Information Theoretically Secure Multi Party Set Intersection Re-visited

Arpita Patra*, Ashish Choudhary**, and C. Pandu Rangan***

Dept of Computer Science and Engineering
IIT Madras, Chennai India 600036
arpitapatra10@gmail.com, partho_31@yahoo.co.in, prangan55@gmail.com

Abstract. We re-visit the problem of secure multiparty set intersection (MPSI) in information theoretic settings. In [15], Li et.al have proposed a protocol for MPSI with $n = 3t + 1$ parties, that provides information theoretic security, when t out of those n parties are corrupted by an active adversary having *unbounded computing power*. In [15], the authors have claimed that their protocol takes six rounds of communication and communicates $\mathcal{O}(n^4m^2)$ field elements, where each party has a set containing m field elements. However, we show that the round and communication complexity of the protocol in [15] is much more than what is claimed in [15]. We then propose a *novel* information theoretically secure protocol for MPSI with $n \geq 3t+1$, which significantly improves the "actual" round and communication complexity of the protocol of [15]. Our protocols employ several tools which are of independent interest.

Keywords: Multiparty Computation, Information Theoretic Security.

1 Introduction

Secure Multiparty Set Intersection (MPSI): Consider a complete synchronous network $\mathcal{N}$, consisting of n parties $\mathcal{P} = \{P_1, \ldots, P_n\}$, who are pairwise connected by a reliable and private channel. The parties do not trust each other and the distrust in the network is modeled by a centralized adversary $\mathcal{A}_t$, who has *unbounded computing* power and can actively corrupt at most t parties in Byzantine fashion, where $t < \frac{n}{3}$. A Byzantine (or actively) corrupted party is under complete control of $\mathcal{A}_t$, who may force the party to behave arbitrarily. Any protocol over $\mathcal{N}$ is assumed to operate in a sequence of rounds. In each round, a party performs some local computation, sends new messages to the other parties through the private channels and publicly *broadcasts* some information, receives the messages that were sent by the other parties in current round on the private channels and the messages that were publicly broadcast by the other parties in

* Financial Support from Microsoft Research India Acknowledged.
** Financial Support from Infosys Technology India Acknowledged.
*** Work Supported by Project No. CSE/05-06/076/DITX/CPAN on Protocols for Secure Communication and Computation Sponsored by Department of Information Technology, Government of India.

M.J. Jacobson Jr., V. Rijmen, and R. Safavi-Naini (Eds.): SAC 2009, LNCS 5867, pp. 71–91, 2009.

current round. Here broadcast is a primitive, which allows a party to send some information *identically* to all other parties. If a physical broadcast channel is available in the system, then broadcast will take one round. Otherwise, we can simulate broadcast using a protocol among the parties in $\mathcal{P}$, which will have the same effect as a physical broadcast channel. Each party P_i has a private data-set S_i, containing m elements from a finite field $\mathbb{F}$, where $|\mathbb{F}| \geq n$. The goal of MPSI is to design a protocol that can compute the intersection of these n sets, satisfying the following properties:

1. CORRECTNESS: At the end of the protocol, each honest party correctly gets the intersection of the n sets, irrespective of the behavior of $\mathcal{A}_t$ and
2. SECRECY: The protocol should not leak any *extra* information to the corrupted parties, other than what is implied by the input of the corrupted parties (i.e., the data-sets possessed by corrupted parties) and the final output (i.e., the intersection of all the n data-sets).

MPSI problem is an interesting secure distributed computing problem and has huge practical applications such as online recommendation services, medical databases, data mining etc. [10].

Existing Literature on MPSI: The MPSI problem was first studied in *cryptographic* model in [10, 14], under the assumption that the adversary has *bounded computing power*. By representing the data-sets as polynomials, the set intersection problem is converted into the task of computing the common roots of n polynomials in [10, 14]. This is done as follows: Let $S = \{s_1, s_2, \ldots, s_m\}$ be a set of size m, where $\forall i, s_i \in \mathbb{F}$. Now set S can be represented by a polynomial $f(x)$ of degree-m, where $f(x) = \prod_{i=1}^{m}(x - s_i) = a_0 + a_1 x + \ldots + a_m x^m$. It is obvious that if an element s is a root of $f(x)$, then s is a root of $r(x)f(x)$ too, where $r(x)$ is a *random* polynomial of degree-m over $\mathbb{F}$. Now for MPSI, party P_i represents his set S_i, by a degree-m polynomial $f^{(P_i)}(x)$ and supplies $f^{(P_i)}(x)$ (i.e. its $m+1$ coefficients), as his input, in a secure manner. Then all the parties jointly and securely compute

$$F(x) = (r^{(1)}(x)f^{(P_1)}(x) + r^{(2)}(x)f^{(P_2)}(x) + \ldots + r^{(n)}(x)f^{(P_n)}(x)) \qquad (1)$$

where $r^{(1)}(x), \ldots r^{(n)}(x)$ are n secret random polynomials of degree-m over $\mathbb{F}$, jointly generated by the n parties. Note that $F(x)$ preserves all the common roots of $f^{(P_1)}(x), \ldots, f^{(P_n)}(x)$. Every element $s \in (S_1 \cap S_2 \cap \ldots \cap S_n)$ is a root of $F(x)$, i.e. $F(s) = 0$. Hence after computing $F(x)$ in a secure manner, it can be reconstructed by every party, who locally checks if $F(s) = 0$ for every s in his private set. All s's at which the evaluation of $F(x)$ is zero forms the intersection set $(S_1 \cap S_2 \cap \ldots \cap S_n)$. In [14], it has been proved formally that $F(x)$ does not reveal any *extra* information to the adversary, other than what is deduced from $(S_1 \cap S_2 \cap \ldots \cap S_n)$ and input set S_i of the corrupted parties.

Remark 1. Even though every $s \in (S_1 \cap S_2 \cap \ldots \cap S_n)$ is a root of $F(x)$, there may exist some $s' \in \mathbb{F}$, such that $F(s') = 0$, even though $s' \notin (S_1 \cap S_2 \cap \ldots \cap S_n)$. This is possible if s' happens to be the common root of all $r^{(i)}(x)$'s. However, as stated in [14], the probability of this event is negligible.

In [14], the MPSI problem is solved by securely computing $F(x)$, assuming $\mathcal{A}_t$ to be *computationally bounded*. In [15], the authors have presented the first information theoretically secure protocol for MPSI, assuming $\mathcal{A}_t$ to be *computationally unbounded* and $n \geq 3t+1$. Specifically, the authors have shown how to securely compute $F(x)$ in the presence of a computationally unbounded $\mathcal{A}_t$. To the best of our knowledge, this is the only known information theoretically secure MPSI protocol. *Notice that, although not explicitly stated in [15], the MPSI protocol of [15] involves a negligible error probability in* CORRECTNESS. *This is due to the argument given in Remark 1.*

Our Motivation and Contribution: The authors in [15] claimed that their MPSI protocol takes *six* rounds and communicates $\mathcal{O}(n^4m^2)$ elements from $\mathbb{F}$. However, we show that the round and communication complexity of the MPSI protocol of [15] is much more than what is claimed in [15]. We then propose a new information theoretically secure protocol for MPSI with $n \geq 3t+1$, which significantly improves the "actual" round and communication complexity of the MPSI protocol given in [15].

2 Analysis of the MPSI Protocol of [15]

In order to securely compute $F(x)$ given in (1) against a computationally unbounded $\mathcal{A}_t$, the MPSI protocol of [15] is divided into three phases. We briefly recall the steps performed in first two phases, which are the most expensive phases in terms of round and communication complexity.

1. Input Phase: Here each party represents his private data-set as a polynomial and t-shares the coefficients of the polynomial among the n parties. To do so, the parties use a two dimensional verifiable secret sharing (VSS). A two dimensional VSS [9, 11, 13], ensures that each party (including a corrupted party) "consistently" and correctly t-shares the coefficients of his polynomial with everybody. Now, the authors in [15] claimed that this takes two rounds, where in the first round, each party does the sharing and in the second round verification is done by all parties to ensure whether everybody has received correct and consistent shares (see sec. 4.2 in [15]). However, no estimation is done for the communication complexity of this phase. Now it is well known that the minimum number of rounds taken by any VSS protocol with $n \geq 3t+1$ is at least *three* [9, 11, 13]. Moreover, the current best three round VSS protocol with $n = 3t+1$ requires a private communication of $\mathcal{O}(n^3)$ and broadcast of $\mathcal{O}(n^3)$ field elements [9, 13]. Now in the **Input Phase** of [15], each party executes $(m+1)$ VSS's to share the coefficients of his secret polynomial. In addition, each party also executes $n(m+1)$ VSS's to share the coefficients of n random polynomials, each of degree m. These polynomials are used to generate secret random polynomials $r^{(1)}(x), \ldots, r^{(n)}(x)$. So the total number of VSS done in **Input Phase** is $\mathcal{O}(n^2m)$. Hence, the **Input Phase** will take at least three rounds, with a private communication of $\mathcal{O}(n^5m)$ and broadcast of $\mathcal{O}(n^5m)$ field elements. If the broadcast channel is not available, then simulation of broadcast of a single

field element requires a private communication of $\mathcal{O}(n^2)$ field elements and $\Omega(t)$ rounds [16]. Thus, in the absence of broadcast channel, the **Input Phase** will require $\Omega(t)$ rounds and a communication complexity of $\mathcal{O}(n^7 m)$ field elements.

2. Computation Phase: Given that the coefficients of $f^{(P_1)}(x), \ldots, f^{(P_n)}(x)$, $r^{(1)}(x), \ldots, r^{(n)}(x)$ are t-shared, in the computation phase, the parties jointly try to compute $F(x) = r^{(1)}(x)f^{(P_1)}(x) + r^{(2)}(x)f^{(P_2)}(x) + \ldots + r^{(n)}(x)f^{(P_n)}(x)$, such that the coefficients of $F(x)$ are t-shared. For this, the parties execute a sequence of steps. But we recall only first two steps, which are crucial in the communication and round complexity analysis of **Computation Phase**.

During **step 1**, the parties locally multiply the shares of the coefficients of $r^{(i)}(x)$ and $f^{(P_i)}(x)$, for $1 \leq i \leq n$. This results in $2t$-sharing of the coefficients of $f^{(P_i)}(x)r^{(i)}(x)$ for $1 \leq i \leq n$. During **step 2**, each party invokes a *re-sharing protocol* and converts the $2t$-sharing of the coefficients of $f^{(P_i)}(x)r^{(i)}(x)$ into t-sharing, for $1 \leq i \leq n$. The re-sharing protocol enables a party to generate t-sharing of an element, given the t'-sharing of the same element, where $t' > t$. In [15], the authors have given the reference of [12] for the details of re-sharing protocol and claimed that the re-sharing and other additional verifications will take *only three rounds*, with a private communication of $\mathcal{O}(n^4 m^2)$ field elements (see sec. 4.2 of [15]). However, [12] presents a protocol for general secure *Multiparty Computation* (MPC), which uses "circuit based approach" to securely evaluate a function. Specifically, the MPC protocol of [12] assumes that the (general) function to be computed is represented as an arithmetic circuit over $\mathbb{F}$, consisting of addition, multiplication, random, input and output gates. The re-sharing protocol of [12] was used to evaluate a multiplication gate. But the protocol was *non-robust* in the sense that it fails to achieve its goal when at least one of the parties misbehaves, in which case the protocol outputs a pair of parties such that at least one of them is corrupted. In fact, the MPC protocol of [12] takes $\Omega(t)$ rounds in the presence of broadcast channel in the system, whereas in the absence of broadcast channel it will take $\Omega(t^2)$ rounds. The authors in [15] have not mentioned what will be the outcome of their protocol if the re-sharing protocol (whose details they have not given) fails during the **computation phase**. In fact, computing t-sharing of the coefficients of $F(x)$ by using the ideas of best known general MPC protocol with $n = 3t+1$ [2, 8, 12] will require a communication complexity of $\Omega(m^2 n^2)$ field elements and round complexity of $\Omega(t)$ rounds in the presence of a broadcast channel.

To summarize, a more accurate estimation of the round complexity and communication complexity of the MPSI protocol of [15] in the presence and in the absence of a physical broadcast channel is as follows:

1. If a physical broadcast channel is available in the system, then the **Input Phase** will require a private communication of $\Omega(n^5 m)$ field elements and broadcast of $\Omega(n^5 m)$ field elements. Moreover, the **Computation Phase** will take $\Omega(t)$ rounds and communication complexity of $\Omega(m^2 n^2)$.
2. If a physical broadcast channel is not available in the system, then the **Input Phase** will require a private communication of $\Omega(n^7 m)$ field elements.

Moreover, the **Computation Phase** will take $\Omega(t^2)$ rounds and communication complexity of $\Omega(m^2n^2)$ field elements.

3 Our Results

We propose a new, information theoretically secure MPSI protocol with $n = 3t + 1$, tolerating a *computationally unbounded* $\mathcal{A}_t$. Our protocol is based on the approach of solving the MPSI by securely computing the function given in (1). Moreover, our protocol involves a negligible error probability in correctness. However, as mentioned in Remark 1, any protocol for MPSI, based on computing the function in (1) will involve a negligible error probability. In the following tables, we compare the round complexity (RC) and communication complexity (CC) of our MPSI protocol with the *estimated* RC and CC of the MPSI protocol of [15] (as stated in previous section). In the tables, the CC is in terms of field elements. Moreover, CC/RC with (out) BC stands for communication complexity/round complexity in presence (absence) of physical broadcast channel[1].

Reference	CC with BC		RC with BC
	Private	Broadcast	
[15]	$\Omega(n^5m + m^2n^2)$	$\Omega(n^5m)$	$\Omega(t)$
This Paper	$\mathcal{O}((m^2n^3 + n^4\log(\lvert\mathbb{F}\rvert))$	$\mathcal{O}(m^2n^3 + n^4\log(\lvert\mathbb{F}\rvert))$	58

Reference	CC without BC	RC without BC
	Private	
[15]	$\Omega(n^7m)$	$\Omega(t^2)$
This Paper	$\mathcal{O}(m^2n^5 + n^6\log(\lvert\mathbb{F}\rvert))$	$\mathcal{O}(t)$

From the table, we find that our MPSI protocol significantly improves the *estimated* round and communication complexity of the MPSI protocol of [15].

3.1 Our MPSI Protocol vs. Existing General MPC Protocols

The MPSI problem is a particular variant of general secure MPC problem [20]. Informally, in MPC problem, each party P_i has a private input $x_i \in \mathbb{F}$. There is a publicly known function $f : \mathbb{F}^n \to \mathbb{F}^n$. At the end of computation of f, party P_i gets $y_i \in \mathbb{F}$, such that $(y_1, \ldots, y_n) = f(x_1, \ldots, x_n)$. The goal of any general MPC protocol is to securely compute f, where at the end of the protocol, all parties (honest) receive correct outputs, irrespective of the behavior of $\mathcal{A}_t$. Moreover, the messages seen by $\mathcal{A}_t$ during the protocol, should contain no *additional* information about the inputs and outputs of honest parties, other than what can be computed from the inputs and outputs of corrupted parties. The function f is represented as an arithmetic circuit over the finite field $\mathbb{F}$, consisting of five type of gates, namely addition, multiplication, random, input and output. The

[1] If a physical broadcast channel is not available, then we use the protocol of [4, 5], which takes $\mathcal{O}(t)$ rounds and private communication of $\mathcal{O}(n^2\ell)$ bits to simulate broadcast of ℓ bit message.

number of gates of these types are denoted by c_A, c_M, c_R, c_I and c_O respectively. Any general MPC protocol tries to securely evaluate the circuit gate-by-gate, keeping all the inputs and intermediate results of the circuit as t-shared [3, 18].

The MPSI problem can be solved using any general MPC protocol. However, since a general MPC protocol does not exploit the nuances and the special properties of the problem, it is not efficient in general. Moreover, we do not know how to customize the generic MPC protocols to solve MPSI problem in an optimal fashion. However, we outline below a general approach and use the same to estimate the complexity of MPSI protocols, that could have been derived from general MPC protocols.

Suppose, we try to solve MPSI by computing the function given in (1), using general MPC protocol. The arithmetic circuit, representing the function in (1), will roughly require $c_I = n(m+1)$ input gates (every party P_i inputs $(m+1)$ coefficients of $f^{(P_i)}(x)$), $c_R = n(m+1)$ random gates (n polynomials $r^{(1)}(x), \ldots, r^{(n)}(x)$ have $n(m+1)$ random coefficients), $c_M = n(m+1)^2$ multiplication gates (computing $r^{(1)}(x)f^{(P_i)}(x)$ requires $(m+1)^2$ co-efficient multiplications) and $c_O = 2m+1$ output gates (the $2m+1$ coefficients of $F(x)$ should be output). In the following tables we give the round complexity (RC) and communication complexity (CC) of best known general MPC protocols with $n = 3t+1$, to securely compute (1) with above number of gates.

Reference	CC with BC		RC with BC
	Private	Broadcast	
[3]	$\mathcal{O}(n^5m^2)$	$\mathcal{O}(n^5m^2)$	$\mathcal{O}(1)$
[12]	$\mathcal{O}(n^4m^2)$	$\mathcal{O}(n^2)$	$\mathcal{O}(n)$
[8]	$\mathcal{O}(n^2m^2)$	$\mathcal{O}(n^2)$	$\mathcal{O}(n)$
[2]	$\mathcal{O}(n^2m^2)$	$\mathcal{O}(n^3)$	$\mathcal{O}(n)$
This Paper	$\mathcal{O}(m^2n^3 + n^4\log(\lvert\mathbb{F}\rvert))$	$\mathcal{O}(m^2n^3 + n^4\log(\lvert\mathbb{F}\rvert))$	58

Reference	CC without BC	RC without BC
	Private	
[3]	$\mathcal{O}(n^7m^2)$	$\mathcal{O}(n)$
[12]	$\mathcal{O}(n^6m^2)$	$\mathcal{O}(n^2)$
[8]	$\mathcal{O}(n^4m^2)$	$\mathcal{O}(n^2)$
[2]	$\mathcal{O}(n^4m^2 + n^5)$	$\mathcal{O}(n^2)$
This Paper	$\mathcal{O}(m^2n^5 + n^6\log(\lvert\mathbb{F}\rvert))$	$\mathcal{O}(n)$

From the table, we find that our protocol incurs much lesser communication complexity than the protocol of [3], while keeping round complexity same. But the protocols of [2, 8, 12] provides slightly better communication complexity than ours at the cost of increased round complexity. Round complexity and communication complexity are two important parameters of any distributed protocol. If we ever hope to practically implement MPSI protocols, then we should look for a solution, which tries to *simultaneously optimize* both these parameters. In this context, our MPSI protocol fits the bill more appropriately, than the protocols mentioned in the table.

Though our main motive in this paper is to present a clean solution for MPSI, as a bi-product we have shown that our protocol simultaneously improves both

communication and round complexity, whereas existing general MPC protocols (when applied to solve MPSI) improve only one of these two parameters.

3.2 Overview of Our Protocol

As mentioned earlier, our MPSI protocol tries to securely compute the function given in (1). Our protocol is divided into three phases, namely (a) Input and Preparation phase; (b) Computation Phase and (c) Output Phase. In the Input and Preparation phase, the parties t-share the coefficients of their input polynomials. Moreover, the parties jointly generate the t-sharing of the secret random $r^{(i)}(x)$ polynomials. To achieve the first task, we design a new protocol called 1DShare, which further uses a new information checking protocol (ICP) called Multi-Secret-Multi-Verifier-ICP. The second task is achieved by a sub-protocol called Random. In the Computation Phase, the parties generate the t-sharing of the coefficients of $r^{(i)}(x)f^{(i)}(x)$. For this, we use sub-protocol Mult, which is a combination of few existing ideas from the literature and few new ideas presented in this paper. Finally, in the Output Phase, the coefficients of $F(x)$ are reconstructed by each party, by using sub-protocol ReconsPublic. In the next section, we give the technical details of each of the above mentioned sub-protocols.

4 Tools Used

Here we present a number of sub-protocols each solving a specific task. Finally, we combine them to design our MPSI protocol. All the sub-protocols that are presented here are designed to *concurrently* deal with $\ell \geq 1$ values. In the literature, the sub-protocols that achieve the same functionality as ours, were designed to deal with *single* value at a time. Our sub-protocols, concurrently dealing with ℓ values, are better in terms of communication complexity, than ℓ concurrent executions of the existing sub-protocols working with single value. Thus, our sub-protocols harness the advantage offered by dealing with multiple values concurrently (this fact will be more clear in the following sections).

For convenience, we analyze the round and communication complexity of the sub-protocols assuming the existence of physical broadcast channel in the system. While presenting the sub-protocols, we assume that all computations and communications are done over a finite field $\mathbb{F}$, where $\mathbb{F} = GF(2^\kappa)$ and κ is the error parameter. Thus, each field element can be represented by $\log(\mathbb{F}) = \mathcal{O}(\kappa)$ bits. Moreover, without loss of generality, we assume that $n = \text{poly}(\kappa)$.

4.1 Information Checking Protocol and IC Signatures

Information Checking Protocol (ICP) is a tool for authenticating messages in the presence of $\mathcal{A}_t$. The notion of ICP was first introduced by Rabin et.al [18]. As described in [7, 18], an ICP is executed among three parties: a dealer D, an intermediary INT and a verifier R. The dealer D hands over a secret value $s \in \mathbb{F}$ to INT. At a later stage, INT is required to hand over s to R and convince R that s is indeed the value which INT received from D.

The basic definition of ICP involves only a *single* verifier R and deals with *only one* secret s [7, 18]. We extend this notion to *multiple* verifiers, where all the n parties in $\mathcal{P}$ act as verifiers. Thus our ICP is executed among three entities: the dealer $D \in \mathcal{P}$, an intermediary $INT \in \mathcal{P}$ and entire set $\mathcal{P}$ acting as verifiers. This will be later helpful in using ICP as a tool in our MPSI protocol. Moreover, we extend our ICP to deal with *multiple* secrets, denoted by S, which contains $\ell \geq 1$ secret values. Thus, our ICP is executed with respect to *multiple* verifiers and deals with *multiple* secrets concurrently. We call our ICP as Multi-Secret-Multi-Verifier-ICP. Now similar to the ICP defined in [7, 18], our Multi-Secret-Multi-Verifier-ICP is a sequence of following three protocols:

1. Distr$(D, INT, \mathcal{P}, S)$: is initiated by D, who hands over secret $S = (s^{(1)} \ \ldots \ s^{(\ell)})$, containing $\ell \geq 1$ elements from $\mathbb{F}$ to INT. In addition, D hands over some **authentication information** to INT and **verification information** to the individual parties (verifiers) in $\mathcal{P}$.

2. AuthVal$(D, INT, \mathcal{P}, S)$: is initiated by INT to ensure that in RevealVal, secret S held by INT will be accepted by all (honest) parties (verifiers) in $\mathcal{P}$.

3. RevealVal $(D, INT, \mathcal{P}, S)$: is carried out by INT and the verifiers. Here INT produces S, along with **authentication information** and individual verifiers in $\mathcal{P}$ produce **verification information**. Depending upon the values produced by INT and verifiers, either S is accepted or rejected by all the parties.

The **authentication information**, along with S, which is held by INT at the end of AuthVal is called D's *IC signature* on S, denoted as $ICSig(D, INT, S)$. Multi-Secret-Multi-Verifier-ICP satisfies the following properties (which are almost same as the properties, satisfied by the ICP of [7, 18]):

1. If D and INT are uncorrupted, then S will be accepted in RevealVal by each honest verifier.
2. If INT is uncorrupted, then at the end of AuthVal, INT knows an S, which will be accepted in RevealVal by each honest verifier, except with probability $2^{-\Omega(\kappa)}$.
3. If D is uncorrupted, then during RevealVal, with probability at least $1 - 2^{-\Omega(\kappa)}$, every $S' \neq S$ produced by a corrupted INT will be rejected by each honest verifier.
4. If D and INT are uncorrupted, then at the end of AuthVal, $\mathcal{A}_t$ has no information about S.

We now present our novel protocol Multi-Secret-Multi-Verifier-ICP, with $n = 3t + 1$. The high level idea of the protocol is as follows: D selects a random polynomial $F(x)$ of degree $\ell + n\kappa$ over $\mathbb{F}$, whose lower order ℓ coefficients are elements of S. In addition, D also selects a random polynomial $R(x)$ of degree $\ell + n\kappa$ over $\mathbb{F}$, which is independent of $F(x)$. D hands over $F(x)$ and $R(x)$ to INT. D then associates κ *random evaluation points* with each verifier P_i and gives the value of $F(x), R(x)$ at those evaluation points to P_i. This prevents with very high probability, a corrupted INT, to produce an incorrect $F(x)$ during RevealVal, without being un-noticed by an honest verifier P_i. This ensures third property

of ICP. In order to ensure second property, an honest INT has to ensure that his $F(x)$ is consistent with the evaluation points of the honest verifiers. For this, INT and the verifiers interact in a zero knowledge fashion and check the consistency of $F(x)$ and secret evaluation points. To maintain the secrecy of S during the zero knowledge interaction, INT uses the $R(x)$ polynomial.

Multi-Secret-Multi-Verifier-ICP($D, INT, \mathcal{P}, \ell, S = (s^{(1)}, \ldots, s^{(\ell)})$)

Distr($D, INT, \mathcal{P}, \ell, S$) **Round 1**: D selects a random polynomial $F(x)$ of degree $\ell + n\kappa$ over $\mathbb{F}$, whose lower order ℓ coefficients are elements of S. In addition, D selects another random polynomial $R(x)$ of degree $\ell + n\kappa$ over $\mathbb{F}$. D also selects $n\kappa$ random, non-zero, distinct evaluation points from $\mathbb{F}$, denoted by $\alpha_{i,1}, \alpha_{i,2}, \ldots, \alpha_{i,\kappa}$, for $1 \leq i \leq n$. D privately gives $F(x)$ and $R(x)$ to INT. To verifier $P_i \in \mathcal{P}$, D privately gives $(\alpha_{i,l}, a_{i,l}, b_{i,l})$, for $l = 1, \ldots, \kappa$, where $a_{i,l} = F(\alpha_{i,l})$ and $b_{i,l} = R(\alpha_{i,l})$.

AuthVal($D, INT, \mathcal{P}, \ell, S$) **Round 2**: INT chooses a random $d \in \mathbb{F} \setminus \{0\}$ and broadcasts $(d, B(x) = F(x) + dR(x))$. Parallely, each verifier $P_i \in \mathcal{P}$ broadcasts a random subset of indices $l_1, ..., l_{\frac{\kappa}{2}}$, the evaluation points $\alpha_{i,l_1}, ..., \alpha_{i,l_{\frac{\kappa}{2}}}$ and the values $a_{i,l_1}, ..., a_{i,l_{\frac{\kappa}{2}}}$ and $b_{i,l_1}, ..., b_{i,l_{\frac{\kappa}{2}}}$. *Notice that each verifier randomly selects the subset of indices $l_1, ..., l_{\frac{\kappa}{2}}$, independent of other verifiers.*

Round 3: D checks if for at least $2t+1$ verifiers P_i, it holds that $a_{i,l} + db_{i,l} = B(\alpha_{i,l})$, for all l in the set of random indices broadcasted by P_i in **Round 2**. If the above condition is not satisfied for at least $2t+1$ verifiers, then D broadcasts the polynomial $F(x)$.

Local Computation (by each party): IF $F(x)$ is broadcasted in **Round 3**, then INT replaces the $F(x)$ received from D during **Round 1**, with the $F(x)$ which is broadcasted in **Round 3**. Accordingly, each verifier P_i adjust his $a_{i,l}$ (as received in **Round 1**), for $l = 1, \ldots, \kappa$, such that $F(\alpha_{i,l}) = a_{i,l}$ holds. ELSE say that verifier P_i *accepts* INT if $a_{i,l} + db_{i,l} = B(\alpha_{i,l})$, for all l in the set of random indices, broadcasted by P_i in **Round 2**.

The polynomial $F(x)$ is called D's **IC signature** on $S = (s^{(1)}, \ldots, s^{(\ell)})$ given to INT, which is denoted by $ICSig(D, INT, S)$.

RevealVal($D, INT, \mathcal{P}, \ell, S$): (a) **Round 4**: INT broadcasts $F(x)$; (b) **Round 5**: Each verifier $P_i \in \mathcal{P}$ broadcasts all the evaluation points $\alpha_{i,l}$ which were not broadcasted during **Round 2** and $a_{i,l}$ corresponding those indices.

Local Computation (by each party): Say that verifier P_i *re-accepts* INT if for *at least one* of the newly revealed (by P_i) points, it holds that $a_{i,l} = F(\alpha_{i,l})$. If there are at least $t+1$ verifiers who *re-accepts* INT, then accept the lower order ℓ coefficients of $F(x)$ as $S = (s^{(1)}, \ldots, s^{(\ell)})$. In this case, we say that D's signature on S is correct. Else reject $F(x)$ broadcasted by INT and we say that INT has failed to produce D's signature.

Lemma 1 (Property 1). *If D and INT are honest, then each honest verifier will accept S at the end of RevealVal, without any error.*

Lemma 2 (Property 2). *If D is uncorrupted, then at the end of AuthVal, INT knows an S, which will be accepted in RevealVal by each honest verifier, except with probability $2^{-\Omega(\kappa)}$.*

PROOF: If D is honest, then the proof follows from Lemma 1. So we consider the case when D is corrupted. Now there are two possible sub-cases. If D broadcasts $F(x)$ during **Round 3**, then the lemma holds trivially, without any error. So we consider the case, when D (corrupted) has not broadcasted $F(x)$ during **Round 3**. This implies that at least $2t+1$ verifiers have *accepted* INT during AuthVal. Now, out of these $2t+1$ verifiers, at least $t+1$ are honest. If we can show that these honest verifiers will *re-accept* INT during RevealVal with high probability, then the proof is over. So we now proceed to prove the same.

In order that an honest P_i *accept* INT during AuthVal but does not *re-accept* it during RevealVal, it must be the case that the data (evaluation points and values) that P_i exposed during AuthVal satisfies the polynomial $B(x)$ that INT broadcasted during AuthVal, but on the other hand, out of the remaining evaluation points that are used by P_i in RevealVal, none satisfy the polynomial $F(x)$ produced by INT. That is, for the selected $\frac{\kappa}{2}$ indices $l_1, ..., l_{\frac{\kappa}{2}}$, it holds that $a_{i,l} + db_{i,l} = B(\alpha_{i,l})$, for all l in the set of indices $\{l_1, ..., l_{\frac{\kappa}{2}}\}$ and $F(\alpha_{i,l}) \neq a_{i,l}$ for all l in the *remaining* set of indices. Notice that INT chooses d independently of the values given by D. Also, P_i chooses the $\frac{\kappa}{2}$ indices randomly out of κ indices. So the probability that the above event happens is $\frac{1}{\binom{\kappa}{\kappa/2}} \approx 2^{-\Omega(\kappa)}$, which is negligible. This shows that with high probability all honest verifiers (at least $t+1$), who have *accepted* INT during AuthVal, will *re-accept* INT during RevealVal, thus proving our lemma. □

Lemma 3 (Property 3). *If D is uncorrupted, then during RevealVal, with probability at least $1 - 2^{-\Omega(\kappa)}$, every $S' \neq S$ produced by a corrupted INT will be rejected by each honest verifier.*

PROOF: If a corrupted INT produces $S' \neq S$ during RevealVal, then it implies that INT has broadcasted $F'(x) \neq F(x)$ during **Round 4**. Moreover, while broadcasting $F'(x)$, INT will have no information about the $\frac{\kappa}{2}$ random secret evaluation points (which were not broadcasted during AuthVal), corresponding to each honest verifier. Without knowing the $\frac{\kappa}{2}$ secret evaluation points of an honest verifier, say P_i, the probability that INT will be re-accepted by P_i is at most $\frac{\ell+nk}{|\mathbb{F}|}$. Thus, the total probability that any honest verifier will accept INT (who broadcasts $F'(x) \neq F(x)$) is $\frac{(\ell+nk)(2t+1)}{|\mathbb{F}|} \approx 2^{-\Omega(k)}$. □

Lemma 4 (Property 4). *If D and INT are honest, then $\mathcal{A}_t$ will have no information about S at the end of AuthVal.*

PROOF: For simplicity, let the first t verifiers are corrupted. So in the **Round 1**, the adversary will know κt points on $F(x)$ and $R(x)$. In **Round 2**, the adversary will come to know about additional $\frac{k}{2}(2t+1)$ points on $F(x)$ and $R(x)$. Moreover, since D and INT are both honest, $2t+1$ honest verifiers will accept INT and hence D will not broadcast $F(x)$ during **Round 3**. So at the end of AuthVal, adversary will know $\kappa t + \frac{\kappa}{2}(2t+1)$ points on each of $F(x)$ and $R(x)$. However, since $n = 3t+1$ and degree of $F(x)$ and $R(x)$ is $\ell + n\kappa$, the adversary will have no information about the lower order ℓ coefficients of $F(x)$. □

Lemma 5. *Protocol Multi-Secret-Multi-Verifier-ICP takes five rounds and correctly generates IC signature on ℓ field elements, by privately communicating $\mathcal{O}((\ell + n\kappa)\kappa)$ bits and broadcasting $\mathcal{O}((\ell + n\kappa)\kappa)$ bits. The protocol works correctly, except with error probability of $2^{-\Omega(\kappa)}$.*

Important Notation: *In the rest of the paper, whenever we say that D hands over $ICSig(D, INT, S)$ to INT, we mean that Distr and AuthVal are executed in the background. Similarly, INT reveals $ICSig(D, INT, S)$ can be interpreted as INT, along with other parties, invoking RevealVal.*

Remark 2 (Comparison with Existing ICP). The current best known ICP is due to [7], which privately communicates and broadcasts $\mathcal{O}(n\kappa)$ bits, to generate IC signature on a single secret (though the ICP of [7] is designed with $n = 2t+1$, the protocol when executed with $n = 3t + 1$, will result in the same communication complexity). Had we executed ℓ times the ICP of [7], dealing with single secret, the communication complexity would turn out to be $\mathcal{O}(\ell n\kappa)$ bits (both private and broadcast). However, the communication complexity of Multi-Secret-Multi-Verifier-ICP considering all the ℓ secrets concurrently is $\mathcal{O}((\ell + n\kappa)\kappa)$ bits (both private and broadcast). This clearly shows that if ℓ is significantly large, which is the case in our MPSI protocol, then executing a *single instance* of Multi-Secret-Multi-Verifier-ICP, dealing with *multiple secrets* concurrently, is advantageous over executing *multiple instances* of ICP of [7], dealing with *single secret*. The same principle holds for other sub-protocols, which are described in the sequel.

4.2 Generating ℓ Length Random Vector

We now present a protocol called RandomVector$(\mathcal{P}, \ell)$, which allows the parties in $\mathcal{P}$ to jointly generate a vector, containing ℓ random elements from $\mathbb{F}$. Following the idea of [8], protocol RandomVector uses Vandermonde Matrix and its capability to extract randomness.

$(r^{(1)}, \ldots, r^{(\ell)})$ = **RandomVector**$(\mathcal{P}, \ell)$

1. Every party $P_i \in \mathcal{P}$ selects $L = \lceil \frac{\ell}{2t+1} \rceil$ random elements $r^{(1,P_i)}, \ldots, r^{(L,P_i)}$ from $\mathbb{F}$.
2. Every party $P_i \in \mathcal{P}$ as a dealer invokes Sharing Phase of four round VSS protocol of [11] with $n \geq 3t + 1$ for sharing each of the values $r^{(1,P_i)}, \ldots, r^{(L,P_i)}$.
3. For reconstructing the values $r^{(1,P_i)}, \ldots, r^{(L,P_i)}$ (shared by P_i in Sharing Phase), the Reconstruction Phase of four round VSS of [11] with $n \geq 3t+1$ is invoked for L times separately. Now corresponding to every $P_i \in \mathcal{P}$, the values $r^{(1,P_i)}, \ldots, r^{(L,P_i)}$ are public.
4. Now parties compute $(r^{(1,1)}, \ldots, r^{(1,2t+1)}) = (r^{(1,P_1)}, \ldots, r^{(1,P_n)})V$, $(r^{(2,1)}, \ldots, r^{(2,2t+1)}) = (r^{(2,P_1)}, \ldots, r^{(2,P_n)})V, \ldots$, $(r^{(L,1)}, \ldots, r^{(L,2t+1)}) = (r^{(L,P_1)}, \ldots, r^{(L,P_n)})V$. Here V is a $n \times (2t + 1)$ publicly known Vandermonde matrix over $\mathbb{F}$.

The values $r^{(1,1)}, \ldots, r^{(1,2t+1)}, \ldots, r^{(L,1)}, \ldots, r^{(L,2t+1)}$ constitute the elements of ℓ length random vector.

Protocol RandomVector also uses the four round perfect VSS (verifiable secret sharing) protocol of [11] (see Fig 2 of [11]) as black box. The perfect VSS (see the definition of VSS in Section 2.1 of [11]) with $n \geq 3t+1$ parties consists of two phases, namely Sharing Phase and Reconstruction Phase. The Sharing Phase takes four rounds and allows a dealer D (which can be any party from the set of n parties) to verifiably share a secret $s \in \mathbb{F}$ by privately communicating $\mathcal{O}(n^2 \log |\mathbb{F}|)$ bits and broadcasting $\mathcal{O}(n^2 \log |\mathbb{F}|)$ bits where $|\mathbb{F}| \geq n$. The Reconstruction Phase takes single round and allows all the (honest) parties to reconstruct the secret s (shared by D in Sharing Phase) by broadcasting $\mathcal{O}(n \log |\mathbb{F}|)$ bits in total. Notice that, in our context, $|\mathbb{F}| = 2^{\kappa} \geq n$. The VSS protocol has an important property that once D (possibly corrupted) shares a secret s during Sharing Phase, then D is *committed* to s. Later, in the Reconstruction Phase, irrespective of the behavior of the corrupted parties, the same s will be reconstructed. Thus a corrupted D will not be able to change his commitment from s to any other value, with the help of corrupted parties, during Reconstruction Phase.

Lemma 6. *Protocol RandomVector generates ℓ length random vector in five rounds and privately communicates $\mathcal{O}(\ell n^2 \kappa)$ bits and broadcasts $\mathcal{O}(\ell n^2 \kappa)$ bits.*

4.3 Unconditional Verifiable Secret Sharing and Reconstruction

Definition 1 (d-1D-sharing [1]). *: We say that a secret s is d-1D-shared, if there exists a degree-d polynomial $f(x)$, with $f(0) = s$, such that each (honest) P_i in $\mathcal{P}$ holds the i^{th} share $f(i) = s_i$ of s. The vector $(s_1, s_2, \ldots, s_n)$ of shares is called a d-sharing of s and is denoted as $[s]_d$. We may skip the subscript d when it is clear from the context.*

If s is d-1D-shared by $D \in \mathcal{P}$, then we denote it as $[s]_d^D$. In the sequel, we describe a new protocol 1DShare, which allows a dealer $D \in \mathcal{P}$ to t-1D-share ℓ secret values $s^{(1)}, \ldots, s^{(\ell)}$, where $\ell \geq 1$, with very high probability. If D behaves correctly during the protocol, then each honest $P_i \in \mathcal{P}$ will hold i^{th} shares $s_i^{(1)}, \ldots, s_i^{(\ell)}$, of $s^{(1)}, \ldots, s^{(\ell)}$ (respectively), at the end of the protocol.

Notice that the desired sharing for each $s^{(i)}$ (separately) can be produced using a perfect (i.e., without any error) VSS protocol with $n \geq 3t+1$ [9, 11, 13]. However, this will involve more communication complexity than 1DShare which performs the same task with less communication complexity (but with a negligible error probability). 1DShare achieves this by incorporating one of the ideas used in [8] and using Multi-Secret-Multi-Verifier-ICP as building block.

The goal of 1DShare is as follows: (a) If D is honest, then the protocol generates $[s^{(1)}]_t, \ldots, [s^{(\ell)}]_t$ with very high probability, such that the secrets $s^{(1)}, \ldots, s^{(\ell)}$ remain information theoretically secure from $\mathcal{A}_t$. (b) If D is corrupted and has not generated t-1D-sharing of secrets, then with high probability, D will be detected as corrupted during a public verification process. Moreover, every honest party accepts a pre-defined t-1D-sharing of ℓ 1's, namely $[1]_t, [1]_t, \ldots, [1]_t$ (ℓ times), on behalf of D.

Informally, the protocol works as follows: D chooses $\ell+1$ random polynomials $f^{(0)}(x), \ldots, f^{(\ell)}(x)$ over $\mathbb{F}$, each of degree t, such that $f^{(0)}(0) = r$ and $f^{(l)}(0) =$

$s^{(l)}$ for $l = 1, \ldots, \ell$. Here r is a random non-zero element from $\mathbb{F}$. D then hands over his IC signature on i^{th} points of $f^{(l)}(x)$ polynomials *concurrently* to party P_i. After this, the parties jointly produce a non-zero random value z. Now D is asked to broadcast a linear combination of the $\ell + 1$ polynomials, where the scalars of the linear combination are function of z. At the same time, each party P_i is asked to broadcast his corresponding linear combination of points. Ideally, the linear combination of points, broadcasted by the individual parties, should lie on the linear combination of the polynomial broadcasted by D. If this happens, then with very high probability, D has correctly t-1D-shared each $s^{(l)}$. Otherwise, there is a party, say P_i, for which the above condition is not satisfied. In this case, P_i is asked to reveal D's signature on the i^{th} points of $f^{(l)}(x)$ polynomials that he has received from D. In case P_i is able to correctly produce the signature, D is detected to be corrupted and the protocol terminates, with each party assuming predefined t-1D-sharing of ℓ 1's, namely $[1]_t, [1]_t, \ldots, [1]_t$, on behalf of D.

$([s^{(1)}]_t^D, \ldots, [s^{(\ell)}]_t^D) = \mathbf{1DShare}(D, \mathcal{P}, \ell, s^{(1)}, s^{(2)}, \ldots, s^{(\ell)})$

1. For $l = 1, \ldots, \ell$, D picks a random polynomial $f^{(l)}(x)$ over $\mathbb{F}$ of degree-t, with $f^{(l)}(0) = s^{(l)}$. D also chooses a random polynomial $f^{(0)}(x)$ of degree-t with $f^{(0)}(0) = r$ where r is a random, non-zero element from $\mathbb{F}$. For $i = 1, \ldots, n$, let $S_i = (r_i, s_i^{(1)}, s_i^{(2)}, \ldots, s_i^{(\ell)})$, where $r_i = f^{(0)}(i)$ and $s_i^{(l)} = f^{(l)}(i)$. D hands over $ICSig(D, P_i, S_i)$ to party P_i.
2. All the parties in $\mathcal{P}$ invoke $\mathsf{RandomVector}(\mathcal{P}, 1)$ to generate a non-zero random value $z \in \mathbb{F}$.
3. D broadcasts the polynomial $f(x) = f^{(0)}(x) + \sum_{l=1}^{\ell} f^{(l)}(x) z^l = \sum_{l=0}^{\ell} f^{(l)}(x) z^l$. Parallely, every party P_i computes and broadcasts $y_i = r_i + \sum_{l=1}^{\ell} s_i^{(l)} z^l$.
4. If the polynomial $f(x)$ broadcasted by D is of degree more than t, then each party agrees that D is corrupted and outputs t-1D-sharing of ℓ 1's i.e $[1]_t, [1]_t, \ldots, [1]_t$. The protocol terminates here.
5. Every party checks whether $f(i) \stackrel{?}{=} y_i$ for all $i = 1, \ldots, n$. If yes then everybody accepts the t-1D-sharings $[s^{(1)}]_t, [s^{(2)}]_t, \ldots, [s^{(\ell)}]_t$ and the protocol terminates. Otherwise, let $P_i \in \mathcal{P}$, such that $f(i) \neq y_i$. In this case, P_i reveals $ICSig(D, P_i, S_i)$. If P_i succeeds to prove D's signature on $S_i = (r_i, s_i^{(1)}, \ldots, s_i^{(\ell)})$ and $f(i) \neq r_i + \sum_{l=1}^{\ell} s_i^{(l)} z^l$, then each party agrees that D is corrupted and outputs t-1D-sharing of ℓ 1's i.e $[1]_t, [1]_t, \ldots, [1]_t$ (ℓ times) and the protocol terminates here. We say that P_i has raised a valid **complaint** against D. But if the signature is invalid then ignore P_i's complaint against D and everybody accepts $[s^{(1)}]_t, \ldots, [s^{(\ell)}]_t$.

Lemma 7. *In protocol 1DShare, if D is honest, then t-1D-sharing of $s^{(1)}, \ldots, s^{(\ell)}$ are generated, except with error probability of $2^{-\Omega(\kappa)}$. Moreover, $\mathcal{A}_t$ will have no information about the secrets. On the other hand, if D is corrupted and any of the values $r, s^{(1)}, \ldots, s^{(\ell)}$ is not t-1D-shared, then D will be caught, except with error probability of $2^{-\Omega(\kappa)}$. The protocol takes eleven rounds, privately communicates $\mathcal{O}((\ell n + n^2\kappa)\kappa)$ bits and broadcasts $\mathcal{O}((\ell n + n^2\kappa)\kappa)$ bits.*

Proof: The communication and round complexity can be checked easily by inspection. We now prove the correctness. If D is honest, then $f(i) = y_i$ will

hold, corresponding to every honest P_i. However, a corrupted party P_i may broadcast incorrect $y_i' \neq y_i$, such that $y_i' \neq f(i)$. Moreover, P_i can forge honest D's IC signature on corresponding incorrect $r_i' \neq r_i$ or/and $s_i^{'(j)} \neq s_i^{(j)}$, for $j = 1 \ldots \ell$. In this case, everybody will reject the sharing done by D. However, from properties of Multi-Secret-Multi-Verifier-ICP protocol, this can happen with probability $2^{-\Omega(\kappa)}$. The secrecy of $s^{(1)}, s^{(2)}, \ldots, s^{(\ell)}$ for an honest D, follows from the fact that $\mathcal{A}_t$ will have only t shares for each $s^{(i)}, 1 \leq i \leq n$ and random r. In addition, the value $f(0)$ is blinded with a random value r, chosen by D. Thus, $\mathcal{A}_t$ will have no information about the secrets.

Next, we consider the case, when D is corrupted and the sharing of at least one of the values $r, s^{(1)}, s^{(2)}, \ldots, s^{(\ell)}$ is not a correct t-1D-sharing, i.e., the shares of the honest parties lie on a polynomial of degree higher than t. Let H be the set of honest parties in $\mathcal{P}$. Moreover, let $h^0(x), \ldots, h^\ell(x)$ denote the minimum degree polynomial, defined by the points on $f^{(0)}(x), \ldots, f^{(\ell)}(x)$ respectively, held by the parties in H. Then according to the condition, degree of at least one of the polynomials $h^0(x), \ldots, h^\ell(x)$ is more than t. Moreover, degree of $h^0(x), \ldots, h^\ell(x)$ can be at most $|H|-1$. This is because $|H|$ distinct points can define a polynomial of degree at most $|H|-1$. Now the value y_i broadcasted by an honest P_i can be defined as $y_i = \sum_{j=0}^{\ell} z^j h^j(i)$.

We next claim that if degree of at least one of $h^0(x), \ldots, h^\ell(x)$ is more than t, then the minimum degree polynomial, say $h^{min}(x)$, defined by y_i's, corresponding to $P_i \in H$ will be of degree more than t, with very high probability. This will clearly imply that $f(x) \neq h^{min}(x)$ and hence $y_i \neq f(i)$, for at least one $P_i \in H$.

So we proceed to prove that $h^{min}(x)$ will be of degree more than t, when one of $h^0(x), \ldots, h^\ell(x)$ has degree more than t. For this, we show the following:

1. We first show that $h^{def}(x) = \Sigma_{j=0}^{\ell} z^j h^j(x)$ will of degree more than t with very high probability, if one of $h^0(x), \ldots, h^\ell(x)$ has degree more than t.
2. We then show that $h^{min}(x) = h^{def}(x)$, implying that $h^{min}(x)$ will be of degree more than t with very high probability

The first claim is easy to prove. If one of $h^0(x), \ldots, h^\ell(x)$ has degree more than t, then the linear combination of these polynomials, namely $h^{def}(x)$, can be written as $h^{def}(x) = h_1^{def}(x) + h_2^{def}(x)$. Here $h_1^{def}(x)$ contains all the coefficients of $h^{def}(x)$, having exponent more than t, while $h_2^{def}(x)$ contains all the remaining coefficients of $h^{def}(x)$. Now $h^{def}(x)$ will be of degree-t, if $h_1^{def}(x) = 0$, which can happen for at most ℓ possible values of z. Since z is generated randomly from $\mathbb{F} \setminus \{0\}$, independent of $h^0(x), \ldots, h^\ell(x)$, the probability that $h_1^{def}(x) = 0$ is at most $\frac{\ell}{|\mathbb{F}|-1} \approx 2^{-\Omega(\kappa)}$. This implies that $h^{def}(x)$ will be of degree $t_m > t$. Notice that each y_i broadcasted by an honest P_i, will lie on $h^{def}(x)$.

Now we will show that $h^{min}(x) = h^{def}(x)$ and thus $h^{min}(x)$ has degree at least t_m, which is greater than t. So consider the difference polynomial $dp(x) = h^{def}(x) - h^{min}(x)$. Clearly, $dp(x) = 0$, for all $x = i$, where $P_i \in H$. Thus $dp(x)$ will have at least $|H|$ roots. On the other hand, maximum degree of $dp(x)$ could be t_m, which is at most $|H|-1$. These two facts imply that $dp(x)$ is zero polynomial, implying that $h^{def}(x) = h^{min}(x)$ and thus $h^{min}(x)$ has degree $t_m > t$.

Since $h^{min}(x)$ has degree more than t, it implies that $h^{min}(x) \neq f(x)$ (which is of degree-t and broadcasted by D). This further imply that $f(i) \neq y_i$, for at least one $P_i \in H$. So P_i will raise a valid **complaint** against D by revealing $ICSig(D, P_i, S_i)$, where $S_i = (r_i, s_i^{(1)}, \ldots, s_i^{(\ell)})$. Since P_i is honest, the signature will be revealed successfully, except with an error probability of $2^{-\Omega(\kappa)}$ (this follows from the properties of Multi-Secret-Multi-Verifier-ICP). Moreover, everybody will publicly verify that $f(i) \neq r_i + \sum_{l=1}^{\ell} s_i^{(l)} z^l$ and hence will catch the corrupted D with very high probability. □

Reconstruction of t-1D-Sharing: We now present a protocol called ReconsPublic, that reconstructs a secret s, given $[s]_t$. In the protocol, every party broadcasts his share of s. Now out of these n shares, at most t could be corrupted. But since $n = 3t+1$, by applying Reed-Solomon error correction algorithm (e.g. Berlekamp Welch Algorithm [17]), s can be recovered.

$s = \mathsf{ReconsPublic}(\mathcal{P}, [s]_t)$

Each party P_i broadcasts his share s_i of s. The parties apply error correction to reconstruct s from the n shares.

Lemma 8. *ReconsPublic takes one round and broadcasts $\mathcal{O}(n\kappa)$ bits.*

Important Notation: We now define few notations which are used in subsequent sections (these notations are also commonly used in the literature). By saying that parties in $\mathcal{P}$ compute (locally) $([y^{(1)}]_d, \ldots, [y^{(\ell')}]_d) = \varphi([x^{(1)}]_d, \ldots, [x^{(\ell)}]_d)$ (for any function $\varphi : \mathbb{F}^{\ell} \rightarrow \mathbb{F}^{\ell'}$), we mean that each P_i computes $(y_i^{(1)}, \ldots, y_i^{(\ell')}) = \varphi(x_i^{(1)}, \ldots, x_i^{(\ell)})$. Note that applying an affine (linear) function φ to a number of d-1D-sharings, we get d-1D-sharings of the outputs. So by adding two d-1D-sharings of secrets, we get d-1D-sharing of the sum of the secrets, i.e. $[a]_d + [b]_d = [a+b]_d$. However, by multiplying two d-1D-sharings of secrets, we get $2d$-1D-sharing of the product of the secrets, i.e. $[a]_d[b]_d = [ab]_{2d}$. ♢

4.4 Upgrading t-1D-Sharing to t-2D-Sharing

Definition 2. *A value s is d-2D-shared among the parties in $\mathcal{P}$, if there exists degree-d polynomials $f, f^1, f^2 \ldots, f^n$ with $f(0) = s$ and for $i = 1, \ldots, n$, $f^i(0) = f(i)$. Moreover, every (honest) party $P_i \in \mathcal{P}$ holds a share $s_i = f(i)$ of s, the polynomial $f^i(x)$ for sharing s_i and share-share $s_{ji} = f^j(i)$ for the share s_j of every other (honest) party P_j. We denote d-2D-sharing of s as $[[s]]_d$.*

If a secret s is d-2D-shared by a party $D \in \mathcal{P}$, then we denote it as $[[s]]_d^D$. Notice that if s is d-2D-shared, then its i^{th} share s_i is d-1D-shared. We now present a new protocol, called Upgrade1Dto2D for upgrading t-1D-sharing to t-2D-sharing. Specifically, given t-1D-sharing of ℓ secrets, namely $[s^{(1)}]_t, \ldots, [s^{(\ell)}]_t$, Upgrade1Dto2D, outputs t-2D-sharing $[[s^{(1)}]]_t, [[s^{(2)}]]_t, \ldots, [[s^{(\ell)}]]_t$, except with probability of $2^{-\Omega(\kappa)}$. Moreover, $\mathcal{A}_t$ learns nothing about the secrets during Upgrade1Dto2D. Furthermore, if a party tries to cheat during the protocol, then with very high probability, he will be caught.

Lemma 9. *Protocol Upgrade1Dto2D upgrades t-1D-sharing of ℓ secrets to t-2D-sharing, except with negligible error probability. The protocol consumes twenty eight rounds, privately communicates $\mathcal{O}((\ell n^2+n^3\kappa)\kappa)$ bits and broadcasts $\mathcal{O}((\ell n^2 + n^3\kappa)\kappa)$ bits. Moreover, $\mathcal{A}_t$ learns nothing about the secrets.*

Proof: The communication and round complexity of the protocol is easy to follow. We now prove the correctness. Provided ℓ t-1D-sharing $[s^{(1)}]_t, [s^{(2)}]_t, \ldots, [s^{(\ell)}]_t$, every honest party P_i correctly t-1D-shares his shares $s_i^{(0)}, s_i^{(1)}, \ldots, s_i^{(\ell)}$. Now for every honest party P_h, the value $s_h = s_h^{(0)} + \sum_{l=1}^{\ell} r^{(l)} s_h^{(l)}$ will be reconstructed correctly, where s_h is the h^{th} share of $s = s^{(0)} + \sum_{l=1}^{\ell} r^{(l)} s^{(l)}$. But a corrupted party P_c may share $\overline{s_c^{(0)}}, \overline{s_c^{(1)}}, \ldots, \overline{s_c^{(\ell)}}$ with $\overline{s_c^{(l)}} \neq s_c^{(l)}$ for some $l \in \{0, 1, \ldots, \ell\}$. In this case with very high probability $\overline{s_c} = \overline{s_c^{(0)}} + \sum_{l=1}^{\ell} r^{(l)} \overline{s_c^{(l)}}$ will not be equal to s_c (which is the actual c^{th} share of s) as the ℓ length vector $(r^{(1)}, \ldots, r^{(\ell)})$ is chosen uniformly at random. Hence Reed-Solomon Error correction algorithm will point $\bar{s}_c$ as a corrupted share, in which case P_c will be caught. It is easy to see that at any stage of the protocol, $\mathcal{A}_t$ learns not more than t shares for each $s^{(l)}, 1 \leq l \leq \ell$. Hence all the secrets will be secure. □

$([[s^{(1)}]]_t, [[s^{(2)}]]_t, \ldots, [[s^{(\ell)}]]_t) = \mathsf{Upgrade1Dto2D}(\mathcal{P}, \ell, [s^{(1)}]_t, [s^{(2)}]_t, \ldots, [s^{(\ell)}]_t)$

1. Each $P_i \in \mathcal{P}$ invokes $\mathsf{1DShare}(P_i, \mathcal{P}, 1, s^{(0,P_i)})$ to generate $[s^{(0,P_i)}]_t$, where $s^{(0,P_i)} \in \mathbb{F} \setminus \{0\}$ is a random value.
2. The parties in $\mathcal{P}$ computes $[s^{(0)}]_t = \sum_{j=1}^{n} [s^{(0,P_j)}]_t$.
3. Now every P_i invokes $\mathsf{1DShare}(P_i, \mathcal{P}, \ell+1, s_i^{(0)}, s_i^{(1)}, \ldots, s_i^{(\ell)})$ to generate $[s_i^{(0)}]_t, [s_i^{(1)}]_t, \ldots, [s_i^{(\ell)}]_t$, where $s_i^{(0)}, s_i^{(1)}, \ldots, s_i^{(\ell)}$ are the i^{th} shares of secrets $s^{(0)}, s^{(1)}, \ldots, s^{(\ell)}$ respectively.
4. Now to detect the parties P_k (at most t), who have generated $[\overline{s_k^{(0)}}]_t, [\overline{s_k^{(1)}}]_t, \ldots, [\overline{s_k^{(\ell)}}]_t$ such that $\overline{s_k^{(l)}} \neq s_k^{(l)}$ for some $l \in \{0, 1, \ldots, \ell\}$, the parties in $\mathcal{P}$ jointly generate an ℓ length random vector $(r^{(1)}, \ldots, r^{(\ell)})$ by invoking Protocol $\mathsf{RandomVector}(\mathcal{P}, \ell)$. Now all the parties publicly reconstruct $s_i = s_i^{(0)} + \sum_{l=1}^{\ell} r^{(l)} s_i^{(l)}$ and $s = s^{(0)} + \sum_{l=1}^{\ell} r^{(l)} s^{(l)}$ by executing following steps:
   (a) The parties in $\mathcal{P}$ compute $[s_i]_t = [s_i^{(0)}]_t + \sum_{l=1}^{\ell} r^{(l)} [s_i^{(l)}]_t$ and invoke $\mathsf{ReconsPublic}(\mathcal{P}, [s_i]_t)$ to publicly reconstruct s_i, for $i = 1, \ldots, n$.
   (b) Every party apply Reed-Solomon error correction algorithm (e.g. Berlekamp Welch Algorithm [17]) to $s_1, s_2, \ldots, s_n$, to recover s. Reed-Solomon error correction algorithm also points out the corrupted shares. Hence if s_i is pointed as a corrupted share, then $[s_i^{(0)}]_t, [s_i^{(1)}]_t, \ldots, [s_i^{(\ell)}]_t$ are ignored by every party.
5. Output $[[s^{(1)}]]_t, [[s^{(2)}]]_t, \ldots, [[s^{(\ell)}]]_t$.

Remark 3 (Comparison with Existing Protocols). In [1], the authors have given a protocol to upgrade d-1D-Sharing to d-2D-Sharing, where $n = 2t+1$. However, the protocol is non-robust. That is, if all the n parties behave honestly, then the protocol will perform the upgradation. Otherwise, the protocol will fail to do the upgradation, but will output a pair of parties, of which at least one is corrupted.

4.5 Proving $c = ab$

Given t-1D-sharing of ℓ pairs, $([a^{(1)}]_t^D, [b^{(1)}]_t^D), \ldots, ([a^{(\ell)}]_t^D, [b^{(\ell)}]_t^D)$, let $c^{(l)} = a^{(l)}b^{(l)}$ for $l = 1, \ldots, \ell$. $D \in \mathcal{P}$ now wants to generate $[c^{(1)}]_t^D, \ldots, [c^{(\ell)}]_t^D$ such that the (honest) parties in $\mathcal{P}$ know that the shared $c^{(l)}$ values satisfy $c^{(l)} = a^{(l)}b^{(l)}$ for $l = 1, \ldots, \ell$. If D is honest, then during this process all $a^{(l)}$, $b^{(l)}$ and $c^{(l)}$ values should remain secure.

We propose a protocol ProveCeqAB to achieve the above task. The idea of the protocol is inspired from [7], where a protocol for the same purpose is proposed, with *a single* pair of values, namely (a, b). Our protocol concurrently deals with ℓ pairs, which leads to a gain in communication complexity. Our protocol uses 1DShare as a building block.

$([c^{(1)}]_t^D, \ldots, [c^{(\ell)}]_t^D) = \mathsf{ProveCeqAB}(\mathsf{D}, \mathcal{P}, \ell, [a^{(1)}]_t^D, [b^{(1)}]_t^D, \ldots, [a^{(\ell)}]_t^D, [b^{(\ell)}]_t^D)$

1. D chooses a random non-zero ℓ length tuple $(\beta^{(1)}, \ldots, \beta^{(\ell)}) \in \mathbb{F}^\ell$. D then invokes $\mathsf{1DShare}(D, \mathcal{P}, \ell, c^{(1)}, \ldots, c^{(\ell)})$, $\mathsf{1DShare}(D, \mathcal{P}, \ell, \beta^{(1)}, \ldots, \beta^{(\ell)})$ and $\mathsf{1DShare}(D, \mathcal{P}, \ell, b^{(1)}\beta^{(1)}, \ldots, b^{(\ell)}\beta^{(\ell)})$ to verifiably t-1D-share $(c^{(1)}, \ldots, c^{(\ell)})$, $(\beta^{(1)}, \ldots, \beta^{(\ell)})$ and $(b^{(1)}\beta^{(1)}, \ldots, b^{(\ell)}\beta^{(\ell)})$ respectively. If in any of these 1DShare protocol, D is found to be corrupted, then every party conclude that D fails in this protocol and hence this protocol terminates.
2. Now all the parties in $\mathcal{P}$ invoke $\mathsf{RandomVector}(\mathcal{P}, 1)$ to generate a random value $r \in \mathbb{F}$.
3. For every $l \in \{1, \ldots, \ell\}$, all parties locally compute $[Y^{(l)}]_t = (r[a^{(l)}]_t + [\beta^{(l)}]_t)$ and invoke $\mathsf{ReconsPublic}(\mathcal{P}, [Y^{(l)}]_t)$ to reconstruct $Y^{(l)}$. Parallely, D broadcasts the values $Z^{(1)} = (ra^{(1)} + \beta^{(1)}), \ldots, Z^{(\ell)} = (ra^{(\ell)} + \beta^{(\ell)})$. All the parties check whether $Z^{(l)} \stackrel{?}{=} Y^{(l)}$. If not then every party concludes that D fails in this protocol and hence the protocol terminates.
4. For every $l \in \{1, \ldots, \ell\}$, the parties locally compute $[X^{(l)}]_t = \left(Y^{(l)}[b^{(l)}]_t - [b^{(l)}\beta^{(l)}]_t - r[c^{(l)}]_t\right)$ and invoke $\mathsf{ReconsPublic}(\mathcal{P}, [X^{(l)}]_t)$ to reconstruct $X^{(l)}$. The parties then check $X^{(l)} \stackrel{?}{=} 0$. If not then every party concludes that D fails in this protocol and hence the protocol terminates. Otherwise D has proved that $c^{(l)} = a^{(l)}b^{(l)}$.

Lemma 10. *In protocol ProveCeqAB, if D does not fail, then $(a^{(l)}, b^{(l)})$, $c^{(l)}$ satisfies $c^{(l)} = a^{(l)}b^{(l)}$ for $l = 1, \ldots, \ell$, except with negligible error probability. ProveCeqAB takes eighteen rounds, privately communicates $\mathcal{O}((\ell n + n^2\kappa)\kappa)$ bits and broadcasts $\mathcal{O}((\ell n + n^2\kappa)\kappa)$ bits. Moreover, if D is honest then $\mathcal{A}_t$ learns no information about $a^{(l)}, b^{(l)}$ and $c^{(l)}$, for $1 \leq l \leq \ell$.*

4.6 Multiplication

Given t-1D-sharing of ℓ pairs of secrets, say $([a^{(1)}]_t, [b^{(1)}]_t), \ldots, ([a^{(\ell)}]_t, [b^{(\ell)}]_t)$, we now present a protocol called Mult which allows the parties to compute t-1D-sharing $[c^{(1)}]_t, \ldots, [c^{(\ell)}]_t$ such that $c^{(l)} = a^{(l)}b^{(l)}$ for $l = 1, \ldots, \ell$. Our protocol is motivated from the protocol of [7], which deals with *a single* pair of t-1D-sharing. However, our protocol concurrently deals with ℓ pairs of t-1D-sharing. This leads to a gain in communication complexity.

Lemma 11. *Except with negligible error probability,* Mult *produces* $[c^{(1)}]_t, \ldots, [c^{(\ell)}]_t$ *from* ℓ *pairs* $([a^{(1)}]_t, [b^{(1)}]_t), \ldots, ([a^{(\ell)}]_t, [b^{(\ell)}]_t)$. *The protocol takes 46 rounds, privately communicates* $\mathcal{O}((\ell n^2 + n^3\kappa)\kappa)$ *bits and broadcasts* $\mathcal{O}((\ell n^2 + n^3\kappa)\kappa)$ *bits. Moreover,* $\mathcal{A}_t$ *learns nothing about* $c^{(l)}, a^{(l)}$ *and* $b^{(l)}$, *for* $1 \leq l \leq \ell$.

$([c^{(1)}]_t, \ldots, [c^{(\ell)}]_t) = \mathsf{Mult}(\mathcal{P}, \ell, ([a^{(1)}]_t, [b^{(1)}]_t), \ldots, ([a^{(\ell)}]_t, [b^{(\ell)}]_t))$

1. All the parties invoke $\mathsf{Upgrade1Dto2D}(\mathcal{P}, \ell, [a^{(1)}]_t, \ldots, [a^{(\ell)}]_t)$ and $\mathsf{Upgrade1Dto2D}(\mathcal{P}, \ell, [b^{(1)}]_t, \ldots, [b^{(\ell)}]_t)$ to upgrade t-1D-sharings of 2ℓ values to t-2D-sharings, i.e., to generate $[[a^{(1)}]]_t, \ldots, [[a^{(\ell)}]]_t$ and $[[b^{(1)}]]_t, \ldots, [[b^{(\ell)}]]_t$ respectively.
2. Let $(a_1^{(l)}, \ldots, a_n^{(l)})$ and $(b_1^{(l)}, \ldots, b_n^{(l)})$ denote the 1D sharings of $a^{(l)}$ and $b^{(l)}$ respectively. Since $a^{(l)}$ and $b^{(l)}$ is t-2D-shared, their i^{th} shares $a_i^{(l)}$ and $b_i^{(l)}$ are t-1D-shared (see the definition of t-2D-sharing). The parties in $\mathcal{P}$ locally compute $[c^{(l)}]_{2t} = [a^{(l)}]_t [b^{(l)}]_t$ for $l = 1, \ldots, \ell$ where $[c^{(l)}]_{2t} = (a_1^{(l)} b_1^{(l)}, \ldots, a_n^{(l)} b_n^{(l)})$.
3. Each party P_i has in his possession i^{th} share of $[c^{(l)}]_{2t}$ i.e. $a_i^{(l)} b_i^{(l)}$ for $l = 1, \ldots, \ell$ where both $a_i^{(l)}$ and $b_i^{(l)}$ are already t-1D-shared by P_i during Protocol Upgrade1Dto2D executed in step 1 of this protocol. Now each party P_i invokes $\mathsf{ProveCeqAB}(P_i, \mathcal{P}, \ell, [a_i^{(1)}]_t^{P_i}, [b_i^{(1)}]_t^{P_i}, \ldots, [a_i^{(\ell)}]_t^{P_i}, [b_i^{(\ell)}]_t^{P_i})$ to produce $[c_i^{(1)}]_t^{P_i}, \ldots, [c_i^{(\ell)}]_t^{P_i}$ such that $c_i^{(l)} = a_i^{(l)} b_i^{(l)}$ for $l = 1, \ldots, \ell$. At most t (corrupted) parties may fail to execute ProveCeqAB. For simplicity assume first $2t+1$ parties are successful in executing ProveCeqAB.
4. Now for each $l \in \{1, \ldots, \ell\}$, first $(2t+1)$ parties have produced $[c_1^{(l)}]_t^{P_1}, \ldots, [c_{(2t+1)}^{(l)}]_t^{P_{(2t+1)}}$. So for $l = 1, \ldots, \ell$, parties in $\mathcal{P}$ compute $[c^{(l)}]_t$ as follows: $[c^{(l)}]_t = r_1 [c_1^{(l)}]_t^{P_1} + \ldots + r_{2t+1} [c_{(2t+1)}^{(l)}]_t^{P_{(2t+1)}}$. Here $r_i = \prod_{j=1, j\neq i}^{n} \frac{-j}{i-j}$. The vector $(r_1, \ldots, r_{2t+1})$ is called recombination vector [6] which is public and known to every party.

4.7 Generating Random t-1D-Sharing

We now present a protocol called $\mathsf{Random}(\mathcal{P}, \ell)$, which allows the parties in $\mathcal{P}$ to jointly generate ℓ random t-1D-sharings, $[r^{(1)}]_t, \ldots, [r^{(\ell)}]_t$, where each $r^{(i)} \in \mathbb{F}$.

$\mathsf{Random}(\mathcal{P}, \ell)$

Every party $P_i \in \mathcal{P}$ invokes $\mathsf{1DShare}(P_i, \mathcal{P}, \ell, r^{(1,P_i)}, \ldots, r^{(\ell,P_i)})$ to verifiably t-1D-share ℓ random values $r^{(1,P_i)}, \ldots, r^{(\ell,P_i)}$ from $\mathbb{F}$. Now all the parties in $\mathcal{P}$ jointly computes $[r^{(l)}]_t = \sum_{i=1}^{n} [r^{(l,P_i)}]_t$ for $l = 1, \ldots, \ell$

Lemma 12. *With overwhelming probability,* Random *generates* ℓ *random* t*-1D-sharing* $[r^{(1)}]_t, \ldots, [r^{(\ell)}]_t$ *in 11 rounds, by privately communicating* $\mathcal{O}((\ell n^2 + n^3\kappa)\kappa)$ *bits and broadcasting* $\mathcal{O}((\ell n^2 + n^3\kappa)\kappa)$ *bits.*

5 Unconditionally Secure MPSI Protocol with $n = 3t + 1$

We now present our unconditionally secure MPSI protocol with $n = 3t + 1$.

Remark 4. In any MPSI protocol that computes the intersection of the sets using the function given in (1), $\mathcal{A}_t$ may disrupt the security of the protocol by forcing a corrupted party to input a zero polynomial representing his set [14, 15]. To avoid this, the authors of [14, 15] specified the following trick. They noticed that the coefficient of m^{th} degree term in every P_j's polynomial $f^{(P_j)}(x)$ is 1 always. Hence, every party assumes a predefined $[1]_t$ on behalf of the m^{th} coefficient of every $f^{(P_j)}(x)$ (instead of allowing individual parties to t-1D-share the m^{th} coefficient). This stops the corrupted parties to commit a zero polynomial.

Input and Preparation Phase

1. Every $P_i \in \mathcal{P}$ represents his set $S_i = \{e_i^{(1)}, e_i^{(2)}, \ldots, e_i^{(m)}\}$ by a polynomial $f^{(P_i)}(x)$ of degree m such that $f^{(P_i)}(x) = (x - e_i^{(1)}) \ldots (x - e_i^{(m)}) = a^{(0,P_i)} + a^{(1,P_i)}x + \ldots + a^{(m,P_i)}x^m$. P_i then invokes $\mathsf{1DShare}(P_i, \mathcal{P}, m, a^{(0,P_i)}, \ldots, a^{(m-1,P_i)})$ to generate $[a^{(0,P_i)}]_t, \ldots, [a^{(m-1,P_i)}]_t$. Since $a^{(m,P_i)} = 1$ always, every party in $\mathcal{P}$ assumes a predefined t-1D-sharing for 1, namely $[1]_t$ on behalf of $a^{(m,P_i)}$ (see Remark 4).
2. The parties in $\mathcal{P}$ invoke n times $\mathsf{Random}(\mathcal{P}, m+1)$ parallely, where i^{th} invocation of $\mathsf{Random}(\mathcal{P}, m+1)$ generates $m+1$ t-1D-sharings $[b^{(0,i)}]_t, \ldots, [b^{(m,i)}]_t$. Now the parties assume that $r^{(i)}(x) = b^{(0,i)} + b^{(1,i)}x + \ldots + b^{(m,i)}x^m$ for $i = 1, \ldots, n$. This step can be executed parallely with step 1.

Computation Phase

1. Let $F^{(i)}(x) = r^{(i)}(x)f^{(P_i)}(x) = c^{(0,i)} + c^{(1,i)}x + \ldots + c^{(2m,i)}x^{2m}$ for $i = 1, \ldots, n$. For $i = 1, \ldots, n$, to generate $[c^{(0,i)}]_t, \ldots, [c^{(2m,i)}]_t$, the parties in $\mathcal{P}$ do the following:
   (a) The parties invoke $\mathsf{Mult}(\mathcal{P}, (m+1)^2, ([a^{(0,i)}]_t, [b^{(0,i)}]_t), ([a^{(0,i)}]_t, [b^{(1,i)}]_t), \ldots, ([a^{(m,i)}]_t, [b^{(m-1,i)}]_t), ([a^{(m,i)}]_t, [b^{(m,i)}]_t))$ with $(m+1)^2$ pairs (every coefficient of $r^{(i)}(x)$ should be multiplied with every coefficient of $f^{(P_i)}(x)$) to produce $(m+1)^2$ t-1D-sharings $[a^{(0,i)}b^{(0,i)}]_t, [a^{(0,i)}b^{(1,i)}]_t, \ldots, [a^{(m,i)}b^{(m-1,i)}]_t, [a^{(m,i)}b^{(m,i)}]_t$.
   (b) The parties compute $[c^{(0,i)}]_t = [a^{(0,i)}b^{(0,i)}]_t$, $[c^{(1,i)}]_t = [a^{(0,i)}b^{(1,i)}]_t + [a^{(1,i)}b^{(0,i)}]_t, \ldots, [c^{(2m,i)}]_t = [a^{(m,i)}b^{(m,i)}]_t$.
2. Let $F(x) = \sum_{i=1}^{n} F^{(i)}(x) = d^{(0)} + d^{(1)}x + \ldots + d^{(2m)}x^{2m}$. To generate $[d^{(0)}]_t, \ldots, [d^{(2m)}]_t$, the parties compute $[d^{(j)}]_t = \sum_{i=1}^{n} [c^{(j,i)}]_t$ for $j = 0, \ldots, 2m$.

Output Phase

1. The parties invoke $\mathsf{ReconsPublic}(\mathcal{P}, [d^{(j)}]_t)$ to publicly reconstruct $d^{(j)}$ for $j = 0, \ldots, 2m$. Thus now parties have reconstructed $F(x)$.
2. Each P_i with his private set $S_i = \{e_i^{(1)}, \ldots, e_i^{(m)}\}$ locally checks whether $F(e_i^{(l)}) \stackrel{?}{=} 0$ for $l = 1, \ldots, m$. If $F(e_i^{(l)}) = 0$, the P_i adds $e_i^{(l)}$ in a set IS_i (initially $IS_i = \emptyset$). P_i outputs IS_i as the intersection set $S_1 \cap S_2 \ldots, \cap S_n$.

Theorem 1. *MPSI protocol with $3t+1$ takes 58 rounds, privately communicates $\mathcal{O}((m^2n^3+n^4\kappa)\kappa)$ and broadcasts $\mathcal{O}((m^2n^3+n^4\kappa)\kappa)$ bits, when physical broadcast channel is available in the system. In the absence of a physical broadcast channel, the protocol takes $\mathcal{O}(t)$ rounds and privately communicates $\mathcal{O}((m^2n^5 + n^6\kappa)\kappa)$ bits. The protocol solves MPSI problem with very high probability.*

6 Open Problem

Designing efficient information theoretically secure MPSI protocol with optimal resilience (i.e., with $n = 2t + 1$) is left as an open problem.

References

1. Beerliová-Trubíniová, Z., Hirt, M.: Efficient Multi-party Computation with Dispute Control. In: Halevi, S., Rabin, T. (eds.) TCC 2006. LNCS, vol. 3876, pp. 305–328. Springer, Heidelberg (2006)
2. Beerliová-Trubíniová, Z., Hirt, M.: Perfectly-Secure MPC with Linear Communication Complexity. In: Canetti, R. (ed.) TCC 2008. LNCS, vol. 4948, pp. 213–230. Springer, Heidelberg (2008)
3. Ben-Or, M., Goldwasser, S., Wigderson, A.: Completeness Theorems for Non-Cryptographic Fault Tolerant Distributed Computation. In: 20th ACM Symposium on Theory of Computing, pp. 1–10. ACM Press, New York (1988)
4. Berman, P., Garay, J.A., Perry, K.J.: Bit Optimal Distributed Consensus. Comp. Sci. Research, 313–322 (1992)
5. Carter, L., Wegman, M.N.: Universal Classes of Hash Functions. J. of Comp. and Sys. Sci. 18(4), 143–154 (1979)
6. Cramer, R., Damgård, I.: Multiparty Computation: An Introduction: Contemporary Cryptography. Birkhäuser, Basel (2005)
7. Cramer, R., Damgård, I., Dziembowski, S., Hirt, M., Rabin, T.: Efficient Multiparty Computations Secure Against an Adaptive Adversary. In: Stern, J. (ed.) EUROCRYPT 1999. LNCS, vol. 1592, pp. 311–326. Springer, Heidelberg (1999)
8. Damgård, I., Nielsen, J.B.: Scalable and Unconditionally Secure Multiparty Computation. In: Menezes, A. (ed.) CRYPTO 2007. LNCS, vol. 4622, pp. 572–590. Springer, Heidelberg (2007)
9. Fitzi, M., Garay, J., Gollakota, S., Pandu Rangan, C., Srinathan, K.: Round-Optimal and Efficient Verifiable Secret Sharing. In: Halevi, S., Rabin, T. (eds.) TCC 2006. LNCS, vol. 3876, pp. 329–342. Springer, Heidelberg (2006)
10. Freedman, M.J., Nissim, K., Pinkas, B.: Efficient Private Matching and Set Intersection. In: Cachin, C., Camenisch, J.L. (eds.) EUROCRYPT 2004. LNCS, vol. 3027, pp. 1–19. Springer, Heidelberg (2004)
11. Gennaro, R., Ishai, Y., Kushilevitz, E., Rabin, T.: The Round Complexity of Verifiable Secret Sharing and Secure Multicast. In: 33rd ACM Symposium on Theory of Computing, pp. 580–589. ACM Press, New York (2001)
12. Hirt, M., Maurer, U., Przydatek, B.: Efficient Secure Multi-party Computation. In: Okamoto, T. (ed.) ASIACRYPT 2000. LNCS, vol. 1976, pp. 143–161. Springer, Heidelberg (2000)
13. Katz, J., Koo, C.Y., Kumaresan, R.: Improving the Round Complexity of VSS in Point-to-Point Networks. In: Aceto, L., Damgård, I., Goldberg, L.A., Halldórsson, M.M., Ingólfsdóttir, A., Walukiewicz, I. (eds.) ICALP 2008, Part II. LNCS, vol. 5126, pp. 499–510. Springer, Heidelberg (2008)
14. Kissner, L., Song, D.: Privacy-Preserving Set Operations. In: Shoup, V. (ed.) CRYPTO 2005. LNCS, vol. 3621, pp. 241–257. Springer, Heidelberg (2005)
15. Li, R., Wu, C.: An Unconditionally Secure Protocol for Multi-Party Set Intersection. In: Katz, J., Yung, M. (eds.) ACNS 2007. LNCS, vol. 4521, pp. 226–236. Springer, Heidelberg (2007)

16. Lynch, N.A.: Distributed Algorithms. Morgan Kaufmann, San Francisco (1996)
17. MacWilliams, F.J., Sloane, N.J.A.: The Theory of Error Correcting Codes. North-Holland Publishing Company, Amsterdam (1978)
18. Rabin, T., Ben-Or, M.: Verifiable Secret Sharing and Multiparty Protocols with Honest Majority. In: 21st ACM Symposium on Theory of Computing, pp. 73–85. ACM Press, New York (1989)
19. Srinathan, K., Narayanan, A., Pandu Rangan, C.: Optimal Perfectly Secure Message Transmission. In: Franklin, M. (ed.) CRYPTO 2004. LNCS, vol. 3152, pp. 545–561. Springer, Heidelberg (2004)
20. Yao, A.C.: Protocols for Secure Computations. In: 23rd IEEE Symposium on Foundations of Computer Science, pp. 160–164. IEEE Press, Los Alamitos (1982)