Laconic Private Set-Intersection From Pairings

Diego F. Aranha[1], Chuanwei Lin[1], Claudio Orlandi[1], and Mark Simkin[2]

[1] Aarhus University, Aarhus, Denmark
{dfaranha,orlandi}@cs.au.dk, chuanwei.lin@au.dk.
[2] Ethereum Foundation
mark.simkin@ethereum.org

Abstract. Private set-intersection (PSI) is one of the most practically relevant special-purpose secure multiparty computation tasks, as it is motivated by many real-world applications. In this paper we present a new private set-intersection protocol which is *laconic*, meaning that the protocol only has two rounds and that the first message is independent of the set sizes. Laconic PSI can be useful in applications, where servers with large sets would like to learn the intersection of their set with smaller sets owned by resource-constrained clients and where multiple rounds of interactions are not possible. Previously, practically relevant laconic PSI protocols were only known from factoring-type assumptions. The contributions of this work are twofold: 1) We present the first laconic PSI protocol based on assumptions over pairing-friendly elliptic curves; and 2) For the first time we provide empirical evaluation of any laconic PSI protocol by carefully implementing and optimizing both our and previous protocols. Our experimental results shows that our protocol outperforms prior laconic PSI protocols.

Table of Contents

1 Introduction

Private set intersection (PSI) protocols allow two parties holding private sets X and Y respectively to jointly compute $X \cap Y$ without revealing any other information about their input sets to each other. Such protocols turn out to be useful in settings like botnet detection [NMH$^+$10], private contact discovery [Mar14], online advertising [PSSZ15], and contact tracing [DPT20]. For this reason there have been numerous works [Mea86, FNP04, KS05, DCW13, PSSZ15, KKRT16, PRTY19, PRTY20] that study both theoretically and practically efficient PSI protocols.

In many of these applications, we have a powerful server with a large database, who communicates with many different computationally weak clients with small databases. One such example is the Signal[3] messenger, where the server stores a list of all registered users and different clients would like to determine which of their contacts use the messenger as well, without revealing the contacts that do not use it. Unfortunately, privacy does not come for free and even executing state-of-the-art PSI protocols incurs a significant computational overhead on both the server and the client.

Client-side devices, such as smartphones, are often resource constrained and if a protocol is too slow to be executed by the client, then there is little that can be done. In practice, it is not realistic to require a client to either download excessively large amounts of data or to perform computationally intensive and battery-draining computations. For the server, however, a somewhat inefficient protocol does not immediately mean that all is lost. A pragmatic, but effective method to alleviate server-side inefficiencies in practice is to follow the Kill-It-With-Iron (KIWI) approach, which simply dictates to improve performance by means of buying either better or just more hardware. This approach is clearly no silver bullet and it can only offset protocol inefficiencies within limits, but it raises the following natural question:

How small can we make the overhead of the party holding the smaller set?

Laconic cryptography [CDG$^+$17, QWW18, DGI$^+$19, DGGM19] is an emerging field within cryptography that asks the same question for general cryptographic tasks. In laconic cryptography, protocols are run between a receiver $\mathcal{R}$ with a potentially large input and a sender $\mathcal{S}$ with a small input. At the end $\mathcal{R}$ learns the output. The protocol should have the following properties: 1) The protocol should only have two messages; 2) The total communication and the work of $\mathcal{S}$ cannot depend on the size of $\mathcal{R}$'s input; and 3) The first message of $\mathcal{R}$ can be re-used by multiple senders $\mathcal{S}$. Predating laconic cryptography, Ateniese et al. [ADT11] provided the first *succinct* PSI protocol which satisfies requirement 2 above but fails to be fully laconic. After theoretical results that showed that any function can be securely evaluated with a laconic protocol, the work of Alamati et al. [ABD$^+$21] showed for the first time that there exist special purpose protocols for PSI which are potentially of practical relevance (e.g., they don't make non-black box use of the underlying groups). Both protocols use techniques inspired by RSA-based accumulators, and our work follows in this line of "accumulator based" PSI protocols.

1.1 Contribution and Technical Overview

In this work, we present a new two-party *laconic* PSI protocol which aims at minimizing the overhead for the sender both from a theoretical and practical perspective.

The protocol is conceptually very simple, which makes implementing it in practice less error-prone: During the protocol execution we only send two messages, sending one message from the receiver to the sender and one message back. The size of the receiver's message is a single group element (independent on the size of the receiver's input set), whereas the sender's message size is linear in the sender's input set size. The computational complexity of the sender is independent of the size of the receiver's input set.

Our main protocol provides semi-honest security under a new, but natural assumption over pairing-friendly curves. To provide some evidence about the soundness of our new assumption, we prove that it holds in the generic group model. We also discuss easy countermeasures to guarantee privacy in the presence of active attacks and present a protocol which gives full simulation-based security against active attacks.

[3] `https://signal.org/`

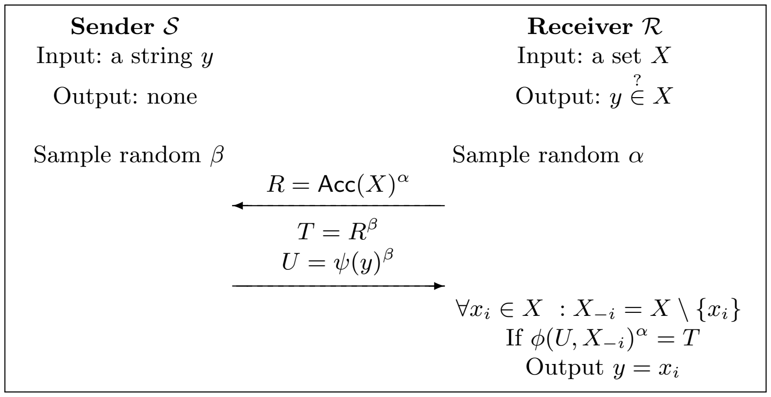
Sender $\mathcal{S}$
Input: a string y
Output: none
Receiver $\mathcal{R}$
Input: a set X
Output: $y \overset{?}{\in} X$
Sample random β
Sample random α
$R = \mathsf{Acc}(X)^\alpha$
$T = R^\beta$
$U = \psi(y)^\beta$
$\forall x_i \in X \ : X_{-i} = X \setminus \{x_i\}$
If $\phi(U, X_{-i})^\alpha = T$
Output $y = x_i$

Fig. 1: Our protocol in a nutshell. For the sake of simplicity the figure considers the special case of *private set membership (PSM)* e.g., a PSI protocol where one of the two parties has as single element as input. This can be extended to a PSI protocol by having the sender send a different (T, U) tuple for each y in their input set.

To evaluate the performances of our construction we have implemented it together with the semi-honest versions of the protocols of [ADT11] and [ABD$^+$21], for which no public implementations were available so far. Note that while the protocol of [ADT11] is not laconic, it still provides a useful baseline since it has a very small communication overhead thanks to the use of the random oracle model and private setup. In order to achieve a fair comparison, we have performed the same optimizations to all protocols, and we provide extensive benchmarks and comparisons to existing works.

PSI from Algebraic Accumulators.. We give an high-level overview of our protocol through the lens of cryptographic accumulators [Bd94] that satisfy algebraic properties (e.g., those based on the RSA [Bd94, CL02, LLX07] or pairings [Ngu05, CKS09, DT08, ATSM09, KB21] assumptions). This is only an informal description and neglects several details but we still believe it can be helpful in understanding the logic of the protocol (the actual protocol description in Section 3 presents the protocol directly without using accumulator notations). A cryptographic accumulator $A = \mathsf{Acc}(X)$ is a function which allows to compress a large set X into a small representation A. It is then possible to create, for any $y \in X$, a witness w that y was included in the accumulator which can be verified with a function Ver. The *soundness* property of an accumulator requires that if $y \notin X$ then it should be infeasible to produce a witness w such that $\mathsf{Ver}(\mathsf{Acc}(X), y, w) = 1$.

Natural algebraic accumulators satify the following properties:

1. The witness w is *not* a function of y, but only of the remaining elements of X once y is removed i.e., $w = \mathsf{Wit}(X_{-y})$
2. The verification function can be decomposed in the following way:
$$\mathsf{Ver}(\mathsf{Acc}(X), y, \mathsf{Wit}(X_{-y})) = 1 \Leftrightarrow \phi(\psi(y), X_{-y}) = \mathsf{Acc}(X)$$
3. The functions ϕ, ψ output elements in a group and for all y, X and scalar β it holds that:
$$\phi(\psi(y)^\beta, X) = \phi(\psi(y), X)^\beta$$

These properties lead to a natural construction for a PSI protocol as visualized in Figure 1. Intuitively the receiver sends a randomized version of the accumulator $R = A^\alpha$ for some random α. The sender replies with $\psi(y)^\beta$ and R^β (with a different β for each element in their set). Now the receiver computes witnesses for each subset of X of size $|X| - 1$ and checks whether the element sent by the sender matches the one removed from X. Security of the protocol, intuitively, follows from the usage of the randomizers α, β and from the fact that it is computationally hard for the receiver to find witnesses for elements which were not in the original set X, thus the receiver cannot perform a brute force attack on y. This is only a very general intutition, and the security of the protocol is formally proven in a different way later in the paper.

Pairing Based Accumulators.. We instantiate the above blueprint with a pairing based accumulator such as the ones from [Ngu05, CKS09, DT08, ATSM09, KB21]: since we only need a *one-shot accumulator* (i.e., elements are only inserted into the accumulator, and all at once), the construction is quite simple. In such an accumulator $\mathsf{Acc}(X) = g^{\prod_{x \in X}(x-s)}$, $w = \mathsf{Wit}(X_{-i}) := g^{\prod_{x \in X_{-i}}(x-s)}$ and the verification function $\mathsf{Ver}(\mathsf{Acc}(X), y, w)$ outputs 1 if

$$e(\mathsf{Acc}(y), w) = e(\mathsf{Acc}(X), g)$$

where e is the bilinear map, and s is some secret unknown to the receiver. Due to the algebraic nature of the accumulators it is possible to randomize each message simply by raising them to random exponents and, thanks to the bilinearity of e, the verification equation is satisfied e.g., we get:

$$e(\mathsf{Acc}(y)^{\beta}, \mathsf{Wit}(X \setminus \{y\}))^{\alpha} = e(\mathsf{Acc}(X)^{\alpha \cdot \beta}, g)$$

Note that to evaluate the polynomials in s required by the protocol the parties must have access to the generator g raised to all powers of s i.e., $(g, g^{s}, \ldots, g^{s^2}, \ldots, g^{s^{|X|}})$. Such a string can be generated by some trusted third-party and be re-used by multiple parties (privacy for the sender relies on the receiver not knowing s, while privacy for the receiver holds unconditionally). We refer to to Section 3 for the actual description of our protocol.

	[ADT11]	[ABD$^+$21]	Ours
Assumption	RSA	RSA	SBDDH
Random Oracle	Yes	No	No
Communication	$(\lvert\mathbb{Z}_N\rvert, \lvert\mathbb{Z}_N\rvert + n\lvert H\rvert)$	$(\lvert\mathbb{Z}_N\rvert, n\lvert\mathbb{Z}_N\rvert + n\lvert H\rvert)$	$(\lvert\mathbb{G}_2\rvert, n\lvert\mathbb{G}_1\rvert + n\lvert H\rvert)$
Setup Size $\mathcal{S}$	$O(1)$ *(private)*	$O(1)$	$O(1)$
Setup Size $\mathcal{R}$	$O(1)$	$O(1)$	$O(m)$
Multi-Sender/Laconic	No	Yes	Yes

Table 1: Comparison of existing succinct PSI protocols e.g., where the communication and computation complexity of the sender is independent of the size of the receivers' set. We denote by m, n the size of $\mathcal{R}, \mathcal{S}$ inputs respectively. Communication complexity is expressed as (*first message*,*second message*), with $|\mathbb{Z}_N|, |\mathbb{G}_1|, |\mathbb{G}_2|$ representing the size of elements from the RSA group, and the two base groups in a type-3 pairing friendly elliptic curve, and H the output of a hash function.

Comparison with Prior Accumulator-Based PSI. Table 1 provides a qualitative comparison of our protocol vs. the protocols of [ADT11, ABD$^+$21]. The original protocol of [ADT11] has three rounds, but it can be made two rounds assuming trusted setup. There is however a caveat, since the sender in [ADT11] needs to learn the trapdoor of the setup. Thus, [ADT11] does not allow multiple senders to reply to the same first message from the receiver, which means it is not a laconic protocol. The main disadvantage of our protocol looking at the table is clearly that the size of the setup for the receiver scales with the size of their inputs. Note however that our setup has the same structure as the setup of widely deployed preprocessing SNARKs, and therefore such setups have already been generated using the so-called "powers of tau" ceremonies of ZCash, FileCoin, etc. for sizes up to 100 millions. Moreover, this dependency also provides an interesting additional security guarantee: in our protocol the sender knows that the size of the set input by the receiver cannot exceed the size of the setup, while in RSA-based protocols there is no such upper bound, thus a receiver might maliciously choose their input to be the entire (or a large fraction of) the universe from which the inputs are picked. Note also that [ADT11] offers the best communication complexity. This is due to their use of private setup and random oracle model. We briefly discuss an variant of our protocol in the same model achieving the same communication in Section 4.5.

Other Related Work.. There exists a very large body of research for PSI protocols. We will only mention the ones most relevant for our work here: several works have studied the question of private set intersection between a party with a large and a party with a small set. In applications where the communication patterns guaranteed by laconic protocols is not required, these protocols offer very attractive features. The protocols of Chen et al. [CLR17] and Chen et al. [CHLR18] are both based on (lattice-based) fully homomorphic encryption and offer good performances in the case of unbalanced set sizes. In these protocols the roles of the parties are swapped and it is the party with the smaller set which obtains the output of the computation, and the communication scales logarithmically with the size of the larger set, thus they are not laconic. Moreoever, due to the parameter size for lattice-based cryptography their communication overhead in practice is larger than for RSA or discrete logarithm based constructions. The works of [RA18] and [KLS$^+$17] independently optimize PSI for unbalanced set sizes and introduce a large receiver side storage overhead to achieve an online phase which only scales in the size of the smaller set. The protocol of [KRS$^+$19] offers a solution with impressive performances for mobile private contact discovery, yet the communication complexity scales linearly with the size of the larger set, and their protocol requires several rounds of interaction. The work of [RT21] is the state of the art for PSI with small input sets, and we compare empirically against them in the implementation section.

To conclude, PSI is a fundamental tool in modern cryptography and it is used in a variety of different applications with different requirements. For this reason we believe that it is generally important to study PSI with different efficiency properties. In this work we design the most efficient laconic PSI protocol to date.

2 Preliminaries

2.1 Notation

We write $a \xleftarrow{\$} A$ to sample a uniform element a from a set A. We write $[n]$ for the set $\{1, \dots, n\}$. Given a set $X = \{x_1, \dots, x_m\}$, where $x_i \in \mathbb{Z}_q$ we write X_{-i} for the set $X_{-i} := X \setminus \{x_i\}$. We write $P(X, s)$ for the polynomial $\prod_{x \in X}(x - s)$ of degree $|X|$ in $\mathbb{Z}_q$ whose roots are all the elements in X and we write $P(X, s) = \sum_{i=0}^{|X|} p(X, i)s^i$ for its coefficients. Note that using our notation we have $P(X, s) = P(X_{-i}, s) \cdot (x_i - s)$.

2.2 Assumptions

To get the best concrete efficiency, we work with an asymmetric (type 3) bilinear map e from groups $\mathbb{G}_1 \times \mathbb{G}_2$ to $\mathbb{G}_T$, both of prime order q generated by g_1, g_2, $g_T = e(g_1, g_2)$ respectively.

We describe here the computational assumption on which the security of our protocol relies on. We define the (B_1, B_2)-*Strong Bilinear Decisional Diffie-Hellman* ((B_1, B_2)-*SBDDH*) problem as follows: A challenger picks a random $s \in \mathbb{Z}_q^*$ and outputs a $(B_1 + B_2 + 2)$-tuple of elements $\left(g_1, g_1^s, \dots, g_1^{s^{B_1}}, g_2, g_2^s, \dots, g_2^{s^{B_2}}\right) \in {\mathbb{G}_1}^{B_1+1} \times {\mathbb{G}_2}^{B_2+1}$. The adversary, on input this string, outputs $y \in \mathbb{Z}_q$. Then the challenger flips a random bit b and, if $b = 0$, outputs $T = e(g_1, g_2)^{1/(y+s)}$, whereas if $b = 1$ it samples T as a random element from the target group. The goal of the adversary is to guess the bit b given the T as input. More formally:

Definition 1 **((B_1, B_2)-Strong Bilinear Decisional Diffie-Hellman Problem ((B_1, B_2)-SBDDH)).** *Consider the game $\mathcal{G}^{SBDDH}_{b,(B_1,B_2),\mathcal{A}}$ described in Figure 2. We say that the (B_1, B_2)-SBDDH problem is hard if, for every PPT adversary $\mathcal{A}$, the advantage*

$$Adv^{SBDDH}_{(B_1,B_2),\mathcal{A}}(\kappa) := \left|\Pr\left(\mathcal{G}^{SBDDH}_{0,(B_1,B_2),\mathcal{A}}(\kappa) = 1\right) - \Pr\left(\mathcal{G}^{SBDDH}_{1,(B_1,B_2),\mathcal{A}}(\kappa) = 1\right)\right|$$

is negligible in the security parameter κ.

We say the (B_1, B_2)-SBDDH assumption holds if no PPT adversary can solve the (B_1, B_2)-SBDDH problem with more than negligible probability.

The Game $\mathcal{G}^{\mathsf{SBDDH}}_{b,(B_1,B_2),\mathcal{A}}(\kappa)$

The game is parametrized by a stateful adversary $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$, a parameter (B_1, B_2) and a bit b.

1. $s \overset{\$}{\leftarrow} \mathbb{Z}_q^*$;
2. $y \leftarrow \mathcal{A}_1\left(\left\{g_1^{s^i}\right\}_{i=0}^{B_1}, \left\{g_2^{s^i}\right\}_{i=0}^{B_2}\right)$
3. $T_0 = e(g_1, g_2)^{1/(y+s)}$;
4. $T_1 \overset{\$}{\leftarrow} \mathbb{G}_T$;
5. $b' \leftarrow \mathcal{A}_2(T_b)$;
6. Output 1 if $b = b'$.

Fig. 2: The (B_1, B_2)-Strong Bilinear Decisional Diffie-Hellman Game ((B_1, B_2)-SBDDH)

Looking ahead, the security of our protocol will rely on the $(1, B)$-SBDDH assumption with B being an upper bound on the size of the set of the receiver.

The assumption can also be extended to symmetric (type 1) pairings by setting $g = g_1 = g_2$ (note in this case some of the elements given as input to $\mathcal{A}_1$ become redundant). Our assumption can be seen as the natural decisional extension of the *Strong Diffie-Hellman (B-SDH)* assumption proposed in [BB08]. This assumption says that given a set $\{g^{s^i}\}_{i\in[B]}$ it is hard to come up with (y, h) such that $h = g^{1/(y+s)}$. Note however that it is not hard to distinguish h from a random group element, since it is possible to compute the pairing $e(g^s \cdot g^y, h)$ and check whether this is the identity in the target group. But it is indeed plausible to assume that $e(g, g)^{1/(y+s)}$ is indistinguishable from random. In fact a very similar assumption has been used before under the name *Decisional Bilinear Diffie-Hellman inversion assumption (B-DBDHI)* by [DY05], which is equivalent to ours when $y = 0$. To increase confidence that our assumption is indeed solid, in Section 5 we prove that our assumption holds in the generic group model.

2.3 PSI Notation and Functionalities

A *private set intersection protocol (PSI)* is a two party protocols between a receiver $\mathcal{R}$ and a sender $\mathcal{S}$ with input respectively $X = \{x_1, \ldots, x_m\}$ and $Y = \{y_1, \ldots, y_n\}$. We assume all elements in both sets can be efficiently encoded as elements of $\mathbb{Z}_q$. At the end of the protocol the receiver $\mathcal{R}$ learns the intersection $Z = X \cap Y$ while the sender learns nothing. This is formally captured in the standard ideal functionality $\mathcal{F}_{\mathsf{PSI}}$ which is deferred to the Appendix in Figure 4. For completeness, we also state the zero-knowledge functionality $\mathcal{F}_{\mathsf{zk}}$ (which will be used in the actively secure version of our protocol) in the Appendix in Figure 5.

3 Our Laconic PSI Protocol – Semi-Honest Security

We give an high-level overview of our protocol using symmetric (type 1) pairings. The actual description of the protocol uses asymmetric (type 3) pairings since this leads to a faster implementation. Intuitively, our protocol works as follows: during the setup phase some trusted party[4] samples a random secret s and publishes g raised to all the powers of s, i.e. g^{s^i} up to the maximum size of the set of the receiver. Then, the receiver $\mathcal{R}$ uses this information to generate an accumulator of his set X. The accumulator is then randomized to hide the receiver's set. In other words, the receiver sends a single group element $R = g^{r \cdot P(X,s)}$ where $P(X, s)$ is the evaluation in s of the polynomial whose roots are all elements in X (note that the receiver does not know s but can anyway evaluate this polynomial in the exponent using the elements from

[4] Note that the same setup can be re-used by any number of senders and receivers. Moreover, we will later discuss how to reduce the amount of trust in the setup phase.

the setup as detailed in the formal description of the protocol), and r is some random value. Note that the receiver can also compute the accumulator for all subsets X_{-k} obtained by removing a single x_k from X for all k. Call R_{-k} those randomized accumulators, computed as $R_{-k} = g^{r \cdot P(X_{-k}, s)}$. Assume for this informal description that the set of the sender $\mathcal{S}$ is a singleton containing y. Now the sender $\mathcal{S}$ can send the accumulator computed on its own singleton set, again randomized to hide its input, i.e., the sender sends $U = g^{t(s-y)}$ for some random t. The sender also sends the "target value" $T = e(R, g)^t$ for reasons that will be apparent shortly. Finally the receiver can use the bilinear pairing to "combine in the exponent" the accumulator of the sender with each of its own "subset accumulators" R_{-k} by computing $e(R_{-k}, U)$ and checking whether this is equal to the target element T. Note that if y is equal to the element missing from any X_{-k}, the output of the pairing is an element in the target group whose exponent contains $P(X, s)$ and the randomness chosen by both parties. It turns out that this is exactly the target value sent by the sender, thus the output of the comparison will reveal to the receiver whether $y \in X$.

For security, note that the input of the receiver is in fact protected unconditionally[5]: the randomizer r turns R into a uniformly random group element independent of the set X. Security for the sender is more interesting. Note that the sender re-uses the randomness t in both $T = e(R, g)^t$ and $U = g^{t(s-y)}$. One might be tempted to directly argue security using some DDH-type assumption, however note that this would fail since R, which determines the base in the first element, is chosen by the receiver. It turns out that the security of the protocol can be reduced to the SBDDH assumption (Definition 1), as we show in the proof of Theorem 1.

The formal description of the protocol is given in Figure 3. The main differences with this informal description are the following: the protocol is presented using asymmetric (type 3) pairings; the set of the sender can contain multiple elements, and the sender will compute a pair (T, U) as described above for each element in their set using independent randomness for each pair; to save on communication, a hash of the target is sent instead; finally, the sender randomly permutes their set before performing the protocol, otherwise the receiver might extrapolate additional information about the input of the receiver from the positions of the matches (e.g., if the sender orders its input in lexicographical order and the receiver finds a match in the first position, then the receiver learns that all elements in the set of the sender are larger than this, which is not an information that could have been derived from the intersection of the sets alone).

Theorem 1. *The protocol Π_{PSI} securely implements $\mathcal{F}_{\mathsf{PSI}}$ in the presence of passively corrupted adversary assuming hardness of the $(1, B)$-SBDDH problem (Definition 1) and collision-resistance of H.*

Proof. We now show that the protocol is correct and that it is possible to generate simulated transcripts for both parties that are computationally indistinguishable from the view of the parties in a real execution of the protocol.

Correctness:. Assume Y is a singleton set equal to y (e.g., for simplicity we skip the j index here) and that $y = x_k$. Then the receiver concludes that x_k is in the intersection if: $H\left(e\left(U_j, R_{-k}\right)\right) \overset{?}{=} H\left(e\left(g_1^t, R\right)\right)$. Since we assume H to be collision resistant, except with negligible probability this happens iff the two inputs to H are equal. Then the receiver includes x_k in their output iff $y = x_k$ since:

$$e\left(U, R_{-k}\right) = e\left(U, \left(\prod_{i=0}^{m-1} S_i^{p(X_{-k}, i)}\right)^r\right) = e\left(g_1^{t(s-y)}, g_2^{r \cdot P(X_{-k}, s)}\right)$$
$$= e\left(g_1, g_2\right)^{r \cdot t \cdot P(X, s)}$$

and also

$$e\left(g_1^t, R\right) = e\left(g_1^t, \left(\prod_{i=0}^{m} S_i^{p(X, i)}\right)^r\right) = e\left(g_1^t, g_2^{r \cdot P(X, s)}\right)$$
$$= e\left(g_1, g_2\right)^{r \cdot t \cdot P(X, s)}$$

[5] This implies that it is not necessary for the receiver to trust the setup.

The Protocol Π_{PSI}

Inputs/Outputs: The protocol is run between a receiver $\mathcal{R}$ with input $X = \{x_1, \ldots, x_m\}$ and a sender $\mathcal{S}$ with input $Y = \{y_1, \ldots, y_n\}$. We assume the inputs can be efficiently encoded as elements in $\mathbb{Z}_q$. At the end the receiver learns $Z = X \cap Y$ while the sender learns nothing.

Setup: Let λ be security parameter and $H(\cdot)$ be a collision-resistant hash function $H : \{0,1\}^* \to \{0,1\}^\lambda$. The receiver and the sender agree on a bilinear map $e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$ where the generators are $g_1, g_2, g_T = e(g_1, g_2)$ respectively. During the setup phase a party trusted by $\mathcal{S}$ picks a random $s \in \mathbb{Z}_q^*$, then computes and send:

$$\mathsf{setup}_{\mathcal{R}} = (S_1, S_2, \ldots, S_m) \text{ and } \mathsf{setup}_{\mathcal{S}} = (S')$$

to $\mathcal{R}$ and $\mathcal{S}$ respectively where $S' = g_1^s$ and $S_i = g_2^{s^i}$ for $i = 0, \ldots, m$.

First Round: In any session *sid*, the receiver $\mathcal{R}$ sends

$$\mathsf{msg1} = (sid, R)$$

to the sender, computed as follows:

1. Pick random $r \xleftarrow{\$} \mathbb{Z}_q$;
2. Compute $R = \left(\prod_{i=0}^{m} S_i^{p(X,i)}\right)^r$ if $X \neq \emptyset$;
3. Compute $R = (S_0)^r$ if $X = \emptyset$.

Second round: In any session *sid*, the sender $\mathcal{S}$ sends to the receiver $\mathcal{R}$ the message

$$\mathsf{msg2} = (sid, T_1, U_1, \ldots, T_n, U_n)$$

computed as follows. Pick a random permutation $\pi : [n] \to [n]$. Then for all $j \in [n]$:

1. Pick random $t_j \xleftarrow{\$} \mathbb{Z}_q$;
2. Compute
$$(T_j, U_j) = \left(H\left(e\left(g_1^{t_j}, R\right)\right), \left(S' \cdot g_1^{-y_{\pi(j)}}\right)^{t_j}\right)$$

Retrieve Output: To retrieve the output from a session *sid*, the receiver $\mathcal{R}$ does the following; For all $j \in [n], k \in [m]$, the receiver initializes $Z = \emptyset$ and then does the following:

1. Compute
$$R_{-k} = \left(\prod_{i=0}^{m-1} S_i^{p\left(X_{-k}, i\right)}\right)^r$$
2. If $H\left(e\left(U_j, R_{-k}\right)\right) \stackrel{?}{=} T_j$, then add x_k to the output Z;

The receiver outputs Z.

Fig. 3: Our Private Set Intersection Protocol

Security – Corrupted Sender:. We simulate the view of a corrupted sender in the following way: the simulator Sim, on input the set Y of the sender and nothing else (the sender has no output in the protocol), picks a random $r \in \mathbb{Z}_q$ and outputs $\mathsf{msg1} = (sid, R)$ with $R = g_2^r$. Note that this completes the view of the sender $\mathcal{S}$ in the protocol since we only need to simulate the incoming messages. We argue that the output of this simulator is indistinguishable from the view of a corrupted sender in the real protocol in a strong, unconditional sense. By construction of R in the protocol, if $X = \emptyset$ then the simulation is trivially perfect. If $X \neq \emptyset$, the distribution are identically distributed as long as $P(X, s) \neq 0$. Note however that this only happens with negligible probability bounded by $|X|/q = m/q$. This concludes the argument in the case of a corrupted sender.

Security – Corrupted Receiver:. We simulate the view of a corrupted receiver, with output Z, in the following way: the simulator Sim on input the sets X, Z follows the following instructions.

1. Simulate the setup as an honest party would do i.e., pick a random $s \in \mathbb{Z}_q^*$ and add
$$(\mathsf{setup}_{\mathcal{S}}, \mathsf{setup}_{\mathcal{R}}) = (S', S_1, S_2, \ldots, S_m)$$
to the simulated view, where $S' = g_1^s$ and $S_i = g_2^{s^i}$ for $i = 0, \ldots, m$.
2. Let $Z = \{z_1, \ldots, z_\zeta\}$.
3. Compute R as in the protocol.
4. Pick a random subset $\Gamma \subseteq [n]$ with $|\Gamma| = |Z| = \zeta$. This simulates the positions for which $\mathcal{R}$ gets a match in the protocol.
5. Define the complimentary set $\Delta = [n] \setminus \Gamma$. This simulates the positions for which $\mathcal{R}$ does not get a match in the protocol.
6. Let $\Gamma = \{\gamma_1, \ldots, \gamma_\zeta\}$ i.e., the positions from which $\mathcal{R}$ should learn their output. For every $j \in [\zeta]$ generate $(T_{\gamma_j}, U_{\gamma_j})$ as in the protocol i.e., pick a random $t_j \in \mathbb{Z}_q$ and compute
$$(T_{\gamma_j}, U_{\gamma_j}) = \left(H\left(e\left(g_1^{t_j}, R\right)\right), \left(S' \cdot g_1^{-z_j}\right)^{t_j}\right)$$
7. Let $\Delta = \{\delta_1, \ldots, \delta_\omega\}$ with $\omega = m - \zeta$ i.e., the positions from which $\mathcal{R}$ does not get a match. For every $j \in [\omega]$ simulate $(T_{\delta_j}, U_{\delta_j})$ as follows: pick random $t_j, u_j \in \mathbb{Z}_q$ and compute
$$(T_{\delta_j}, U_{\delta_j}) = \left(H\left(e\left(g_1, g_2\right)^{t_j}\right), g_1^{u_j}\right)$$
8. Finally include
$$\mathsf{msg2} = (sid, T_1, U_1, \ldots, T_n, U_n)$$
to the simulated view.

In other words, the simulator picks at random which indices j correspond to a match and which do not, then makes sure that the j's that should produce a match indeed do so by computing the corresponding group elements as an honest party would do, while it just sends uniformly random group elements for the j's which should not give a match. We now argue indistinguishability of the real protocol and the simulated execution using the following hybrids.

Hybrid 0.. In this hybrid the distribution of the view of $\mathcal{R}$ is the same as in the real protocol.

Hybrid (1,0).. The same as Hybrid 0, with the exception that now we do not directly pick the permutation π uniformly at random. Instead, we first pick sets (Γ, Δ) at random like in the simulation i.e., Γ is a random subset of $[n]$ with $\Gamma = (\gamma_1, \ldots, \gamma_\zeta)$ and Δ is the complimentary set. Now, for each j we find the index $\rho(j)$ such that $z_j = y_{\rho(j)}$ and we choose a random permutation π under the the constraint that $\pi(\gamma_j) = \rho(j)$. We then fill the remaining positions in the permutation at random.

The distribution produced by Hybrid (1,0) is identical to the distribution produced by Hybrid 0, since in both cases π is an uniform permutation in $[n]$.

Hybrid (1,j) for all $j \in [n]$.. We now have a sequence of m hybrids, where in each hybrid we change the distribution of a single pair (T_j, U_j), so that in Hybrid $(1, j-1)$ the pair (T_j, U_j) is generated like in the real protocol, whereas in Hybrid $(1, j)$ the pair (T_j, U_j) is generated like in the simulation. Each hybrid j uses the output set Z and the last $n-j$ elements of the input set Y (thus, by construction, hybrid $(1, n)$ does not use Y).

Note that by construction of π in the previous hybrid, for all $j \in \Gamma$ (i.e., the position for which the receiver gets a match), the pair (T_j, U_j) is identically distributed in the real protocol and the simulation, and therefore for all $j \in \Gamma$ the distribution of Hybrid $(1, j-1)$ and Hybrid $(1, j)$ are trivially identical.

We prove in Claim 3 that the distributions of Hybrid $(1, j-1)$ and Hybrid $(1, j)$ for $j \in \Delta$ are computationally indistinguishable under the $(1, m)$-SBDDH assumption.

Hybrid 2.. In this hybrid the distribution of the view of $\mathcal{R}$ is the same as in the simulated execution.

The distribution of Hybrid 2 is identical to the one in Hybrid $(1, m)$, since in Hybrid $(1, m)$ the distribution every single pair (T_j, U_j) from the real protocol has been replaced with the corresponding pair from the simulated transcript. Therefore, this concludes the proof.

Claim. For all $j \in \Delta$, the distribution generated by Hybrid $(1, j-1)$ and Hybrid $(1, j)$ are computationally indistinguishable under the $(1, m)$-SBDDH assumption (Definition 1).

Proof. We show here that a distinguisher $\mathcal{A}$ that can tell Hybrid $(1, j-1)$ and Hybrid $(1, j)$ apart with non-negligible advantage can be turned into an adversary $\mathcal{B} = (\mathcal{B}_1, \mathcal{B}_2)$ that has a non-negligible advantage against the $(1, m)$-SBDDH problem. Note that the reduction can choose and therefore knows the inputs X, Y for both parties. The reduction goes as follows:

1. $\mathcal{B}_1$ receives as input $(g_1, S', g_2, S_1, S_2, \ldots, S_m)$ from the $(1, m)$-SBDDH challenger.
2. The reduction picks a random permutation π and lets $\mathcal{B}_1$ output $y^* = -y_{\pi(j)}$.
3. Now $\mathcal{B}_2$ receives T^* as input from the challenger, and uses it to complete the transcripts of the protocol in the following way:
   (a) The reduction includes the message $\mathsf{setup} = (g_1, S', g_2, S_1, S_2, \ldots, S_m)$ in the transcript of the protocol.
   (b) The reduction picks a random $r \in \mathbb{Z}_q$, computes $R = g_2^{r \cdot P(X,s)}$ and includes the message $\mathsf{msg1} = (sid, R)$ in the transcript of the protocol.
   (c) For all $i < j$, the reduction generates a pair (T_i, U_i) like a real world sender $\mathcal{S}$ would do.
   (d) For all $i > j$, the reduction generates a pair (T_i, U_i) following the instructions of the simulator Sim.
   (e) For $i = j$, the reduction generates (T_j, U_j) as follows: pick a random $\tau \in \mathbb{Z}_q$ and compute
$$(T_j, U_j) = \left(H\left((T^*)^{\tau \cdot P(X,s)} \right), g_1^{\tau} \right)$$
4. The reduction includes the message $\mathsf{msg2} = (sid, T_1, U_1, \ldots, T_m, U_m)$ in the transcript of the protocol.
5. The reduction invokes the distinguisher $\mathcal{A}$ on input the transcripts of the protocol. If $\mathcal{A}$'s guess is "real protocol", then $\mathcal{B}_2$ outputs 0, whereas if $\mathcal{A}$'s guess is "simulated protocol", then $\mathcal{B}_2$ outputs 1.

We now analyze the success probability of the reduction. It suffices to notice that when $T^* = e(g_1, g_2)^{1/(s+y^*)}$ then the input of the distinguisher is distributed identically as in Hybrid $(1, j-1)$ whereas then T^* is a random element from $\mathbb{G}_T$ then the input of the distinguisher is distributed identically as in Hybrid $(1, j)$. When $b = 0$, this can be observed using the following renaming of variable $t := \tau/(s + y^*)$. Note that, as long as $s + y^* \neq 0$, which only happens with negligible probability, if τ is uniformly random in $\mathbb{Z}_q$ then so is

t. Therefore when $b = 0$ we get that:

$$\begin{aligned}(T_j, U_j) &= \left(H\left((T^*)^{\tau \cdot P(X,s)}\right), g_1^{\tau}\right)\\ &= \left(H\left(e(g_1, g_2)^{\tau \cdot P(X,s)/(s+y^*)}\right), g_1^{\tau}\right)\\ &= \left(H\left(e(g_1, g_2)^{t \cdot P(X,s)}\right), g_1^{t(s+y^*)}\right)\\ &= \left(H\left(e(g_1, g_2)^{t \cdot P(X,s)}\right), g_1^{t(s-y_{\pi(j)})}\right)\\ &= \left(H\left(e(g_1^t, R)\right), \left(S' \cdot g_1^{-y_{\pi(j)}}\right)^t\right)\end{aligned}$$

(in the first equality we replace $T^* = e(g_1, g_2)^{1/(s+y^*)}$, in the second equality we replace $t = \tau/(s + y^*)$, in the third equality we replace $y^* = -y_{\pi(j)}$, and in the final equality we replace the definitions of R and S') which is the distribution in the real protocol. When $b = 1$ and T^* is a random element from $\mathbb{G}_T$ we can write $T^* = e(g_1, g_2)^{t^*}$ and thus get:

$$\begin{aligned}(T_i, U_i) &= \left(H\left((T^*)^{\tau \cdot P(X,s)}\right), g_1^{\tau}\right)\\ &= \left(H\left(e(g_1, g_2)^{t^* \cdot \tau \cdot P(X,s)}\right), g_1^{\tau}\right)\\ &= \left(H\left(e(g_1, g_2)^{t}\right), g_1^{u}\right)\end{aligned}$$

(in the first equality we replace $T^* = e(g_1, g_2)^{t^*}$, in the second equality we replace $u := \tau$ and $t := t^* \cdot \tau \cdot P(X, s)$; note that if $P(X, s) \neq 0$ then t, u are uniform in $\mathbb{Z}_q$ since τ, t^* are) which is exactly the distribution of the simulated execution.

This concludes the proof of the claim.

4 Active Security and Extensions

In this section we first (Section 4.1) analyze to which extent the original protocol provides any (weak but) meaningful security guarantees against active adversaries *essentially as it is* (i.e., we argue for some game-based properties of the protocol). Then (Section 4.2) we describe how to achieve full active security (in a strong ideal world/real world simulation sense). We then discuss some extensions of the protocol including: how to let multiple parties contribute randomness to the setup phase (Section 4.3), introducing tradeoffs to reduce the computational overhead of the receiver, at the price of increased overhead for the sender, using standard bucketing techniques (Section 4.4), a variant of the semi-honest protocol achieving better communication complexity using random oracles and private setup (Section 4.5).

4.1 Active Attacks vs. Π_{PSI} And Countermeasures

We analyse possible active attacks on our protocol Π_{PSI} and some simple countermeasures which allow to achieve game-based security properties in this setting. This also serves as a warm-up towards our final fully-active secure protocol.

Privacy vs. Malicious Sender.. Consider an actively corrupt sender $\mathcal{S}^*$, who also controls the setup generation. As we have discussed in the proof of Theorem 1, the output of the simulator is indistinguishable to the view of a passively corrupt $\mathcal{S}$ in the protocol in a strong, unconditional sense. Therefore the only way that an active attack can hope to break the privacy of the receiver is by choosing the setup $\mathsf{setup}_{\mathcal{R}}$ maliciously (we will argue later that in fact it is always possible to check that the setup is well formed, but an adversary could still choose s maliciously). Note that there is indeed an attack in this case: consider for simplicity the case where $X = \{x\}$ contains a single element. Then an active attacker would be able to check whether $x^* \in X$ by picking $s = -x^*$ and publishing $\mathsf{setup}_{\mathcal{R}} = g_2^s$, which leads to $R = 1_{\mathbb{G}_2}$ iff $x = x^*$. This can be easily fixed by adding a simple step to the protocol: we let the receiver 1) check that the setup message $\mathsf{setup}_{\mathcal{R}}$ is indeed a vector of valid group elements in $\mathbb{G}_2$; and 2) we let the receiver $\mathcal{R}$ pick a random value σ (after seeing the setup) which is used to "shift" both input sets X, Y into $\tilde{X}, \tilde{Y}$ where for all i, j $\tilde{x}_i = x_i + \sigma$ and $\tilde{y}_j = y_j + \sigma$. Then, both parties run the original protocol with inputs $\tilde{X}, \tilde{Y}$ instead. Since this is a bijective mapping this transformation has no effect on correctness.

With these adjustments, we can easily prove that the message R is statistically independent of the set X for any (even possibly ill-formed) choice of $\mathsf{setup}_{\mathcal{R}}$. To see why consider $\mathsf{setup}_{\mathcal{R}} = (S_1, \ldots, S_m)$. If $\mathcal{R}$ checks that $\mathsf{setup}_{\mathcal{R}} \in {\mathbb{G}_2}^m$ then there exists $s_1, \ldots, s_m \in \mathbb{Z}_q$ such that

$$\mathsf{setup}_{\mathcal{R}} = (g_2^{s_1}, \ldots, g_2^{s_m})$$

(even if the sender $\mathcal{S}$ might not know these values). Let $\mathsf{p}(\tilde{X}, i)$ be the coefficients of the polynomial whose roots are the elements of $\tilde{X}$, then it holds that $R = \left(g_2^{\sum s_i \cdot \mathsf{p}(\tilde{X}, i)}\right)^r$ is a uniformly random element in $\mathbb{G}_2$ as long as $\sum s_i \cdot \mathsf{p}(\tilde{X}, i) \neq 0$. But since σ is random and sampled after the adversary chosen $s_0, \ldots, s_m$, this only happens with negligible probability.

Correctness vs. Malicious Sender.. We ask whether there are attacks that a malicious $\mathcal{S}^*$ could mount on our protocol which would lead the honest receiver $\mathcal{R}$ to output some "incorrect value". By construction the protocol only allows $\mathcal{R}$ to output a set Z such that $Z \subseteq X$ so even a corrupt sender cannot make the receiver output elements which are not already in their own set. But we can still ask whether there is a strategy for $\mathcal{S}^*$ to force the output of $\mathcal{R}$ to contain some high-entropy elements (imagine an application of PSI where $\mathcal{S}$ is supposed to input some secret, e.g. a password, and $\mathcal{R}$ would later grant $\mathcal{S}$ access if the output of the protocol is not empty). Again, there is a simple attack against this by choosing $(T, U) = (H(1_{\mathbb{G}_T}), 1_{\mathbb{G}_1})$ which leads, for any k, to $H(e(U, R_{-k})) = T$. We can easily counteract this by asking the honest receiver to abort if they receive such tuples.

We can analyze what happens then: Suppose for simplicity that the honest receiver has a single, uniformly random input element $X = \{x\}$. Then, after receiving some S_1 from $\mathcal{S}^*$, $\mathcal{R}$ replies with $\left(R = \left(S_1 g_2^{-x}\right)^r\right)$. In the retrieve phase, $\mathcal{R}$ outputs $Z = \{x\}$ iff

$$H(e(U, g_2^r)) = T$$

where S_1, U, T are chosen by the malicious $\mathcal{S}^*$. Now, since x is uniformly random the message R contains statistically no information about r independently of the choice of S_1, and therefore the probability that $\mathcal{S}^*$ produces (U, T) that satisfy the check is negligible. (In the final fully active secure protocol this class of attacks will be prevented without assuming any distribution of x, but by asking the malicious $\mathcal{S}^*$ to prove that they computed (T, U) according to the protocol specification).

Privacy vs. Malicious Receiver.. A malicious receiver might try to send an ill-formed message R in the hope to extract sensitive information about the input of the sender. Note however that *any* $R \neq 1_{\mathbb{G}_2}$ is in fact a possible valid choice for R corresponding to some input: Intuitively, privacy vs. a malicious receiver holds since (under the discrete logarithm assumption) even a malicious receiver knows at most one representation of R in base $(S_0, \ldots, S_m)$, and any such representation corresponds to an input set of X. So intuitively the worse that a malicious receiver can do is to pick its input X^* maliciously. Note in this sense that the fact that our protocol by construction limits the receiver to pick a set X^* of size at most m (defined by the setup phase) might even be seen as a feature vs. the RSA based constructions which would allow a malicious receiver to pick a set X^* of arbitrary size.

Inspecting our semi-honest proof of security in Theorem 1 it is possible to notice that neither our simulator nor the reduction to our assumption needs to use the knowledge of the discrete logarithm of R to prove the claim, nor it needs to compute the setup parameters $\mathsf{setup}_{\mathcal{R}}$ as a function of the value of R. Note finally that not even the invalid choice $R = 1_{\mathbb{G}_2}$ would help the corrupt receiver, since this just leads T to be $H(1_{\mathbb{G}_T})$, thus losing information about the sender's random choice t, and leading U to be perfectly indistinguishable from a random element in the group.

Correctness vs. Malicious Receiver.. Since the sender has no output, there are trivially no active attacks in this case.

4.2 Full Active Security via ZK-Proofs

Here we explain how to achieve full active security (i.e., simulation-based) in the presence of active adversaries. We describe the differences with the semi-honest protocols and defer the full description of the active secure protocol to Appendix B. At a very high-level, active security is achieved employing the countermeasures described above and by letting both parties prove in zero-knowledge that the messages they send are well formed, to allow for simulation-based security. We stress that the ZK-relations we need in the protocol can be proven with succinct communication using recent advances in zero-knowledge proofs.

Laconic PSI with Active Security.. Our active secure protocol, which we denote by Π^*_{PSI}, is similar to the one in Figure 3 with the following modifications (the protocol makes use of the $\mathcal{F}_{\mathsf{zk}}$ functionality in Figure 5):

1. Parties abort if they ever receive the identity element in any group.
2. Add a step to **First Round** where $\mathcal{R}$ verifies that the setup is of the correct form by checking, for all $i \in [m]$, that $e(S', S_{i-1}) = e(g_1, S_i)$. When $i = 1$ this asserts that $\log_{g_1}(S') = \log_{g_2}(S_1) = s$, and for all other i it asserts that $\log_{g_2}(S_{i-1}) = s \cdot \log_{g_2}(S_i)$. Noticing that our setup is very similar to those of recent SNARKs, we borrow this technique from recent works on SNARKs with universal and updatable setup [GKM+18].
3. In **First Round**, add a step where $\mathcal{R}$ invokes the $\mathcal{F}_{\mathsf{zk}}$ functionality for the following relation (Greek letters indicate values which are not already defined as part of the protocol):

$$R_{\mathcal{R}} = \left\{ \begin{array}{c} x = (\mathsf{setup}_{\mathcal{R}}, R) \\ w = (\omega_0, \ldots, \omega_m) \end{array} : R = S_0^{\omega_0} \cdot S_1^{\omega_1} \cdot \ldots \cdot S_m^{\omega_m} \right\}$$

   where $\omega_i = p(X, i) \cdot r$. Using recent results in compressed Σ-protocols, we can construct practically efficient ZK-proofs (in the sense that they don't need to represent group exponentiation as a Boolean circuit) for such relation with size only $O(\log(m))$ [AC20, BCC+16].
4. In **Second Round**, modify Step 2 to remove the collision-resistant hash function from the computation of T_j i.e., T_j is now computed as $T_j = e(g_1^{t_j}, R)$. Similarly, remove the hash function from the check in Step 2 in **Retrieve Output**. (The efficiency loss from having to prove the hash function in zero-knowledge outweights the savings in communication.)
5. In **Second Round**, add a step where $\mathcal{S}$ invokes the $\mathcal{F}_{\mathsf{zk}}$ functionality for the following relation (Greek letters indicate values which are not already defined as part of the protocol):

$$R_{\mathcal{S}} = \left\{ \begin{array}{c} x = (\mathsf{setup}_{\mathcal{S}}, \delta, \{T_j, U_j\}_{j\in[n]}) \\ w = (\{t_j, \alpha_j\}_{j\in[n]}) \end{array} : \begin{array}{c} T_j = \delta^{t_j} \wedge \\ U_j = (S')^{t_j} g_1^{\alpha} \end{array} \right\}$$

   where $\delta = e(g_1, R)$ and $\alpha_j = -t_j \cdot y_{\pi(j)}$. Using [ABC+22], we can construct proofs for this relation of size only $O(\log(n))$.

We can then show the following:

Theorem 2. *The protocol Π^*_{PSI} securely implements $\mathcal{F}_{\mathsf{PSI}}$ in the presence of an actively corrupted adversary assuming the hardness of the $(1, B)$-SBDDH problem (Definition 1) in the $\mathcal{F}_{\mathsf{zk}}$-hybrid model.*

The full proof is deferred to Appendix B. Intuitively, the simulator will be able to extract the inputs of both parties thanks to the $\mathcal{F}_{\mathsf{zk}}$ functionality. In particular, extracting the input of the sender from α_j, t_j and σ is quite straightforward once one notices that $T_j \neq 1_{\mathbb{G}_T} \Rightarrow t_j \neq 0$. To extract the input of the receiver we first extract the coefficients $\omega_0, \ldots, \omega_m$ and then interpret them as the coefficients of a degree m polynomial Ω. The simulator can find the roots of Ω using an efficient polynomial factoring algorithm [CZ81,vzGS92,Sho93] which define an input set $\tilde{X}^*$ for the corrupted $\mathcal{R}^*$, which can then be easily turned into X^* by subtracting σ. The views of both parties are still simulated essentially as in the semi-honest case.

4.3 Updatable Setup

Recall that correctness of the setup phase can be easily checked using pairing relations and that the sender does not need to know the setup trapdoor s (in fact no one does). In other words, the only property required by the setup is that the receiver does not learn s.

It is therefore natural to think of a setting in which multiple parties jointly generate the setup, such that if at least one of these parties is honest then the protocol is secure. As already noticed the structure of our setup is very similar to those of recent SNARKs with universal and updatable setup [GKM$^+$18]. In particular this means that we can use the same techniques to let multiple parties contribute randomness to the setup in a sequential way in the following way: each party P_ι takes as input the setup output by party $P_{\iota-1}$ and outputs an updated setup string. For convenience let:

$$(S'_0, S_{0,1}, \ldots, S_{0,m}) = (g_1, g_2, \ldots, g_2)$$

Then party ι can first verify that the setup string $\iota - 1$ is correct by checking that for all $i \in [m]$:

$$S_{\iota-1,i} \neq 1_{\mathbb{G}_2} \text{ and } e(S'_{\iota-1}, S_{\iota-1,i-1}) = e(g_1, S_{\iota-1,i})$$

and then contribute randomness s_ι by outputting:

$$(S'_\iota, S_{\iota,1}, \ldots, S_{\iota,m}) = ((S'_{\iota-1})^{s_\iota}, S_{\iota-1,1}^{s_\iota}, \ldots, S_{\iota,m}^{{s_\iota}^m})$$

Which leads to a final secret $s = \prod_\iota s_\iota$ being uniformly random in the view of the adversary as long as at least one party is honest. This is a big advantage w.r.t., the RSA-based construction since multiparty generation of RSA moduli is notoriously a complex secure multiparty computation task.

4.4 Reducing Receiver's Overhead Using Bucketing

The main computational bottleneck in our protocol are the $O(m \cdot n)$ pairings performed by the receiver during the **Retrieve Output** phase. While in this work, we chose to focus on minimizing the overhead for the sender, there are value of m, n for which our protocol becomes too slow on the receiver side to be practically relevant.

We note here that standard bucketing techniques (originating from [PSZ14]), such as simple hashing, can be used in our setting to reduce the receiver's computational costs at the expense of a larger first round message from the receiver to the sender. More concretely, given a random function $H : \{0,1\}^* \to [k]$, the receiver can first partition the set X by throwing elements x_i into buckets $H(x_i)$ and then compute the first round message of our laconic PSI protocol for each of those buckets independently. The sender can partition the set Y consistently with function H, then, roughly speaking, pad each bucket slightly to hide the concrete number of elements in the bucket, and finally compute the second round message independently for each bucket and each of the first round messages that it received.

Using this strategy the first round message of the receiver increases by a multiplicative factor of k. Ignoring the padding for a second, one can see that in expectation each bucket will contain n/k elements and thus the receiver will need to perform roughly

$$k \cdot n/k \cdot m/k = nm/k$$

pairing evaluations, which is a factor k cheaper than the computational costs for the naive receiver.

To make this intuition actually work, a little bit of care is needed. Since bucketing is not a contribution of this work we don't expand further but we refer instead to [PSZ14] for more details on this technique.

4.5 Reducing Communication Complexity with Private Setup and Random Oracle

Note from Table 1 that the protocol of [ADT11] offers the best communication complexity. This is due to their use of private setup and random oracle model. We briefly discuss here a variant of our protocol achieving the same communication complexity: Following the blueprint of [ADT11], in the setup phase we let the sender $\mathcal{S}$ learn the setup trapdoor s. Thanks to the knowledge of s, $\mathcal{S}$ can compute for each $y \in Y$

$$T = e(g_1^{t/(s-y)}, R) = e(g_1, g_2)^{r \cdot t \cdot P(X,s)/(s-y)}$$

and send it along with $U = g_1^t$. The check performed by the receiver in **Retrieve Output** does not need to change, and the result of the following computation on the receiver side

$$e(U, R_{-k}) = e(g_1, g_2)^{r \cdot t \cdot P(X_{-k},s)}$$

is equal to T when $y = x_k$. If the sender has a set of size larger than 1, in the standard model we would need to use a different t for each y to preserve privacy. However, thanks to the properties of the random oracle model, we can use a single randomizer t and transfer a single U together with $H(T_1), \ldots, H(T_m)$ and still achieve security. Intuitively this is because, in the random oracle model, the hardness of computing $e(g_1, g_2)^{1/(s-y)}$ is turned into entropy. Since the resulting protocol is not laconic and requires the sender to know the trapdoor from the setup (which makes the setup much harder to realize in practice) we do not investigate this variant further.

5 Generic Security of the (B_1, B_2)-SBDDH Assumption

In this section, we examine the (B_1, B_2)-SBDDH assumption in the generic group model: In the generic group model, elements of $\mathbb{G}_1$, $\mathbb{G}_2$ and $\mathbb{G}_T$ are encoded as unique random strings, so that the adversary can only test string equality. We define a family of injective functions Θ_1 where $\theta_1 \in \Theta_1$ is a function $\theta_1 : \mathbb{Z}_q \mapsto \{0,1\}^\ell$ mapping a scalar $a \in \mathbb{Z}_q$ to the string representation $\theta_1(a)$, with $\ell \geq \log_2 q$. Similarly, we define families Θ_2, Θ_T for $\mathbb{G}_2$ and $\mathbb{G}_T$ such that $\theta_2 \in \Theta_2$ is of the form $\theta_2 : \mathbb{Z}_q \mapsto \{0,1\}^\ell$ and $\theta_T \in \Theta_T$ is of the form $\theta_T : \mathbb{Z}_q \mapsto \{0,1\}^\ell$. The adversary performs operations on group elements by interacting with an oracle G which in turns gives access to several sub-oracles: three oracles for the group operation in each of the three groups $\mathbb{G}_1$, $\mathbb{G}_2$ and $\mathbb{G}_T$, two oracles for the homomorphism $\psi : \mathbb{G}_2 \mapsto \mathbb{G}_1$ and its inverse ψ^{-1}, and one oracle for the bilinear pairing $e : \mathbb{G}_1 \times \mathbb{G}_2 \mapsto \mathbb{G}_T$. Note that such homomorphism are not efficiently computable in the type of curves that we use to instantiate the protocol. However, by giving more power to the adversary in model we make our claim stronger and we cover security of the SBDDH assumptions also in other settings in which such homomorphism are efficiently computable, or in the symmetric case where $\mathbb{G}_1 = \mathbb{G}_2$.

Theorem 3. *Let $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$ be an stateful algorithm that solves the (B_1, B_2)-SBDDH problem. Assume that $\mathcal{A}$ makes at most q_G oracle to G queries for the group operations in $\mathbb{G}_1$, $\mathbb{G}_2$ and $\mathbb{G}_T$, the homomorphisms ψ and ψ^{-1}, and the bilinear pairing $e : \mathbb{G}_1 \times \mathbb{G}_2 \mapsto \mathbb{G}_T$, all counted together. Then,*

$$\left| \Pr\left[\mathcal{A}_2^G\left(\theta_T\left(\Gamma_0\right), \theta_T\left(\Gamma_1\right)\right) = b \;\middle|\; \begin{array}{l} x \xleftarrow{\$} \mathbb{Z}_q^*; \\ \theta_1, \theta_2, \theta_T \xleftarrow{\$} \Theta_1, \Theta_2, \Theta_T; \\ y \leftarrow \mathcal{A}_1^G\left(\begin{array}{l} q, \\ \{\theta_1\left(x^i\right)\}_{i\in[0,B_1]}, \\ \{\theta_2\left(x^i\right)\}_{i\in[0,B_2]} \end{array}\right), \\ b \xleftarrow{\$} \{0,1\}; \\ \Gamma_b \leftarrow 1/(x+y); \\ \Gamma_{1-b} \xleftarrow{\$} \mathbb{Z}_q^* \end{array} \right] - \frac{1}{2} \right|$$

is bounded by $\varepsilon < 2\left(q_G + 2B + 4\right)^2 \frac{(B+1)}{q}$ and $B = \max(B_1, B_2)$.

Proof. Instead of letting $\mathcal{A}$ interact with the actual oracles, we consider an algorithm $\mathcal{B}$ simulating the game with $\mathcal{A}$. The main idea is that $\mathcal{B}$ will not pick an actual value x until the very end of the game, and will instead just keep internally the symbolic polynomial representation of all the elements queried by $\mathcal{A}$. Since in the generic group model $\mathcal{A}$ sees random strings anyway, the simulation is perfect as long as $\mathcal{B}$ do not send two different representations to $\mathcal{A}$ for the same group element. This could happen e.g., if two of the polynomials internally stored by $\mathcal{B}$ evaluate to same the value once the formal variable is replaced by the sampled x. This proof structure is very standard in generic group model proofs see e.g., [Sho97, DY05, BB08, CDK$^+$12].

The simulated oracle $\mathcal{B}$ maintains three lists of pairs $L_1 = \{(F_{1,i}, s_{1,i}) : i = 0, 1, \ldots, t_1 - 1\}$, $L_2 = \{(F_{2,i}, s_{2,i}) : i = 0, 1, \ldots, t_2 - 1\}$ and $L_T = \{(F_{T,i}, s_{T,i}) : i = 0, 1, \ldots, t_T - 1\}$. Let B denote $\max(B_1, B_2)$. Here, the entries $F_{1,i}$ and $F_{2,i}$ will be multivariate polynomials of degree $\leq B$ in $\mathbb{Z}_q[X, \mathit{\Gamma}_0, \mathit{\Gamma}_1]$ and $F_{T,i}$ will be of degree $\leq 2B$ in $\mathbb{Z}_q[X, \mathit{\Gamma}_0, \mathit{\Gamma}_1]$. The entries $s_{1,i}$, $s_{2,i}$ and $s_{T,i}$ will be all the encoding strings given out to the adversary. At the beginning of the game, the lists are initialized at step $\tau = 0$ by assigning $F_{1,i} = X^i$ for $i = 0, \ldots, B_1$, $F_{2,i} = X^i$ for $i = 0, \ldots, B_2$, $F_{T,0} = \mathit{\Gamma}_0$ and $F_{T,1} = \mathit{\Gamma}_1$. The corresponding encodings are set to random distinct ℓ-bit strings. All polynomials are stored as coefficients of powers of variables. The lists have length $t_1 = B_1 + 1$, $t_2 = B_2 + 1$ and $t_T = 2$.

We assume that $\mathcal{A}$ only makes oracle queries on encoding strings of elements it had previously received. For any query encoding string, $\mathcal{B}$ can find the corresponding polynomial by determining the index of string. It is required that the strings in each list should be distinct.

To start the game, $\mathcal{B}$ provides $\mathcal{A}_1$ with the $B_1 + B_2 + 2$ encoding strings $(s_{1,0}, \ldots, s_{1,B_1}, s_{2,0}, \ldots, s_{2,B_2})$ that correspond to the first part of the SBDDH instance. $\mathcal{A}_1$ begins to issue oracle queries. $\mathcal{B}$ responds to $\mathcal{A}_1$'s queries as follows:

Group operations: Given two operands $s_{1,i}$ and $s_{1,j}$ with $0 \leq i, j < t_1$ and a multiply/divide bit, $\mathcal{B}$ computes the polynomial $F_{1,t_1} = F_{1,i} \pm F_{1,j}$ accordingly. If $F_{1,t_1} = F_{1,l}$ for some $l < t_1$, $\mathcal{B}$ sets $s_{1,t_1} = s_{1,l}$. Otherwise, s_{1,t_1} is set as a random string in $\{0,1\}^\ell \backslash \{s_{1,0}, \ldots, s_{1,t_1-1}\}$. The pair (F_{1,t_1}, s_{1,t_1}) is added to the list L_1, the string s_{1,t_1} is given to $\mathcal{A}_1$ as the answer, and t_1 is incremented by 1.
Group operation queries in $\mathbb{G}_2$ and $\mathbb{G}_T$ are answered similarly, except $\mathcal{B}$ operates on the lists L_2 and L_T respectively.

Homomorphisms: Given a homomorphism query from $\mathbb{G}_2$ to $\mathbb{G}_1$ with an operand $s_{2,i}, 0 \leq i < t_2$, $\mathcal{B}$ makes a copy of the associated polynomial $F_{2,i}$ into L_1, i.e. $F_{1,t_1} = F_{2,i}$. If $F_{1,t_1} = F_{1,l}$ for some $l < t_1$, $\mathcal{B}$ sets $s_{1,t_1} = s_{1,l}$. Otherwise, s_{1,t_1} is set as a random string in $\{0,1\}^\ell \backslash \{s_{1,0}, \ldots, s_{1,t_1-1}\}$. The pair (F_{1,t_1}, s_{1,t_1}) is added to the list L_1, the string s_{1,t_1} is given to $\mathcal{A}_1$ as the answer to the query, and t_1 is incremented by 1.
Inverse homomorphism queries from $\mathbb{G}_1$ to $\mathbb{G}_2$ are answered similarly, except $\mathcal{B}$ makes a copy of the polynomial from $\mathbb{G}_1$ and operates on the list L_2.

Pairing: Given a pairing query consisting of two operands $s_{1,i}$ and $s_{2,j}$ with $0 \leq i < t_1$ and $0 \leq j < t_2$, $\mathcal{B}$ computes the product $F_{T,t_T} = F_{1,i} \cdot F_{2,j}$. If $F_{T,t_T} = F_{T,l}$ for some $l < t_T$, $\mathcal{B}$ sets $s_{T,t_T} = s_{T,l}$. Otherwise, s_{T,t_T} is set as a random string in $\{0,1\}^\ell \backslash \{s_{T,0}, \ldots, s_{T,t_T-1}\}$. The pair (F_{T,t_T}, s_{T,t_T}) is added to the list L_T, the string s_{T,t_T} is given to $\mathcal{A}_1$ as the answer to the query, and t_T is incremented by 1.

After making $q_{G,1}$ queries, $\mathcal{A}_1$ outputs y, and $\mathcal{A}_2$ receives the strings $(s_{T,0}, s_{T,1})$ corresponding to the second part of the SBDDH instance. Now $\mathcal{A}_2$ can perform more oracle queries similarly to $\mathcal{A}_1$ (remember that since $\mathcal{A}$ is stateful, $\mathcal{A}_2$ has full information about the queries performed by $\mathcal{A}_1$ and the responses received by $\mathcal{B}$).

After making at most $q_{G,2}$ queries with $q_G = q_{G,1} + q_{G_2}$, $\mathcal{A}_2$ returns a guess $\hat{b} \in \{0,1\}$. $\mathcal{B}$ chooses a bit b and uniform value $x, z \xleftarrow{\$} \mathbb{Z}_q^*$ and sets $\mathit{\Gamma}_b \leftarrow 1/(x+y), \mathit{\Gamma}_{1-b} \leftarrow z$.

If the simulation provided by $\mathcal{B}$ is consistent, it reveals nothing about b. The only way in which the simulation could be inconsistent is that, after we choose values for $X, \mathit{\Gamma}_0, \mathit{\Gamma}_1$, two different polynomials in each list happen to produce the same value. Specifically, $\mathcal{A}$ wins if either of the following holds, for any $\dagger \in \{1, 2, T\}$:

1. $F_{\dagger,i}(x, 1/(x+y), z) = F_{\dagger,j}(x, 1/(x+y), z)$

2. $F_{\dagger,i}(x,z,1/(x+y)) = F_{\dagger,j}(x,z,1/(x+y))$

in $\mathbb{Z}_q$ for some i,j such that $F_{\dagger,i} \neq F_{\dagger,j}$ in $\mathbb{Z}_q[X,\Gamma_0,\Gamma_1]$,

Assume that $x+y \neq 0$ (which happens with probability $1-1/q$). Then note that $\forall i$ the degree of the polynomials $F_{1,i}, F_{2,i}$ is upper bounded by B and the degree of the polynomials $F_{T,i}$ is upper bounded by $2B$. In particular, no polynomial that the adversary created interacting with the oracle contains the term $1/X$. Therefore we can bound the probability that any two polynomials assume the same value when substituting with the formal variable with the random choices using the Schwartz-Zippel lemma. In particular we get that for all i,j, $\Pr[F_{1,i}-F_{1,j}=0] = \Pr[F_{2,i}-F_{2,j}=0] \leq B/q$ and $\Pr[F_{T,i}-F_{T,j}=0] \leq 2B/q$. Summing up we get that the probability that the simulation fails is bounded by:

$$\varepsilon \leq 2 \cdot \left(\binom{t_1}{2} \frac{B}{q} + \binom{t_2}{2} \frac{B}{q} + \binom{t_T}{2} \frac{2B}{q} \right) \left(1 - \frac{1}{q}\right) + \frac{1}{q}$$

Remembering that the total length of the lists of polynomials $t_1+t_2+t_T$ after q_G queries is at most $q_G + B_1 + B_2 + 4$ we get that:

$$\varepsilon < 2\left(q_G + 2B + 4\right)^2 \frac{(B+1)}{q} = O\left(\frac{q_G^2 B + B^3}{q}\right)$$

6 Experimental Results

We implemented our protocol together with related work using the RELIC library [AGM$^+$], and benchmarked the implementations on an Intel Core i7-7820X CPU running at 3.60GHz. RELIC is well-suited for implementing the different protocols, since it contains efficient implementations of the RSA cryptosystem, record-setting implementations of the pairing computation and some advanced functionality including multi-scalar exponentiations within the pairing groups. We selected parameters at the 128-bit security for the experiment, which translates to 3072-bit RSA moduli and the BLS12-381 pairing-friendly curve, as estimated in [BD16, MSS16]. Since some of the benchmarks were computationally intensive and could not be executed more than once, we disabled HyperThreading and TurboBoost to reduce randomness. We conducted two sets of experiments: the first to compare our protocol against other accumulator-based protocols, and the second to compare against other protocols in the literature. The code to implement the protocols and execute the experiments is available at `https://github.com/relic-toolkit/relic/tree/main/demo/psi-client-server`.

We compared our protocol with the works of Alamati et al. [ABD$^+$21] and Ateniese et al. [ADT11], which both use RSA-based accumulators. For implementing the two latter protocols, we apply a collection of optimizations for a fair comparison. The description of the optimized protocols is provided in Appendix C. We implement the hash-to-prime function by simply testing a truncated value of the hash result for primality, and incrementing until a positive answer is found. We selected the hash length to $\lambda_1 = 80$ bits to achieve statistical security of 40 bits, which is comparable to other semi-honest protocols in the literature. The hash results are also cached by the Receiver from the First Round until the very last step (Retrieve Output) to avoid excessive primality testing. For the particular case of the [ADT11] protocol, we allow the sender to optimize the exponentiations by using the Chinese Remainder Theorem with knowledge of the factorization $N = PQ$. In order to reduce traffic requirements, in the last step we compare hash results instead of full elements, while preventing collisions by setting the hash output length to $\lambda_2 = 256$ bits. For the RSA-based protocols, a complexity of $O(m^2)$ is required in the Retrieve step to rebuild the accumulator m times (one for each missing element) containing the $m-1$ elements. That is repeated for each of the $O(n)$ elements U_j received from the Sender in the case of [ABD$^+$21]. It is not possible to precompute this step, since it depends on the Sender output. A dynamic programming optimization proposed in [ADT11] could reduce this cost further, but it was not implemented due to our emphasis on Sender latency. For our protocol, we allow the Receiver to precompute and cache m versions of the randomized accumulator as R_{-k}, each missing one element, as to reduce complexity of the last step. This allows to implement the Retrieve Output operation in effectively $O(nm)$ operations. The group elements U_j sent by the Sender are transmitted in compressed form to save bandwidth, since point compression is trivial on the Sender side and involves an expensive square root in a finite field for decompression at the Receiver, again trading off costs to favor the Sender.

Table 2 contains the first set of results. We present numbers for computation by both Sender and Receiver, for equal set sizes ranging from 2^7 to 2^{10}. The Receiver's computation is further broken down in the First Round and Retrieve Output steps of the protocol. For our protocol, the First Round contains the precomputation of all R_{-k}, which explains the higher values. In this table, we do not take into account the communication latency or bandwidth, but just the communication complexity in bits. From the table, one can see that our protocol is close to 20 times more efficient than [ABD$^+$21], and 3.25 times faster than [ADT11] at the Sender side. For the Receiver, the speedups compared to [ABD$^+$21] range from 40 to 160 with growing set sizes, directly accompanying the growth in complexity for the last step. That protocol does not scale well beyond $m = 2^{10}$ elements and results were thus omitted. Our protocol presents the best performances both for computation and communication among the analyzed laconic protocols, being at least approximately 3.25 and 40 times faster than [ABD$^+$21] for Sender and Receiver, respectively. In comparison to [ADT11], our protocol is around 3.3 times slower for the Receiver, a rate quite similar to the Sender speedup, hence a trade-off that is admissible in laconic cryptography.

Size ($n = m$)	Protocol	Sender (ms)	Receiver (ms)	First (ms)	Retrieve (ms)	Comm. (bits)
2^7	[ABD$^+$21]	2814.6	680605.6	87.6	680518.0	429056
	[ADT11]	475.5	**5261.8**	87.8	5174.0	**38912**
	Ours	**142.8**	16822.2	2562.2	14260.0	83456
2^8	[ABD$^+$21]	5645.8	5452745.2	164.9	5452580.3	855040
	[ADT11]	931.7	**20899.3**	163.9	20735.4	**71680**
	Ours	**285.3**	69524.9	10562.7	58962.2	165376
2^9	[ABD$^+$21]	11319.8	43620974.2	331.5	43620642.7	1707008
	[ADT11]	1863.1	**83743.6**	332.7	83410.9	**137216**
	Ours	**572.3**	275742.7	47533.6	228209.1	329216
2^{10}	[ABD$^+$21]	-	-	-	-	3410944
	[ADT11]	3691.0	**333508.3**	628.7	332879.6	**268288**
	Ours	**1137.5**	1117925.5	210579.7	907345.8	656896

Table 2: Comparison of succinct/laconic PSI protocols based on accumulators. The lowest numbers for each performance metric (Sender/Receiver latency and communication complexity) highlighted in bold for each set size. Due to the high latency, timings come from single measurements.

We conducted a second set of experiments to evaluate the performance of our protocol in comparison with other works. For this experiment, we implemented a dedicated set of client and server programs using sockets and multi-threading in C, such that we could control the transmission bandwidth. Following related work, we executed both client and server in different cores of the same machine communicating over `localhost`, while using the Linux `tc` tool to limit the bandwidth of the loopback interface. In this implementation, the Sender initiates the connection, receives the randomized accumulator from the Receiver and then proceeds to close the connection immediately after finishing its round of the protocol, thus optimizing for Sender latency. The Receiver continues the execution until the output is retrieved.

For comparison, we executed the protocol optimized for small sets by Rosulek and Trieu [RT21] using their implementation, and their experimental setup by using three network configurations combining bandwidth and round-trip time: 10Gbps with 0.2 ms, 50Mbps with 80ms and finally 1Mbps with 80ms. We first validated

				Online running time (ms)					
				10 Gbps		50 Mbps		1 Mbps	
n	m	Protocol	Comm. (KB)	Sender	Receiver	Sender	Receiver	Sender	Receiver
2^8	2^8	[RT21]	**9.79**	**79.0**	**85.8**	**247.9**	**248.1**	**267.4**	**267.4**
		[CMdG$^+$21]	1130.70	569.0	373.9	2574.9	388.9	11243.3	398.8
		Ours	20.67	289.0	57259.1	465.2	57581.7	467.3	57618.3
	2^{10}	[RT21]	34.04	**219.4**	**242.5**	**383.4**	**363.4**	487.5	515.6
		[CMdG$^+$21]	1129.41	593.90	391.0	2569.2	386.5	1197.4	**391.9**
		Ours	**20.67**	305.3	228061.6	467.3	22839.4	**465.1**	228381.4
	2^{12}	[RT21]	130.04	704.9	851.3	1060.7	1149.7	2161.3	2081.5
		[CMdG$^+$21]	1129.62	593.7	**385.0**	2564.9	**390.1**	11226.7	**388.8**
		Ours	**20.67**	**306.9**	1140396.3	**468.7**	1155221.4	**467.1**	1164645.6

Table 3: Comparison of protocols at 128-bit security under different bandwidth requirements. The lowest numbers for each combination of performance metric (Sender/Receiver latency and communication complexity) and set sizes are highlighted in bold. Timings are computed as the average of 10 measurements.

that we could approximately reproduce results from their paper in terms of traffic and execution time. We then proceeded to execute the protocol with set sizes $n = 2^8$ and $m \in \{2^8, 2^{10}, 2^{12}\}$ under the same conditions to measure the online Sender and Receiver latency reported by their implementation. Table 3 contains the resulting data. The variance in set sizes and available bandwidth captures the cut-off point quite precisely. Our protocol is less efficient for the Sender than [RT21] for the smallest sets $n = m = 2^8$ due to higher communication requirements (20.67 vs 9.79 KB). By growing the Receiver's set size to 2^{10} such that the communication requirements are similar for both protocols, we observe that the performance difference is reduced, and latencies are competitive under the more constrained bandwidth of 1 Mbps, with a small gain for our protocol. With the largest Receiver's set size, the constant communication complexity for the Sender in our protocol starts to clearly pay off and the resulting Sender latency is from 2.3 to 4.6 times lower. Naturally, this comes at the expense of the Receiver's complexity which is much greater than in [RT21]. Hence, we can observe that our protocol is more efficient than [RT21] in terms of Sender latency for unbalanced yet small set sizes under rigorous bandwidth constraints.

Lastly, we benchmarked the state-of-the-art unbalanced PSI protocol by Cong et al. [CMdG$^+$21] under the same network conditions as before, also using their implementation and the parameter configuration for available for the smallest sets. Because the provided benchmarks only measured steps within the protocol, we instrumented their implementation to measure latency in the outer Sender and Receiver functions in order to make sure that all communication was captured. We can again observe that the communication requirements of the protocol (a factor of 55 higher) penalize Sender latency substantially, especially under worse network conditions. For this set of benchmarks, the Receiver latency does not appear penalized because the of how the bulk of communication is scheduled (the receiver ends as soon as the final transmission step starts).

References

ABC$^+$22. Diego F. Aranha, Emil Madsen Bennedsen, Matteo Campanelli, Chaya Ganesh, Claudio Orlandi, and Akira Takahashi. ECLIPSE: enhanced compiling method for pedersen-committed zksnark engines. In Goichiro Hanaoka, Junji Shikata, and Yohei Watanabe, editors, *Public-Key Cryptography - PKC 2022 - 25th IACR International Conference on Practice and Theory of Public-Key Cryptography, Virtual Event,* *March 8-11, 2022, Proceedings, Part I*, volume 13177 of *Lecture Notes in Computer Science*, pages 584–614. Springer, 2022.

ABD$^+$21. Navid Alamati, Pedro Branco, Nico Döttling, Sanjam Garg, Mohammad Hajiabadi, and Sihang Pu. Laconic private set intersection and applications. In Kobbi Nissim and Brent Waters, editors, *Theory of Cryptography - 19th International Conference, TCC 2021, Raleigh, NC, USA, November 8-11, 2021, Proceedings, Part III*, volume 13044 of *Lecture Notes in Computer Science*, pages 94–125. Springer, 2021.

AC20. Thomas Attema and Ronald Cramer. Compressed Σ-protocol theory and practical application to plug & play secure algorithmics. In Daniele Micciancio and Thomas Ristenpart, editors, *CRYPTO 2020, Part III*, volume 12172 of *LNCS*, pages 513–543. Springer, Heidelberg, August 2020.

ADT11. Giuseppe Ateniese, Emiliano De Cristofaro, and Gene Tsudik. (If) size matters: Size-hiding private set intersection. In Dario Catalano, Nelly Fazio, Rosario Gennaro, and Antonio Nicolosi, editors, *PKC 2011*, volume 6571 of *LNCS*, pages 156–173. Springer, Heidelberg, March 2011.

AGM$^+$. D. F. Aranha, C. P. L. Gouvêa, T. Markmann, R. S. Wahby, and K. Liao. RELIC is an Efficient LIbrary for Cryptography. `https://github.com/relic-toolkit/relic`.

ATSM09. Man Ho Au, Patrick P. Tsang, Willy Susilo, and Yi Mu. Dynamic universal accumulators for DDH groups and their application to attribute-based anonymous credential systems. In Marc Fischlin, editor, *CT-RSA 2009*, volume 5473 of *LNCS*, pages 295–308. Springer, Heidelberg, April 2009.

BB08. Dan Boneh and Xavier Boyen. Short signatures without random oracles and the SDH assumption in bilinear groups. *Journal of Cryptology*, 21(2):149–177, April 2008.

BCC$^+$16. Jonathan Bootle, Andrea Cerulli, Pyrros Chaidos, Jens Groth, and Christophe Petit. Efficient zero-knowledge arguments for arithmetic circuits in the discrete log setting. In Marc Fischlin and Jean-Sébastien Coron, editors, *EUROCRYPT 2016, Part II*, volume 9666 of *LNCS*, pages 327–357. Springer, Heidelberg, May 2016.

Bd94. Josh Cohen Benaloh and Michael de Mare. One-way accumulators: A decentralized alternative to digital sinatures (extended abstract). In Tor Helleseth, editor, *EUROCRYPT'93*, volume 765 of *LNCS*, pages 274–285. Springer, Heidelberg, May 1994.

BD16. Elaine Barker and Quynh Dang. Nist special publication 800-57 part 1, revision 4. *NIST, Tech. Rep*, 16, 2016.

CDG$^+$17. Chongwon Cho, Nico Döttling, Sanjam Garg, Divya Gupta, Peihan Miao, and Antigoni Polychroniadou. Laconic oblivious transfer and its applications. In Jonathan Katz and Hovav Shacham, editors, *CRYPTO 2017, Part II*, volume 10402 of *LNCS*, pages 33–65. Springer, Heidelberg, August 2017.

CDK$^+$12. Ronald Cramer, Ivan Damgård, Eike Kiltz, Sarah Zakarias, and Angela Zottarel. DDH-like assumptions based on extension rings. In Marc Fischlin, Johannes Buchmann, and Mark Manulis, editors, *PKC 2012*, volume 7293 of *LNCS*, pages 644–661. Springer, Heidelberg, May 2012.

CHLR18. Hao Chen, Zhicong Huang, Kim Laine, and Peter Rindal. Labeled PSI from fully homomorphic encryption with malicious security. In David Lie, Mohammad Mannan, Michael Backes, and XiaoFeng Wang, editors, *ACM CCS 2018*, pages 1223–1237. ACM Press, October 2018.

CKS09. Jan Camenisch, Markulf Kohlweiss, and Claudio Soriente. An accumulator based on bilinear maps and efficient revocation for anonymous credentials. In Stanislaw Jarecki and Gene Tsudik, editors, *PKC 2009*, volume 5443 of *LNCS*, pages 481–500. Springer, Heidelberg, March 2009.

CL02. Jan Camenisch and Anna Lysyanskaya. Dynamic accumulators and application to efficient revocation of anonymous credentials. In Moti Yung, editor, *CRYPTO 2002*, volume 2442 of *LNCS*, pages 61–76. Springer, Heidelberg, August 2002.

CLR17. Hao Chen, Kim Laine, and Peter Rindal. Fast private set intersection from homomorphic encryption. In Bhavani M. Thuraisingham, David Evans, Tal Malkin, and Dongyan Xu, editors, *ACM CCS 2017*, pages 1243–1255. ACM Press, October / November 2017.

CMdG$^+$21. Kelong Cong, Radames Cruz Moreno, Mariana Botelho da Gama, Wei Dai, Ilia Iliashenko, Kim Laine, and Michael Rosenberg. Labeled PSI from homomorphic encryption with reduced computation and communication. In Giovanni Vigna and Elaine Shi, editors, *ACM CCS 2021*, pages 1135–1150. ACM Press, November 2021.

CZ81. David G Cantor and Hans Zassenhaus. A new algorithm for factoring polynomials over finite fields. *Mathematics of Computation*, pages 587–592, 1981.

DCW13. Changyu Dong, Liqun Chen, and Zikai Wen. When private set intersection meets big data: an efficient and scalable protocol. In Ahmad-Reza Sadeghi, Virgil D. Gligor, and Moti Yung, editors, *ACM CCS 2013*, pages 789–800. ACM Press, November 2013.

DGGM19. Nico Döttling, Sanjam Garg, Vipul Goyal, and Giulio Malavolta. Laconic conditional disclosure of secrets and applications. In David Zuckerman, editor, *60th FOCS*, pages 661–685. IEEE Computer Society Press, November 2019.

DGI$^+$19. Nico Döttling, Sanjam Garg, Yuval Ishai, Giulio Malavolta, Tamer Mour, and Rafail Ostrovsky. Trapdoor hash functions and their applications. In Alexandra Boldyreva and Daniele Micciancio, editors, *CRYPTO 2019, Part III*, volume 11694 of *LNCS*, pages 3–32. Springer, Heidelberg, August 2019.

DPT20. Thai Duong, Duong Hieu Phan, and Ni Trieu. Catalic: Delegated PSI cardinality with applications to contact tracing. In Shiho Moriai and Huaxiong Wang, editors, *ASIACRYPT 2020, Part III*, volume 12493 of *LNCS*, pages 870–899. Springer, Heidelberg, December 2020.

DT08. Ivan Damgård and Nikos Triandopoulos. Supporting non-membership proofs with bilinear-map accumulators. Cryptology ePrint Archive, Report 2008/538, 2008. `https://eprint.iacr.org/2008/538`.

DY05. Yevgeniy Dodis and Aleksandr Yampolskiy. A verifiable random function with short proofs and keys. In Serge Vaudenay, editor, *PKC 2005*, volume 3386 of *LNCS*, pages 416–431. Springer, Heidelberg, January 2005.

FNP04. Michael J. Freedman, Kobbi Nissim, and Benny Pinkas. Efficient private matching and set intersection. In Christian Cachin and Jan Camenisch, editors, *EUROCRYPT 2004*, volume 3027 of *LNCS*, pages 1–19. Springer, Heidelberg, May 2004.

GKM$^+$18. Jens Groth, Markulf Kohlweiss, Mary Maller, Sarah Meiklejohn, and Ian Miers. Updatable and universal common reference strings with applications to zk-SNARKs. In Hovav Shacham and Alexandra Boldyreva, editors, *CRYPTO 2018, Part III*, volume 10993 of *LNCS*, pages 698–728. Springer, Heidelberg, August 2018.

KB21. Ioanna Karantaidou and Foteini Baldimtsi. Efficient constructions of pairing based accumulators. In Ralf Küsters and Dave Naumann, editors, *CSF 2021 Computer Security Foundations Symposium*, pages 1–16. IEEE Computer Society Press, 2021.

KKRT16. Vladimir Kolesnikov, Ranjit Kumaresan, Mike Rosulek, and Ni Trieu. Efficient batched oblivious PRF with applications to private set intersection. In Edgar R. Weippl, Stefan Katzenbeisser, Christopher Kruegel, Andrew C. Myers, and Shai Halevi, editors, *ACM CCS 2016*, pages 818–829. ACM Press, October 2016.

KLS$^+$17. Ágnes Kiss, Jian Liu, Thomas Schneider, N. Asokan, and Benny Pinkas. Private set intersection for unequal set sizes with mobile applications. *PoPETs*, 2017(4):177–197, October 2017.

KRS$^+$19. Daniel Kales, Christian Rechberger, Thomas Schneider, Matthias Senker, and Christian Weinert. Mobile private contact discovery at scale. In Nadia Heninger and Patrick Traynor, editors, *USENIX Security 2019*, pages 1447–1464. USENIX Association, August 2019.

KS05. Lea Kissner and Dawn Xiaodong Song. Privacy-preserving set operations. In Victor Shoup, editor, *CRYPTO 2005*, volume 3621 of *LNCS*, pages 241–257. Springer, Heidelberg, August 2005.

LLX07. Jiangtao Li, Ninghui Li, and Rui Xue. Universal accumulators with efficient nonmembership proofs. In Jonathan Katz and Moti Yung, editors, *ACNS 07*, volume 4521 of *LNCS*, pages 253–269. Springer, Heidelberg, June 2007.

Mar14. Moxie Marlinspike. The difficulty of private contact discovery. `whispersystems.org/blog/contact-discovery.`, 2014.

Mea86. Catherine A. Meadows. A more efficient cryptographic matchmaking protocol for use in the absence of a continuously available third party. In *Proceedings of the 1986 IEEE Symposium on Security and Privacy, Oakland, California, USA, April 7-9, 1986*, pages 134–137, 1986.

MSS16. Alfred Menezes, Palash Sarkar, and Shashank Singh. Challenges with assessing the impact of NFS advances on the security of pairing-based cryptography. In *Mycrypt*, volume 10311 of *Lecture Notes in Computer Science*, pages 83–108. Springer, 2016.

Ngu05. Lan Nguyen. Accumulators from bilinear pairings and applications. In Alfred Menezes, editor, *CT-RSA 2005*, volume 3376 of *LNCS*, pages 275–292. Springer, Heidelberg, February 2005.

NMH$^+$10. Shishir Nagaraja, Prateek Mittal, Chi-Yao Hong, Matthew Caesar, and Nikita Borisov. Botgrep: Finding P2P bots with structured graph analysis. In *19th USENIX Security Symposium, Washington, DC, USA, August 11-13, 2010, Proceedings*, pages 95–110, 2010.

PRTY19. Benny Pinkas, Mike Rosulek, Ni Trieu, and Avishay Yanai. SpOT-light: Lightweight private set intersection from sparse OT extension. In Alexandra Boldyreva and Daniele Micciancio, editors, *CRYPTO 2019, Part III*, volume 11694 of *LNCS*, pages 401–431. Springer, Heidelberg, August 2019.

PRTY20. Benny Pinkas, Mike Rosulek, Ni Trieu, and Avishay Yanai. PSI from PaXoS: Fast, malicious private set intersection. In Anne Canteaut and Yuval Ishai, editors, *EUROCRYPT 2020, Part II*, volume 12106 of *LNCS*, pages 739–767. Springer, Heidelberg, May 2020.

PSSZ15. Benny Pinkas, Thomas Schneider, Gil Segev, and Michael Zohner. Phasing: Private set intersection using permutation-based hashing. In *24th USENIX Security Symposium, USENIX Security 15, Washington, D.C., USA, August 12-14, 2015.*, pages 515–530, 2015.

PSZ14. Benny Pinkas, Thomas Schneider, and Michael Zohner. Faster private set intersection based on OT extension. In Kevin Fu and Jaeyeon Jung, editors, *USENIX Security 2014*, pages 797–812. USENIX Association, August 2014.

QWW18. Willy Quach, Hoeteck Wee, and Daniel Wichs. Laconic function evaluation and applications. In Mikkel Thorup, editor, *59th FOCS*, pages 859–870. IEEE Computer Society Press, October 2018.

RA18. Amanda C. Davi Resende and Diego F. Aranha. Faster unbalanced private set intersection. In Sarah Meiklejohn and Kazue Sako, editors, *FC 2018*, volume 10957 of *LNCS*, pages 203–221. Springer, Heidelberg, February / March 2018.

RT21. Mike Rosulek and Ni Trieu. Compact and malicious private set intersection for small sets. In Giovanni Vigna and Elaine Shi, editors, *ACM CCS 2021*, pages 1166–1181. ACM Press, November 2021.

Sho93. Victor Shoup. Factoring polynomials over finite fields: asymptotic complexity vs. reality. In *Proc. IMACS Symposium, Lille, France.* Citeseer, 1993.

Sho97. Victor Shoup. Lower bounds for discrete logarithms and related problems. In Walter Fumy, editor, *EUROCRYPT'97*, volume 1233 of *LNCS*, pages 256–266. Springer, Heidelberg, May 1997.

vzGS92. Joachim von zur Gathen and Victor Shoup. Computing frobenius maps and factoring polynomials (extended abstract). In *24th ACM STOC*, pages 97–105. ACM Press, May 1992.

A Standard Ideal Functionalities

The Ideal Functionality $\mathcal{F}_{\mathsf{PSI}}$

The functionality $\mathcal{F}_{\mathsf{PSI}}$ works with two parties, the sender $\mathcal{S}$ and the receiver $\mathcal{R}$, and an ideal world adversary $\mathcal{A}$, controlling the corrupted parties. The functionality is parameterized with two integers m, n specifying the maximum set size for $\mathcal{R}, \mathcal{S}$ respectively.
Receiver Input. Upon receiving input (sid, Rec, X) from the receiver $\mathcal{R}$, ignore the message if another message of the form $(sid, \mathsf{Rec}, *)$ had been received earlier or if $|X| > m$. Otherwise, store the message and send (sid, Rec) to the adversary.
Sender Input. Upon receiving input (sid, Sen, Y) from the sender $\mathcal{S}$, ignore the message if another message of the form $(sid, \mathsf{Sen}, *)$ had been received earlier or if $|Y| > n$. Otherwise, output (sid, Sen) to the adversary.
Output. When the adversary sends $(sid, \mathsf{continue})$, if messages for session sid where received by both parties, send $(sid, \mathsf{Output}, Y = X \cap Y)$ to the receiver $\mathcal{R}$.

Fig. 4: The PSI ideal functionality $\mathcal{F}_{\mathsf{PSI}}$

The Ideal Functionality $\mathcal{F}_{\mathsf{zk}}$

The zero-knowledge functionality $\mathcal{F}_{\mathsf{zk}}$ works with two parties, a prover $\mathcal{P}$ and a verifier $\mathcal{V}$. The functionality is parameterized with a relation $R(x, w)$.
Verifier Input. Upon receiving input $(sid, \mathsf{V}, x_{\mathsf{V}})$ from the verifier $\mathcal{V}$, ignore the message if another message of the form $(sid, \mathsf{V}, *)$ had been received earlier. Otherwise, store the message and send $(sid, \mathsf{V}, x_{\mathsf{V}})$ to the adversary.
Prover Input. Upon receiving input $(sid, \mathsf{P}, x_{\mathsf{P}}, w)$ from the prover $\mathcal{P}$, ignore the message if another message of the form $(sid, \mathsf{P}, *)$ had been received earlier. Otherwise, send $(sid, \mathsf{P}, x_{\mathsf{P}})$ to the adversary.
Output. When the adversary sends $(sid, \mathsf{continue})$, if messages for session sid where received by both parties such that $x = x_{\mathsf{V}} = x_{\mathsf{P}}$ output $(sid, \mathsf{Output}, R(x, w))$ to the verifier $\mathcal{V}$.

Fig. 5: The Zero-Knowledge ideal functionality $\mathcal{F}_{\mathsf{zk}}$.

B Active Secure Protocol

Proof of Theorem 2..

Proof. Correctness of the modified protocol is immediate by inspection.

To prove security against an actively corrupted receiver $\mathcal{R}^*$, we build the following simulator: the simulator:

1. Produces strings $(\mathsf{setup}_{\mathcal{S}}, \mathsf{setup}_{\mathcal{R}})$ as in the protocol and sends them to $\mathcal{R}^*$;
2. Receives (σ, R) from the receiver $\mathcal{R}^*$;
3. Receives the witness $(\omega_0, \ldots, \omega_m)$ from the $\mathcal{F}_{\mathsf{zk}}$ functionality;
4. If the witness does not satisfy the relation, abort. Otherwise, interpret the witness as the coefficients of a polynomial Ω. Find the roots $\tilde{X}$ of the polynomial Ω using an polynomial factoring algorithm [CZ81, vzGS92, Sho93].
5. Recover the input set X from $\tilde{X}$ by subtracting σ from all entries.
6. Send the input X to $\mathcal{F}_{\mathsf{PSI}}$ on behalf of the corrupted receiver;
7. Receive the output set Z;
8. Complete the simulation of **Second Round** exactly as the semi-honest simulator i.e., for each $z \in Z$ pick a random index j and compute (T_j, U_j) as in the protocol. For all remaining indices j, sample uniformly random pairs (T_j, U_j);
9. Trivially simulate the zero-knowledge proof using the functionality $\mathcal{F}_{\mathsf{zk}}$;

Indistinguishability of the view of $\mathcal{R}^*$ follows since the output Z received from the functionality is the same as the output in the real protocol, since the polynomial Ω extracted by the simulator is the same as the polynomial P used in the protocol up to a multiplicative factor. Then, the argument follows the same idea as in the semi-honest protocol, since for any choice of $R \neq 1_{\mathbb{G}_2}$ the distribution of (T_j, U_j) in the real protocol is indistinguishable from random under the $(1, B)$-SBDDH assumption as showed in Theorem 1.

To prove security against an actively corrupted sender $\mathcal{S}^*$ we build the following simulator:

1. The simulator receives $(\mathsf{setup}_{\mathcal{S}}, \mathsf{setup}_{\mathcal{R}})$ from the malicious sender $\mathcal{S}^*$, performs the same checks as an honest receiver would and aborts if any check fails;
2. The simulator sends an uniformly random R and σ to $\mathcal{S}^*$ (and trivially simulates the zero-knowledge proof using the functionality $\mathcal{F}_{\mathsf{zk}}$);
3. Then for each $j \in [n]$:
   (a) The simulator receives the tuples (T_j, U_j) from the corrupted $\mathcal{S}^*$ and aborts if any $U_j = 1_{\mathbb{G}_2}$ or $T_j = 1_{\mathbb{G}_T}$;
   (b) The simulator receives the witnesses (t_j, α_j) from the $\mathcal{F}_{\mathsf{zk}}$ functionality and aborts if the witness does not satisfy the relation;
   (c) The simulator computes $\tilde{y}_j = -\alpha_j / t_j$ (note that $T_j \neq 1_{\mathbb{G}_T}$ implies $t_j \neq 0$);
   (d) The simulator computes $y_j = \tilde{y}_j - \sigma$;
4. Finally the simulator inputs the set $Y = \{y_1, \ldots, y_n\}$ to the $\mathcal{F}_{\mathsf{PSI}}$ functionality on behalf of the corrupted sender.

Indistinguishability of the view of the corrupted $\mathcal{S}^*$ follows since, as argued earlier, the distribution of R in the real protocol is perfectly indistinguishable from the uniform distribution for any (non-trivial) choice of the setup parameters and the output of the honest $\mathcal{R}$ in the ideal world is distributed as in the real protocol since we successfully extract the input of the sender.

C Semi-honest Laconic PSI Protocols

We present in Figures 7 and 8 the protocols of [ADT11, ABD$^+$21] using our notation and applying our optimizations. Note that the original description of [ABD$^+$21] the functions H, F are respectively a programmable pseudorandom function and a seeded extractor, instead of two random oracles. In order to help the performances of their protocol towards achieving a comparison as fair as possible, we implement both of these primitivies with hash functions modeled as random oracles instead.

The Active Secure PSI Protocol Π^*_{PSI}

Inputs/Outputs: The protocol is run between a receiver $\mathcal{R}$ with input $X = \{x_1, \ldots, x_m\}$ and a sender $\mathcal{S}$ with input $Y = \{y_1, \ldots, y_n\}$. We assume the inputs can efficiently be encoded as elements of $\mathbb{Z}_q$. At the end the receiver learns $Z = X \cap Y$ while the sender learns nothing.

Setup: The receiver and the sender agree on a bilinear map $e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$ where the generators are $g_1, g_2, g_T = e(g_1, g_2)$ respectively. During the setup phase a party trusted by $\mathcal{S}$ picks a random $s \in \mathbb{Z}_q^*$, then computes and send:

$$\mathsf{setup}_{\mathcal{R}} = (S', S_1, S_2, \ldots, S_m) \text{ and } \mathsf{setup}_{\mathcal{S}} = (S')$$

to $\mathcal{R}$ and $\mathcal{S}$ respectively where $S' = g_1^s$ and $S_i = g_2^{s^i}$ for $i = 0, \ldots, m$. The receiver checks that for all $i = [m]$, $S_i \neq 1_{\mathbb{G}_2}$ and that

$$e(S', S_{i-1}) = e(g_1, S_i)$$

First Round: In any session *sid*, the receiver $\mathcal{R}$ sends

$$\mathsf{msg1} = (sid, \sigma, R, \pi_{\mathcal{S}})$$

to the sender, computed as follows:

1. Pick a random $\sigma \xleftarrow{\$} \mathbb{Z}_q$;
2. Compute the set $\tilde{X} = \{\tilde{x}_1, \ldots, \tilde{x}_m\}$ as $\tilde{x}_i = x_i + \sigma$;
3. Pick random $r \xleftarrow{\$} \mathbb{Z}_q$;
4. Compute $R = \left(\prod_{i=0}^m S_i^{p(\tilde{X},i)}\right)^r$;
5. If $R = 1_{\mathbb{G}_2}$, then let $R = (S_0)^r$ instead.
6. Compute a zero-knowledge proof $\pi_{\mathcal{S}}$ that R is computed according to the protocol specification using the relation $R_{\mathcal{S}}$ defined in Section 4.

Second round: In any session *sid*, the sender $\mathcal{S}$ checks that $R \neq 1_{\mathbb{G}_2}$ and if so sends to the receiver $\mathcal{R}$ the message

$$\mathsf{msg2} = (sid, T_1, U_1, \ldots, T_n, U_n, \pi_{\mathcal{R}})$$

computed as follows. Pick a random permutation $\pi : [n] \to [n]$. Then for all $j \in [n]$:

1. Pick random $t_j \xleftarrow{\$} \mathbb{Z}_q$;
2. Compute the set $\tilde{Y} = \{\tilde{y}_1, \ldots, \tilde{y}_n\}$ as $\tilde{y}_i = y_i + \sigma$;
3. Compute
$$(T_j, U_j) = \left(\left(e\left(g_1^{t_j}, R\right)\right), \left(S' \cdot g_1^{-\tilde{y}_{\pi(j)}}\right)^{t_j}\right)$$
4. Compute a zero-knowledge proof $\pi_{\mathcal{R}}$ that each (T_j, U_j) is computed according to the protocol specification using the relation $R_{\mathcal{R}}$ defined in Section 4.

Retrieve Output: To retrieve the output from a session *sid*, the receiver $\mathcal{R}$ does the following; Check that for all $j \in [n]$ $T_j \neq 1_{\mathbb{G}_T}$ and $U_j \neq 1_{\mathbb{G}_2}$ and abort otherwise; Else, for all $j \in [n], k \in [m]$, the receiver initializes $Z = \emptyset$ and then does the following:

1. Compute
$$R_{-k} = \left(\prod_{i=0}^{m-1} S_i^{p(\tilde{X}_{-k},i)}\right)^r$$
2. If $e(U_j, R_{-k}) \overset{?}{=} T_j$, then add x_k to the output Z;

The receiver outputs Z.

Fig. 6: Our Actively Secure Private Set Intersection Protocol

The Protocol $\Pi_{\mathsf{PSI}}^{\mathsf{ADT11}}$

Let λ be security parameter and λ_1, λ_2 be the ones that depend on λ, $H(\cdot), F(\cdot)$ be two random oracles such that $H : \{0,1\}^* \to \{0,1\}^{\lambda_1}$ and $F : \{0,1\}^* \to \{0,1\}^{\lambda_2}$.

Inputs/Outputs: The protocol is run between a receiver $\mathcal{R}$ with input $X = \{x_1, \ldots, x_m\}$ and a sender $\mathcal{S}$ with input $Y = \{y_1, \ldots, y_n\}$. At the end the receiver learns $Z = X \cap Y$ while the sender learns nothing.

Setup: A trusted party samples $N \xleftarrow{\$} \mathsf{RSA}(\lambda)$ where $N = PQ$ and P, Q are safe prime numbers, i.e., $P = 2P' + 1, Q = 2Q' + 1$, a uniformly random generator g of the set of quadratic residues in $\mathbb{Z}_N^*$. The party distributes (P', Q') to the sender $\mathcal{S}$ and output the common reference string

$$\mathsf{setup} = (N, g, k).$$

First Round: In any session *sid*, the receiver $\mathcal{R}$ sends

$$\mathsf{msg1} = (sid, R)$$

to the sender, computed as follows:

1. Pick random $r \xleftarrow{\$} [N]$;
2. Compute $R = g^{r \prod_{i \in [m]} H(x_i)} \bmod N$ if $X \neq \emptyset$;
3. Compute $R = g^r \bmod N$ if $X = \emptyset$.

Second round: In any session *sid*, the sender $\mathcal{S}$ sends to the receiver $\mathcal{R}$ the message

$$\mathsf{msg2} = (sid, U, T_1, \ldots, T_n)$$

computed as follows. Sample a random $t \xleftarrow{\$} \{0, \ldots, P'Q' - 1\}$ and compute $U = g^t \bmod N$. Pick a random permutation $\pi : [n] \to [n]$ and for all $j \in [n]$, compute

$$T_j = F\left(R^{t \cdot (1/H(y_{\pi(j)}))} \bmod N\right)$$

Retrieve Output: To retrieve the output from a session *sid*, the receiver $\mathcal{R}$ initializes $Z = \emptyset$ and then does the following: For all $k \in [m]$,

1. Compute

$$R_{-k} = U^{r \cdot \prod_{i \in [m] \setminus k} H(x_i)} \bmod N$$

2. If there exist a $j \in [m]$ such that $F(R_{-k}) \stackrel{?}{=} T_j$, then add x_k to the output Z;

The receiver outputs Z.

Fig. 7: The (non-laconic) PSI protocols of [ADT11] based on RSA-accumulators.

The Protocol $\Pi_{\mathsf{PSI}}^{\mathsf{ABD+21}}$

Let $\mathcal{U}$ be a universe which contains the input sets of the parties. Let $\kappa \in \mathbb{Z}$ such that $5\kappa \leq \lambda$. Let $H(\cdot), F(\cdot)$ be two random oracles such that $H : \{0,1\}^* \to \mathsf{Primes}(\kappa)$ and $F : \{0,1\}^* \to \{0,1\}^\lambda$.

Inputs/Outputs: The protocol is run between a receiver $\mathcal{R}$ with input $X = \{x_1, \ldots, x_m\}$ and a sender $\mathcal{S}$ with input $Y = \{y_1, \ldots, y_n\}$. At the end the receiver learns $Z = X \cap Y$ while the sender learns nothing.

Setup: A trusted party samples $N \xleftarrow{\$} \mathsf{RSA}(\lambda)$ where $N = PQ$ and P, Q are safe prime numbers, a uniformly random generator $g \in \mathbb{Z}_N^*$. The party outputs

$$\mathsf{setup} = (N, g).$$

First Round: In any session sid, the receiver $\mathcal{R}$ sends

$$\mathsf{msg1} = (sid, R)$$

to the sender, computed as follows:

1. Pick random $r \xleftarrow{\$} [N]$;
2. Compute $R = g^{r \prod_{i\in[m]} H(x_i)} \bmod N$ if $X \neq \emptyset$;
3. Compute $R = g^r \bmod N$ if $X = \emptyset$.

Second round: In any session sid, the sender $\mathcal{S}$ sends to the receiver $\mathcal{R}$ the message

$$\mathsf{msg2} = (sid, T_1, U_1, \ldots, T_n, U_n)$$

computed as follows. Pick a random permutation $\pi : [n] \to [n]$. Then for all $j \in [n]$:

1. Pick random $t_j \xleftarrow{\$} [N]$;
2. Compute
$$(T_j, U_j) = \left(F\left(R^{t_j} \bmod N\right), g^{t_j \cdot H\left(y_{\pi(j)}\right)} \bmod N\right)$$

Retrieve Output: To retrieve the output from a session sid, the receiver $\mathcal{R}$ initializes $Z = \emptyset$ and then does the following: For all $j \in [n], k \in [m]$,

1. Compute
$$R_{-k,j} = U_j^{r \cdot \prod_{i\in[m]\setminus k} H(x_i)} \bmod N$$
2. If $F\left(R_{-k,j}\right) \overset{?}{=} T_j$, then add x_k to the output Z;

The receiver outputs Z.

Fig. 8: The laconic PSI protocols of [ABD$^+$21] based on RSA-accumulators.