be honest-but-curious. The system does not protect against malicious DPs sending false infomations, but a solution will be discuss in Section the **Input range validation for Elliptic Curve ElGamal**.
Queriers are assumed to be malicious, and can collude between themselves or with a subset of the CA servers.
It is also assumed tha all network communication is encrypted and authenticated, by using a protocol like TLS for example.

1.3 Pipeline and proof

The protocol start when a querier wants to retrieve some information about sensitive data. It sends the query to one of the server of the CA. Upon receiving, the server broadcast this query to the other servers in the collective authority.
From here the data are privately and securely processed by the CA, before sending back the result to the querier, encrypter over the public key of the Querrier. During all the steps of the protocol, the server will never get the data in clear.
The pipeline is the following: Encryption, Verification Shuffle, Distributed Deterministig Tag, Collective Aggregation, Distributed Results Obfuscation, Key Switch. At the end of this pipeline, the querier get the data and can decrypt them to get the aggregate statistics he asked for, without any server seeing the data in clear, or knowing from which data provider the data are from.

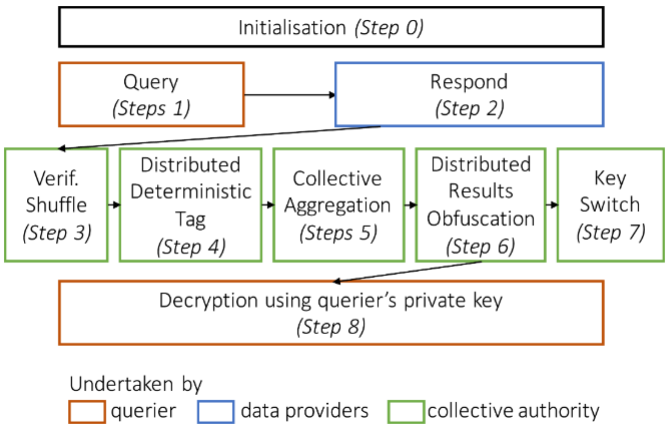

Figure 1: Unlynx query processing pipeline

Proofs are done thanks to zero-knowledge systems, to preserve privacy. There is one for each state of the pipeline, to illustrate a zero knowledge proof, while doing the aggregation phase it publishes the ciphertexts, and the result of their aggregation. As the data is encrypted using Elliptic curve ElGamal, it leaks nothing about the data alone. One of the problem in Unlynx is that it exists no Input range proof for data coming from the data providers, meaning the DPs can send fraudulous data (by giving a very big number or really small) and as data are encrypted we have no way to directly check this. This make the whole result and computation obsolete, and this is why in the basic Threat model, data providers are assumed honest-but-curious. This input validation proof is implemented and described in the following lines.

1.4 Input range validation for Elliptic Curve ElGamal

A problem with the encryption of data is that we can not be sure if the data sent are correct. An example is if we want to aggregate a field were values should be in range $[0, 100)$, in the current system if a malicious data provider want to send a value of 15000 to falsify information, it can do it. This section present an interactive algorithm based on a classic ElGamal input range validation [3], and we will discuss its non-interactive form in the section dedicated to implementation.
We define the scalar in the elliptic curve over the field $\mathbb{Z}_{\mathfrak{l}}$. Also we call $e(.,.)$ the billinear mapping that is describe in the Background section.
This is the algorithm that allows to check the validity of a secret σ in a range $[0, u^l]$. The algortihm can be adapted to check any range $[a, b]$.

This algorithm as been adapted from the original one [3], used on classic ElGamal encryption. The sound-

Common Input: B, P, u, l and commitment C
Prover Input: σ, r such that $C = B\sigma + Pr$, $\sigma \in [0, u^l)0$
$\mathbf{P} \leftarrow \mathbf{V}$ verifier picks $x \in \mathbb{Z}_p$ defines $y \leftarrow Bx$ and sends $A_i \leftarrow B(x+i)^{-1}\ \forall i \in \mathbb{Z}_u$
$\mathbf{P} \rightarrow \mathbf{V}$ Then prover encode the signature of the value to check in base u with randomly picked v_j.
So $\forall j \in \mathbb{Z}_l$ st $\sigma = \sum_j \sigma_j u^j$, it picks $v_j \in \mathbb{Z}_p$ and sends $V_j = A_{\sigma_j} v_j$
$\mathbf{P} \rightarrow \mathbf{V}$ prover pick 3 values $s_j, t_j, m_j \in \mathbb{Z}_p$, $\forall j \in \mathbb{Z}_l$ and sends:
$a_j \leftarrow e(V_j, B)(-s_j) + e(B, B)(t_j)$
$D \leftarrow \sum_j (Bu^j s_j + Pm_j)$
$\mathbf{P} \leftarrow \mathbf{V}$ Verifier sends a random challenge $c \in \mathbb{Z}_p$
$\mathbf{P} \rightarrow \mathbf{V}$ prover sends the following value for Verifier to compute verification.
$\forall j \in \mathbb{Z}_l$, $Z_{\sigma_j} \leftarrow s_j - \sigma_j c$ and $Z_{v_j} \leftarrow t_j - v_j c$
$Z_r = m - rc$, where $m = \sum_j m_j$
Verifier checks that $D = Cc + PZ_r + \sum_j (Bu^j Z_{\sigma_j})$
$a_j = e(V_j, y)c + e(V_j, B)(-Z_{\sigma_j}) + e(B, B)(Z_{v_j})$, $\forall j \in \mathbb{Z}_l$

ness follows from the unforgeability of the Boneh-Boyen signature. The prover need to compute $5l$ point multiplication in the protocol.

Arbitrarty Range

To handle an arbitrary range $[a, b]$, it is needed to show that $\sigma \in [a, a + u^l]$ AND $\sigma \in [b - u^l, b]$, this leads to the following formula:

$$\sigma \in [b - u^l, b] \iff \sigma - b + u^l \in [0, u^l)$$
$$\sigma \in [a, a + u^l] \iff \sigma - a \in [0, u^l)$$

The only modification necessary in the algortihm is the verifier's check which is now:

$$D = Cc + B(-B + u^l) + P(Z_r) + \sum_j B(Z_{\sigma_j})$$
$$D = Cc + B(-A) + P(Z_r) + \sum_j B(Z_{\sigma_j})$$

2 Prio System

2.1 Model

Prio is also a privacy-preserving data sharing system developed Standford University.
It consists of a collective authority (CA) formed by a number m of server $S_1, ...S_m$,and each data provider holds a private value x_i that is sensitive. Unlike Unlynx, Prio does not encrypt private value x_i that why it is a more challenging aggregation in terms of privacy.
Communication is assumed to be done in secure channels as previously described.
Functionality: Prio should permit the collective authority to compute some aggregation function $f(x_1, ...x_n)$ over private values of data providers, in a way that leaks as little as possible about these, except what can be inferred from the aggregation itself.
Privacy and Robustness: Privacy is assured if at least one server is trusted, but robustness is satisfied if and only if all server are trusted, as you cannot be assured that computation at the server are correct.

2.2 Threats

Collective authority servers In Prio, it is needed that one server is not malicious and trusted, so that security and privacy are guaranteed.
Robustness against malicious server seems desirable but doing so would cost privacy and perfomance degradation, which is not wanted.
Data providers are assumed to be malicious. The system protects itself against malicious DPs. All data that does not pass the SNIPs proof, will be discarded.

2.3 Pipeline and proof

Pipeline is a little different than Unlynx, as first the proof is run on the data when they arrived, if it passes the proof, it is stored and aggregate later with more data.
First, it is needed to define an arithmetic circuit for each data provider, $Valid(.)$. When the data provider run the circuit with its secret value as input, it output 1, i.e $Valid(x_i) = 1$.
This circuit is only defined in function of the number of bit of each shares $[x_i]_j$ from the data provider, so it will also send a configuration file to the server so that it an reconstruct the circuit to verify the input too.
From this circuit, 3 polynomials are extracted f, g and h.

The data providers first upload their shares $[x_i]_1, ..., [x_i]_m$ of private value and a share of polynomial h extracted from arithmetic circuit.
The servers verify the SNIPs provided by data providers to jointly confirm that $Valid(x_i) = 1$. If it fails, the server discard the submission.
Then each servers saved in an accumulator the data they need to aggregate and run the collective aggregation over the verifed datas.
When received enough input, they each publish their aggregation result to yield the final aggregation (which is the sum of all aggregation) result.

2.4 Prio SNIPs

In this section, the SNIP protocol will be detailled a litlle more.
Assumption: The **Valid** circuit have M multiplication gates, we work over a field $\mathbb{F}$ such that $2M << |\mathbb{F}|$

2.4.1 DP evaluation

First the DP evaluates the circuit **Valid** on its input x. It construct three polynomials f, g and h which encode respectively the inputs wire and the ouput wire of each of the M multiplication gates in the **Valid(x)** circuit.
This step is done by polynomial interpolation to construct f, g and get h by multiplying those.
So polynomials f, g have a degree at most $M - 1$ while $h = f \cdot g$ have a degree at most $2M - 2$.
Then the DP splits the polynomial h using additive secre sharing and send the ith ($[h]_i$) share to server i

2.4.2 Consistency checking at the server

At this time each server i holds a share $[x]_i$ and $[h]_i$ send by the data provider. From both this values, the servers can reproduce $[f]_i$ and $[g]_i$ without communicating with each others.
Indeed $[x]_i$ is a share of the input, and $[h]_i$ contains a share of each wire value coming out of a multiplication gate. Thus, it can derive all other values via affine operation on the wire.

2.4.3 Polynomial identity test

Now each server has reconstructed shares $[\hat{f}]_i, [\hat{g}]_i$ from $[h]_i$ and $[x]_i$. It holds that $\hat{f} \cdot \hat{g} = h$ if and only if the servers collectively hold a set of wire value that, when summed is equal to the internal wire value of the **Valid**(x) circuit computation.
All exeute the Schwartz-Zippel randomized polynomial identity test [REF] to check if relation holds and no data have been corrupted or malicious DPs have tried to send wrong data.
The principle is that if $\hat{f} \cdot \hat{g} \neq h$, then the polinomial $\hat{f} \cdot \hat{g} - h$ is a non-zero polynomial of degree at most $2M - 2$ zeros in $\mathbb{F}$. It can have at most $2M - 2$ zeros, so we choose a random $r \in \mathbb{F}$ and evaluate this polynomial, the servers will detect with probability at least $1 - \frac{2M-2}{|F|}$ that $\hat{f} \cdot \hat{g} \neq h$.
The servers can use a linear operation to get share $\sigma_i = [\hat{f}(r) \cdot \hat{g}(r) - h(r)]_i$. Then publish to ensure that $\sum_i \sigma_i = 0 \in \mathbb{F}$. If it is not 0 the servers reject the client submission.

2.4.4 Multiplication of shares

Finally all servers need to multiply the shares $[\hat{f}(r)]_i$ and $[\hat{g}(r)]_i$ to get the share $[\hat{f}(r)]_i \cdot [\hat{g}(r)]_i$ without leaking anything to each other about the values of the two polynomials. It is here that the Beaver MPC enter the computation. Each servers also received from a trusted dealer a one-time-use shares $([a]_i, [b]_i, [c]_i) \in \mathbb{F}^3$ such that $a \cdot b = c \in \mathbb{F}$. We can from this share, efficiently compute a multiparty multiplication of a pair secret-shared values. This only require each server to broadcast a single message.
The triple is generated by the data provider. The servers will be able to compute correcctly if values are corect. Moreover, even if the data provider send wrong values, the server still catch the cheating client with high probability. Indeed if $a \cdot b \neq c \in \mathbb{F}$ then we can write $a \cdot b = (c + \alpha) \in \mathbb{F}$. We can then run the polynomial identity test with $\hat{f}(r) \cdot \hat{f}(r) - h(r) + \alpha = 0$. The servers will catch a wrong input with probability at least $1 - \frac{2M-2}{|F|}$.

2.4.5 Output verification

If all servers are honest, each will hold a set of shares of the values if the **Valid** circuit. To confirm that **Valid**(x) is indeed 1 they only need to publish their share of the *output wire*. Then, all server sum up to confirm that its indeed equal to 1, except with some small failure probability due to the polynomail identity test.

3 Implementation

This section will detail what was implemented more throughouly. First we will abord the implementation of Prio:
Prio code [15] was implemented in Go, with 3 depedencies in C (FLINT, GMP and MPFR) to execute polynomial operations by Henry Corrigan-Gibbs.
This paper contribution is mostly porting code to use the multiparty computation and SNIPs in the Unlynx framework. So most of the code is used directly from the repository, but all the communication server/data provider has been reworked to be compatible with the Tree structure used in Unlynx [16].
To follow the structure, the aggregation and verification protocol were splitted in two different protocols, even if they both work together to get the final result. To be more precise, data from client are send, server do the SNIPs proof, if it is valid, it will aggregate the result of the SNIPs, else it will discard.
Let's describe the easiest protocol, the aggregation:
The protocols are run at each server j, and each protocol has received a share $[x_i]_j$ from dataprovider i, represented by a *type big.Int* in Go (it can actually be another type but the SNIPS protocol transform all the type in a *big.Int*). The protocol structure is a binary Tree.
When several data have been received, aggregation start by the root protocol, that notify all children that

the aggregation will start, and wait for the children to send their local sum. This notification goes down the Tree until there are no more children, and at this point each Leaf l aggregate locally the share by simply summing $\sum_i [x_i]_l$, and send the result to their unique parents. On receiving the response from its children, the other nodes aggregates locally their own shares, and
Now we're going to look at the input range validation implementation:

1: **Common Input:** B, P, u, l and commitment C. There will be several bases at the servers, as we will want to test several possible range.
2: **Prover Input:** σ, r such that $C = B\sigma + Pr$, $\sigma \in [0, u^l)$
3: **Server** picks a $x \in \mathbb{Z}_p$ every q defined request, defines $y \leftarrow Bx$ and compute for the all basis $A_i \leftarrow B(x+i)^{-1}$ $\forall i \in \mathbb{Z}_u$. It makes the signature public as well as the key y.
4: **Then Data provider** encodes the signature of the value to check in base u with randomly picked v_j.
5: So $\forall j \in \mathbb{Z}_l$ st $\sigma = \sum_j \sigma_j u^j$, it picks $v_j \in \mathbb{Z}_p$ and compute $V_j = A_{\sigma_j} v_j$
6: It also picks 3 values $s_j, t_j, m_j \in \mathbb{Z}_p$, $\forall j \in \mathbb{Z}_l$ and sends:
7: First : a public challenge c, $Z_r = m - rc$ and $D \leftarrow \sum_j (Bu^j s_j + Pm_j)$
8: and then $\forall j \in \mathbb{Z}_l$
9: $a_j \leftarrow e(V_j, B)(-s_j) + e(B, B)(t_j)$
10: $Z_{\sigma_j} \leftarrow s_j - \sigma_j c$ and $Z_{v_j} \leftarrow t_j - v_j c$
11:
12: Server checks that $D = Cc + PZ_r + \sum_j (Bu^j Z_{\sigma_j})$
13: $a_j = e(V_j, y)c + e(V_j, B)(-Z_{\sigma_j}) + e(B, B)(Z_{v_j})$, $\forall j \in \mathbb{Z}_l$

- what is implemented in a little more detailled
- optimization apported by code from github and not aborded in details

4 Performance evaluation

Explain a little more about comparison and test settings

- Scaling with number of server
- Scaling with number of client
- Scaling with fairly high number of both
- Time and Bandwidth dilemna

5 Comparison Theory and Performance

Comparison but explaining why it's cannot be perfectlhy matched, because of the differents assumptions.

6 Future Work

Conclusion

iufznfiozfoezizefez fze fezfezfezfezfezfezf ezfezfezfzefezfezfezfezfezfezfzefzefezfezfezfezfezfezfezfz

References

[1] David Froelicher, Patricia Egger, Joo Sa Sousa, Jean Louis Raisaro, Zhicong Huang, Christian Mouchet, Bryan Ford and Jean-Pierre Hubaux.
UnLynx: A Decentralized System for Privacy-Conscious Data Sharing. *EPFL*

[2] Henry Corrigan-Gibbs and Dan Boneh.
Prio: Private, Robust, and Scalable Computation of Aggregate Statistics. *Standford University*

[3] Jan Camenisch, Rafik Chaabouni, and abhi shelat
Efficient Protocols for Set Membership and Range Proofs. *IBM Research, EPFL, U. of Virginia*

[4] Keller,J., Lai,K.R., and Pelroth, N
How many times has your personal information been exposed to hackers ?

`http://www.nytimes.com/interactives/2015/07/29technology/personaltech/what-parts-of-your-information-`

[5] Classified Pentagon data leaked on the public cloud, BBC news
`http://www.bbc.com/news/technology-42166004`

[6] Greenberg, A.
Apple's 'differential privacy' is about collecting your data- but not your data.
`https://www.wired.com/2016/06/apples-differential-privacy-collection-data/`

[7] Departement federal de l'economie, de la formation et de la recherche DEFR.
`https://www.amstat.ch`

[8] LCA1 laboratory, EPFL.

[9] DeDis laboratory, EPFL.

[10] Warner, S. L.
Randomized response: A survey technique for eliminating evasive bias.
Journal of the American Statistical Association 60,309 (1965),63-69

[11] Smith, B.
Uber executive suggest digging up dirt on journalits.
`http://www.buzzfeed.com/bensmith/uber-executive-suggests-digging-up-dirt-on-journalists`

[12] Dyadic security `https://www.dyadicsec.com/`

[13] Dan Bogdanov, Sven Laur, and Jan Wilemson.
Sharemind: A framework for fast privacy-preserving computations. In *European Symposium on Research in Computer Security*

[14] David I Wolinsky, Hery Corrigan-Gibbs, Bryan Ford, and Aaron Jonhson.
Scalable anonymous group communication in the anytrust model. In *5th European Workshop on System Security. 2012*

[15] Prototype implementation of Prio in Go `https://github.com/henrycg/prio`

[16] Decentralized privacy-preserving data sharing tool : Unlynx
`https://github.com/lca1/unlynx`