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

Due on Autumn 2017

D.Froelicher, J.Troncoso-Pastoriza

Max Premi

November 28, 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 Curves and zero knowledge proofs, Prio uses Secret-sharing encoding and *secret-shared non-interactive proofs*(SNIP's), which should perform much better in term of computation time, by doing more exchange between servers.
We consider m servers that consitute the collective authority whose goal is to verifiably compute aggregation functions over data send by n data providers.
Moreover, privacy is assured if at least one client is trusted, and it then assure robustness and scalability, with several other mechanics such as Noise addition for Unlynx or SNIPs proof for Prio. In both case the number of servers should be significantly smaller than the number of client.
Prio also extends classic private aggregation techniques to enable collection of different class of statistics such as least-square regression.
This paper presents the implementation of Prio aggregation system and input proof into Unlynx, as well as an input validation proof for Unlynx Aggregation system, and then compare both of those to see which one scale better.

Contents

Introduction

Nowadays, tons of data are generated around us and 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 collect and store private data from client.
The need of collecting data and sharing them in a privacy-preserving way has become crucial in this context. A lot of techniques have been developed through the years, by major technology companies such as Google [references], but also researcher in Universities [references].

- **Put some example applications**

However, by gaining *privacy*, these protocols sacrifice *robustness* and *scalability*,

- **Explain the link between the two and why**

, leading to the use of technical agreements rather than technical solutions.
In this paper, we present the implementation of Prio into Unlynx and compare them with input validation proof.

Contributions

Background

1 Unlynx System

- How does unlynx work (with a figure) (system of client, queries and servers)
- How Aggregation is done
- How proof is done

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 $DP_1, ...DP_n$. These DPs combined represent a distributed databased 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.

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 TLS.

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.
The pipeline is the following: Encryption, Verification Shuffle, Distributed Determinstig 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 data he asked, without any server seeing the data in clear, or knowing from which data provider the data are from.

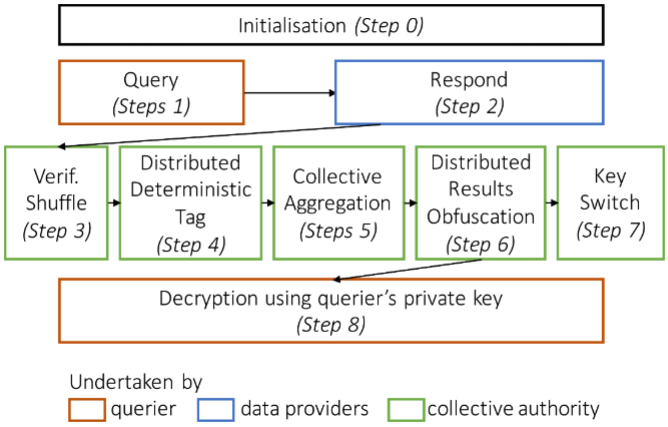

Figure 1: Unlynx query processing pipeline

Proofs are done with zero-knowledge systems, to preserve privacy. There is one for each state of the pipeline, for example when aggregating, it publishes the result of the local aggregation, as the data is encrypted using ElGamal Elliptic curves, 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 is implemented and described in section [SECTION]

2 Prio System

- How does Prio work (same as before)
- How does Aggregation is done
- New proof system SNIPs

2.1 Model

Prio is also a privacy-preserving sharing sytem, where each DPs holds a private value x_i. The goal of the CA is to compute an aggregation function $f(x_1, ...x_n)$, in a way that leaks as little as possible about each x_i

values to the server. Unlike Unlynx, Prio do not encrypt private value x_i that why it is a more challenging aggregation in terms of privacy.
As Unlynx communication is assumed to be done in secure channels. Privacy is assured as long as one server is trusted and correctness is provided if and only if all server are honest.

2.2 Threats

do

2.3 Pipeline and proof

do

Prio SNIPs

- Client evaluation
- Consistency checking at the server
- Polynomial identity test
- Multiplication of shares
- Output verification

2.4 Input range validation for ElGamal

do

Implementation

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

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

El Gamal range input checking

- TO be seen

Conclusion

References