UNIVERSITY OF CALIFORNIA,
IRVINE

Privacy-Preserving Identity Verification Methods for Accountless Users via Private List Intersection and Variants

THESIS

submitted in partial satisfaction of the requirements
for the degree of

MASTER OF SCIENCE

in Computer Science

by

Zane Karl

2024

DEDICATION

To my family and to my friends–

A Road Taken

I had balked at the road's utility.
As the struggle begot futility.
But the words of a few,
both near me and near you,
lent worth to my time's suitability.

TABLE OF CONTENTS

LIST OF FIGURES

Page

LIST OF TABLES

ACKNOWLEDGMENTS

I would like to thank my advisor and committee chair Professor Gene Tsudik for his frank yet sagacious input and his immutable curiosity without which this work would never have existed.

I would like to express my deepest thanks to my committee member and co-advisor Professor Stanislaw Jarecki for his indefatigable enthusiasm and energetic correspondence.

I would like to thank Elina van Kempen and Seoyeon Hwang for their tireless persistence and aid toward making this work a reality. Each of you is a pleasure to work with and I could not have asked for more assiduous and affable co-researchers and friends.

Finally, I would like to thank my final committee member Professor Faisal Nawab for his patience and flexibility. I am lucky to have been a student of yours.

VITA

Zane Karl

EDUCATION

Master of Science in Computer Science **2024**
University of California, Irvine *Irvine, California*

Bachelor of Science in Applied Mathematics **2017**
University of California, Los Angeles *Los Angeles, California*

RESEARCH EXPERIENCE

Graduate Researcher **2021–2024**
University of California, Irvine *Irvine, California*

Undergraduate Researcher **2013–2017**
University of California, Los Angeles *Los Angeles, California*

TEACHING EXPERIENCE

Teaching Assistant **2022–2023**
University of California, Irvine *Irvine, California*

Program in Computing Teaching Assistant **2015–2017**
University of California, Los Angeles *Los Angeles, California*

REFEREED CONFERENCE PUBLICATIONS

Under Pressure: Effectiveness and Usability of the Apple Pencil as a Biometric Authentication Tool **Feb 2024**
Symposium on Usable Security and Privacy (USEC)

You've Got a Friend in ME (Mobile Edge): Blockchain Processing with Cloud Node Backup. **Aug 2022**
IEEE International Conference on Blockchain

SOFTWARE

PLI and Variants `https://anonymous.4open.science/r/PLI-D874/README.md`
A C implementation of PLI and its variants using different cryptographic mechanisms

ABSTRACT OF THE THESIS

Privacy-Preserving Identity Verification Methods for Accountless Users via Private List Intersection and Variants

By

Zane Karl

Master of Science in Computer Science

University of California, Irvine, 2024

Professor Gene Tsudik, Chair

Several prominent privacy laws require service providers to let consumers request access, correction, or deletion of their personal data. Compliance with such laws necessitates the verification of consumers' identities. This is not a problem for consumers who already have an account with a service provider since they can authenticate themselves via a successful account log-in, e.g., using username/password, or two-factor authentication. However, there are no such methods for accountless consumers, even though service providers (e.g., data aggregators) routinely accumulate data on consumers without accounts. Currently, accountless consumers are asked to share Personally Identifiable Information (PII) with service providers, which is privacy-invasive.

This paper proposes $\mathcal{P}$IVA*: a Privacy-Preserving Identity Verification for Accountless Users* using *Private List Intersection (PLI)* and its variants. First, we introduce PLI, a close relative of *private set intersection (PSI)*, a well-known cryptographic primitive that allows two or more mutually distrusting parties to compute the intersection of their private input sets. PLI takes advantage of the (ordered and fixed) list structure of the parties' private sets. As a result, PLI can be made more efficient than PSI. We also explore PLI variants: PLI-cardinality (PLI-CA), threshold-PLI (t-PLI), and threshold-PLI-cardinality (t-PLI-CA), all of which yield less information than PLI. These variants are progressively better suited for addressing the accountless consumer authentication problem. We prototype $\mathcal{P}\mathsf{IVA}$ and evaluate its performance, using regular PSI and garbled circuits as the basis for comparison. Results show that our PLI and PLI-CA constructions are more efficient than a garbled circuit approach, in both computation and communication overheads. While the garbled circuit-based implementations of t-PLI and t-PLI-CA have a faster execution time, our constructions greatly outperform garbled circuits in terms of bandwidth, with the garbled circuit-based implementation of t-PLI requiring $16\times$ the bandwidth of our construction. Additionally, the constructed t-PLI protocol is faster than existing threshold PSI protocols, taking advantage of the ordered property of lists. All implementation and evaluation are available in [2].

Chapter 1

Introduction

Numerous privacy regulations – including European Union General Data Protection Regulation (GDPR) [19], California Consumer Privacy Act (CCPA) [17], and Brazilian Lei Geral de Proteção de Dados (LGPD) [7] – mandate that service providers that collect any consumer-specific data must grant consumers access to that data [1]. In order to allow consumers to access, correct, or delete their personal data, a service provider must employ effective identity verification methods.

For instance, CCPA introduces the notion of a Verifiable Consumer Request (VCR) which is defined as "a request that is made by a consumer [...], and that the business can verify, using commercially reasonable methods, pursuant to regulations adopted by the Attorney General pursuant to paragraph (7) of subdivision (a) of Section 1798.185 to be the consumer about whom the business has collected personal information." (1798.140 (ak)) [17]. CCPA additionally states that "the business may require authentication of the consumer that is reasonable in light of the nature of the personal information requested" (1798.130 (a)(2)(A))

[1]While GDPR and LGPD apply to any entity processing residents' data in their respective regions, CCPA only applies to for-profit businesses that have a gross revenue above $25 million, handle the data of more than 100,000 California residents, or generate 50% or more of their revenue by selling California residents' PII (in 1798.140 (d)) [17].

[17].

For a consumer who has an account with a service provider, authentication translates into account log-in via, e.g., username and password, and two-factor authentication, such as email or SMS verification. In fact, GDPR guidelines [5] state that a user log-in, using a password, "should be sufficient to authenticate a data subject". CCPA also mentions that, for consumers with accounts, "the business may require the consumer to use that account to submit a verifiable consumer request" [17].

On the other hand, for the *account-less* users who have not set up an account with the service provider, or who have even not interacted with the service provider before submitting a VCR, only limited options exist for user authentication.

Without an account and hence without any shared secrets (such as passwords), service providers tend to ask consumers to reveal some amount of Personally Identifiable Information (PII) in the form of: (1) simply entering it [10, 18, 22, 6, 8], (2) providing cookies from prior interactions [25, 26], (3) showing government-issued IDs [10, 18, 22, 6, 25, 26, 8], or (4) producing notarized documents [22, 25, 26, 8]. The user study in [24] dealing with users' experiences with removing PII from the Internet shows that current removal procedures are not thorough, transparent, or responsive. This naturally prompts user concerns about service providers acquiring new PII in the nebulous process authenticating accountless users.

Motivated by the current state of affairs, this paper proposes $\mathcal{P}$IVA*: $\underline{P}$rivacy-Preserving $\underline{I}$dentity $\underline{V}$erification for $\underline{A}$ccountless Users.* $\mathcal{P}$IVA involves two parties: an accountless user $\mathcal{C}$ who possesses their PII as an ordered/labeled list and a service provider $\mathcal{S}$ that holds a collection of such PII-s (ordered/labeled lists) for each accountless user. It allows $\mathcal{S}$ to authenticate $\mathcal{C}$ if there is a sufficient number of list elements in the intersection of their respective PII lists. Meanwhile, $\mathcal{S}$ is prevented from learning any new $\mathcal{C}$ PII and vice-versa.

As a basic building block, we use *Private List Intersection (PLI)*, a variant of the well-known cryptographic primitive, *private set intersection (PSI)*. While PSI allows two parties to jointly compute the intersection of their private *sets*, PLI computes the intersection of the private *lists*, i.e., the list order matters. Due to the fixed order of PII assumed in PLI, the use of PLI instead of PSI can yield better performance. We also consider PLI variants that reveal strictly less information than PLI: (1) *PLI-Cardinality (PLI-CA)*, which outputs only the cardinality of the intersection list; (2) *threshold-PLI (t-PLI)*, which outputs the intersection only if its cardinality exceeds a threshold t; and (3) *threshold-PLI-Cardinality (t-PLI-CA)*, which outputs the cardinality of the intersection only if it exceeds the threshold t. We implement $\mathcal{P}\mathsf{IVA}$ using these protocols and evaluate its efficiency. The implementation is publicly available at [2]. [2]

Organization: After inspecting the related work in Chapter 2, Section 3 presents the necessary preliminary background, and Section 4 shows the system and threat models of our paper. Then we propose $\mathcal{P}\mathsf{IVA}$ with underlying primitives in Section 5 and their security analysis in Section 6. Implementation and evaluation details are given in Section 7, and the paper concludes with Section 8.

[2]This work is a reformatted version of the submitted for publication, but as of yet unpublished, paper [3].

Chapter 2

Related Work

2.1 Current Privacy-Invasive Authentication

Several surveys [10, 18, 22, 6, 25, 26, 8] investigate current authentication methods that service providers use for data requests submissions. These results show that the information requested is often privacy-invasive [6].

For example, [10] and [18] analyzed information requested by 55 service providers and used to identify and authenticate a user submitting a data request. Several require the user to take part in a live phone call and/or send (by mail) copies of a government-issued ID. Also, [8] shows that, while most companies receiving a data request require an account login, others request a government-issued ID, answering user-specific questions, or providing a sworn declaration. Furthermore, the study in [6] shows that some companies demand additional PII from users.

Similar results are observed in Android applications. For instance, [22] analyzed responses of application developers of 109 Android apps upon receiving a VCR. To verify the identity of the user, they require email- or account-based authentication, or ask for PII, e.g., the user's name, address, phone number, Android Advertising ID, date of birth, or a signed affidavit.

2.2 Identity Verification for Accountless Users

Verifying the identity of an accountless user is challenging; sometimes, sharing additional PII does not solve the problem. For example, [1] shows that while IP addresses are considered PII, they may not be sufficient to authenticate a user since they can be spoofed, or can be attributed to more than one user.

[6] suggests implementing a cookie generated by the user (instead of the service provider), to safely authenticate accountless users. A cookie can include an email address and the public key of the user, which the service provider could use to send a challenge to the user for further authentication.

Another approach is outlined in [15], which proposes VICEROY, an authentication method for accountless users using a public-private key pair generated by a local trusted device. For each web session initiation, the user generates a public key and supplies it to the server, so that the user can later digitally sign its request using the corresponding private key for the session.

To the best of our (current) knowledge, there is no prior work investigating privacy-preserving authentication for users who have never interacted with the service providers.

Chapter 3

Background

We use the notation $[1, n]$ to represent the set $\{1, 2, \ldots, n\}$ throughout the paper. Where applicable, we denote by $x_{i,1}$ and $x_{i,2}$ the first and second elements of a tuple x_i with two elements, for $\forall i \in [1, n]$, i.e., $x_i := (x_{i,1}, x_{i,2})$, where $i = 1, ..., n$.

3.1 Additively Homomorphic ElGamal Encryption

Below we define the ElGamal variant [9], which is additively homomorphic. Compared to the original ElGamal encryption [12] which is multiplicatively homomorphic, this variant multiplies g^m instead of m during encryption, where m is the plaintext message and g is a group generator of a group G of prime order q.

Definition 3.1.1. Additively homomorphic ElGamal encryption [9] consists of four algorithms: $(\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec})$, where:

- $\mathsf{Setup}(1^n) \rightarrow (G, q, g)$: outputs the system parameters (G, q, g), where G is a cyclic group of prime order q with a generator g.

- $\mathsf{KeyGen}(G, q, g) \to (sk, pk)$: outputs a pair of secret and public keys, where $sk = x \xleftarrow{\$} Z_q^*$ and $pk = (G, q, g, h)$ with $h = g^x$.

- $\mathsf{Enc}_{pk}(m; y) \to c$: Given a message $m \in Z_q$ and a public key $pk = (G, q, g, h)$, it outputs the ciphertext of m:

  $$c = (g^y, h^y g^m),$$

  where y is chosen at random from Z_q.

- $\mathsf{Dec}_{sk}(c) \to m$: Given a secret key $sk = x$ and a ciphertext $c = (c_1, c_2)$, it outputs the decryption m of c by sequentially running the following:

  - $\mathsf{PreDec}_{sk}(c) \to M$: Given a secret key $sk = x$ and a ciphertext $c = (c_1, c_2)$, it outputs $M := c_2(c_1^x)^{-1}$; and

  - $\mathsf{DLP}(M) \to m$: Given a group element $M \in G$, it outputs m such that $g^m = M$ in G. i.e., DLP is an algorithm that solves the discrete logarithm problem (DLP) in G with a small prime q.

Note that in general, message space is limited to $\mathbb{Z}_q$ with a small q due to decryption relying on the hardness of DLP. However, checking whether m is zero can be efficiently done over any prime q, as it only requires checking whether $c_2(c_1^x)^{-1} \overset{?}{=} 1$, without solving DLP.

3.2 Shamir Secret Sharing

Shamir secret sharing scheme [23] is a t-out-of-n threshold secret sharing scheme based on Lagrange interpolation, with a threshold t. i.e., Given $\geq t$ shares, the secret s can be reconstructed, while the possession of $< t$ shares reveals no information about s.

Definition 3.2.1. Shamir secret sharing scheme [23] is composed of two algorithms: $(\mathsf{Share}, \mathsf{Reconstruct})$, where:

- $\mathsf{Share}_{(t,n)}(s) \rightarrow (s_1, ..., s_n)$: generates n shares of s, given a secret s. It randomly chooses $(t-1)$ coefficients of a polynomial f of degree $(t-1)$ and sets s as the constant term of f, i.e.,

$$f(x) = s + a_1x + a_2x^2 + ... + a_{t-1}x^{t-1},$$

  with random a_i's. Then, it outputs n points of f over a set of agreed locations (other than zero) as n shares, e.g., $(s_1 := f(1), s_2 := f(2), ..., s_n := f(n))$.

- $\mathsf{Reconstruct}(s_{i_1}, ..., s_{i_m}) \rightarrow s/\perp$: reconstructs a secret s, given m shares $(s_{i_1}, ..., s_{i_m})$. It computes the polynomial $f'(x)$, using t out of m shares, e.g., via Lagrange interpolation, and outputs the y-intercept of f', i.e., $s' := f'(0)$. The output s' is equal to s only if m shares contain t or more correct shares.

Note that $\mathsf{Reconstruct}$ is not efficient if the input shares include some incorrect shares. This is because it requires trying all possible subsets of size t out of m shares, i.e., $\binom{m}{t}$ trials are needed.

3.3 Reed-Solomon Codes and Berlekamp-Welch Algorithm

The Shamir secret sharing scheme can be seen as a special case of Reed-Solomon (RS) coding scheme [20], where the number of errors is zero. (n, t)-RS codes are a group of error correction codes, where its encoder adds some parity bits to encode t messages into n messages (called *symbols* in coding theory) so that its decoder can correct up to $\frac{(n-t)}{2}$ errors. Specifically, t messages are assigned as the coefficients of a polynomial p of degree $t-1$, and the encoding of t messages are the evaluation of $p(x)$ over some n evaluation points,

$x_1, ..., x_n$, i.e., $\mathsf{Encode}(m_1, ..., m_t) = (p(x_1), ..., p(x_n))$, where $p(x) := m_1 + m_2x + ..., m_tx^{t-1}$. For decoding a codeword $c := (c_1, ..., c_n)$ with n symbols, the original method interpolates polynomials over all possible t points over n points, $(x_1, c_1), ..., (x_n, c_n)$, and outputs the t coefficients of the majority polynomial p.

The Berlekamp-Welch (BW) algorithm [4] is an efficient decoding algorithm that corrects up to $\lfloor \frac{n-t}{2} \rfloor$ errors in RS codes. It not only recovers the polynomial p representing the original messages but also outputs the error locator polynomial err by solving n linear equations, where the x-intercepts of err represent the location where the errors occurred among n symbols. We denote this by $\mathsf{BW}_{(k,n)}(c_1, c_2, ..., c_n) \rightarrow (p, err)$, where $k := \lceil \frac{n+t}{2} \rceil$, the minimum number of correct symbols in the input codeword. We use this algorithm to efficiently reconstruct the secret of Shamir secret sharing, where the input shares may include some invalid shares, i.e., errors.

Chapter 4

System & Threat Models

The system model includes two mutually distrusting parties: a client ($\mathcal{C}$) and a server ($\mathcal{S}$), where $\mathcal{S}$ initiates the protocol to verify $\mathcal{C}$'s identity and grants access to $\mathcal{C}$ once the verification succeeds. Each party has a list of PII of size n, $X = (x_1, x_2, ..., x_n)$ and $Y = (y_1, y_2, ..., y_n)$, respectively. We assume that the order of attributes in lists is public and agreed between parties in advance.

We consider a static Honest-but-Curious (HbC) adversary that can corrupt either $\mathcal{C}$ or $\mathcal{S}$ before the protocol starts, and during the protocol aims to learn maximal information about the other party's private input. Definition 4.0.1 presents the standard formal security definition [14] of a two-party protocol Π computing a function f in the presence of HbC adversaries.

Definition 4.0.1 (Security against HbC adversaries)**.** Let $f(X, Y) = (f_{\mathcal{C}}(X, Y), f_{\mathcal{S}}(X, Y))$ be a deterministic function, where $f_{\mathcal{C}}(X, Y)$ is the output to $\mathcal{C}$ and $f_{\mathcal{S}}(X, Y)$ is the output to $\mathcal{S}$, given the respective inputs X and Y. We say that for arbitrary inputs X and Y from

$\mathcal{C}$ and $\mathcal{S}$, a protocol Π securely computes f in the presence of a static HbC adversary if:

$$\{output^{\Pi}(X, Y, \lambda)\}_{X,Y\in\{0,1\}^*,\lambda\in\mathbb{N}} \overset{c}{\equiv} \{f(X, Y)\}_{X,Y\in\{0,1\}^*} \quad (4.1)$$

and there exists PPT algorithms, $S_{\mathcal{C}}$ and $S_{\mathcal{S}}$, such that:

$$\{S_{\mathcal{C}}(1^{\lambda}, X, f_{\mathcal{C}}(X, Y))\}_{X,Y\in\{0,1\}^*,\lambda\in\mathbb{N}} \overset{c}{\equiv} \{view^{\Pi}_{\mathcal{C}}(X, Y, \lambda)\}_{X,Y\in\{0,1\}^*,\lambda\in\mathbb{N}} \quad (4.2)$$

$$\{S_{\mathcal{S}}(1^{\lambda}, Y, f_{\mathcal{S}}(X, Y))\}_{X,Y\in\{0,1\}^*,\lambda\in\mathbb{N}} \overset{c}{\equiv} \{view^{\Pi}_{\mathcal{S}}(X, Y, \lambda)\}_{X,Y\in\{0,1\}^*,\lambda\in\mathbb{N}} \quad (4.3)$$

where λ is the security parameter. $output^{\Pi}$ denotes the output of the execution of Π, and $view^{\Pi}_{\mathcal{C}} = (X, r_{\mathcal{C}}, m_{\mathcal{C}})$ and $view^{\Pi}_{\mathcal{S}} = (Y, r_{\mathcal{S}}, m_{\mathcal{S}})$ denote the views of $\mathcal{C}$ and $\mathcal{S}$, respectively, during an execution of Π on inputs X, Y, and λ, where $r_{\mathcal{C}}$ is $\mathcal{C}$'s random tape, $r_{\mathcal{S}}$ is $\mathcal{S}$'s random tape, $m_{\mathcal{C}}$ is the message received by $\mathcal{C}$, and $m_{\mathcal{S}}$ is the message received by $\mathcal{S}$.

Chapter 5

$\mathcal{P}$IVA Design

In this section, we present four protocol variants for $\mathcal{P}$IVA: (1) *private list intersection (PLI)*, (2) *PLI-Cardinality (PLI-CA)*, (3) *threshold-PLI (t-PLI)*, and (4) *threshold-PLI-Cardinality (tPLI-CA)*. We analyze their security and costs in the subsequent section.

5.1 PLI

Fig. 5.1 shows the ideal functionality $\mathcal{F}_{\mathsf{PLI}}$ of a PLI protocol. It takes as input lists X and Y of size n from $\mathcal{C}$ and $\mathcal{S}$, respectively, and outputs their intersection $X \cap Y$ to $\mathcal{S}$.

In the context of our motivation of service providers and accountless consumers, a particular position $0 \leq \ i \leq \ n$ in each list ($X[i]$ or $Y[i]$) represents a specific type of PII, e.g., home address, driver's license number, or mother's maiden name. One important assumption for PLI is that n must be fixed and global, and each position i is well-known to be associated with a particular PII type.

Once obtaining the intersection, $\mathcal{S}$ decides whether the $\mathcal{C}$-provided PII is sufficient to verify their identity corresponding to some PII held by $\mathcal{S}$. As part of PLI, $\mathcal{S}$ learns no new PII about $\mathcal{C}$. This follows GDPR guidelines [5] stating that information requested by a service provider for verification purposes:

> "must be proportionate to the type of data processed, the damage that could occur, etc. in order to avoid excessive data collection",

Parameters: List size n
Ideal Functionality $\mathcal{F}_{\mathsf{PLI}}$:

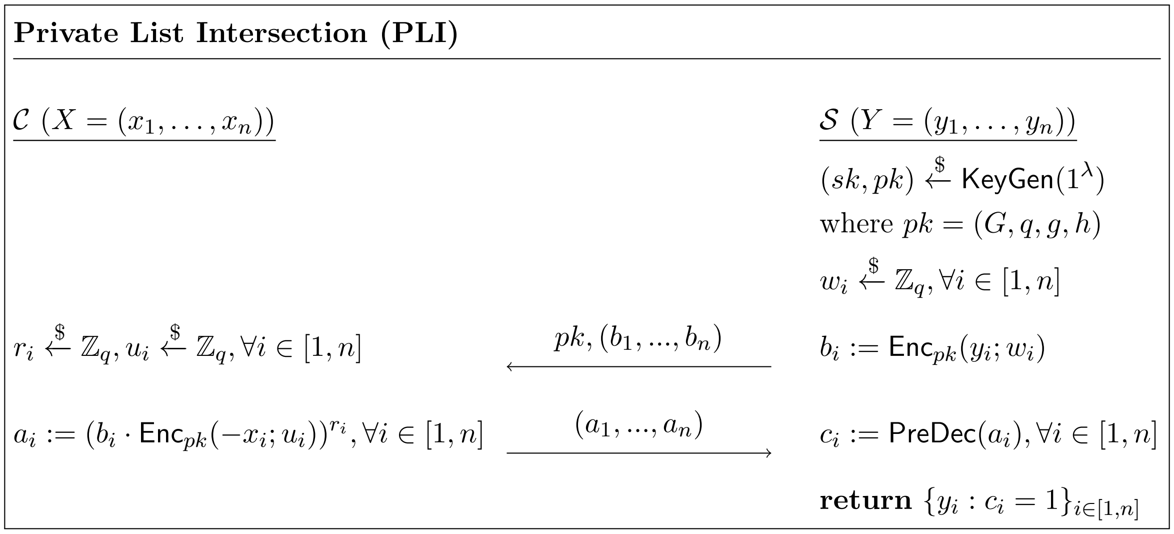
1. Wait for inputs $Y = (y_1, y_2, ..., y_n)$ from $\mathcal{S}$ and $X = (x_1, x_2, ..., x_n)$ from $\mathcal{C}$.
2. Compute the list intersection $X \cap Y$.
3. Give output $X \cap Y$ to $\mathcal{S}$, and $\perp$ to $\mathcal{C}$.

Figure 5.1: PLI Ideal Functionality

We construct a PLI protocol using the additively homomorphic ElGamal variant from Section 3.1. First, $\mathcal{S}$ generates ElGamal encryption keys (sk, pk). After generating n random values $(w_1, ..., w_n)$, $\mathcal{S}$ encrypts each element y_i under its own public key, such that $b_i := \mathsf{Enc}_{pk}(y_i; w_i) = (g^{w_i}, h^{w_i} g^{y_i})$ for all $i \in [1, n]$. $\mathcal{S}$ sends its public key, $pk = (G, q, g, h)$, and the ciphertexts of its input elements, $(b_1, ..., b_n)$, to $\mathcal{C}$.

Upon receipt, $\mathcal{C}$ chooses $r_i \in G$ and $u_i \in [1, q-1]$ at random, for all $i \in [1, n]$. Then, $\mathcal{C}$ encrypts the additive inverse of each element x_i in its list, using pk and u_i's, i.e., $\mathsf{Enc}_{pk}(-x_i; u_i) = (g^{u_i}, h^{u_i} g^{-x_i})$. $\mathcal{C}$ multiplies the resulting values by b_i's in the list order, re-randomizes them with r_i to hide its input values, and sends them to $\mathcal{S}$. i.e., $\mathcal{C}$ sends $(a_1, ..., a_n)$, where $a_i := ((b_{i,1} \cdot g^{u_i})^{r_i}, (b_{i,2} \cdot h^{u_i} g^{-x_i})^{r_i})$ for all $i \in [1, n]$.

Next, $\mathcal{S}$ checks if the received ciphertext is encryption of zero, by computing $c_i := \mathsf{PreDec}_{sk}(a_i) = \frac{a_{i,2}}{a_{i,1}^{sk}} \overset{?}{=} 1$, for each a_i, and outputs all y_i's with such c_i's. It is easy to see that the output of



Private List Intersection (PLI)

$\mathcal{C}\ (X = (x_1, \ldots, x_n))$

$\mathcal{S}\ (Y = (y_1, \ldots, y_n))$

$(sk, pk) \xleftarrow{\$} \mathsf{KeyGen}(1^\lambda)$

where $pk = (G, q, g, h)$

$w_i \xleftarrow{\$} \mathbb{Z}_q, \forall i \in [1, n]$

$r_i \xleftarrow{\$} \mathbb{Z}_q, u_i \xleftarrow{\$} \mathbb{Z}_q, \forall i \in [1, n]$ $\quad \xleftarrow{pk, (b_1, ..., b_n)} \quad$ $b_i := \mathsf{Enc}_{pk}(y_i; w_i)$

$a_i := (b_i \cdot \mathsf{Enc}_{pk}(-x_i; u_i))^{r_i}, \forall i \in [1, n]$ $\quad \xrightarrow{(a_1, ..., a_n)} \quad$ $c_i := \mathsf{PreDec}(a_i), \forall i \in [1, n]$

return $\{y_i : c_i = 1\}_{i \in [1,n]}$

Figure 5.2: PLI Construction based on ElGamal in Def. 3.1.1, where $\cdot$ in a_i computation is the element-wise product, i.e., $x \cdot y = (x_1y_1, x_2y_2)$ for some tuples $x = (x_1, x_2)$ and $y = (y_1, y_2)$.

the protocol is $X \cap Y$, since a_i is the encryption of $(y_i - x_i)r_i$ as follows:

$$\begin{aligned} a_i &= ((b_{i,1} \cdot g^{u_i})^{r_i}, (b_{i,2} \cdot h^{u_i} g^{-x_i})^{r_i}) \\ &= ((g^{w_i} \cdot g^{u_i})^{r_i}, (h^{w_i} g^{y_i} \cdot h^{u_i} g^{-x_i})^{r_i}) \\ &= (g^{(w_i+u_i)r_i}, h^{(w_i+u_i)r_i} g^{(y_i-x_i)r_i}) \\ &= \mathsf{Enc}_{pk}((y_i - x_i)r_i; R_i), \end{aligned}$$

where $R_i := (w_i + u_i)r_i$. Consequently, when $y_i = x_i$, $\mathcal{S}$ gets $c_i = g^{(y_i-x_i)r_i} = g^0 = 1$. Otherwise, it learns a random c_i. The above protocol is reflected in Fig. 5.2.

5.2 PLI-CA

PLI reveals to $\mathcal{S}$ which list elements match and, hence, which do not. $\mathcal{S}$ might use this information to confirm whether the PII previously collected for a given $\mathcal{C}$ is accurate. To prevent this, we include PLI-cardinality (PLI-CA), which outputs only the cardinality of the intersection. Fig. 5.3 shows the ideal functionality of PLI-CA.

In the context of our motivation, PLI-CA is suitable since it suffices for the service provider ($\mathcal{S}$) to learn how many elements are in the intersection to decide whether this is enough to verify the identity of a consumer ($\mathcal{C}$).

Parameters: List size n
Ideal Functionality $\mathcal{F}_{\mathsf{PLI-CA}}$**:**

1. Wait for inputs $Y = (y_1, y_2, ..., y_n)$ from $\mathcal{S}$ and $X = (x_1, x_2, ..., x_n)$ from $\mathcal{C}$.
2. Compute cardinality of list intersection $|X \cap Y|$.
3. Give output $|X \cap Y|$ to $\mathcal{S}$, and $\perp$ to $\mathcal{C}$.

Figure 5.3: PLI-CA Ideal Functionality

We build a PLI-CA protocol atop the PLI protocol from Section 5.1. To hide locations of matching elements, $\mathcal{C}$ randomly shuffles the computation results using a pseudorandom permutation Π and sends the permuted results. Fig. 5.4 shows the PLI-CA construction. $\mathcal{S}$ now outputs the number of elements where $c_i = 1$. Because of the pseudorandom permutation, indices where $c_i = 1$ do not reveal the real location of intersecting elements to $\mathcal{S}$.

5.3 t-PLI

While PLI and PLI-CA output their results to $\mathcal{S}$ no matter what, threshold PLI (t-PLI) outputs the intersection to $\mathcal{S}$ only if the number of common data exceeds some agreed-upon (fixed a priori) threshold t, i.e., $|X \cap Y| \geq t$. See Fig. 5.5 for the ideal functionality of t-PLI. It does not reveal anything to $\mathcal{S}$ if either party does not have sufficient PII. The threshold t can be determined by either agreement between parties, or by some legal standard. For instance, CCPA considers this, mentioning the “degree of certainty" required for identity verification. It sets “reasonable degree of certainty" as matching at least two pieces of information and “reasonably high degree of certainty" as matching at least three pieces of information and obtaining a signed declaration, under penalty of perjury [13].

The t-PLI protocol, shown in Fig. 5.6, adds the t-out-of-n Shamir Secret Sharing to the

PLI-Cardinality (PLI-CA)

$\underline{\mathcal{C}\ (X = (x_1, \ldots, x_n))}$		$\underline{\mathcal{S}\ (Y = (y_1, \ldots, y_n))}$
		$(sk, pk) \xleftarrow{\$} \mathsf{KeyGen}(1^\lambda)$
		where $pk = (G, q, g, h)$
		$w_i \xleftarrow{\$} \mathbb{Z}_q \forall i \in [1, n]$
$r_i \xleftarrow{\$} \mathbb{Z}_q, u_i \xleftarrow{\$} \mathbb{Z}_q, \forall i \in [1, n]$	$\xleftarrow{pk, (b_1, ..., b_n)}$	$b_i := \mathsf{Enc}_{pk}(y_i; w_i)$
$a'_i := (b_i \cdot \mathsf{Enc}_{pk}(-x_i; u_i))^{r_i}, \forall i \in [1, n]$		
$(a_1, ..., a_n) = \Pi((a'_1, ..., a'_n)),$ for $\Pi \xleftarrow{\$} \mathcal{P}_n$	$\xrightarrow{(a_1, ..., a_n)}$	$c_i := \mathsf{PreDec}(a_i), \forall i \in [1, n]$
		return $\lvert\{c_i : c_i = 1\}_{i \in [1,n]}\rvert$

Figure 5.4: A PLI-CA Construction atop PLI in Fig. 5.2. $\mathcal{P}_n$ is the pseudorandom permutation.

Parameters: List size n, threshold t.
Ideal Functionality $\mathcal{F}_{\mathsf{t-PLI}}$:

1. Wait for inputs $Y = (y_1, y_2, ..., y_n)$ from $\mathcal{S}$ and $X = (x_1, x_2, ..., x_n)$ from $\mathcal{C}$.
2. Compute the list intersection $X \cap Y$.
3. Give output $X \cap Y$ to $\mathcal{S}$ if $|X \cap Y| \geq t$, or $\perp$, otherwise. Give output $\perp$ to $\mathcal{C}$.

Figure 5.5: t-PLI Ideal Functionality

PLI protocol in Section 5.1. After computing a_i's, $\mathcal{C}$ randomly chooses a secret value s and generates n shares of s using Share algorithm in Definition 3.2.1. Then, $\mathcal{C}$ XORs each i-th share with the hash of the second element of a_i's, such that

$$e_i := H(a_{i,2}) \oplus s_i, \text{ for each } i.$$

Next, it sends the first component of a_i's, i.e., $(a_{i,1})_i$, and e_i's to $\mathcal{S}$, along with the hash of s.

$\mathcal{S}$ computes $H(a_{i,1}^{sk})$ with its private key sk and XORs to each e_i for each i, i.e., $s'_i := H(a_{i,1}^{sk}) \oplus e_i$. Then, $\mathcal{S}$ derives s' using the $\mathsf{Reconstruct}$ algorithm, or the Berlekamp-Welch algorithm BW for better performance, with $(s'_1, ..., s'_n)$ as input shares. Finally, if the hash of

Threshold PLI (t-PLI)

$\mathcal{C}\ (X = (x_1, \ldots, x_n))$		$\mathcal{S}\ (Y = (y_1, \ldots, y_n))$
		$(sk, pk) \xleftarrow{\$} \mathsf{KeyGen}(1^\lambda)$
		where $pk = (G, q, g, h)$
		$w_i \xleftarrow{\$} \mathbb{Z}_q, \forall i \in [1, n]$
$r_i \xleftarrow{\$} \mathbb{Z}_q, u_i \xleftarrow{\$} \mathbb{Z}_q, \forall i \in [1, n]$	$\xleftarrow{pk, (b_1, ..., b_n)}$	$b_i := \mathsf{Enc}_{pk}(y_i; w_i)$
$a_i := (b_i \cdot \mathsf{Enc}_{pk}(-x_i; u_i))^{r_i}, \forall i$		
$s \xleftarrow{\$} S$		
$(s_1, \ldots, s_n) \xleftarrow{\$} \mathsf{Share}_{(t,n)}(s)$		
$e_i := H(a_{i,2}) \oplus s_i, \forall i \in [1, n]$	$\xrightarrow[(e_1, ..., e_n), H(s)]{(a_{1,1}, ..., a_{n,1}),}$	$s'_i := H(a_{i,1}^{sk}) \oplus (e_i), \forall i$
		$(p, err) \leftarrow \mathsf{BW}_{(k,n)}(s'_1, ..., s'_n)$
		$s' := p(0)$
		return $\{y_i : err(i) \neq 0\}_i$, if $H(s') = H(s)$

Figure 5.6: A t-PLI Construction atop PLI in Fig. 5.2, using the Shamir secret sharing scheme in Def. 3.2.1 and the Berlekamp-Welch algorithm BW. $\oplus$ denotes the XOR operation.

s' matches the received hash value, $\mathcal{S}$ outputs its element with the indices of correct shares.

Note that s'_i is the same as the original share s_i if $x_i = y_i$. Therefore, if at least $k := \lceil \frac{n+t}{2} \rceil$ obtained s'_i are correct, $\mathcal{S}$ can use BW to efficiently recover the original polynomial p, and the error locating polynomial err. However, even if between t and $k - 1$ shares are correct, $\mathcal{S}$ can still recover p and locate errors by examining all the possible subsets of size t of the set $\{s'_i\}_i$ of size n.

5.4 t-PLI-CA

Threshold-PLI-Cardinality (t-PLI-CA) combines t-PLI and PLI-CA to further enhance the subject's privacy: $\mathcal{S}$ learns only the cardinality of the intersection, only if it exceeds some threshold t. i.e., The ideal functionality $\mathcal{F}_{\mathsf{t-PLI}}$ of t-PLI-CA (in Fig. 5.7) outputs $|X \cap Y|$ to

$\mathcal{S}$ only if $|X \cap Y| \geq t$.

Parameters: List size n, threshold t
Ideal Functionality $\mathcal{F}_{\mathsf{t-PLI-CA}}$**:**

1. Wait for inputs $Y = (y_1, y_2, ..., y_n)$ from $\mathcal{S}$ and $X = (x_1, x_2, ..., x_n)$ from $\mathcal{C}$.
2. Compute the cardinality of the list intersection $|X \cap Y|$.
3. Give output $|X \cap Y|$ to $\mathcal{S}$ if $|X \cap Y| \geq t$, or $\perp$, otherwise. Give output $\perp$ to $\mathcal{C}$.

Figure 5.7: Ideal functionality of t-PLI cardinality (t-PLI-CA)

We construct a t-PLI-CA protocol atop the t-PLI protocol by adding random shuffling. The idea is to prevent $\mathcal{S}$ from learning the indices of the matching elements given by the error locating polynomial by randomly permuting $(a_1, ..., a_n)$. As a result, $\mathcal{S}$ outputs the subtraction of the number of x-intercepts of the error locating polynomial err from the size of list n. Recall that $\mathcal{S}$ can find s' such that $H(s') = H(s)$ only when the number of matching elements exceeds the threshold, and the x-intercepts of err represent the wrong shares. Thus, the output means the cardinality of the intersection of the private lists. The detailed protocol is given in Fig. 5.8.

t-PLI Cardinality (t-PLI-CA)

$\mathcal{C}$ $(X = (x_1, \ldots, x_n))$		$\mathcal{S}$ $(Y = (y_1, \ldots, y_n))$
		$(sk, pk) \overset{\$}{\leftarrow} \mathsf{KeyGen}(1^\lambda)$
		where $pk = (G, q, g, h)$
		$w_i \overset{\$}{\leftarrow} \mathbb{Z}_q, \forall i \in [1, n]$
$r_i \overset{\$}{\leftarrow} \mathbb{Z}_q, u_i \overset{\$}{\leftarrow} \mathbb{Z}_q \forall i \in [1, n]$	$\xleftarrow{pk, (b_1, ..., b_n)}$	$b_i := \mathsf{Enc}_{pk}(y_i; w_i)$
$a_i := (b_i \cdot \mathsf{Enc}_{pk}(-x_i; u_i))^{r_i}, \forall i$		
$(a_1, ..., a_n) = \Pi((a'_1, ..., a'_n)),$ for $\Pi \overset{\$}{\leftarrow} \mathcal{P}_n$		
$s \overset{\$}{\leftarrow} S$		
$(s_1, \ldots, s_n) \overset{\$}{\leftarrow} \mathsf{Share}_{(t,n)}(s)$		
$e_i := H(a_{i,2}) \oplus s_i, \forall i \in [1, n]$	$\xrightarrow[(e_1, ..., e_n), H(s)]{(a_{1,1}, ..., a_{n,1}),}$	$s'_i := H(a_{i,1}^{sk}) \oplus (e_i), \forall i$
		$(p, err) \leftarrow \mathsf{BW}_{(k,n)}(s'_1, ..., s'_n)$
		$s' := p(0)$
		If $H(s') = H(s)$,
		return $(\vert\{s'_i : err(i) \neq 0\}_i\vert)$

Figure 5.8: A t-PLI-CA Construction atop t-PLI in Fig. 5.6. $\mathcal{P}_n$ is the set of random permutations of size n.

Chapter 6

Security & Cost Analysis

6.1 Security Analysis

Theorem 1 (Security of PLI)**.** The protocol presented in Fig. 5.2 securely computes the ideal functionality $\mathcal{F}_{\mathsf{PLI}}$ in Fig. 5.1 in the presence of HbC adversaries according to the Definition 4.0.1.

Proof. **Correctness:** Assume a protocol execution with honest $\mathcal{C}$ and honest $\mathcal{S}$. $\mathcal{C}$ gets output $\perp$ from the protocol Π, while $\mathcal{S}$ computes $c_i = \frac{a_{i,2}}{a_{i,1}^{sk}}$, which is equal to

$$\begin{aligned}
\frac{(b_{i,2} \cdot h^{u_i} g^{-x_i})^{r_i}}{((b_{i,1} \cdot g^{u_i})^{r_i})^{sk}} &= \frac{(h^{w_i} g^{y_i} h^{u_i} g^{-x_i})^{r_i}}{((g^{w_i})^{sk}(g^{u_i})^{sk})^{r_i}} \\
&= \frac{(h^{w_i} h^{u_i} g^{y_i - x_i})^{r_i}}{(h^{w_i} h^{u_i})^{r_i}} \\
&= \frac{(h^{w_i} h^{u_i})^{r_i} (g^{y_i - x_i})^{r_i}}{(h^{w_i} h^{u_i})^{r_i}} \\
&= (g^{y_i - x_i})^{r_i}
\end{aligned}$$

Thus, $c_i = g^0 = 1$, if $y_i = x_i$, or a random element in G, otherwise. $\mathcal{S}$ outputs $X \cap Y$.

Now, we build two simulators, one for each party.

Server privacy: Considering the corrupted $\mathcal{C}$, we construct a simulator SIM_C as follows: SIM_C chooses n random values $(z_1, ...z_n)$ and encrypts them under pk, i.e., $b'_i = \mathsf{Enc}_{pk}(z_i)$ for all i. It sends to $\mathcal{C}$ the public key pk and the encrypted random values $(b'_1, ..., b'_n)$. Because of the IND-CPA security of ElGamal encryption under the hardness assumption of the decisional Diffie-Hellman (DDH) problem, $(b'_1, ..., b'_n)$ is indistinguishable from the real protocol view of $(b_1, ..., b_n)$.

Client privacy: Considering the corrupted $\mathcal{S}$, $\mathcal{S}^*$, we construct a simulator SIM_S as follows: as input, SIM_S is given Y and $X \cap Y$ from the ideal functionality $\mathcal{F}_{\mathsf{PLI}}$.

- SIM_S receives the public key pk and $(b_1, ..., b_n)$ from $\mathcal{S}^*$. It sets $z_i = 0$ if $y_i \in X \cap Y$, and $z_i \xleftarrow{\$} G$, otherwise.
- SIM_S computes $A_i = Enc_{pk}(z_i)$ for all $i \in [1, n]$, and sends $(A_1, ..., A_n)$ to $\mathcal{S}^*$.

$\mathcal{S}^*$'s view (after PreDec) in the interaction with SIM_S is

$$\{\{1\}_{y_i \in X \cap Y}, \{z_i\}_{y_i \notin X \cap Y}\}$$

while the output of the protocol execution with real $\mathcal{C}$ is

$$\{\{1\}_{y_i \in X \cap Y}, \{(y_i - x_i)r_i\}_{y_i \notin X \cap Y}\}$$

. The two distributions above are indistinguishable because both z_i's and r_i's are chosen at random in G. $\square$

Theorem 2 (Security of PLI-CA)**.** The protocol presented in Fig. 5.4 securely computes the ideal functionality $\mathcal{F}_{\mathsf{PLI-CA}}$ in Fig. 5.3 in the presence of HbC adversaries according to the Definition 4.0.1.

Proof. **Correctness:** Similar to Theorem 1, c_i is 1 if $y_i = x_i$, or is a random, otherwise. Thus, $|\{c_i : c_i = 1\}| = |X \cap Y|$, which the server outputs.

Server privacy: Since the view of the (corrupted) $\mathcal{C}$ is the same as the one in PLI, server privacy is also met in PLI-CA.

Client privacy: Due to the similarity of the protocols, the construction of the simulator SIM_S, considering the corrupted $\mathcal{S}$, $\mathcal{S}^*$, is similar to the one in Theorem 1. In addition to SIM_S in Theorem 1, the following step is added:

- SIM_S sends $\Pi((A_1, ..., A_n))$ to $\mathcal{S}^*$, for a randomly chosen Π in $\mathcal{P}_n$.

For some $\Pi, \Pi' \in \mathcal{P}_n$, $\mathcal{S}^*$'s view in the interaction with SIM_S, $\Pi\{\{1\}_{y_i \in X \cap Y}, \{z_i\}_{y_i \notin X \cap Y}\}$, and the output of the protocol execution with real $\mathcal{C}$, $\Pi'\{\{1\}_{y_i \in X \cap Y}, \{(y_i - x_i)r_i\}_{y_i \notin X \cap Y}\}$, are indistinguishable. □

Theorem 3 (Security of t-PLI)**.** The protocol presented in Fig. 5.6 securely computes the ideal functionality $\mathcal{F}_{\mathsf{t-PLI}}$ in Fig. 5.5 in the presence of HbC adversaries according to the Definition 4.0.1.

Proof. **Correctness:** Assume a protocol execution with honest $\mathcal{C}$ and honest $\mathcal{S}$. $\mathcal{C}$ gets output $\perp$ from the protocol Π, while $\mathcal{S}$ computes

$$s'_i = H(a_{i,1}^{sk}) \oplus (e_i) = H(a_{i,1}^{sk}) \oplus H(a_{i,2}) \oplus s_i, \forall i$$

Since

$$\begin{aligned} H(a_{i,1}^{sk}) &= H(((b_{i,1} \cdot g^{u_i})^{r_i})^{sk}) \\ &= H(((g^{w_i} \cdot g^{u_i})^{sk})^{r_i}) \\ &= H((h^{w_i} \cdot h^{u_i})^{r_i}) \end{aligned}$$

and

$$\begin{aligned} H(a_{i,2}) &= H((b_{i,2} \cdot h^{u_i} g^{-x_i})^{r_i}) \\ &= H((h^{w_i} g^{y_i} \cdot h^{u_i} g^{-x_i}))^{r_i}) \\ &= H((h^{w_i} \cdot h^{u_i})^{r_i} (g^{y_i - x_i})^{r_i}), \end{aligned}$$

$H(a_{i,2}) = H(a_{i,1}^{sk})$, if $y_i = x_i$. Otherwise, $H(a_{i,2})$ is a hash of a random element in G. Consequently, if $y_i = x_i$, $s'_i = H(a_{i,2}) \oplus H(a_{i,2}) \oplus s_i = s_i$, or s'_i is random, otherwise. Depending on the number of intersecting elements, $|I|$, where $I := \{s'_i : s'_i = s_i\}_{i\in[1,n]}|$, three possible cases exist:

1. $|I| \geq k$: $\mathcal{S}$ can apply the Berlekamp-Welch algorithm to recover (p, err). $\mathcal{S}$ outputs $\{y_i : err(i) \neq 0\}_{i\in[1,n]}$, which is the set of input elements where their corresponding shares are correct, i.e., indices of intersecting elements.
2. $t \leq |I| < k$: $\mathcal{S}$ can examine every subset of size t, and check which subset reconstruct s' such that $H(s') = H(s)$.
3. $|I| < t$: $\mathcal{S}$ cannot reconstruct s' and cannot learn anything about the $\mathcal{C}$ input elements.

Server privacy: Since the view of the (corrupted) $\mathcal{C}$ is the same as the one in PLI (and PLI-CA), server privacy is also met in t-PLI.

Client privacy: Considering corrupted $\mathcal{S}$, $\mathcal{S}^*$, a simulator SIM_S can be constructed. Given Y and $|X \cap Y|$ from $\mathcal{F}_{\mathsf{t-PLI}}$,

- SIM_S receives the public key pk and $(b_1, ..., b_n)$ from $\mathcal{S}^*$. It sets $z_i = 0$ if $y_i \in X \cap Y$, and $z_i \xleftarrow{\$} G$, otherwise.
- SIM_S computes $A_i = \mathsf{Enc}_{pk}(z_i)$ for all i.

- SIM_S generates $S \leftarrow \$$, and computes the shares $(S_1, ..., S_n) \leftarrow \mathsf{Share}_{(t,n)}(S)$.

- SIM_S computes $E_i = H(A_{i,2}) \oplus S_i$ for all i.

- SIM_S sends $(A_{1,1}, ..., A_{n,1}), (E_1, ..., E_n)$, and $H(S)$ to $\mathcal{S}^*$.

Comparing $\mathcal{S}^*$'s view in interaction with SIM_S and in real execution of Π with $\mathcal{C}$, first, $H(s)$ and $H(S)$ are indistinguishable, as they are each a hash of a randomly selected value. Second, the tuples, $(a_{1,1}, ..., a_{n,1})$ and $(A_{1,1}, ..., A_{n,1})$, are indistinguishable because of the IND-CPA security of ElGamal encryption, under the DDH problem. Lastly, the tuples, $(e_1, ..., e_n)$ and $(E_1, ..., E_n)$ are also indistinguishable, as $(a_{1,2}, ..., a_{n,2})$ and $(A_{1,2}, ..., A_{n,2})$ are indistinguishable because of the IND-CPA security of ElGamal encryption, and through the security of one-time pad encryption with the randomly generated shares guaranteed by Shamir secret sharing scheme. □

Theorem 4 (Security of t-PLI-CA)**.** The protocol presented in Fig. 5.8 securely computes the ideal functionality $\mathcal{F}_{\mathsf{t-PLI-CA}}$ in Fig. 5.7 in the presence of HbC adversaries according to the Definition 4.0.1.

Proof. **Correctness:** Similar to Theorem 3, $H(a_i, 2) = H(a_i^{sk}, 1)$ if $y_i = x_i$, or is otherwise random. Thus, if the number of intersecting elements $|I| \geq t$, $\mathcal{S}$ can apply the Berlekamp-Welch algorithm or examine every subset to obtain the number of errors and obtain $|\{s'_i : err(i) \neq 0\}_i| = |X \cap Y|$, which the server outputs. If $|I| < t$, $\mathcal{S}$ outputs $\perp$.

Server privacy: Since the view of the (corrupted) $\mathcal{C}$ is the same as the one in PLI (and PLI-CA and t-PLI), server privacy is also met in PLI-CA.

Client privacy: Due to the similarity of the protocols, the construction of the simulator SIM_S, considering the corrupted $\mathcal{S}$, $\mathcal{S}^*$, is similar to the one in Theorem 3. In addition to SIM_S in Theorem 3, the following step is modified:

- SIM_S computes $A_i = \Pi(\mathsf{Enc}_{pk}(z_i))$ for all i.

For some $\Pi, \Pi' \in \mathcal{P}_n$, $\mathcal{S}^*$'s view in the interaction with SIM_S, $\Pi\{\{s_i\}_{y_i \in X \cap Y}, \{H(a_{i,1}^{sk}) \oplus H(a_{i,2}) \oplus (s_i)\}_{y_i \notin X \cap Y}\}$, and the output of the protocol execution with real $\mathcal{C}$, $\Pi'\{\{s_i\}_{y_i \in X \cap Y}, \{H(a_{i,1}^{sk}) \oplus H(a_{i,2}) \oplus (s_i)\}_{y_i \notin X \cap Y}\}$, are indistinguishable. □

6.2 Cost Analysis

Table 6.1: Communication and computation complexities for each protocol

	Server computation	Server communication	Client computation	Client communication
PLI	$O(n)$	$O(n)$	$O(n)$	$O(n)$
PLI-CA	$O(n)$	$O(n)$	$O(n)$	$O(n)$
t-PLI	$O(n^3)$ or $O(C(n,t))$	$O(n)$	$O(tn)$	$O(n)$
t-PLI-CA	$O(n^3)$ or $O(C(n,t))$	$O(n)$	$O(tn)$	$O(n)$

Table 6.1 summarizes the communication and computation complexities of each protocol. Since all protocols involve sharing encrypted elements of $\mathcal{S}$ and $\mathcal{C}$ lists, communication complexity for both parties is $O(n)$. Encrypting and computing on n list elements yields computation complexity of $O(n)$ for both $\mathcal{S}$ and $\mathcal{C}$ in the PLI and PLI-CA. In t-PLI and t-PLI-CA, generating shares increases the computation overhead of $\mathcal{C}$ to $O(tn)$, where t is the threshold. $\mathcal{S}$ either successfully runs the Berlekamp-Welch algorithm, which has the time complexity of $O(n^3)$, or examines each possible subset of size t with complexity $O(C(n,t))$, where $C(n,t)$ is the number of t-combinations over n elements.

Chapter 7

Implementation & Evaluation

7.1 Implementation

The performance of each protocol was tested on a MacBook Pro with a 2.3 GHz Dual-Core Intel i5 processor with 16 GB RAM. We implemented each protocol in `C`, using the OpenSSL v3.1.2 library. The Fisher-Yates shuffle [11], instantiated with a cryptographically secure pseudo-random number generator provided by OpenSSL, served as a secure shuffle for the PLI-CA and t-PLI-CA protocols. To evaluate the performance of our constructions, we implemented each protocol using Yao's garbled circuit [27] provided by MP-SPDZ library [16]. The implementation is available at [2].

For the homomorphic encryption schemes, we compared the performance of the PLI implemented with Elliptic Curve-based ElGamal (EC-ElGamal) with a 224-bit secure elliptic curve (provided by OpenSSL) to the one implemented with the standard (multiplicatively homomorphic) ElGamal and the additively homomorphic ElGamal schemes with a 2048-bit key size, which is an equivalent security parameter.

Finally, we measured the communication and computation costs of each protocol for:

1. increasing size n of input lists, and compared it to (1) the garbled circuit implementation, (2) a fast PSI construction for small sets [21], and (3) a threshold PSI construction [28].

2. increasing intersection cardinality $|X \cap Y|$, to observe effects of low or high number of matching elements in the case of t-PLI and t-PLI-CA.

3. increasing reconstruction threshold t, and analyzed the effect of the threshold choice on the execution time of t-PLI and t-PLI-CA.

For each instantiation of the protocol, the average of 10 code executions is reported.

7.2 Comparisons of ElGamal variants

For $n = 10$ through $n = 100$, with a step size of 10, the bandwidth and execution time of the PLI protocol were measured for EC-ElGamal, conventional ElGamal, and additively homomorphic ElGamal schemes. The results are displayed in Fig.s 7.1 and 7.2.

The use of EC-ElGamal in PLI implementation resulted in both lower bandwidth and faster execution time. Its bandwidth is more than 2× lower, while its execution time is about 20× lower than the other two ElGamal schemes at $n = 30$.

These results motivated the use of EC-ElGamal in the implementations of PLI-CA, t-PLI, and t-PLI-CA.

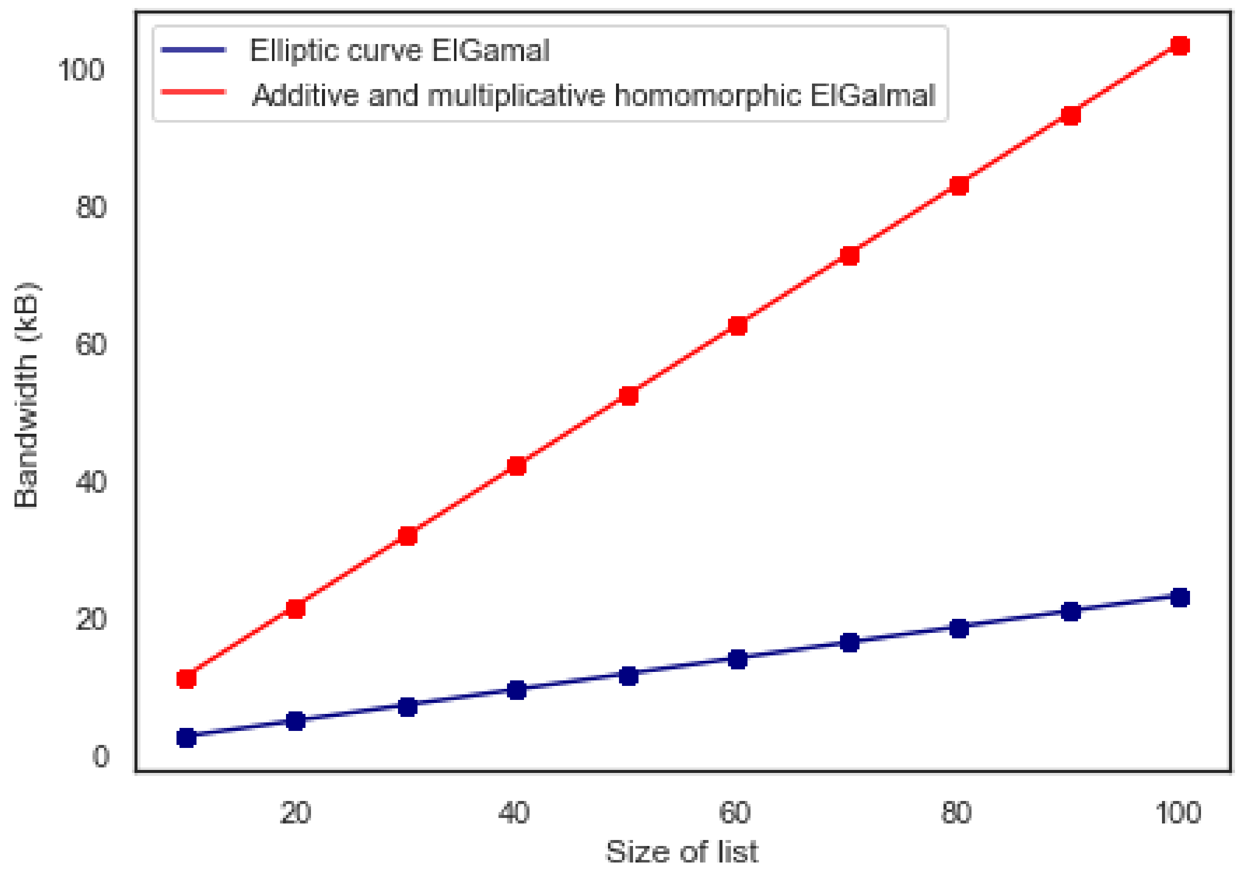


Figure 7.1: Bandwidth of PLI protocol using EC-ElGamal and conventional ElGamal schemes.

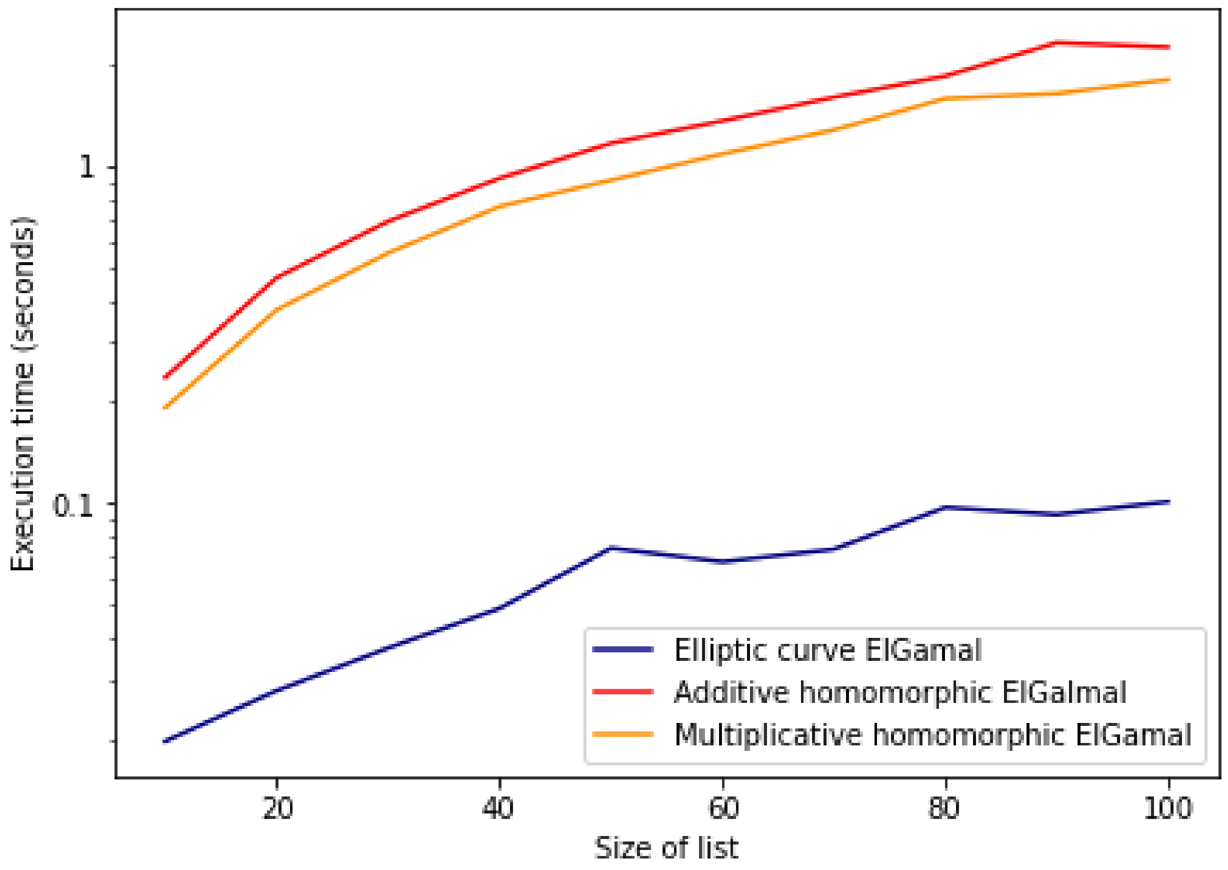


Figure 7.2: Execution time of PLI protocol using EC-ElGamal, original (multiplicatively homomorphic) ElGamal, and additively homomorphic ElGamal schemes.

7.3 Bandwidth Evaluation

For 224-bit list inputs, Fig. 7.3 showcases the bandwidth required to run each protocol for increasing values of the list size, n. We observe that the bandwidth used in the PLI and PLI-CA protocols is the same, which is consistent with the protocol constructions in Section 5. This is because the messages sent in PLI-CA and PLI are equivalent, only permuted. The same is observed for t-PLI and t-PLI-CA. The threshold PLI protocols require less bandwidth, e.g. 11.5 kB for $n = 30$, than the PLI and PLI-CA protocols, which need 14 kB for $n = 30$. This is due to $\mathcal{C}$ sending only the first element of each encrypted value $a_{i,1}$ and a one-time pad of the hash of the second element $H(a_{i,2})$ in the threshold protocols, instead of sending $(a_{i,1}, a_{i,2})$ in the non-threshold protocols.

Compared to MP-SPDZ garbled circuits with 32-bit list inputs, the bandwidth required by PLI constructions with EC-ElGamal encryptions is lower. For example, for $n = 30$, the garbled circuit versions of PLI, PLI-CA, t-PLI, and t-PLI-CA require 60.4 kB, 124.8 kB, 187.4 kB and 127.0 kB, respectively. The bandwidth required for the garbled circuit implementations is illustrated in Fig. 7.4. Since the evaluation through MP-SPDZ outputs errors for higher values of n, we present the dashed lines for the projected values for $n = 50$ and $n = 60$.

The communication overhead does not change as the threshold varies in the t-PLI and t-PLI-CA protocols, since $\mathcal{C}$ must still receive $\mathcal{S}$'s entire list, and send it back to $\mathcal{S}$.

7.4 Execution Time Evaluation

We first evaluated the running time of each protocol as the list size n increased from 10 to 100, with step size 5. Shown in Fig. 7.5, the execution time of PLI and PLI-CA increase

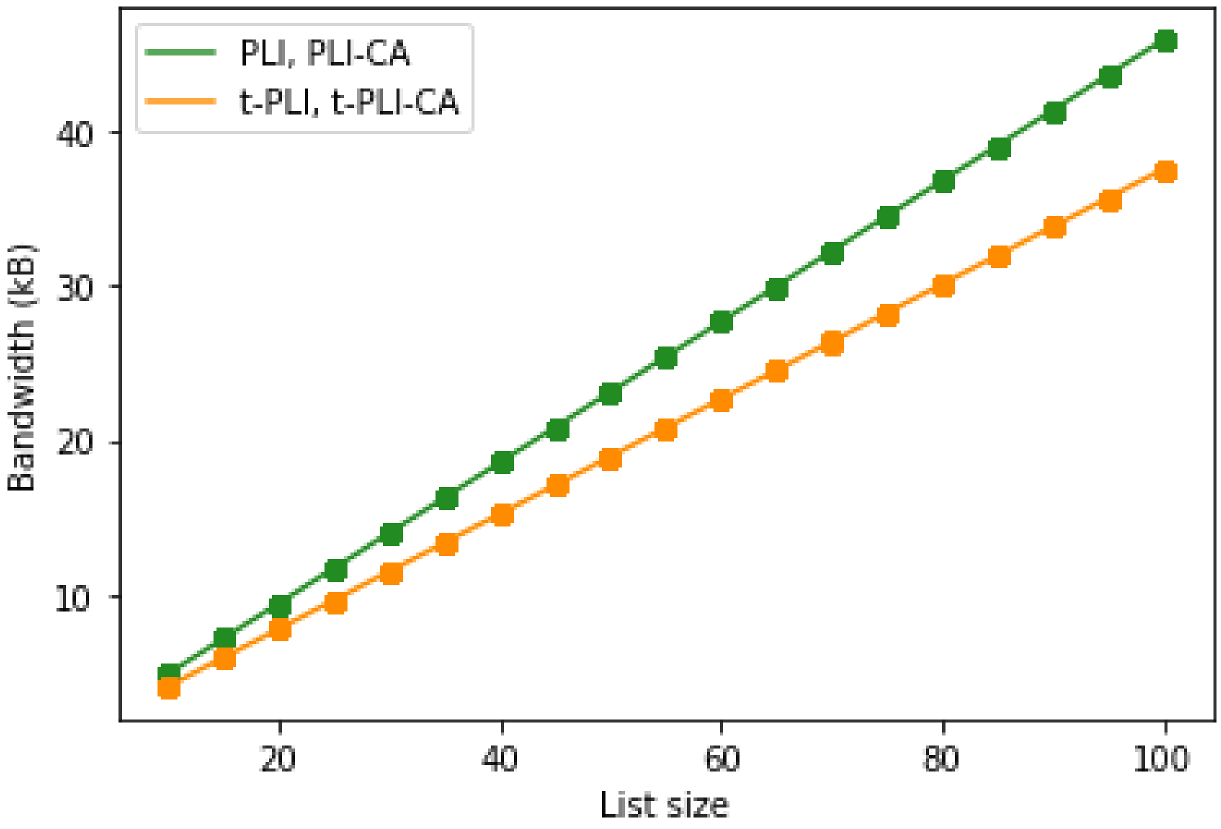


Figure 7.3: Bandwidth consumed by all protocols using EC-ElGamal encryption.

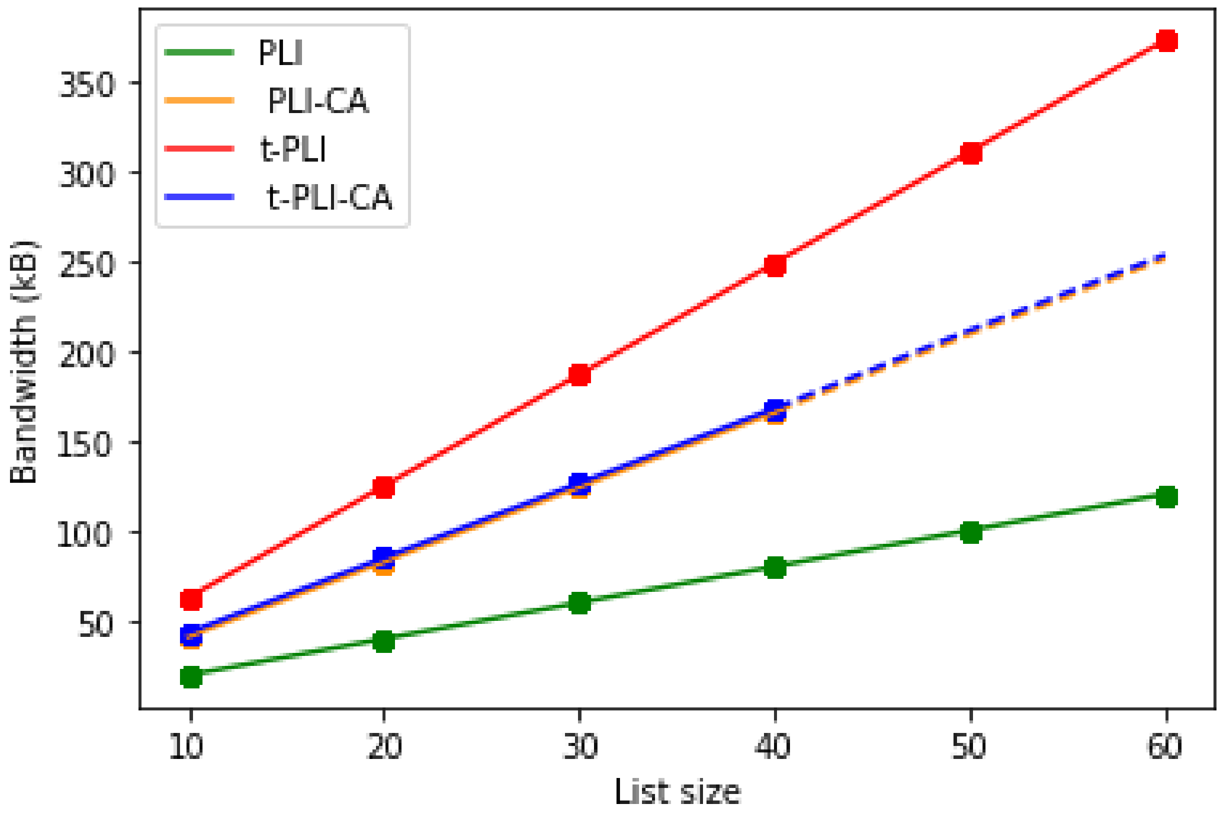


Figure 7.4: Bandwidth consumed by all protocols using garbled circuits in MP-SPDZ [16].

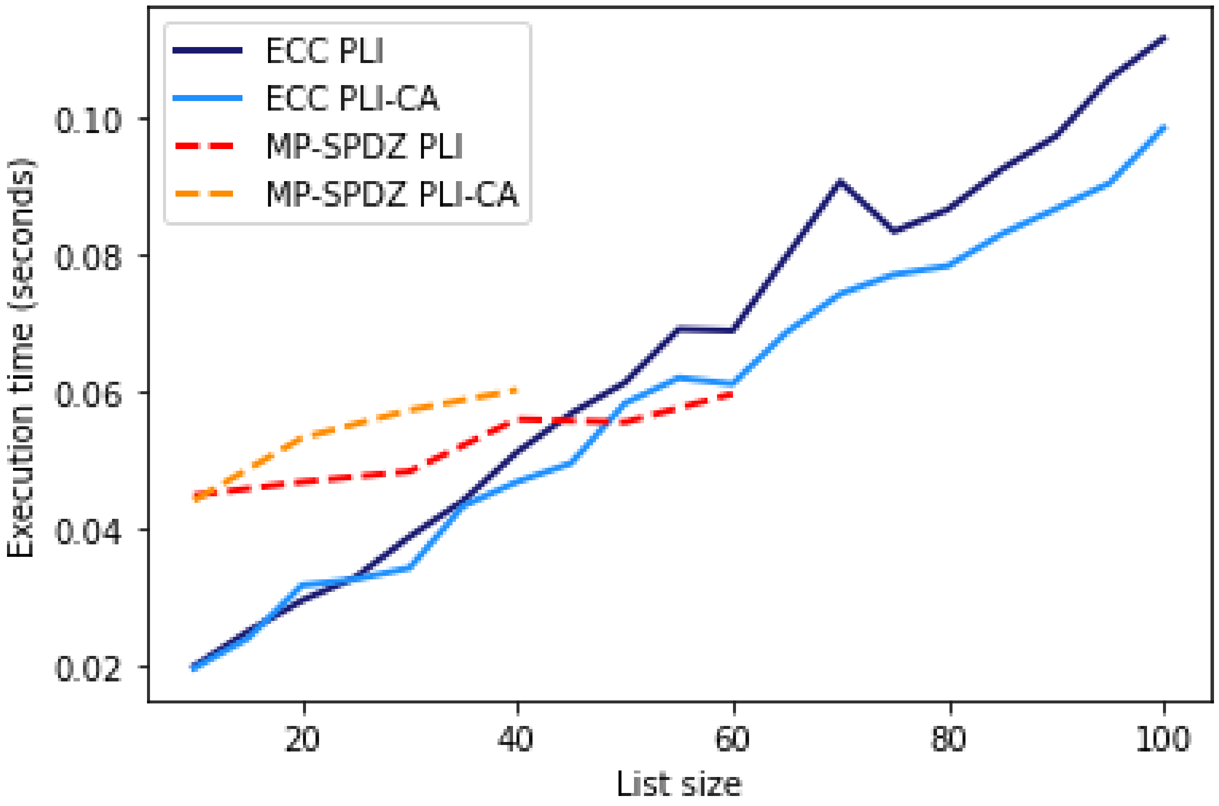


Figure 7.5: Execution time of PLI and PLI-CA protocols for increasing list size, for both the ECC-based implementation and the MP-SPDZ garbled circuits implementation.

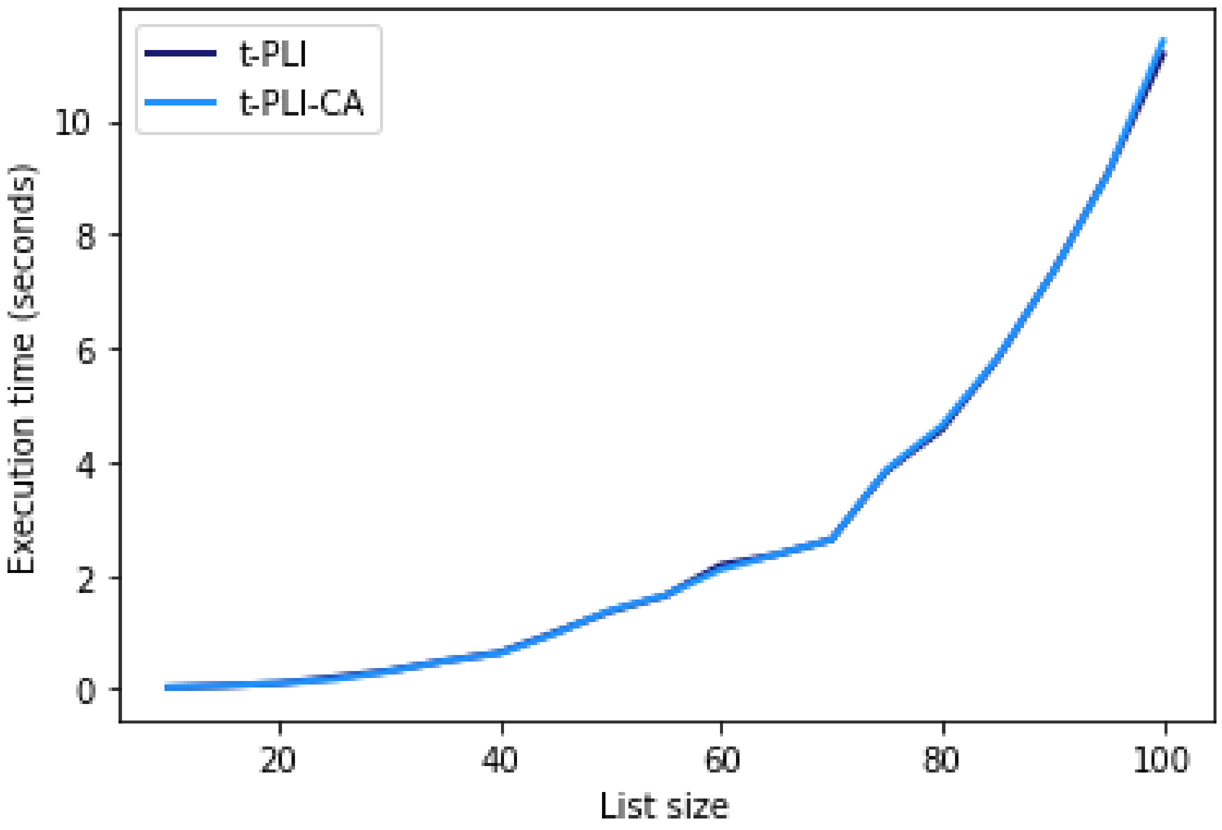


Figure 7.6: Execution time of the t-PLI and t-PLI-CA protocols for increasing list size.

linearly as n increases. For $n = 30$, PLI and PLI-CA take on average 0.039 and 0.034 seconds to complete. The garbled circuit constructions of PLI and PLI-CA, also displayed in Fig. 7.5, are less efficient than our constructions for $n \leq 40$. Correspondingly, for $n = 30$, the garbled circuit-based PLI implementation executes in 0.048 seconds, while one for PLI-CA executes in 0.057 seconds. A fast PSI construction for small sets [21] executes in 0.062 seconds for a set size of 128 on a machine with a single Intel Xeon with 2.30GHz and 256GB RAM, with a 0.2ms round-trip time between parties.

Now, fixing the cardinality $|X \cap Y|$ to 80% of n and the threshold t to $\frac{1}{3}n$, we evaluate the execution time of t-PLI and t-PLI-CA, which results are displayed in Fig. 7.6. Because of the Berlekamp-Welch (BW) protocol, the execution time is $O(n^3)$, which is also reflected in the figure. For $n = 30$, t-PLI and t-PLI-CA take on average 0.32 and 0.31 seconds, while for $n = 100$, t-PLI and t-PLI-CA take on average 11.4 and 11.1 seconds. For t-PLI and t-PLI-CA, the garbled circuits approach is always faster than the ECC construction, e.g. at $n = 30$, the execution time of t-PLI is 0.055 seconds using garbled circuits. In [28], Zhao and Chow build a threshold PSI protocol that executes in 85.3 and 139.8 seconds for each party respectively, for a list size of 100 and a threshold of 50, on a machine with two Intel Core i5-4590 3.30GHz CPUs, and 8GB RAM.

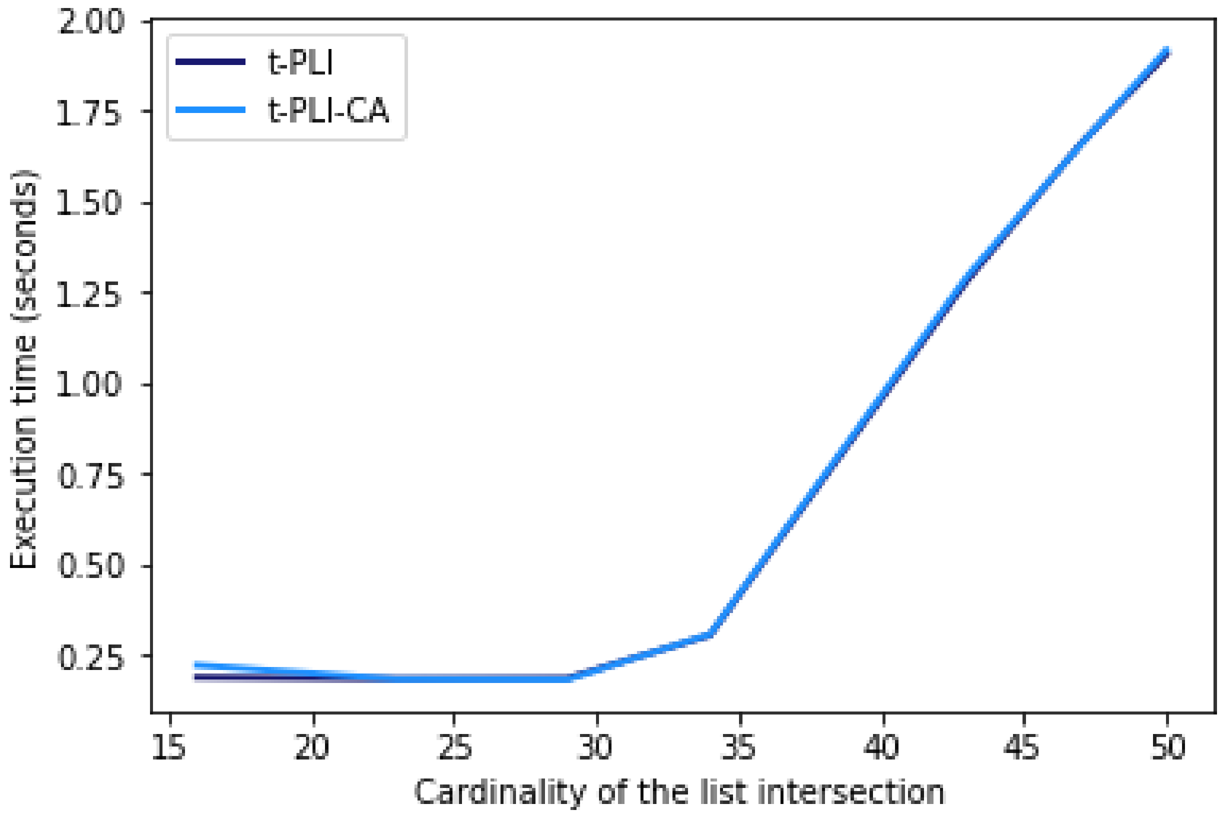


Figure 7.7: Execution time of the t-PLI and t-PLI-CA protocols for increasing list intersection cardinality, for a $n = 50$ and $t = 16$.

Then, we compare the execution time for t-PLI and t-PLI-CA as $|X \cap Y|$ increases and as t increases. In both cases, $n = 50$. In Fig. 7.7, with a t fixed to $\frac{1}{3} \cdot 50 \approx 16$, the protocol is very fast when $|X \cap Y|$ does not allow efficient reconstruction of the secret, i.e. when the BW algorithm fails. On average, when the number of matches is below $k = \lceil \frac{n+t}{2} \rceil$ (here $k = 33$), execution for both t-PLI and t-PLI-CA takes 0.19 seconds. For $|X \cap Y| \geq 35$, the

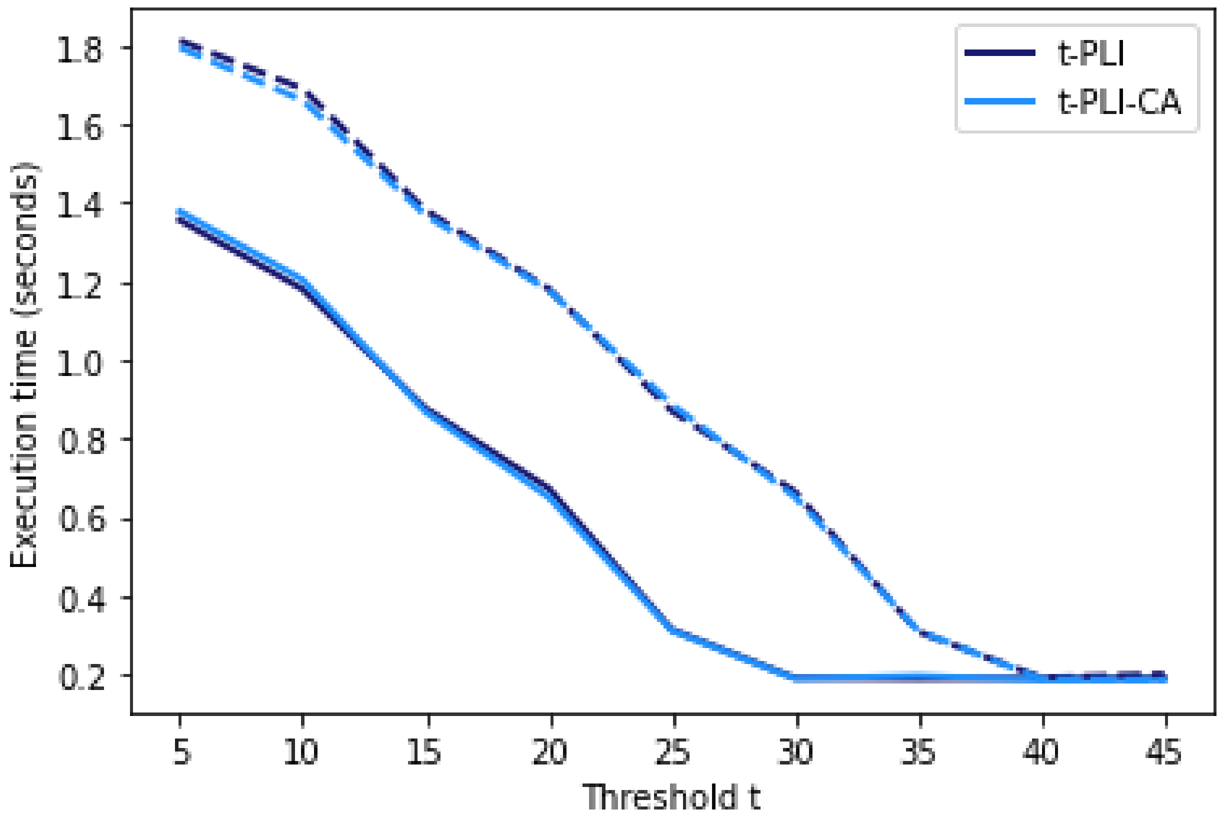


Figure 7.8: Execution time of the t-PLI and t-PLI-CA protocols for increasing threshold, for $n = 50$. For solid lines, $|X \cap Y| = 39$, for dashed lines, $|X \cap Y| = 44$.

BW algorithm succeeds, and the t-PLI and t-PLI-CA protocols take a linearly increasing amount of time. This is because the BW algorithm starts by assuming the maximum number of errors $\lfloor \frac{n-t}{2} \rfloor$, and tries to recover the secret. If it fails to recover the secret, it decreases the assumed number of errors by one unit, tries again, and repeats until successful recovery. Thus, for a higher intersection cardinality $|X \cap Y|$, there are fewer errors, and running the BW algorithm requires more trials.

We observe a similar effect in Fig. 7.8, where the execution time first decreases linearly as the threshold t increases, i.e. as $\lfloor \frac{n-t}{2} \rfloor$ decreases. Then, when t gets too high and efficient reconstruction is impossible, the BW algorithm fails and the t-PLI and t-PLI-CA protocols take about 0.19 seconds to complete.

Chapter 8

Conclusion

This paper proposes $\mathcal{P}\mathsf{IVA}$, privacy-preserving identity verification for accountless users, based on four cryptographic primitives: PLI, PLI-CA, t-PLI, and t-PLI-CA. We constructed a protocol for each primitive and analyzed their security and performance with the prototype implementations. The proposed protocols for PLI and PLI-CA are more efficient than the garbled circuit approach in communication and computation costs for the application of accountless users. Our t-PLI and t-PLI-CA constructions achieved lower bandwidth than garbled circuit implementations, and were faster than existing threshold PSI protocols.

Bibliography

[1] Supriya Adhatarao, Cédric Lauradoux, and Cristiana Santos. Why ip-based subject access requests are denied? *arXiv preprint arXiv:2103.01019*, 2021.

[2] Anonymous Author(s). Anonymous repository for piva, 2023. `https://anonymous.4open.science/r/PLI-D874/README.md`.

[3] Anonymous Author(s). Piva: Privacy-preserving identity verification methods for accountless users via private list intersection and its variants. Submitted to the 19th ACM ASIA Conference on Computer and Communications Security (ACM AsiaCCS 2024), 2023.

[4] Elwyn R Berlekamp and Lloyd R Welch. Error correction for algebraic block codes, Dec 1986.

[5] European Data Protection Board. Guidelines 01/2022 on data subject rights - right of access, version 2.0, 2023.

[6] Coline Boniface, Imane Fouad, Nataliia Bielova, Cédric Lauradoux, and Cristiana Santos. Security analysis of subject access request procedures: How to authenticate data subjects safely when they request for their data. In *Privacy Technologies and Policy: 7th Annual Privacy Forum, APF 2019, Rome, Italy, June 13–14, 2019, Proceedings 7*, pages 182–209. Springer, 2019.

[7] Brazil. Lei nº 13.709, de 14 de agosto de 2018, 2018.

[8] Luca Bufalieri, Massimo La Morgia, Alessandro Mei, and Julinda Stefa. Gdpr: when the right to access personal data becomes a threat. In *2020 IEEE International Conference on Web Services (ICWS)*, pages 75–83. IEEE, 2020.

[9] Ronald Cramer, Rosario Gennaro, and Berry Schoenmakers. A secure and optimally efficient multi-authority election scheme. In Walter Fumy, editor, *Advances in Cryptology — EUROCRYPT '97*, pages 103–118, Berlin, Heidelberg, 1997. Springer Berlin Heidelberg.

[10] Mariano Di Martino, Pieter Robyns, Winnie Weyts, Peter Quax, Wim Lamotte, and Ken Andries. Personal information leakage by abusing the {GDPR}'right of access'. In *Fifteenth Symposium on Usable Privacy and Security (SOUPS 2019)*, pages 371–385, 2019.

[11] Ronald Aylmer Fisher and Frank Yates. *Statistical tables for biological, agricultural, and medical research.* Hafner Publishing Company, 1953.

[12] Taher El Gamal. A public key cryptosystem and a signature scheme based on discrete logarithms. *IEEE Trans. Inf. Theory*, 31(4):469–472, 1985.

[13] California Attorney General. California consumer privacy act regulations, 2020.

[14] Oded Goldreich. *Foundations of Cryptography, volume 2.* Cambridge University Press, 2004.

[15] Scott Jordan, Yoshimichi Nakatsuka, Ercan Ozturk, Andrew Paverd, and Gene Tsudik. VICEROY: gdpr-/ccpa-compliant enforcement of verifiable accountless consumer requests. In *30th Annual Network and Distributed System Security Symposium, NDSS 2023, San Diego, California, USA, February 27 - March 3, 2023.* The Internet Society, 2023.

[16] Marcel Keller. MP-SPDZ: A versatile framework for multi-party computation. In *Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security*, 2020.

[17] California Legislature. Title 1.81.5. california consumer privacy act of 2018, 2018.

[18] Mariano Di Martino, Isaac Meers, Peter Quax, Ken Andries, and Wim Lamotte. Revisiting identification issues in GDPR 'right of access' policies: A technical and longitudinal analysis. *Proc. Priv. Enhancing Technol.*, 2022(2):95–113, 2022.

[19] European Parliament and Council. Regulation (eu) 2016/679, general data protection regulation, 2016.

[20] I. S. Reed and G. Solomon. Polynomial codes over certain finite fields. *Journal of the Society for Industrial and Applied Mathematics*, 8(2):300–304, 1960.

[21] Mike Rosulek and Ni Trieu. Compact and malicious private set intersection for small sets. In Yongdae Kim, Jong Kim, Giovanni Vigna, and Elaine Shi, editors, *CCS '21: 2021 ACM SIGSAC Conference on Computer and Communications Security, Virtual Event, Republic of Korea, November 15 - 19, 2021*, pages 1166–1181. ACM, 2021.

[22] Nikita Samarin, Shayna Kothari, Zaina Siyed, Oscar Bjorkman, Reena Yuan, Primal Wijesekera, Noura Alomar, Jordan Fischer, Chris Jay Hoofnagle, and Serge Egelman. Lessons in VCR repair: Compliance of android app developers with the california consumer privacy act (CCPA). *Proc. Priv. Enhancing Technol.*, 2023(3):103–121, 2023.

[23] Adi Shamir. How to share a secret. *Commun. ACM*, 22(11):612–613, 1979.

[24] Kejsi Take, Kevin Gallagher, Andrea Forte, Damon McCoy, and Rachel Greenstadt. "it feels like whack-a-mole": User experiences of data removal from people search websites. *Proc. Priv. Enhancing Technol.*, 2022(3):159–178, 2022.

[25] Tobias Urban, Dennis Tatang, Martin Degeling, Thorsten Holz, and Norbert Pohlmann. The unwanted sharing economy: An analysis of cookie syncing and user transparency under gdpr. *arXiv preprint arXiv:1811.08660*, 2018.

[26] Tobias Urban, Dennis Tatang, Martin Degeling, Thorsten Holz, and Norbert Pohlmann. A study on subject data access in online advertising after the gdpr. In *Data Privacy Management, Cryptocurrencies and Blockchain Technology: ESORICS 2019 International Workshops, DPM 2019 and CBT 2019, Luxembourg, September 26–27, 2019, Proceedings 14*, pages 61–79. Springer, 2019.

[27] Andrew Chi-Chih Yao. How to generate and exchange secrets (extended abstract). In *27th Annual Symposium on Foundations of Computer Science, Toronto, Canada, 27-29 October 1986*, pages 162–167. IEEE Computer Society, 1986.

[28] Yongjun Zhao and Sherman S. M. Chow. Can you find the one for me? In David Lie, Mohammad Mannan, and Aaron Johnson, editors, *Proceedings of the 2018 Workshop on Privacy in the Electronic Society, WPES@CCS 2018, Toronto, ON, Canada, October 15-19, 2018*, pages 54–65. ACM, 2018.