A Multiparty Commutative Hashing Protocol based on the Discrete Logarithm Problem

Daniel Zentai[1], Mihail Plesa[2], and Robin Frot[3]

{daniel.zentai[1], mihail.plesa[2], robin.frot[3]}@xtendr.io
xtendr
Budapest, Hungary

Abstract. Let $\mathcal{X}$ and $\mathcal{Y}$ be two sets and suppose that a set of participants $P = \{P_1, P_2, \ldots, P_n\}$ would like to calculate the keyed hash value of some message $m \in \mathcal{X}$ known to a single participant in P called the data owner. Also, suppose that each participant P_i knows a secret value $x_i \in \mathcal{X}$. In this paper, we will propose a protocol that enables the participants in this setup to calculate the value $y = H(m, x_1, x_2, \ldots, x_n)$ of a hash function $H : \mathcal{X}^{n+1} \to \mathcal{Y}$ such that:

- The function H is a one-way function.
- Participants in $P \backslash \{P_i\}$ cannot obtain x_i.
- Participants other than the data owner cannot obtain m.
- The hash value $y = H(m, x_1, x_2, \ldots, x_n)$ remains the same regardless the order of the secret x_i values.

Keywords: Hash functions, Discrete logarithm problem, Anonymization

1 Introduction

Hash functions are very common building blocks of cryptographic protocols like digital signatures or message authentication codes. In this paper, we will propose a protocol built around the Chaum-van Heijst-Pfitzmann hash function [1]. The goal of our protocol is to calculate a hash value in a multiparty setup, i.e. the calculation is made by multiple participants collaboratively.

Our motivation was the following. Suppose we have to store personal data in a way that enables us to have access to a certain subset of non-sensitive attributes (e.g. age or height) and also enables us to keep the confidentiality of more sensitive attributes (e.g. passport number or name). Anonymization and pseudonymization may look like very similar concepts, but according to the European Union's General Data Protection Regulation (GDPR [8]) this is not the case. Pseudonymized data can be recovered using some secret information (e.g. a decryption key). Anonymized data on the other hand cannot be recovered under any circumstances, meaning we cannot use encryption to anonymize data.

In this paper, we will propose a protocol that enables a set of participants and a trusted server to calculate a hash value (i.e. anonymize their data) collaboratively.

We impose the following requirements regarding the process of the calculation. The function H should be a one-way function, the participants (including the server) should not learn either each other's secret values or the plaintext, and the hash value should be the same regardless of which participant is the data owner (i.e. what is the order of the participants in the calculation). The reason for the last requirement is that we want to be able to detect whether two different participants have the same plaintext without knowing the plaintext itself.

2 Preliminaries

In this section, we will give some basic definitions required throughout our paper. For the interested reader, we recommend [5] for more information.

Definition 1. *A function $\epsilon : \mathbb{N} \to \mathbb{R}$ is negligible if for all $c \in \mathbb{N}$ there exists an $n_0 \in \mathbb{N}$ such that for all $n \geq n_0$ we have $\epsilon(n) < n^{-c}$.*

Definition 2. *A hash function h is a deterministic polynomial time algorithm with arbitrary input length and fixed output length.*

Definition 3. *A hash function h is said to be one-way if it is*

- *Easy to compute, i.e. h outputs $h(x)$ in polynomial time for all $x \in D(h)$.*
- *Hard to invert, i.e. given a hash value y no probabilistic polynomial time algorithm can calculate an x for which $y = h(x)$ with non-negligible probability.*

Definition 4. *A hash function is said to be collision resistant if no probabilistic polynomial time algorithm can succeed in finding two values $x \neq x' \in D(h)$ such that $h(x) = h(x')$ with non-negligible probability.*

Throughout this paper, we will assume that the discrete logarithm problem defined below is hard, i.e. it cannot be solved by any probabilistic polynomial time algorithm.

Definition 5. *If G is a cyclic group and $a, b \in G$ are group elements then the discrete logarithm problem includes finding an $x \in G$ such that $a^x = b$.*

There are several cryptosystems based on the hardness of the discrete logarithm problem. The most popular ones include the Diffie-Hellman key exchange [3] and the ElGamal encryption [4].

3 Related Work

There are several anonymization techniques available, including but not limited to k-anonymity, differential privacy, or synthetic data [6]. These methods are designed either to add noise to the data, add some artificial data to the real data set, or generalize the data such that the data subject is no longer identifiable from it.

Our method is a bit different in a few aspects. First of all, we propose a purely cryptographic method, the protocol requires the collaboration of multiple participants, and the anonymized (i.e. hashed) version of a fixed data record remains the same regardless of which participant owns the plaintext. The main advantage of this approach is that this way we allow the participants to calculate the intersection of their data set without revealing sensitive identifiers.

To ensure that the hash of a fixed message m remains the same regardless of which participant owns the plaintext, we have to use commutative hashing. In [2] the authors propose $h(x, y) = f(\min(x, y), \max(x, y))$ where f is a collision-resistant hash function. Unfortunately, a similar approach won't work in our case since the participants know nothing about each other's secret values.

Definition 6. *A hash function $h : \mathcal{X}^n \to \mathcal{Y}^n$ is commutative, if $h(x_1, x_2, \ldots, x_n) = h(\sigma(x_1, x_2, \ldots, x_n))$ for all $x_1, x_2, \ldots, x_n$ and all σ permutations.*

Note that we cannot ensure collision resistance if we use a commutative hash function, since $(x_1, x_2, \ldots, x_n) \neq \sigma(x_1, x_2, \ldots, x_n)$ (unless σ is the identity permutation), but $h(x_1, x_2, \ldots, x_n) = h(\sigma(x_1, x_2, \ldots, x_n))$ by definition. Therefore the most we can do without losing the desired functionality is to ensure one-way property.

4 Hashing Protocol

As a building block of our protocol, we will use the discrete logarithm-based collision-resistant hash function defined by Chaum, van Heijst, and Pfitzmann [1]

Definition 7. *Let p be a prime number such that $q = \frac{p-1}{2}$ is also a prime number, and let a and b be primitive elements of the multiplicative group of $GF(p)$. Let $h : GF(q) \times GF(q) \to GF(p)\backslash\{0\}$ be the following function:*

$$h(x, y) = a^x \cdot b^y \mod p$$

The hash function h defined above is collision-resistant if the discrete logarithm problem is hard [1].

Now, suppose that we have a set of participants $P = \{P_1, P_2, \ldots, P_n\}$ and a trusted server represented by a special participant $S \notin P$. Also, suppose that the data owner (i.e. the only participant who knows m is P_1, therefore P_1 initiates the hashing protocol).

Let h be the Chaum-van Heijst-Pfitzmann hash function and $\Pi = (Gen, Enc, Dec)$ a public key encryption scheme. Suppose that each participant P_i has two randomly generated keys $x_i, y_i \in GF(q)$. Also, suppose that P_1 wants to hash the message $m \in GF(q)$. In addition, we will suppose that all the messages in the protocol are sent over a secure channel.

Our protocol proceeds as follows:

Multiparty Commutative Hashing Protocol

1. Upon receiving an upload request from P_1, S generates n random numbers $r_1, r_2, \ldots, r_n$ and sends them to the participants in P.
2. Upon receiving r_1 from S, P_1 calculates $h_1 = h(x_1 + m, y_1)$ and sends
$$h_1 | Enc_{K_S^{pub}}(r_1)$$
to S, where K_S^{pub} is the public key of S.
3. For all $i = 2, \ldots, n$, upon receiving r_i from S, P_i calculates $h_i = h(x_i, y_i)$ and sends
$$h_i | Enc_{K_S^{pub}}(r_i)$$
to S.
4. Upon receiving $h_i | Enc_{K_S^{pub}}(r_i)$ from P_i for all $i = 1, \ldots, n$, S checks the random values $r_1, r_2, \ldots, r_n$ and stores $\prod_{j=1}^n h_j$ if these are the same values S sent out at the first step. Otherwise, S outputs an error message.

Note that the purpose of the r_i values is not authentication, these just make it possible for the server to check that every participant took part in the calculation and therefore $\prod_{j=1}^n h_j$ can be stored as the output of the protocol.

Theorem 1. *If the discrete logarithm problem is hard, then $H(x_1, \ldots, x_n, y_1, \ldots, y_n) = \prod_{i=1}^n h(x_i, y_i)$ is a one-way function.*

Proof. Recall that if the discrete logarithm problem is hard, then h is a collision-resistant hash function. Suppose that H is not a one-way function, thus there exists a PPT algorithm $\mathcal{A}$ that can successfully find a corresponding input to a given value of H. We will use $\mathcal{A}$ to find a collision in h.

From the construction of H, it immediately follows that

$$\begin{aligned} H(x_1, \ldots, x_n, y_1, \ldots, y_n) &= \prod_{i=1}^n h(x_i, y_i) \\ &= \prod_{i=1}^n a^{x_i} \cdot b^{y_i} \\ &= a^{\sum_{i=1}^n x_i} \cdot b^{\sum_{i=1}^n y_i} \\ &= h\left(\sum_{i=1}^n x_i, \sum_{i=1}^n y_i\right) \end{aligned} \tag{1}$$

Now let $\alpha = H(x, 0, \ldots, 0, y, 0, \ldots, 0)$ and run algorithm $\mathcal{A}$ on α. Suppose that $\mathcal{A}(\alpha) = x_1^*, \ldots, x_n^*, y_1^*, \ldots, y_n^*$.

From this, we can successfully find a preimage of $h(x, y)$ since

$$\begin{aligned} h(x,y) &= H(x, 0, \ldots, 0, y, 0, \ldots, 0) \\ &= H(x_1^*, \ldots, x_n^*, y_1^*, \ldots, y_n^*) \\ &= h\left(\sum_{i=1}^{n} x_i^*, \sum_{i=1}^{n} y_i^*\right) \end{aligned} \tag{2}$$

And that is a contradiction since h is collision-resistant (therefore also a one-way function) if the discrete logarithm problem is hard.

□

Note that H is still collision-resistant in a somewhat weaker sense. If we fix the x_i and y_i values (e.g. we do not let the participants to change their keys) then it is hard to find two messages $m \neq m'$ with the same hash value.

5 Hashing with elliptic curves

Although the Chaum-van Heijst-Pfitzmann hash is a provable secure construction, it is not widely used in practice because of its running time. One way to make a discrete logarithm-based protocol more applicable in practice, is to use elliptic curves.

Definition 8. *Let p be a prime number and $\mathcal{E}_p$ be an elliptic curve over the finite field $GF(p)$. Also, let P, Q be points of $\mathcal{E}_p$. The elliptic curve discrete logarithm problem includes finding a $k \in \mathbb{Z}$ such that $k \cdot P = Q$.*

The elliptic curve discrete logarithm problem is thought to be even harder than the aforementioned classical discrete logarithm problem. The Chaum-van Heijst-Pfitzmann hash function can be defined as follows using elliptic curves.

Definition 9. *Let p be a prime number and $\mathcal{E}_p$ be an elliptic curve over the finite field $GF(p)$, moreover assume that $|\mathcal{E}_p|$ is a prime n. Let A and B be base points of $\mathcal{E}_p$ such that the discrete logarithm of B is not known. Let $h : (\mathbb{Z}\backslash n\mathbb{Z})^2 \to \mathcal{E}_p \backslash \{\mathcal{O}\}$ be the following function:*

$$h(k, l) = k \cdot A + l \cdot B$$

Theorem 2. *The hash function h defined above is collision-resistant if the elliptic curve discrete logarithm problem is hard.*

Proof. Suppose that we can find a collision efficiently, i.e. two pairs of integers $(k, l) \neq (k', l')$ such that

$$h(k, l) = k \cdot A + l \cdot B = k' \cdot A + l' \cdot B = h(k', l').$$

We can assume that $l' \neq l$. Otherwise, we would have $kA = k'A$ and thus $k = k'$. Since n is prime, $l - l'$ is invertible modulo n and this gives

$$\frac{k - k'}{l' - l} \cdot A = B.$$

We successfully solved the elliptic curve discrete logarithm problem.

□

Again, suppose that we have a set of participants $P = \{P_1, P_2, \dots, P_n\}$ and a trusted server represented by a special participant $S \notin P$. Also, suppose that the data owner (i.e. the only participant who knows m is P_1, therefore P_1 initiates the hashing protocol.

Let h be the Chaum-van Heijst-Pfitzmann hash function and $\Pi = (Gen, Enc, Dec)$ a public key encryption scheme. Suppose that each participant P_i has two randomly generated keys $k_i, l_i \in \mathbb{Z}$. Also, suppose that P_1 wants to hash the message m where m is a point of an elliptic curve $\mathcal{E}_p$ over the finite field $GF(p)$

Now our modified protocol proceeds as follows:

Multiparty Anonymization Protocol with Elliptic Curves

1. Upon receiving an upload request from P_1, S generates n random numbers $r_1, r_2, \dots, r_n$ and sends them to the participants in P.
2. Upon receiving r_1 from S, P_1 calculates $h_1 = h(k_1 + m, l_1)$ and sends
$$h_1 | Enc_{K_S^{pub}}(r_1)$$
to to S, where K_S^{pub} is the public key of S.
3. For all $i = 2, \dots, n$, upon receiving r_i from S, P_i calculates $h_i = h(k_i, l_i)$ and sends
$$h_i | Enc_{K_S^{pub}}(r_i)$$
to S.
4. Upon receiving $h_i | Enc_{K_S^{pub}}(r_i)$ from P_i for all $i = 1, \dots, n$, S checks the random values $r_1, r_2, \dots, r_n$ and stores $\sum_{j=1}^{n} h_j$ if these are the same values S sent out at the first step. Otherwise, S outputs an error message.

6 Implementation results

We instantiated the protocol both over the multiplicative group of integers modulo p and over the elliptic curve SECP256k1 [13]. The purpose of our experiment was to analyze the running time with respect to the number of participants. We discovered that the elliptic curve implementation is $10\times$ faster than the implementation over the multiplicative group of integers. For each number of participants, we run 100 trials of the protocol and average the running time. The results are presented in Table 1. We used the Levenberg-Marquardt algorithm to determine the coefficients of a linear polynomial that best fit the data [9]. Let N be the number of participants. We denote by $T_{\mathbb{Z}_p}(N)$ and $T_{\mathcal{E}_p}(N)$ the running time in seconds of the protocol implemented over $\mathbb{Z}_p$, respectively $\mathcal{E}_p$. The discovered polynomials are shown in (4) and (5):

$$T_{\mathcal{E}_p}(N) = 0.008N - 0.733 \tag{3}$$

$$T_{\mathbb{Z}_p}(N) = 0.13N - 42.06 \tag{4}$$

Num. of participants	**Running time over $\mathcal{E}_p$ (s)**	**Running time over $\mathbb{Z}_p$ (s)**
4	0.028	0.280
8	0.058	0.561
16	0.111	1.122
32	0.222	2.245
64	0.442	4.511
128	0.882	9.065
256	1.772	18.318
512	3.551	37.178
1024	7.135	76.174
2048	14.431	160.599
4096	29.474	352.496
8192	61.157	837.490
16384	132.225	2242.677

Table 1: The running time (s) with respect to the number of participants

Figures 1 and 2 show the running time with respect to the number of participants as well as the polynomial found by the curve fitting algorithm when implemented over $\mathcal{E}_p$ and over $\mathbb{Z}_p$.

All experiments were performed in Python on an Apple M1 Max platform.

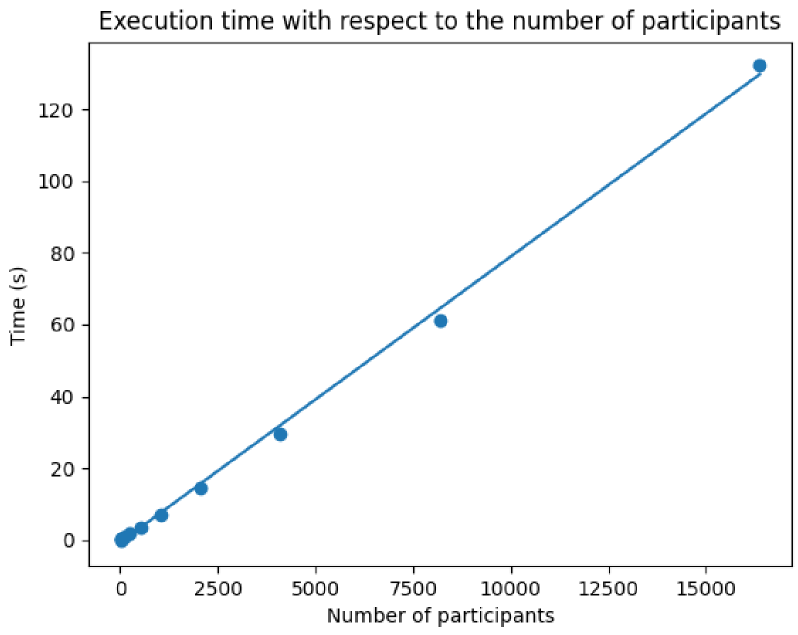

Fig. 1: The running time of the protocol implemented over $\mathcal{E}_p$

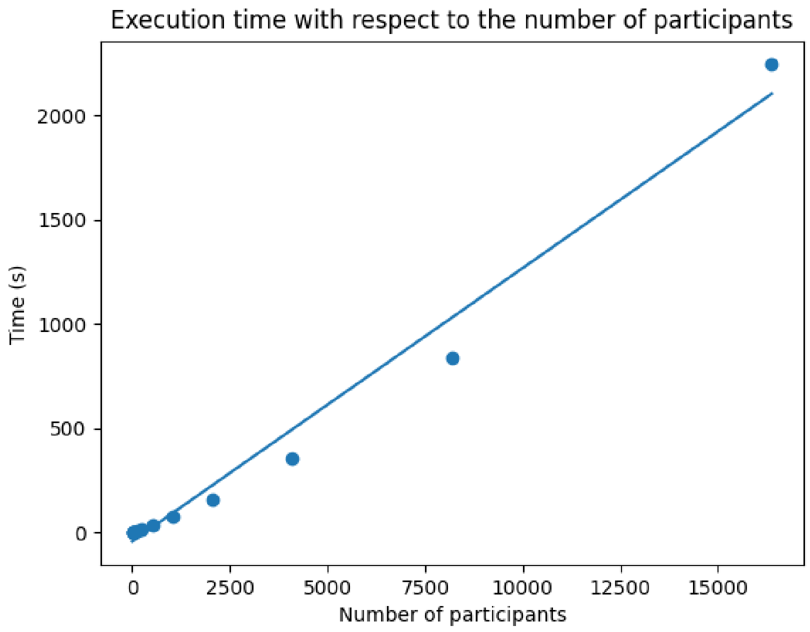

Fig. 2: The running time of the protocol implemented over $\mathbb{Z}_p$

7 A threshold protocol

One possible relaxation of the protocol is to ensure that any k-element subset of the participants can calculate the anonymous ID for some fixed $1 < k \leq n = |P|$. As a building block, we can use Shamir's secret sharing scheme [10] which works as follows.

Let $\mathbb{F}$ be a finite field and let $s_0 \in \mathbb{F}$ be the secret we want to share with the participants. Let $a_1, \ldots, a_{k-1}$ be elements of $\mathbb{F}$ chosen uniformly at random and $f(x) = s_0 + \sum_{i=1}^{k-1} a_i x^i$. Each participant P_i is given a point $(x_i, f(x_i))$ of the polynomial f. With Lagrange interpolation, any k element subset of the participants can now calculate the secret s_0 as follows.

$$s_0 = f(0) = \sum_{i=1}^{k} f(x_i) \cdot \ell_i$$

where

$$\ell_i = \prod_{j=1, j\neq i}^{k} \frac{x_j}{x_j - x_i}$$

Now we can use this secret sharing scheme to modify our anonymization protocol in a way that any k element subset of P can calculate the anonymous IDs.

Let h be the Chaum-van Heijst-Pfitzmann hash function and $\Pi = (Gen, Enc, Dec)$ a Fully Homomorphic Encryption (FHE) scheme. Suppose that each participant P_i has a randomly generated key $x_i \in GF(q)$. Also, suppose that P_1 wants to hash the message $m \in GF(q)$. In addition, we will suppose that all the messages in the protocol are sent over a secure channel.

Multiparty Threshold Anonymization Protocol

1. First, S generates two secret s_0 and t_0 and distributes them using Shamir's secret sharing scheme to the members in P. S uses two different polynomials $f(x) = s_0 + \sum_{i=1}^{k-1} a_i x^i$ and $g(x) = t_0 + \sum_{i=1}^{k-1} b_i x^i$. Let $x_1, \ldots, x_n \in GF(p)$ secret random elements generated by the respective P_i.
2. Upon receiving $Enc_{K_i^{pub}}(x_i)$ from P_i, S sends $f(Enc_{K_i^{pub}}(x_i)) = Enc_{K_i^{pub}}(f(x_i))$ and $g(Enc_{K_i^{pub}}(x_i)) = Enc_{K_i^{pub}}(g(x_i))$ to P_i where K_i^{pub} is the public key of P_i.
3. Every pair (P_i, P_{i+1}) shares with S the quotient x_{i+1}/x_i with the **Multiply** protocol described below.
4. Upon receiving an anonymization request from P_1, S randomly chooses a k element subset $Q \subseteq P$. Without loss of generality suppose now that $Q = \{P_1, \ldots, P_k\}$. Additionally, S generates k random values $r_1, r_2 \ldots, r_k$ and sends r_i to $P_i \in Q$.
5. For every participants $P_i \in Q$, S calculates $\ell_i = \prod_{j=1, j\neq i}^{k} \frac{x_j}{x_j - x_i}$ with the stored quotients and sends ℓ_i to P_i.
6. Upon receiving r_1 and ℓ_1 from S, P_1 calculates $h_1 = h(m + f(x_1)\cdot \ell_1, g(x_1)\cdot \ell_1)$ and sends
$$h_1 | Enc_{K_S^{pub}}(r_1)$$
to S where K_S^{pub} is the public key of S.
7. For all $i = 2, \ldots, k$, upon receiving r_i from S, P_i calculates $h_i = h(f(x_i)\cdot \ell_i, g(x_i)\cdot \ell_i)$ and sends
$$h_i | Enc_{K_S^{pub}}(r_i)$$
to S.
8. Upon receiving $h_i | Enc_{K_S^{pub}}(r_i)$ from P_i for all $i = 1, \ldots, k$, S stores
$$\prod_{i=1}^{k} h_i = a^{m + \sum_{i=1}^{k} f(x_i)\cdot \ell_i} \cdot b^{\sum_{i=1}^{k} g(x_i)\cdot \ell_i} = a^{m+s_0} \cdot b^{t_0}$$
if the random values $r_1, r_2, \ldots, r_k$ are the same values S sent out at the first step. Otherwise, S outputs an error message.

Note that in step 5, the server S needs to calculate $\frac{x_i}{x_i - x_j}$, where x_j is not known by P_i since it is a secret value generated by P_j. We will use a subroutine that solves this problem. The subroutine uses the fact that $\frac{x_i}{x_i - x_j}$ can be calculated securely if we can calculate the product $x_k^{-1} \cdot x_{k+1}$ securely, since

$$\frac{x_i}{x_i - x_j} = \left(\frac{x_i - x_j}{x_i}\right)^{-1} = \left(1 - \frac{x_j}{x_i}\right)^{-1} = \left(1 - x_i^{-1} \cdot x_j\right)^{-1}$$

and, if $j > i$,

$$\frac{x_j}{x_i} = \prod_{k=i}^{j-1} \frac{x_{k+1}}{x_k}.$$

Two-Party Multiplication Protocol

1. **Multiply**(x, y) takes two inputs from two different parties: x from P_1 and y from P_2.
2. P_1 generates a random number r_1 and sends $r_1 \cdot x$ to P_2 who then generates a random number r_2, calculates $r_1 \cdot x \cdot r_2 \cdot y$ and sends it to the server.
3. The server also generates a random value r_S, calculates $r_S \cdot r_1 \cdot x \cdot r_2 \cdot y$ and sends it back to P_1.
4. P_1 calculates $r_1^{-1} \cdot r_S \cdot r_1 \cdot x \cdot r_2 \cdot y = r_S \cdot x \cdot r_2 \cdot y$ and sends it to P_2.
5. P_2 calculates $r_2^{-1} \cdot r_S \cdot x \cdot r_2 \cdot y = r_S \cdot x \cdot y$ and sends it to the server.
6. The server calculates $r_S^{-1} \cdot r_S \cdot x \cdot y = x \cdot y$.

Note that in both versions of the protocol, if the server is corrupted, it can brute-force the hash values. Indeed, if P_i is the data owner during the first run then S learns $h(x_j, y_j)$ (or already knows (s_0, t_0) in the threshold protocol) for all $j \in P \backslash \{P_i\}$. And if $P_u \neq P_i$ is the data owner during the second run then S learns $h(x_i, y_j)$ (or $h(f(x_i), g(x_i)$ in the threshold protocol) as well. We can prevent this if the participants agree on a random number R using some group key exchange protocol and run the protocol on the message $R \cdot m$ instead of m. Other participants besides S cannot perform this attack since every message is sent over a secure channel.

Also note that it is possible to double the size of the plaintext space if P_1 sends $h(m_1 + x_1, m_2 + y_1)$ (or $h(m_1 + f(x_1) \cdot \ell_1, m_2 + g(x_1) \cdot \ell_1)$ in the threshold case) with $(m_1, m_2) \in GF(p)^2$.

8 Conclusion

We proposed a protocol that enables a set of participants to calculate a one-way commutative hash function collaboratively in a way that all of the participants have to take part in the computation, and this fact is verifiable by a trusted server and no participant other than the data owner has access to the plaintext. We also proposed a threshold protocol where any k-element subset of the participants can run the protocol successfully.

A possible future direction would be to construct a decentralized version of the protocol or make the protocol post-quantum.

Our implementation was purely experimental and was not made for efficiency. One possible direction of research is to create an efficient implementation of the protocol using low-level parallelizable instruction or FPGA [11][12].

References

1. D. Chaum, E. Van Heijst, and B. Pfitzmann. *Cryptographically strong undeniable signatures, unconditionally secure for the signer.* Advances in Cryptology-CRYPTO '91, volume 576 of Lecture Notes in Computer Science, pages 470-484. Springer-Verlag, (1992).

2. Goodrich, Michael T., and Roberto Tamassia. Efficient authenticated dictionaries with skip lists and commutative hashing. Technical Report, Johns Hopkins Information Security Institute, 2000.
3. W. Diffie, M. E. Hellman. *"New Directions in Cryptography.* IEEE Transactions on Information Theory. 22 (6): 644–654 (1976).
4. T. ElGamal. *A Public-Key Cryptosystem and a Signature Scheme Based on Discrete Logarithms.* IEEE Transactions on Information Theory. 31 (4): 469–472. (1985.)
5. J. Katz, Y. Lindell. *Introduction to modern cryptography.* CRC Press, (2007).
6. Marques JF, Bernardino J. Analysis of Data Anonymization Techniques. InKEOD 2020 (pp. 235-241).
7. S. Martín, P. Morillo, J.L. Villar, *Computing the order of points on an elliptic curve modulo N is as difficult as factoring N*, Applied Mathematics Letters, Volume 14, Issue 3, Pages 341-346, ISSN 0893-9659 (2001.)
8. *Regulation (EU) 2016/679 of the European Parliament and of the Council of 27 April 2016 on the protection of natural persons with regard to the processing of personal data and on the free movement of such data, and repealing Directive 95/46/EC (General Data Protection Regulation).* OJ L 119/1 (2016).
9. Ranganathan, A. The levenberg-marquardt algorithm. *Tutorial On LM Algorithm.* **11**, 101-110 (2004)
10. A. Shamir. *How to share a secret*, Communications of the ACM, (1979.)
11. Brown, S., Rose, J. FPGA and CPLD architectures: A tutorial. *IEEE Design & Test Of Computers.* **13**, 42-57 (1996)
12. Peleg, A., Weiser, U. MMX technology extension to the Intel architecture. *IEEE Micro.* **16**, 42-50 (1996)
13. Chen, L., Moody, D., Randall, K., Regenscheid, A. & Robinson, A. Recommendations for discrete logarithm-based cryptography: Elliptic curve domain parameters. (Lily Chen, Dustin Moody, Karen Randall, Andrew Regenscheid, Angela Robinson, 2023)