$\mathcal{P}$IVA: Privacy-Preserving Identity Verification Methods for Accountless Users via Private List Intersection and Variants

Seoyeon Hwang, Stanislaw Jarecki, Zane Karl, Elina van Kempen, Gene Tsudik

University of California, Irvine
evankemp@uci.edu

Abstract. Several prominent privacy regulation (e.g., CCPA and GDPR) require service providers to let consumers request access to, correct, or delete, their personal data. Compliance necessitates verification of consumer identity. 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. However, there are no such methods for accountless consumers, even though service providers routinely collect data about casual consumers, i.e., those without accounts. Currently, in order to access their collected data, accountless consumers are asked to provide Personally Identifiable Information (PII) to service providers, which is privacy-invasive.

To address this problem, we propose $\mathcal{P}$IVA*: Privacy-Preserving Identity Verification for Accountless Users*, a technique based on *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 suspicious parties to compute the intersection of their private input sets. PLI takes advantage of the (ordered and fixed) list structure of each party's private set. As a result, PLI is 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 $\mathcal{P}$IVA and compare its performance against techniques based on regular PSI and garbled circuits (GCs). Results show that proposed PLI and PLI-CA constructions are more efficient than GC-based techniques, in terms of both computation and communication overheads. While GC-based t-PLI and t-PLI-CA execute faster, proposed constructs greatly outperform the former in terms of bandwidth, e.g., our t-PLI protocol consumes 16× less bandwidth. We also show that proposed protocols can be made secure against malicious adversaries, with only moderate increases in overhead. These variants outperform their GC-based counterparts by at least one order of magnitude.

Keywords: Identity Verification · Authentication · Accountless Users · Private List Intersection · Threshold Private List Intersection · Cardinality

1 Introduction

Notable privacy regulations – including the European Union General Data Protection Regulation (GDPR) [15], California Consumer Privacy Act (CCPA) [8], and Brazilian Lei Geral de Proteção de Dados (LGPD) [5] – mandate that service providers which collect any consumer-specific data must grant consumers access to that data [1]. To allow consumers to access, correct, or delete their personal data, a service provider must employ secure identity verification methods.

For instance, CCPA [8] 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)). CCPA also 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)).

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 [14] 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" [8].

On the other hand, for *account-less* users (i.e., those without accounts with a service provider) authentication options are limited. 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 [12,24,31,4,6], (2) providing cookies from prior interactions [35,36], (3) showing government-issued IDs [12,24,31,4,35,36,6], or (4) producing notarized documents [31,35,36,6]. A recent user study [33] that studies user experience 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 of authenticating accountless users.

Motivated by the above, this paper proposes $\mathcal{P}$IVA*: Privacy-Preserving Identity Verification for Accountless 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 a sufficiently many list elements are matched as the intersection of their respective PII lists. Meanwhile, $\mathcal{S}$ is prevented from learning any new $\mathcal{C}$ PII and vice-versa.

[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)) [8].

As a basic building block, we use *Private List Intersection (PLI)*, a variant of a well-known cryptographic primitive called *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., it takes list order into account. 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 intersection cardinality only if it exceeds a threshold t. We implement these protocols and evaluate their efficiency. All variants outperform the GC-based implementation in terms of bandwidth, and the PLI and PLI-CA execute faster. The entire implementation is publicly available at [1].

Finally, we propose a modified version of each protocol that achieves security against malicious (as opposed to HbC: Honest-but-Curious) adversaries. These modifications result in relatively low additional communication and computation costs. In particular, the asymptotic overhead of each modified protocol remains the same. Notably, our work yields the first threshold-based PSI and PSI-CA protocols secure in the malicious model, which might be of independent interest.

Organization: After reviewing related work in Section 2, Section 3 presents the preliminary background, and Section 4 describes system and threat models. Next, we present $\mathcal{P}$IVA and PLI variants in Section 5; their security analyses appear in Appendix A. We describe the implementation and evaluation in Section 6. Finally, variants of proposed protocols secure in the malicious model are presented in Section 7.

2 Related Work

Current Privacy-Invasive Authentication. Several surveys [12,24,31,4,35,36,6] investigate current authentication methods that service providers use for data requests submissions. These results show that the information requested is often privacy-invasive [4].

For example, [12] and [24] 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, [6] 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 [4] shows that some companies demand additional PII from users.

Similar results are observed in Android applications. For instance, [31] 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 some PII, e.g., the user's name, address, phone number, Android Advertising ID, date of birth, or a signed affidavit.

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, [2] shows that while IP addresses are considered PII, they may not be sufficient to authenticate a user since they can be spoofed or attributed to more than one user.

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

Another approach is outlined in [22], 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, however, there is no prior work investigating privacy-preserving authentication for users who have never interacted with the service providers.

Private Set Intersection and Its Variants. Private Set Intersection (PSI) protocols, e.g. [30,21,29,27], allow two parties to compute intersection $X \cap Y$ of their private input sets X and Y, without revealing any other information. Works on PSI include PSI-cardinality (PSI-CA), e.g. [11,13,34], which reveals only cardinality $|X \cap Y|$ of the intersection, and hides the intersecting elements, and threshold-PSI (t-PSI), e.g. [38,26,19], which reveals the intersection only if $|X \cap Y| \geq t$ for some fixed threshold t.

Table 1: Comparison of the computational complexity of PLI and PSI variants as a function of n, the size of input lists or sets.[2]

	Comp. Cost	**Active Security**
PSI [30,21,29,27]	$O(n)$	✓
PLI (ours)	$O(n)$	✓
t-PSI [38]	$O(n)$	✗
t-PSI [26]	$O(n^2)$	✗
t-PSI [19]	$O(t)$	✗
t-PLI (ours)	$O(n^3)$	✓

	Comp. Cost	**Active Security**
PSI-CA [11,13]	$O(n)$	✗
PSI-CA [34]	$O(n \log n)$	✗
PLI-CA (ours)	$O(n)$	✓
t-PLI-CA (ours)	$O(n^3)$	✓

Literature on PSI is very rich, but instead of citing all work in PSI literature, we summarize state-of-the-art PSI results and compare them to our PLI protocols in Table 1. Asymptotically, our PLI and PLI-CA protocols match the

[2] To the best of our knowledge, there is no prior work on t-PSI-CA. [39] presents a protocol they call "t-PSI-CA", but it reveals $X \cap Y$ if $|X \cap Y| \geq t$, and $|X \cap Y|$ otherwise, whereas our t-PLI-CA reveals $|X \cap Y|$ if $|X \cap Y| \geq t$ and $\perp$ otherwise.

$O(n)$ computation cost of prior PSI and PSI-CA protocols, while our t-PLI has $O(n^3)$ costs which is higher than t-PSI of [38,26,19]. However, all our protocols achieve *active security*, i.e. security against malicious participants, at the same asymptotic costs (and a small increase to concrete costs), whereas $O(n)$ actively secure PSI is only known as the basic PSI variant, and not for PSI-CA or t-PSI. Active secure (t-)PLI(-CA) can be implemented using generic constant-round actively secure computation using "cut-and-choose" over garbled circuits (see e.g. [23]), but the "cut-and-choose" technique increases protocol costs by two orders of magnitude, rendering them impractical.

3 Preliminaries

We use the notation $[n]$ to represent the set $\{1, 2, \ldots, n\}$. 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 $\forall i \in [n]$, i.e., $x_i := (x_{i,1}, x_{i,2})$, where $i = 1, ..., n$.

Additively Homomorphic ElGamal Encryption. Below, we define the additively homomorphic variant of the ElGamal encryption scheme [10]. Compared to the original ElGamal [18] which is multiplicatively homomorphic, this variant uses g^m in place of the plaintext m in encryption, where g is a generator of a group G of prime order q.

Definition 1. *Additively homomorphic ElGamal encryption [10] is a tuple of algorithms* $(\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Enc}, \mathsf{PreDec}, \mathsf{Test})$*, where:*

- $\mathsf{Setup}(1^\lambda) \to (G, q, g)$*: for security parameter* λ*, it outputs parameters* (G, q, g)*, where* g *is a generator of group* G *with prime order* q *s.t.* $|q| \geq 2\lambda$*.*
- $\mathsf{KeyGen}(G, q, g) \to (sk, pk)$*: outputs a pair of secret and public keys, where* $sk \leftarrow_\$ Z_q$ *and* $pk = (G, q, g, h)$ *for* $h = g^{sk}$*.*
- $\mathsf{Enc}_{pk}(m) \to c$*: On message* $m \in Z_q$ *and public key* $pk = (G, q, g, h)$*, it outputs ciphertext* $c := (g^w, h^w g^m)$*, where* $w \leftarrow_\$ Z_q$*.*
  (We write $\mathsf{Enc}_{pk}(m; w)$ *to denote randomness* w *as an explicit input.)*
- $\mathsf{PreDec}_{sk}(c) \to M$*: On secret key* sk *and ciphertext* $c = (c_1, c_2)$*, output* $M := c_2(c_1)^{-sk}$*. (Note that if* $(c_1, c_2) = \mathsf{Enc}_{pk}(m)$ *then* $M = g^m$*.)*
- $\mathsf{Test}(M, m)$*: Decide if the pre-decryption value* M *corresponds to plaintext target* m *by outputting* 1 *if* $M = g^m$ *and* 0 *otherwise.*

Note that this version of ElGamal does not allow for efficient decryption, but it allows for efficient testing if a ciphertext encrypts a given plaintext m. In our PLI protocol and its (t-)PLI(-CA) variants, the decryptor will only test whether the ciphertext encrypts $m = 0$, by checking if $c_2(c_1)^{-sk} = 1$.

Shamir Secret Sharing. Shamir secret sharing [32] is a t-out-of-n threshold secret sharing scheme based on polynomial interpolation over finite fields, with a threshold t. i.e., When a secret s is shared using Shamir secret sharing, s can be reconstructed only with $\geq\ t$ shares, while the possession of $< t$ shares reveals no information about s.

Definition 2. *Shamir secret sharing scheme [32] over field $\mathbb{F}$ is defined by algorithms* $\mathsf{Share}, \mathsf{Reconstruct}$ *s.t.*

- $\mathsf{Share}_{(t,n)}(s) \rightarrow (s_1, ..., s_n)$*: generates n shares of a secret $s \in \mathbb{F}$, assuming $n < |F|$. It chooses $(t-1)$ random coefficients $a_i \leftarrow_{\$} \mathbb{F}$, for $i \in [t-1]$, defines $(t-1)$-degree polynomial $f(x) = s + a_1 x + a_2 x^2 + ... + a_{t-1} x^{t-1}$ over $\mathbb{F}$, and sets $s_1 := f(1), s_2 := f(2), ..., s_n := f(n)$.*
- $\mathsf{Reconstruct}(s_{i_1}, ..., s_{i_m}) \rightarrow s$*: reconstructs a secret from $m \geq t$ shares $s_{i_1}, ..., s_{i_m}$. It interpolates any t out of these m shares into a $(t-1)$-degree polynomial f' and outputs $s' := f'(0)$ as the reconstructed secret.*

Note that $\mathsf{Reconstruct}$ guarantees to output $s' = s$ if the m shares it receives as input were generated by $\mathsf{Share}(s)$.

Reed-Solomon Codes and Berlekamp-Welch Decoding. Shamir secret sharing scheme can be seen as a special case of Reed-Solomon (RS) coding scheme [28], 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 $t-1$ over a finite field $\mathbb{F}$, 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 $p(x) := m_1 + m_2 x + ..., m_t x^{t-1}$. 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 [3] 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 over $\mathbb{F}$ which represents 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 shared by the Shamir secret sharing, where the input shares potentially include some invalid shares, i.e., errors.

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.

Considering that $\mathcal{S}$ may have a large database of clients, we assume that $\mathcal{S}$ locates $\mathcal{C}$ in its database using a unique public string, e.g. in the context of people search websites, using the URL of the webpage about $\mathcal{C}$.

In Section 5, we consider a static *Honest-but-Curious* (HbC) adversary who corrupts either $\mathcal{C}$ or $\mathcal{S}$ before the protocol starts, and aims to learn maximal information about the other party's private input, but otherwise follows the protocol. In Section 7, we consider *active*, a.k.a. *malicious* adversaries, i.e. we assume that an adversary who corrupts either party can run an arbitrary protocol in place of the prescribed one, and we show that our protocols realize the intended ideal functionality even in the presence of malicious adversaries.

5 $\mathcal{P}$IVA Design

In this section, we present four protocol variants for $\mathcal{P}$IVA, i.e. PLI, PLI-CA, t-PLI, and t-PLI-CA. We describe each variant below, summarizing their security properties with ideal functionalities described in Fig. 3 in Subsection 5.3. Due to the space limits, all security proofs are deferred to Appendix A.

5.1 Private List Intersection (PLI) and PLI-Cardinality (PLI-CA)

PLI: Recall that (one-sided) PSI takes private *sets* as input and outputs their intersection set to one of the parties. PLI takes private *lists* as input instead. i.e. the inputs are two lists, X and Y, of size n, contributed by $\mathcal{C}$ and $\mathcal{S}$, respectively. Entries in list X and Y are arbitrary bitstrings or a special symbol $\perp$ which designates that the corresponding party does not have any value at this position.

The protocol computes and sends to $\mathcal{S}$ the list intersection $X \cap Y$, defined as the set of indexes where the two lists contain the same values, i.e.

$$X \cap Y \;=\; \{\, i \;\mid\; X[i] = Y[i] \;\wedge\; X[i] \neq \perp \,\}$$

(See also the Ideal Functionality for PLI in Fig. 3 in Section 5.3.) In the context of our application, each position in list X or Y represents a specific type of PII, e.g., home address, driver's license number, mother's maiden name, etc, and we assume that the ordered list of these n attributes is fixed and public.

After obtaining the intersection, $\mathcal{S}$ decides whether the $\mathcal{C}$-provided PII is sufficient to verify their identity corresponding to the PII held by $\mathcal{S}$. As part of PLI, $\mathcal{S}$ learns no new PII about $\mathcal{C}$. This follows GDPR guidelines [14] 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",

We construct a PLI protocol using the additively homomorphic ElGamal variant of Section 3. We assume that values contained in input lists X and Y are integers in Z_q. Since $q \geq 2\lambda$ where λ is the security parameter, longer values can be hashed into Z_q using a collision-resistant hash. If $X[i]$, resp. $Y[i]$, is an empty symbol $\perp$, the corresponding party replaces it with a random value in Z_q. (Using a random value at some index prevents either accidental or malicious attempt at creating a match at that index, except for negligible probability $1/q$.)

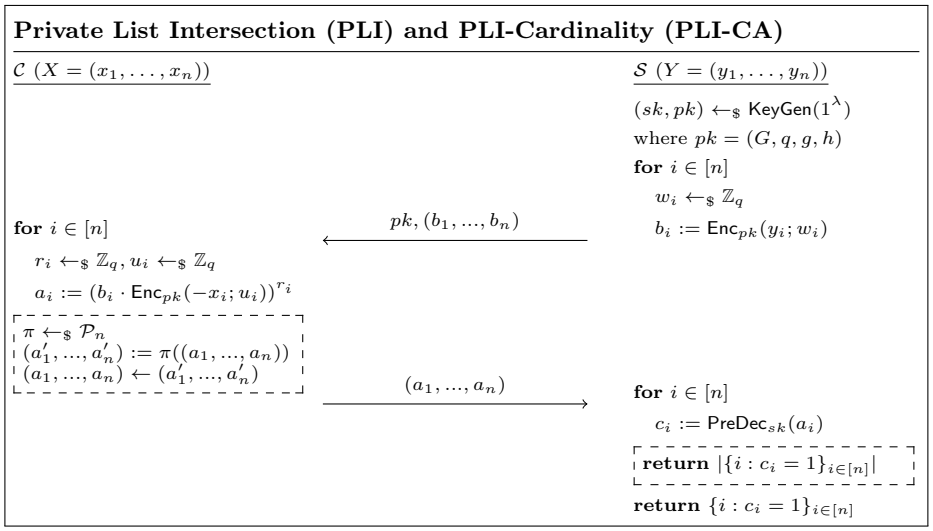


Fig. 1: PLI and PLI-CA protocols based on ElGamal in Def. 1, where $\cdot$ in a_i computation is the element-wise product, i.e., $x \cdot y = (x_1 y_1, x_2 y_2)$, for some tuples $x = (x_1, x_2)$ and $y = (y_1, y_2)$, and $\mathcal{P}_n$ is the set of all pseudorandom permutations of size n. PLI-CA protocol includes the steps in the dashed box, while PLI protocol does not.

The PLI protocol is shown in Fig. 1, and it works as follows. $\mathcal{S}$ first generates a private-public key pair (sk, pk) for ElGamal encryption, and encrypts its each element y_i under the public key. i.e., $b_i := \mathsf{Enc}_{pk}(y_i; w_i) = (g^{w_i}, h^{w_i} g^{y_i})$, where w_i is chosen at random in $\mathbb{Z}_q$, for all $i \in [n]$. Then, $\mathcal{S}$ 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, $\mathcal{C}$ encrypts the additive inverse of each element x_i in its list, using pk. i.e., $\mathsf{Enc}_{pk}(-x_i; u_i) = (g^{u_i}, h^{u_i} g^{-x_i})$, where u_i is chosen at random in $[q-1]$, for all $i \in [n]$. Then, $\mathcal{C}$ multiplies the resulting values by b_i's in the list order and re-randomizes them with a random r_i to hide its input values. i.e., For a randomly chosen $r_i \in G$, $\mathcal{C}$ computes $a_i := ((b_{i,1} \cdot g^{u_i})^{r_i}, (b_{i,2} \cdot h^{u_i} g^{-x_i})^{r_i})$ for all $i \in [n]$. Lastly, $\mathcal{C}$ sends $(a_1, ..., a_n)$ to $\mathcal{S}$ in order.

Now, $\mathcal{S}$ partially decrypts the received ciphertexts to see if they are encryption of zero. i.e., $\mathcal{S}$ computes $c_i := \mathsf{PreDec}_{sk}(a_i) = \frac{a_{i,2}}{a_{i,1}^{sk}}$, and checks if it is equal to 1 or not, for each a_i. Finally, $\mathcal{S}$ outputs all i's s.t. $c_i = 1$, which is the intersection $X \cap Y$. This is because a_i is the encryption of $(y_i - x_i) r_i$ as follows:

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

Consequently, only if $y_i = x_i$, $\mathcal{S}$ gets $c_i = g^{(y_i - x_i)r_i} = g^0 = 1$. Otherwise, it gets a random c_i and learns nothing about x_i's. Due to the space limit, we provide the proof of the following theorem in Appendix A. The PLI protocol is similar to the first protocol in [25].

Theorem 1. *The protocol presented in Fig. 1 securely computes the list intersection in the presence of HbC adversaries under the decisional Diffie-Hellman (DDH) problem. It requires $O(n)$ communication and computation costs for both sides, where n is the input list size.*

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 suggest using PLI-cardinality (PLI-CA), which outputs only the cardinality of the intersection. i.e., It reveals only the number of intersecting elements, while keeping their locations private. Using PLI-CA for $\mathcal{P}$IVA is suitable because 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 ($\mathcal{C}$).

We build a PLI-CA protocol (in Fig. 1 with the dashed box) atop the PLI protocol. To hide the locations of matching elements, $\mathcal{C}$ randomly shuffles the computation results using a pseudorandom permutation π and sends the permuted results. $\mathcal{S}$ 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 $\mathcal{S}$. Similar to PLI, we derive the following theorem, and present the proof in Appendix A.

Theorem 2. *The protocol presented in Fig. 1 including the procedure of the dashed box securely computes the cardinality of list intersection in the presence of HbC adversaries under the DDH problem. It requires $O(n)$ communication and computation costs for both sides, where n is the input list size.*

5.2 Threshold PLI (t-PLI) and t-PLI-Cardinality (t-PLI-CA)

While PLI and PLI-CA output their results to $\mathcal{S}$ no matter what, threshold PLI (t-PLI) and t-PLI-Cardinality (t-PLI-CA) output the intersection to $\mathcal{S}$ only if the number of common data exceeds some agreed-upon (fixed a priori) threshold t, i.e., $|X \cap Y| \geq t$. Therefore, it does not reveal anything to $\mathcal{S}$ if either party does not have sufficient PII.

Note that 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 [7].

t-PLI: We add t-out-of-n Shamir Secret Sharing to the PLI protocol in Section 5.1, i.e., after computing a_i's, $\mathcal{C}$ randomly chooses a secret value s and runs

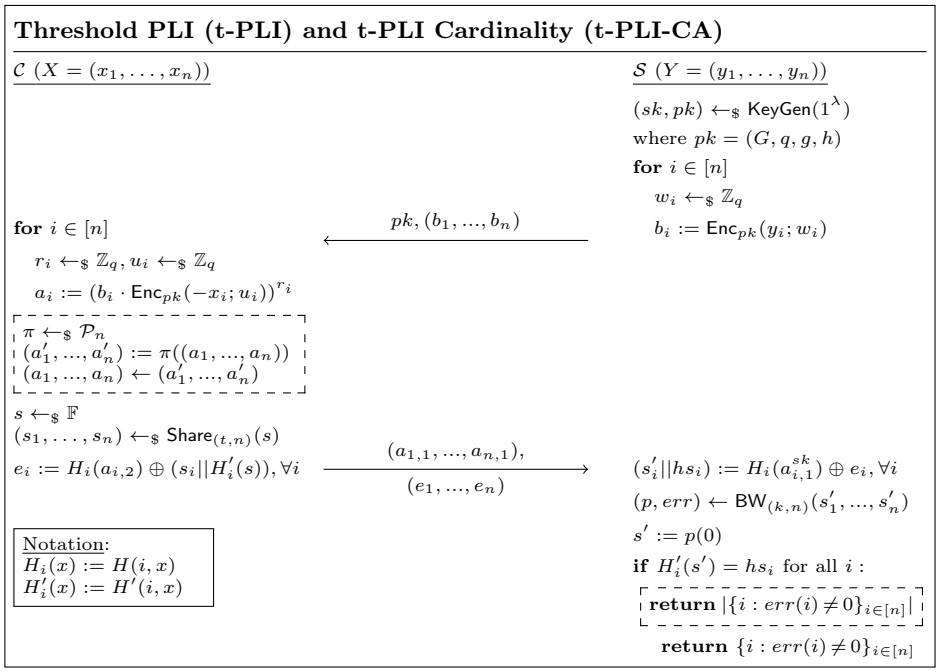


Fig. 2: t-PLI and t-PLI-CA protocols using Shamir secret sharing (Def. 2) and Berlekamp-Welch decoder BW (Section 3). Here $\oplus$ denotes XOR, $||$ denotes string concatenation, and $\mathcal{P}_n$ is the set of all permutations of size n. t-PLI-CA protocol includes the steps in the dashed box, while t-PLI protocol does not.

the Share algorithm, see Definition 2, to generate sharing $(s_1, ..., s_n)$ of s. Then, $\mathcal{C}$ xor's each i-th share with the hash of the second element of a_i's, such that $e_i := H(i, a_{i,2}) \oplus (s_i || H'(i, s))$, for each i, where H, H' are Random Oracle hash functions where $H : \{0,1\}^* \to \{0,1\}^{|\mathbb{F}|+2\lambda}$ and $H' : \{0,1\}^* \to \{0,1\}^{2\lambda}$, where $|\mathbb{F}|$ is the size of bitstrings encoding elements of field $\mathbb{F}$ used in Shamir Secret Sharing. We denote $H(i, x)$ and $H'(i, x)$ by $H_i(x)$ and $H'_i(x)$, respectively. Lastly, $\mathcal{C}$ sends the first component of a_i's, $(a_{i,1})_i$, and e_i's to $\mathcal{S}$.

Once receiving it, $\mathcal{S}$ computes $H_i(a_{i,1}^{sk})$ with its private key sk and XORs to each e_i for each i. Then, it gets the shares and hash values by dividing the computed strings, i.e., $(s'_i || hs_i) := H_i(a_{i,1}^{sk}) \oplus e_i$. Then, $\mathcal{S}$ derives s' using the Reconstruct algorithm, or the Berlekamp-Welch algorithm BW for better performance, with $(s'_1, ..., s'_n)$ as input shares. Finally, if the hash of s' with the position i matches the received hash value for all i, $\mathcal{S}$ outputs its element with the indices of correct shares. Fig. 2 shows the protocol above.

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 algorithm 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. As a result, the following theorem can be derived, with a proof given in Appendix A.

Theorem 3. *The protocol presented in Fig. 2 securely computes the list intersection only when its cardinality exceeds the threshold t in the presence of HbC adversaries under the DDH problem. It requires $O(n)$ communication bits, and $O(tn)$ $\mathcal{C}$-side computation, where n is the input list size and t is the threshold for outputting the result. $\mathcal{S}$-side requires $O(n^3)$ computation if the cardinality of intersection is greater than or equal to $\lceil \frac{n+t}{2} \rceil$, or $O(C(n,t))$ computation, otherwise, where $C(n,t)$ is the number of t-combinations over n elements.*

t-PLI-CA: We combine t-PLI and PLI-CA to further enhance privacy. At the end of t-PLI-CA, $\mathcal{S}$ learns only the cardinality of the intersection, and only if it exceeds threshold t, i.e., $\mathcal{S}$ outputs $|X \cap Y|$ only if $|X \cap Y| \geq t$.

Similarly to PLI-CA built atop the PLI construction, we construct t-PLI-CA by adding a random shuffling to the t-PLI construction. i.e., To prevent $\mathcal{S}$ from learning the indices of the matching elements, given by the error locating polynomial, $\mathcal{C}$ randomly permutes $(a_1, ..., a_n)$. As a result, $\mathcal{S}$ outputs the subtraction of the number of x-intercepts of the error locating polynomial err from the list size n. Recall that $\mathcal{S}$ can find the correct 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. 2 with the dashed box, and the security proof of the following theorem is presented in Appendix A.

Theorem 4. *The protocol presented in Fig. 2 including the procedure of the dashed box securely computes the cardinality of list intersection only when it exceeds the threshold t in the presence of HbC adversaries under the DDH problem. It requires $O(n)$ communication bits, and $O(tn)$ $\mathcal{C}$-side computation, where n is the input list size and t is the threshold for outputting the result. $\mathcal{S}$-side requires $O(n^3)$ computation if the cardinality of intersection is greater than or equal to $\lceil \frac{n+t}{2} \rceil$, or $O(C(n,t))$ computation, otherwise, where $C(n,t)$ is the number of t-combinations over n elements.*

5.3 Ideal Functionalities for PLI and its variants

Fig. 3 shows the ideal functionality of each PLI variant introduced above, i.e., private list intersection (PLI), PLI-Cardinality (PLI-CA), threshold PLI (t-PLI), and t-PLI-Cardinality (t-PLI-CA). Each functonality takes as input lists X and Y of size n from $\mathcal{C}$ and $\mathcal{S}$ respectively, and computes either the intersection (PLI and t-PLI) or the cardinality of the intersection (PLI-CA and t-PLI-CA) of the input lists, and this output is revealed to $\mathcal{S}$ party either unconditionally (PLI and PLI-CA) or only if the intersection size is above t (t-PLI and t-PLI-CA).

Parameters: List size n
Ideal Functionality of (a) PLI, (b) PLI-CA, (c) t-PLI, (d) 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 (a),(c): the list intersection $\mathsf{out}_{\mathcal{S}} = X \cap Y$
   (b),(d): the cardinality of list intersection $\mathsf{out}_{\mathcal{S}} = |X \cap Y|$
3. Send $\perp$ to $\mathcal{C}$ and (a),(b): send $\mathsf{out}_{\mathcal{S}}$ to $\mathcal{S}$
   (c),(d): if $|X \cap Y| \geq t$ send $\mathsf{out}_{\mathcal{S}}$ to $\mathcal{S}$ else send $\perp$ to $\mathcal{S}$

Fig. 3: Ideal Functionality of PLI and its (t-)PLI(-CA) variants

6 Implementation & Evaluation

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 [16], 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 (GC) [37] provided by MP-SPDZ [23].

For the homomorphic encryption schemes, we use the Elliptic Curve-based ElGamal (EC-ElGamal) with a 224-bit key provided by OpenSSL, after comparing it to the standard and the additively homomorphic ElGamal with the equivalent security level (i.e., with a 2048-bit key). From our comparisons, using EC-ElGamal in our PLI implementation has around 2× better bandwidth and 20× better latency than using the other ElGamal schemes. The list elements of the participants were 256-bit integers.

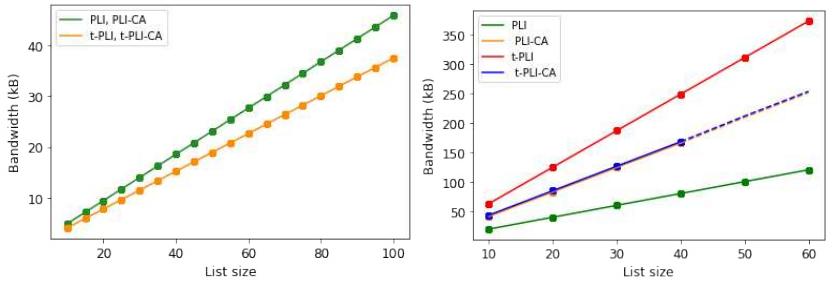
To evaluate, we measured the communication and computation costs for:

1. increasing size n of input lists, and compared it to the GC implementation.
2. increasing intersection cardinality $|X \cap Y|$, to observe the 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, we present the average of 10 code executions as a result. Our implementation codes are publicly available at [1].

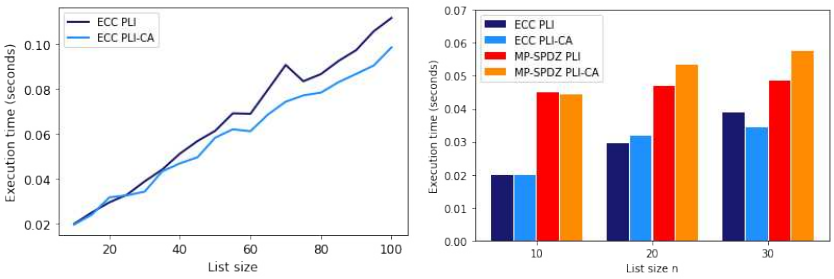
6.1 Bandwidth Evaluation

For 224-bit list inputs, Fig. 4a 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



(a) (b)

Fig. 4: Bandwidth consumed by all protocols using (a) EC-ElGamal and (b) GC in MP-SPDZ [23]. In (b), due to errors occurring for higher n, projected values are shown with dashed lines. t-PLI results coincide with t-PLI-CA.



(a) (b)

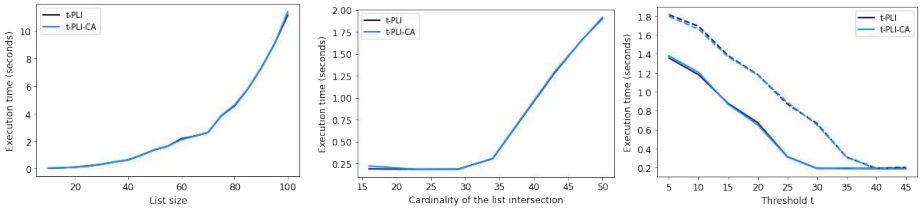
Fig. 5: Execution time of PLI and PLI-CA protocols (a) for increasing list size, and (b) compared to the GC implementation in MP-SPDZ [23]

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 $H_i(a_{i,2})$ 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 integer inputs, the bandwidth required by PLI constructions with EC-ElGamal encryptions is lower. For example, for $n = 30$, the GC 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 (shown in Fig. 4b). 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 $n = 60$.

6.2 Execution Time Evaluation

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



(a) (b) (c)

Fig. 6: Execution time of the t-PLI and t-PLI-CA protocols for (a) increasing list size, (b) increasing cardinality of intersection list, for $n = 50$ and $t = 16$, and (c) increasing threshold, for $n = 50$, $|X \cap Y| = 44$ (solid lines) and $|X \cap Y| = 39$ (dashed lines)

to the GC implementations, our constructions are more efficient in terms of execution time (Fig. 5b). For example, for $n = 30$, our PLI and PLI-CA protocols took 0.039 and 0.034 seconds on average, while the ones using GC took 0.048 and 0.057 seconds to complete.

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. 6a. 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 GC approach is always faster than ours, e.g. at $n = 30$, it takes on average 0.055 seconds to run GC-based tPLI implementation.

Lastly, we compare the execution time for t-PLI and t-PLI-CA as $|X \cap Y|$ increases and as t increases, for a fixed $n = 50$. In Fig. 6b, 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 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. 6c, 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.

7 Security against malicious participants

Each of our protocol variants enables moderate-cost upgrades to security against malicious participants, i.e., parties that arbitrarily diverge from the protocol. Recall that PLI should reveal nothing to $\mathcal{C}$ while $\mathcal{S}$ should learn (only) $X \cap Y$. This would be violated if a malicious $\mathcal{C}$ learns anything about $\mathcal{S}$'s Y and/or makes $\mathcal{S}$ output I s.t. $i \in I$ even if $X[i] \neq Y[i]$, or if a malicious $\mathcal{S}$ learns more about $\mathcal{C}$'s X beyond $X \cap Y$. We argue that this will not happen, i.e., our protocols can become maliciously secure, at a moderate increase to protocol costs. We present the modifications below and proof outlines in Appendix A.

Security against Malicious Server: All four protocol variants offer security for the client in the face of a malicious server, if the protocol is amended in two ways. First, each input x_i and y_i is replaced by resp. $\bar{x}_i = H_i^q(x_i)$ and $\bar{y}_i = H_i^q(y_i)$ where H^q is a Random Oracle (RO) hash onto Z_q and $H_i^q(x) = H^q(i, x)$. Second, $\mathcal{S}$ appends a zero-knowledge proof of knowledge (ZKPoK) of the discrete logarithm $sk = \mathsf{DL}_g(h)$ to its message $(pk, (b_1, ..., b_n))$, where $pk = (G, q, g, h)$.[3] This modification adds 1 (multi)exponentiation for $\mathcal{C}$ and $\mathcal{S}$ and 4λ bits to bandwidth, thus increasing protocol costs by only a small additive factor.

Security of PLI (in Fig. 1) against Malicious Client: can be achieved by pre-processing inputs via RO hash as above, and adding ZKPoK of (δ_i, r_i) s.t. $a_{i,1} = g^{\delta_i}(b_{i,1})^{r_i}$ for all $i \in [n]$ to $\mathcal{C}$'s message ($\delta_i = u_i r_i$ for honest $\mathcal{C}$). In the protocol, $\mathcal{S}$ outputs I s.t. $i \in I$ iff $a_{i,2} = a_{i,1}^{sk}$, but if $(h, a_{i,1}, b_{i,1}, b_{i,2}) = (g^{sk}, g^{\delta_i} b_{i,1}^{r_i}, g^{w_i}, h^{w_i} g^{\bar{y}_i})$, this is equivalent to $a_{i,2} = h^{\delta_i}(b_{i,2} g^{-\bar{y}_i})^{r_i}$, i.e., $g^{\bar{y}_i} = b_{i,2}(a_{i,2}^{-1} h^{\delta_i})^{1/r_i}$. This modification adds n (multi-)exponentiations for both $\mathcal{C}$ and $\mathcal{S}$ and $4\lambda n$ bits to bandwidth, resulting in PLI protocol with two-sided security against malicious parties, with bandwidth and computation costs that are larger than the HbC-secure version of PLI, by resp. factors $\times 1.5$ and $\times 1.4$.

Security of t-PLI (in Fig. 2) against Malicious Client: is accomplished by the same ZKPoK as for PLI above. In particular, $\mathcal{C}$ does not need to prove anything regarding the e_i values or the secret-sharing encoded by them. Since this modification is the same as above, it incurs the same overhead *compared to the PLI protocol*, i.e. it does not affect Berlekamp-Welch decoding costs.

Security of X-CA protocols against Malicious Client: The only difference between our (t-)PLI-CA and (t-)PLI protocols is that $\mathcal{C}$ permutes ciphertexts a_i before processing them further. Since all our ZKPoK's above involve only $a_{i,1}$ values, we just add a ZK *proof of shuffle*, e.g. [20]. Taking the RO-based non-interactive version of the 3-round proof-of-shuffle due to Furukawa [17] adds $15n$ exponentiations to the total computation cost and $6n\lambda$ bits to bandwidth, which together with the changes above increases the bandwidth and computation costs of the HbC versions of these protocols by resp. factors $\times 2.25$ and $\times 4.4$.

[3] All our ZKPoK proofs are variants of Schnorr's proof of discrete logarithm, made non-interactive given an RO hash. They add 2λ bits to bandwidth and 1 (multi-)exponentiation to the computation of both parties per statement, see [9].

8 Conclusion

This paper proposes $\mathcal{P}$IVA, privacy-preserving identity verification for accountless users, that includes four protocols: PLI, PLI-CA, t-PLI, and t-PLI-CA. We analyze each protocol's security and performance via a prototype implementations. Proposed PLI and PLI-CA protocols are more efficient than their GC-based counterparts in terms of communication and computation costs. Whereas, proposed t-PLI and t-PLI-CA protocols incur much less bandwidth than GC-based ones and are faster than current threshold PSI protocols. All proposed protocols can be upgraded to be secure in the malicious model with a small increase in costs.

Acknowledgements: We thank ESORICS'24 reviewers for constructive feedback. This work was supported in part by funding from the NSF Award SATC-1956393.

References

1. Repository for piva (2023), https://github.com/zane-a-karl/PLI
2. Adhatarao, S., Lauradoux, C., Santos, C.: Why ip-based subject access requests are denied? arXiv preprint arXiv:2103.01019 (2021)
3. Berlekamp, E.R., Welch, L.R.: Error correction for algebraic block codes (Dec 1986)
4. Boniface, C., Fouad, I., Bielova, N., Lauradoux, C., Santos, C.: 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. pp. 182–209 (2019)
5. Brazil: Lei nº 13.709, de 14 de agosto de 2018 (2018), http://www.planalto.gov.br/ccivil_03/_Ato2015-2018/2018/Lei/L13709.htm
6. Bufalieri, L., La Morgia, M., Mei, A., Stefa, J.: Gdpr: when the right to access personal data becomes a threat. In: ICWS (2020)
7. California Attorney General: California consumer privacy act regulations (2020), https://oag.ca.gov/sites/all/files/agweb/pdfs/privacy/oal-sub-final-text-of-regs.pdf?
8. California Legislature: Title 1.81.5. california consumer privacy act of 2018 (2018), https://leginfo.legislature.ca.gov/faces/codes_displayText.xhtml?division=3.&part=4.&lawCode=CIV&title=1.81.5
9. Camenisch, J., Stadler, M.: Proof systems for general statements about discrete logarithms. In: Technical Report/ETH Zurich, Dept. of Computer Science (1997)
10. Cramer, R., Gennaro, R., Schoenmakers, B.: A secure and optimally efficient multi-authority election scheme. In: Advances in Cryptology — EUROCRYPT '97 (1997)
11. De Cristofaro, E., Gasti, P., Tsudik, G.: Fast and private computation of cardinality of set intersection and union. In: CANS (2012)
12. Di Martino, M., Robyns, P., Weyts, W., Quax, P., Lamotte, W., Andries, K.: Personal information leakage by abusing the {GDPR}'right of access'. In: Fifteenth Symposium on Usable Privacy and Security (SOUPS 2019). pp. 371–385 (2019)
13. Duong, T., Phan, D.H., Trieu, N.: Catalic: Delegated psi cardinality with applications to contact tracing. In: Eurocrypt (2020)

14. European Data Protection Board: Guidelines 01/2022 on data subject rights - right of access, version 2.0 (2023)
15. European Parliament and Council: Regulation (eu) 2016/679, general data protection regulation (2016), https://eur-lex.europa.eu/eli/reg/2016/679/
16. Fisher, R.A., Yates, F.: Statistical tables for biological, agricultural, and medical research. Hafner Publishing Company (1953)
17. Furukawa, J.: Efficient and verifiable shuffling and shuffle-decryption. IEICE Transactions (2005). https://doi.org/10.1093/ietfec/E88-A.1.172
18. Gamal, T.E.: A public key cryptosystem and a signature scheme based on discrete logarithms. IEEE Trans. Inf. Theory (1985)
19. Ghosh, S., Simkin, M.: The communication complexity of threshold private set intersection. In: CRYPTO (2019)
20. Groth, J.: A verifiable secret shuffle of homomorphic encryptions. Cryptology ePrint Archive, Paper 2005/246 (2005)
21. Heinrich, A., Hollick, M., Schneider, T., Stute, M., Weinert, C.: Privatedrop: Practical privacy-preserving authentication for apple airdrop. In: USENIX Security (2021)
22. Jordan, S., Nakatsuka, Y., Ozturk, E., Paverd, A., Tsudik, G.: VICEROY: gdpr-/ccpa-compliant enforcement of verifiable accountless consumer requests. In: NDSS (2023)
23. Keller, M.: MP-SPDZ: A versatile framework for multi-party computation. In: ACM CCS (2020)
24. Martino, M.D., Meers, I., Quax, P., Andries, K., Lamotte, W.: Revisiting identification issues in GDPR 'right of access' policies: A technical and longitudinal analysis. Proc. Priv. Enhancing Technol. (2022)
25. Narayanan, A., Thiagarajan, N., Lakhani, M., Hamburg, M., Boneh, D., et al.: Location privacy via private proximity testing. In: NDSS. vol. 11 (2011)
26. Pagnin, E., Gunnarsson, G., Talebi, P., Orlandi, C., Sabelfeld, A.: Toppool: Time-aware optimized privacy-preserving ridesharing. Cryptology ePrint Archive (2021)
27. Pinkas, B., Rosulek, M., Trieu, N., Yanai, A.: Psi from paxos: fast, malicious private set intersection. In: Eurocrypt (2020)
28. Reed, I.S., Solomon, G.: Polynomial codes over certain finite fields. Journal of the Society for Industrial and Applied Mathematics (1960)
29. Rindal, P., Schoppmann, P.: Vole-psi: fast oprf and circuit-psi from vector-ole. In: Eurocrypt (2021)
30. Rosulek, M., Trieu, N.: Compact and malicious private set intersection for small sets. In: ACM CCS (2021)
31. Samarin, N., Kothari, S., Siyed, Z., Bjorkman, O., Yuan, R., Wijesekera, P., Alomar, N., Fischer, J., Hoofnagle, C.J., Egelman, S.: Lessons in VCR repair: Compliance of android app developers with the california consumer privacy act (CCPA). Proc. Priv. Enhancing Technol. (2023)
32. Shamir, A.: How to share a secret. Commun. ACM (1979)
33. Take, K., Gallagher, K., Forte, A., McCoy, D., Greenstadt, R.: "it feels like whack-a-mole": User experiences of data removal from people search websites. Proc. Priv. Enhancing Technol. (2022)
34. Trieu, N., Shehata, K., Saxena, P., Shokri, R., Song, D.: Epione: Lightweight contact tracing with strong privacy. arXiv preprint arXiv:2004.13293 (2020)
35. Urban, T., Tatang, D., Degeling, M., Holz, T., Pohlmann, N.: The unwanted sharing economy: An analysis of cookie syncing and user transparency under gdpr. arXiv preprint arXiv:1811.08660 (2018)

36. Urban, T., Tatang, D., Degeling, M., Holz, T., Pohlmann, N.: A study on subject data access in online advertising after the gdpr. In: ESORICS (2019)
37. Yao, A.C.: How to generate and exchange secrets (extended abstract). In: 27th Annual Symposium on Foundations of Computer Science (1986)
38. Zhao, Y., Chow, S.S.M.: Can you find the one for me? In: WPES (2018)
39. Zhao, Y., Chow, S.S.: Are you the one to share? secret transfer with access structure. PETS (2017)

A Security Proofs for $\mathcal{P}$IVA Protocols

Proof of Theorem 1 (PLI security):

Correctness: Assume an execution of the protocol Π with honest $\mathcal{C}$ and honest $\mathcal{S}$. $\mathcal{S}$ computes $c_i := \frac{a_{i,2}}{a_{i,1}^{sk}}$, which is equivalent to $(g^{y_i - x_i})^{r_i}$ as follows:

$$c_i := \frac{a_{i,2}}{a_{i,1}^{sk}} = \frac{(b_{i,2} \cdot h^{u_i} g^{-x_i})^{r_i}}{((b_{i,1} \cdot g^{u_i})^{r_i})^{sk}} = \frac{(h^{w_i} g^{y_i} h^{u_i} g^{-x_i})^{r_i}}{((g^{w_i})^{sk} (g^{u_i})^{sk})^{r_i}}$$
$$= \frac{(h^{w_i} h^{u_i} g^{y_i - x_i})^{r_i}}{(h^{w_i} h^{u_i})^{r_i}} = \frac{(h^{w_i} h^{u_i})^{r_i} (g^{y_i - x_i})^{r_i}}{(h^{w_i} h^{u_i})^{r_i}} = (g^{y_i - x_i})^{r_i}$$

This becomes 1, if y_i is equal to x_i, or looks random in G, otherwise, as the random r_i is not known to $\mathcal{S}$. Therefore, $\mathcal{S}$ outputs $X \cap Y$, while $\mathcal{C}$ outputs $\perp$.
Server privacy: For corrupted $\mathcal{C}$, simulator SIM_C can be constructed as follows: SIM_C chooses n random values $(z_1, ...z_n)$ in Z_q and encrypts them under pk, i.e. it sets $b_i = \mathsf{Enc}_{pk}(z_i)$ for all i, and it sends pk and $(b_1, ..., b_n)$ to $\mathcal{C}$. (Note that SIM_C could use $\mathcal{C}$'s input X in the simulation, but the above algorithm does not need this input.) Because of the IND-CPA security of ElGamal encryption under the hardness assumption of the decisional Diffie-Hellman (DDH) problem, the ciphertexts produced by SIM_C are indistinguishable from the ones produced by $\mathcal{S}$ in the real protocol execution.
Client privacy: For corrupted $\mathcal{S}$, simulator SIM_S can be constructed as follows: Given Y and $X \cap Y$,

- SIM_S receives the public key pk and $(b_1, ..., b_n)$ from $\mathcal{S}$, and it sets $z_i = 0$ for all $i \in X \cap Y$ and picks $z_i \leftarrow_\$ Z_q$ for all $i \notin X \cap Y$.
- SIM_S computes $a_i = Enc_{pk}(z_i)$ for all i, and sends $(a_1, ..., a_n)$ to $\mathcal{S}$.

It follows that $\mathcal{S}$'s view in the interaction with SIM_S matches the interaction with the real-world $\mathcal{C}$: In both cases each a_i is either a random encryption of 1, if $i \in X \cap Y$, or an encryption of a random value in Z_q, if $i \notin X \cap Y$.

Proof of Theorem 2 (PLI-CA security):

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: Considering the corrupted $\mathcal{S}$, a simulator SIM_S can be similarly constructed. In addition to SIM_S in Theorem 1, 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 is indistinguishable from the output of the protocol execution with real $\mathcal{C}$, as $\pi\{\{1\}_{i\in X\cap Y}, \{z_i\}_{i\notin X\cap Y}\} \stackrel{c}{\equiv} \pi'\{\{1\}_{i\in X\cap Y}, \{(y_i - x_i)r_i\}_{i\notin X\cap Y}\}$.

Proof of Theorem 3 (t-PLI security):

Correctness: Assume an execution of the protocol Π with honest $\mathcal{C}$ and honest $\mathcal{S}$. For all $i \in [n]$, $\mathcal{S}$ computes s'_i as follows: $(s'_i||hs_i) := H_i(a_{i,1}^{sk}) \oplus e_i = H_i(a_{i,1}^{sk}) \oplus H_i(a_{i,2}) \oplus s_i, \forall i$. Since $H_i(a_{i,1}^{sk}) = H_i(((b_{i,1} \cdot g^{u_i})^{r_i})^{sk}) = H_i(((g^{w_i} \cdot g^{u_i})^{sk})^{r_i}) = H_i((h^{w_i} \cdot h^{u_i})^{r_i})$ and $H_i(a_{i,2}) = H_i((b_{i,2} \cdot h^{u_i} g^{-x_i})^{r_i}) = H_i((h^{w_i} g^{y_i} \cdot h^{u_i} g^{-x_i}))^{r_i}) = H_i((h^{w_i} \cdot h^{u_i})^{r_i} (g^{y_i - x_i})^{r_i})$, $H_i(a_{i,2}) = H_i(a_{i,1}^{sk})$, if $y_i = x_i$. Otherwise, $H_i(a_{i,2})$ is a hash of a random element in G. Consequently, if $y_i = x_i$, $s'_i = s_i$, or is random, otherwise. Depending on the cardinality of intersection $I := \{i \mid s'_i = s_i\}_{i\in[n]}$, three possible cases exist:

1. $|I| \geq k := \lceil \frac{n+t}{2} \rceil$: $\mathcal{S}$ can apply BW algorithm to recover (p, err). $\mathcal{S}$ outputs $\{y_i : err(i) \neq 0\}_{i\in[n]}$, 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 $H_i(s') = hs_i$.
3. $|I| < t$: $\mathcal{S}$ can neither reconstruct s' nor learn anything about x_i's.

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 $\mathcal{S}$, a simulator SIM_S can be constructed as below. Given Y and $|X \cap Y|$,

- 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 $z_i \leftarrow^{\$} G$, 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 $E_i = H_i(A_{i,2}) \oplus H'_i(S_i)$ for all i.
- SIM_S sends $(A_{1,1}, ..., A_{n,1})$ and $(E_1, ..., E_n)$ to $\mathcal{S}$.

Comparing $\mathcal{S}$'s view in the interaction with SIM_S and in the real execution of Π with $\mathcal{C}$, first, 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 hardness of DDH problem. Then, 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.

Proof of Theorem 4 (t-PLI-CA security):

Correctness: Similar to Theorem 3, $H_i(a_{i,2}) = H_i(a_{i,1}^{sk})$ 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 t-PLI-CA.

Client privacy: Due to the similarity of the protocols, the construction of the simulator SIM_S, considering the corrupted $\mathcal{S}$, 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. As in Theorem 3, $\mathcal{S}$'s view (even without this modification) in the interaction with SIM_S and the output of the protocol execution with real $\mathcal{C}$ are indistinguishable, and adding pseudorandom permutations randomly chosen in $\mathcal{P}_n$ does not change it.

Proof Outlines for Security against Malicious Participants:

For any malicious server $\mathcal{S}^*$, simulator SIM_S can (1) emulate H^q, capturing $\mathcal{S}^*$'s queries to H^q, (2) on $\mathcal{S}$'s message, extracts $sk = \mathsf{DL}_g(pk)$ from the ZKPoK and sets $M_i = \mathsf{PreDec}_{sk}(b_i)$ for each i, and (3) forms $\mathcal{S}^*$'s effective input Y into either protocol, by setting $Y[i]$, for each i, to y s.t. $\mathcal{S}^*$ queried (i, y) to H and $M_i = g^{\bar{y}_i}$ for $\bar{y}_i = H_i^q(y)$, and setting $Y[i] = \perp$ if there is no such H^q query. After sending Y to PLI functionality and getting output out_S back, SIM_S can then simulate honest client's message as in the proofs of Theorems 1-4 above.

For malicious client $\mathcal{C}^*$, SIM_C extracts δ_i, r_i for $i \in [n]$ from $\mathcal{C}^*$'s message, "pre-decrypt" each $a_{i,2}$ as $M_i = b_{i,2}(a_{i,2}^{-1}h^{\delta_i})^{1/r_i}$, and form $\mathcal{C}^*$'s effective input X into PLI, by setting $X[i]$, for each i, to x s.t. $\mathcal{C}^*$ received $\bar{x}_i = H_i^q(x)$ from RO H^q and $M_i = g^{\bar{x}_i}$, and setting $X[i] = \perp$ if $\mathcal{C}^*$ made no such H^q query. An honest $\mathcal{S}$ will output $I = X \cap Y$. In the simulation, SIM_C verifies the same constraint as in the protocol for $\bar{x}_i = H(i, x)$, hence $i \in I$ if $y_i = x_i$. SIM_C misses any match where the constraint is satisfied for $\bar{y}_i = H_i^q(y)$ but $\mathcal{C}^*$ does not query H_i^q on y. Since this can happen only with probability n/q, the simulated and real views are computationally indistinguishable.

For the modified t-PLI, SIM_C will extract (δ_i, r_i)'s as above, but it modifies the pre-decryption and derivation of $\mathcal{C}^*$'s effective input X: SIM_C searches through $\mathcal{C}^*$'s queries to H_i^q, H'_i, and H_i, to determine if there exists s s.t., for some subset of at least t indexes i, $\bar{x}_i = H_i^q(x_i)$, $hs_i = H'_i(s)$, and $ha_i = H_i(a_{i,2})$ satisfy that (1) $(ha_i \oplus e_i)[R] = hs_i$, where $z[L]$ and $z[R]$ stand for resp. the $\mathbb{F}$ and $\{0,1\}^{2\lambda}$ components of z; (2) $g^{\bar{x}_i} = b_{i,2}(a_{i,2}^{-1}h^{\delta_i})^{1/r_i}$; and (3) $s_i = (ha_i \oplus e_i)[L]$ lie on t-degree polynomial which interpolates to s. If SIM_C identifies such a subset, it forms X from all (i, x_i) pairs found above and sets all other values in X to $\perp$. If no such subset of at least t indexes is found, then SIM_C sets X to $(\perp)^n$. SIM_C efficiently and correctly simulates the real protocol because if H', H are RO's then constraint $(ha_i \oplus e_i)[R] = hs_i$ can be satisfied at most by a single (ha_i, hs_i) pair, except for negligible probability. On the other hand, if fewer than $t+1$ matches exist in the real protocol, the hs_i components recovered by $\mathcal{S}$ are indistinguishable from random, and thus the real protocol execution on X s.t. $|X \cap Y| < t$ is indistinguishable from the simulation where SIM_C sets X to an all-empty sequence $(\perp)^n$.