ÉCOLE POLYTECHNIQUE
FÉDÉRALE DE LAUSANNE

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

Due on Autumn 2017

D.Froelicher, J.Troncoso-Pastoriza

Max Premi

December 25, 2017

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.
Prio also extends classic private aggregation techniques to enable collection of different class of statistics such as least-square regression, as Unlynx System 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.
Then 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 in local 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. Even if these statistics are colleted with the goal of learning usefull aggregate informations about the users/population, it might end in collecting and storing private data from client.
The need to collect data and to share them in a privacy-preserving way has become crucial in this context. We can for example illustrate this with the problem of Cloud leaking that happened several times in the past years.[ref]
Leaking private user data bring important privacy and security risks. They might be discloded, sold for profit or even be used by agencies for targeting and mass surveillance goals.
A lot of techniques have been developed through the years, by major technology companies such as Google [references], but also researcher in Universities [references].
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 technical agreements rather than technical solutions, which is not a robust solution.
Indeed, centralized system are still widely used [ref] as they are way simpler and use a trusted third party. But these trusted parties still are a single point of failure in the system and data provider begin to realize the 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.

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.
- The implementation of a proof for input range validation for Elliptic Curve ElGamal, using biparing on a specific Elliptic Curve.
- An evaluation of both this protocol in terms of privacy and efficiency, with a comparison with the most similar settings.

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 $K_i = k_iB$ 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 $\mathbb{F}^L \rightarrow \mathbb{F}$, 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 $f : D^n \rightarrow A$, 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 $xinD$ 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 $f(x_1, ..., x_n)$.

1 Unlynx System

1.1 Model

Unlynx is a privacy-preserving data sharing system developed by LCA1 in collaboration with DeDiS.
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. 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 [REF]. 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 [INPUT RANGE VALID SECTION].
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.
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, whule 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, but there is no Input range proof , meaning the DPs can send fraudulous data (by giving a very big number, and as it is encoded we cannot verify it) and make the computation wrong. This input validation proof is implemented and described in section [SECTION]

1.4 Input range validation for 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

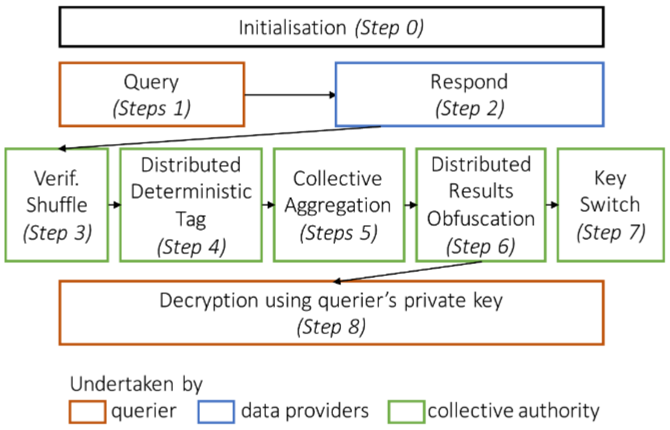
Initialisation (Step 0)
Query (Steps 1)
Respond (Step 2)
Verif. Shuffle (Step 3)
Distributed Deterministic Tag (Step 4)
Collective Aggregation (Steps 5)
Distributed Results Obfuscation (Step 6)
Key Switch (Step 7)
Decryption using querier's private key (Step 8)
Undertaken by
querier
data providers
collective authority

Figure 1: Unlynx query processing pipeline

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 [REF], and we will discuss its non-interactive form in the section dedicated to implementation.

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 $h = f.g$ 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 $\hat{f}.\hat{g} = h$ 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 $\hat{f}.\hat{g} \neq h$, then the polinomial $\hat{f}.\hat{g} - h$ 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 $\hat{f}.\hat{g} \neq h$.

The servers can use a linear operation to get share $\sigma_i = [\hat{f}(r).\hat{g}(r) - h(r)]_i$. 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 $[\hat{f}(r)]_i.[\hat{g}(r)]_i$ 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 $a.b = c \in \mathbb{F}$. 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 $a.b \neq c \in \mathbb{F}$ then we can write $a.b = (c + \alpha) \in \mathbb{F}$. We can then run the polynomial identity test with $\hat{f}(r).\hat{f}(r) - h(r) + \alpha = 0$. 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

nziefnoezifoezfzoioezifoezifoizen oirfezifnioznfzoei nzoi nfzioe

- 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 ezfezfezfzefezfezfezfezfezfzefzefezfezfezfezfezfezfezfz

References

inoinofizefnoezifnezoifnez