Verifiable Computation in Multiparty Protocols with Honest Majority

Peeter Laud Alisa Pankova

Monday 27th January, 2014

Abstract

A lot of cryptographic protocols have been proposed for semihonest model. In general, they are much more efficient than those proposed for the malicious model. In this paper, we propose a method that allows to detect the parties that have violated the protocol rules after the computation has ended, thus making the protocol secure against covert attacks. This approach can be useful in the settings where for any party it is fatal to be accused in violating protocol rules. In this way, up to the verification, all the computation can be performed in semihonest model, which makes it very efficient in practice. The verification is statistical zero-knowledge, and it is based on linear probabilistically checkable proofs (PCP) for verifiable computation. Each malicious party is detected with probability $1 - \varepsilon$ for a negligible ε that is defined by the failure of the corresponding linear PCP. The initial protocol has to be executed only once, and the verification requires in total 3 additional rounds (if some parties act dishonestly, in the worst case they may force the protocol to substitute each round with 4 rounds, due to the transmission functionality that prevents the protocol from stopping). The verification also ensures that all the parties have sampled all the randomness from an appropriate distribution. Its efficiency does not depend on whether the inputs of the parties have been shared, or each party uses its own private input.

The major drawback of the proposed scheme is that the number of values sent before and after the protocol is exponential in the number of parties. Nevertheless, the settings make the verification very efficient for a small number of parties.

1 Introduction

The semihonest and the malicious model are the two main models in which cryptographic protocols are studied. In the semihonest model, the adversary is curious about the values it gets, and it tries to extract information out of them, but it follows the protocol rules honestly. In the malicious model, the adversary is allowed to do whatever it wants. In addition to these traditional models, a notion of covert security was proposed in [AL10]. In this model, the adversary is malicious, but it will not cheat if it will be caught with a non-negligible probability, which can be defined more precisely as a security parameter. This notion is very realistic in many computational models, where the participants care about their reputation and will not cheat even if this probability is not close to 1.

Some works have been dedicated to covert security [Lin13, DGN10], where [Lin13] treats the security for two-party computation based on garbled circuits, both the covert and the malicious cases, and [DGN10] deals with honest majority protocols for an arbitrary number of parties. A more precise definition of *covert security with public verifiability* has been proposed in [AO12]. This allows the cheater to be blamed publicly.

2 Our Contribution

In this paper, we propose a scheme that is based on succinct computation verification. Our work is closely related to [DGN10] that is dealing with honest majority protocols for an arbitrary number of parties. The solution proposed in [DGN10] is based on running the initial protocol on two inputs, the real shares and the dummy shares. In this case, the real shares should be indistinguishable from random, and hence in the beginning the protocol is being rewritten to a shared form. Differently from [DGN10], our solution does not require rewriting the original protocol. The original protocol has to be run only once, and each malicious party is detected with probability $1 - \varepsilon$ for a negligible ε. Our approach is statistical zero-knowledge, and it it based on linear probabilistically checkable proofs (PCP) for verifiable computation. The particular PCP that we are using is the one proposed in [BSCG+13]. The quantity ε is defined by the failure of the corresponding linear PCP behind the protocol. Additionally, the verification ensures that all the parties have sampled all the randomness from an appropriate distribution. If everyone is indeed honest, the verification requires in total 3 additional rounds. If some parties are dishonest, then in the worst case they may force the protocol to substitute each round with 4 rounds, due to the transmission functionality.

One question is whether there are indeed no additional rounds if everyone is honest, since there should be some time offset when the parties wait for possible complaints, and if nothing happens, only then they proceed to the next round. Since there are no complaints if everyone is honest, there is no communication during that time offset, and hence such additional rounds are much cheaper.

Another solution is to assume that the complaints can be presented not immediately, but on the next round, when instead of ordinary messages some party may sent complaints it had on the previous round. In this way, if everyone acts honestly, we will have just 3 additional rounds compared to the initial protocol (2 rounds are needed for the verification, and one more is the final time offset in the end of the protocol, where the parties should wait for the possible lastmost complaints).

The major drawback of our scheme is that the number of values sent per one round is exponential in the number of parties. In [DGN10], efficiency is achieved by reducing the probability of being detected from 1/2 to 1/4. We cannot use the same approach in our case since the probability of being detected would immediately become negligible. Nevertheless, the settings make the verification very efficient for a small number of parties.

Similarly to [DGN10], we prove the security of our scheme in UC model [Can01].

2.1 Notation

Throughout this work, we use the following notation:

- the upper case letters A denote matrices;
- the bold lower case letters $\mathbf{b}$ denote vectors;
- $\langle \mathbf{a}, \mathbf{b} \rangle$ denotes the scalar product of $\mathbf{a}$ and $\mathbf{b}$;
- $(\mathbf{a}||\mathbf{b})$ is a concatenation of vectors $\mathbf{a}$ and $\mathbf{b}$.

2.2 Assumptions

Our verification protocol is based on security of some other schemes. Here is the list of used assumptions.

- Secure point-to-point channels between each pair of parties.
- Broadcast channels between subsets of parties.
- Honest Verifier Statistical Zero-Knowledge Linear Probabilistically Checkable Proofs for verifiable computation [Lip13, GGPR13, BSCG+13, BCI+13]. In particular, all the complexity estimations in this chapter are based on the solution proposed in [BSCG+13].
- Functionality that allows to prove to third parties which messages one received during the protocol, and to further transfer such revealed messages. This allows to protect the initial protocol from stopping, when the computation cannot proceed due to some malicious party that is either just doing nothing, or causes some other party to wait by sending wrong messages. We use the solution proposed in [DGN10].

2.3 The Protocol Outline

We describe briefly the initial settings, and how the new verifiable protocol differs from the original one.

- In the initial settings, we have a set of arithmetic circuits C_i^j over some finite field $\mathbb{F}$, where C_i^j is the circuit computed by the party M_i on the j-th round of computation. Some outputs of C_i^j may be used as inputs for some C_k^{j+1}, so there is some communication between the parties. Each circuit may use some randomness that comes from random uniform distribution in $\mathbb{F}$ (this is sufficient to model any other distribution). The circuits could be boolean as well, since there also exist linear probabilistically checkable proofs based on boolean circuits [Lip13], so our verification is not restricted to computation over some certain field.
- The computation is performed by n parties. Let them be denoted M_i for $i \in \{1, \ldots, n\}$. A necessary condition is that at least $t = \lfloor n/2 \rfloor + 1$ are honest.
- Before the execution of original protocol starts, the inputs of the parties are committed in a special way. Let the input of the party M_i be represented by a vector $\mathbf{x}_i$ over $\mathbb{F}$. M_i represents $\mathbf{x}_i$ as $\binom{n-1}{t}$ distinct sums of the form $\mathbf{x}_i = \sum_{k \in T_j} \mathbf{x}_{ikT_j}$ for $j \in \{1, \ldots, \binom{n-1}{t}\}$, where each T_j represents a distinct subset of t other parties, and k corresponds to one particular party in that subset. The idea is that each party has to prove its honestness to any subset of t other parties. All the shares are signed and distributed amongst the corresponding parties. Although the number $\binom{n-1}{t}$ is exponential, computing all $t \cdot \binom{n-1}{t}$ signatures is not less efficient than computing just one, for example using hash Merkle tree.
- The randomness used in the protocols should also be committed in the same way. Moreover, we want to ensure that it indeed comes from random uniform distribution, without revealing to anyone its value.
  - Let an arbitrary set of t parties be responsible for generating the randomness. Let these parties be called "generators". By honest majority assumption, at least one of them is honest. For each M_i, they generate the randomness $\mathbf{r}_i$ as follows. Each generator M_j generates $\mathbf{r}_{ji}$ of the same length that $\mathbf{r}_i$ should be. The idea is to take $\mathbf{r}_i = \mathbf{r}_{i1} + \ldots + \mathbf{r}_{it}$. Since at least one party is honest, the vector $\mathbf{r}_i$ comes from a random uniform distribution.
  - Each generator M_j represents its $\mathbf{r}_{ij}$ as $\binom{n-1}{t}$ distinct sums of the form $\mathbf{r}_{ij} = \sum_{k \in T_\ell} \mathbf{r}_{ijkT_\ell}$ for $\ell \in \{1, \ldots, \binom{n-1}{t}\}$. All the shares are signed and sent to M_i. After M_i receives $\mathbf{r}_{ij}$ from all generators M_j, it may compute the sum of all $\mathbf{r}_{ij}$ and use it as $\mathbf{r}_i$ (M_i has to verify if the shares for different sets T_ℓ indeed all represent the same value). Then M_i signs all the received shares also by itself, and distributes the shares and the signatures (both signed by M_i and the corresponding generator M_j) amongst appropriate subsets of t parties, similarly to $\mathbf{x}_i$.
- The original protocol is computed in the same way as before. Additionally, each communicated vector $\mathbf{c}_{ij}^\ell$ sent by M_i to M_j on the round ℓ is presented as $\binom{n}{t}$ distinct sums $\mathbf{c}_{ij}^\ell = \sum_{k \in T_{j'}} \mathbf{c}_{ijkT_{j'}}^\ell$ (here we have $\binom{n}{t}$ instead of $\binom{n-1}{t}$ since both communicating parties should later verify the consistency of this value from each other). Along with each $\mathbf{c}_{ij}^\ell$, M_j receives the signature of $\mathbf{c}_{ij}^\ell$ and the signatures of all the shares $\mathbf{c}_{ij\ell kT_{j'}}^\ell$. M_j checks if the signatures are all indeed valid, and in turn signs them. M_j distributes the corresponding signatures (both signed by M_i and M_j) amongst each T_j. Here M_j is unable to check whether the shares under the signatures are valid and indeed sum up to $\mathbf{c}_{ij}^\ell$. All the shares will be distributed after the protocol execution, and then M_i may present the signature of $\mathbf{c}_{ij}^\ell$ to complain.
- After the protocol computation ends, all the communication shares are finally distributed. Each party M_j is verified for honestness. A party is honest iff it can prove that it acted according to the protocol, given the signed input, randomness, and communication that it had with the other parties. It has to perform a 3-round interactive proof with each subset of t parties in parallel. Since each subset of t parties holds all the shares of all the committed values, they are able to reconstruct the committed values and check if the proof indeed corresponds to them.

  In general, in a linear PCP the prover has to prove the knowledge of a vector $\pi = (\mathbf{p}||\mathbf{d})$ such that certain combinations of $\langle \pi, \mathbf{q}_i \rangle$ for special challenges $\mathbf{q}_1, \ldots, \mathbf{q}_5$ should be equal to 0, and $\mathbf{d}$ corresponds to the committed values. The problem is that the prover cannot see any of the $\mathbf{q}_i$ before committing the proof, but at the same time π should remain private.

  In particular, for any subset of t verifiers, the following has to be done (ordered by rounds).

  1. The verifiers agree on a random $\tau \in \mathbb{F}$ that is sufficient to generate all $\mathbf{q}_1, \ldots, \mathbf{q}_5$ (in one round). The prover generates shares $\pi = \pi_1 + \ldots + \pi_t$ (where the $\mathbf{d}$ part is shared in the same way as it

was committed to the given set of t verifiers) and distributes them amongst the parties. Each verifier checks if the part that corresponds to $\mathbf{d}$ is consistent with the signatures of shares sent during the computation.

2. Each verifier V_i computes and publishes $\langle \pi_i, \mathbf{q}_j \rangle$ for $j \in \{1, \ldots, 5\}$. The τ is published. Everyone may compute $\langle \pi_1, \mathbf{q}_j \rangle + \ldots + \langle \pi_t, \mathbf{q}_j \rangle = \langle \pi, \mathbf{q}_j \rangle$ for $j \in \{1, \ldots, 5\}$ and locally verify the necessary combinations. The prover checks if all the scalar products are computed correctly, and complains if necessary.

A party is claimed honest iff it succeeds in all the $\binom{n-1}{t}$ proofs against t other parties. This means that even if it was in collaboration with $t-2$ other malicious parties, there exists a subset of t all-honest parties that will definitely accept only the correct proof. We also need to ensure that the presence of malicious parties will not make the proof fail for an honest prover, and this can be done by revealing the signatures that correspond to the shares of incorrect scalar products. An honest party is safe to open them since they are known by the adversary anyway.

3 A Linear Probabilistically Checkable Proof for Verifiable Computation

In this section we describe in more details the PCP that we use in our verification. This will be necessary since we do not use it just as a black box, but commit some parts of the proof in a special way, so we need to ensure that everything still works. Additionally, writing it down allows to estimate the complexity more precisely.

We start from a statistical honest verifier zero knowledge (HVZK) linear PCP based on translating each arithmetic circuit to a quadratic arithmetic program [BSCG$^+$13].

Definition 1 (Linear PCP). *[BCI$^+$13] Let $\mathcal{R}$ be a binary relation, $\mathbb{F}$ a finite field, P_{LPCP} a deterministic prover algorithm, and V_{LPCP} a probabilistic oracle verifier algorithm. We say that the pair (P_{LPCP}, V_{LPCP}) is an input-oblivious k-query linear PCP for $\mathcal{R}$ over $\mathbb{F}$ with knowledge error ε and query length m if it satisfies the following requirements.*

1. ***Syntax.*** *On any input $\mathbf{v}$ and oracle π, the verifier $V_{LPCP}(\mathbf{v})$ makes k input-oblivious queries to π and then decides whether to accept or reject. More precisely, V_{LPCP} consists of a probabilistic query algorithm Q_{LPCP} and a deterministic decision algorithm D_{LPCP} working as follows. Based on its internal randomness τ, and independently of $\mathbf{v}$, Q_{LPCP} generates k query vectors $\mathbf{q_1}, \ldots, \mathbf{q_k} \in \mathbb{F}^m$ to π and state information u. Then, given $\mathbf{v}$, u, and the k oracle answers $a_1 = \langle \pi, \mathbf{q_1} \rangle, \ldots, a_k = \langle \pi, \mathbf{q_k} \rangle$, D_{LPCP} accepts or rejects.*
2. ***Completeness.*** *For every $(\mathbf{v}, \mathbf{w}) \in \mathcal{R}$, the output of $P_{LPCP}(\mathbf{v}, \mathbf{w})$ is a description of a linear function $\pi : \mathbb{F}^m \to \mathbb{F}$ such that $V^{\pi}_{LPCP}(\mathbf{v})$ accepts with probability 1.*
3. ***Knowledge.*** *There exists a knowledge extractor E_{LPCP} such that for every linear function $\pi^* : \mathbb{F}^m \to \mathbb{F}$ if the probability that $V^{\pi^*}_{LPCP}(\mathbf{v})$ accepts is greater than ε then $E^{\pi^*}_{LPCP}(\mathbf{v})$ outputs $\mathbf{w}$ such that $(\mathbf{v}, \mathbf{w}) \in \mathcal{R}$.*

In relation to verifiable computation, this definition is used in the following context:

1. $\mathbf{v}$ is the vector of committed input/randomness/communication/public variables.
2. $\mathbf{w}$ is an extension of $\mathbf{w}$ that contains non-deterministic auxiliary input and the values of intermediate gates.
3. $(\mathbf{v}, \mathbf{w}) \in \mathcal{R}$ iff $\mathbf{w}$ is a valid vector of values that the prover party indeed would have got if it followed the protocol honestly, according to the committed input/randomness/communication/public variables $\mathbf{v}$.

A particular solution proposed in [BSCG$^+$13] is a statistical HVZK. Namely, it means that the answers to the queries $\langle \pi, \mathbf{q}_i \rangle$ do not reveal any information about the input, unless the randomness $\tau \in \mathbb{F}$ that has been used in query generation is chosen in a bad way, which happens with negligible probability. In this particular solution, it is sufficient to generate 5 challenges $\mathbf{q}_1, \ldots, \mathbf{q}_5$.

Let us describe the solution of [BSCG$^+$13] in a bit more details. First of all, it has been shown that any arithmetic circuit can be represented by a quadratic arithmetic program.

Definition 2 (Quadratic Arithmetic Program)**.** *Consider some integers m, n, k such that $n-1 \geq k$. A* strong quadratic arithmetic program (QAP) *over a field $\mathbb{F}$, denoted $\mathsf{P}(A,B,C)$, consists of three $m \times n$ matrices A, B, C over a field $\mathbb{F}$. P accepts a vector $\mathbf{v} \in \mathbb{F}^k$ iff there exists a vector $\mathbf{w} = (1, w_1, \ldots, w_{n-1})$ such that $(w_1, \ldots, w_k) = \mathbf{v}$ and $A\mathbf{w} \circ B\mathbf{w} = C\mathbf{w}$.*

The relation $R_{\mathsf{P}(A,B,C)}$ is defined as

$$(\mathbf{v}, \mathbf{w}) \in R_{\mathsf{P}(A,B,C)} \iff \mathsf{P}(A,B,C) \text{ accepts on input } \mathbf{v} \ .$$

The problem of program verification is now reduced to the problem of proving the existence of $\mathbf{w}$ such that $(\mathbf{v}, \mathbf{w}) \in R_{\mathsf{P}(A,B,C)}$, where A,B,C are defined by the arithmetic circuits that the parties compute. The values m and n are both $O(|C|)$. More precisely, n is the number of wires in the circuit (all the input/randomness/communication/intermediate variables), and m is the number of multiplication gates.

Preprocessing: Denote $A = (a_{ij})$, $B = (b_{ij})$, $C = (c_{ij})$ for $i \in \{0, \ldots, m-1\}$, $j \in \{0, \ldots, n-1\}$. Let $S = \{\omega^0, \ldots, \omega^{m-1}\} \subseteq \mathbb{F}$ for a principal root of unity ω in $\mathbb{F}$.

Let A_j, B_j, C_j for $j \in \{0, \ldots, n-1\}$ be polynomials of degree $m-1$ defined in such a way that $A_j(\omega^i) = a_{ij}$, $B_j(\omega^i) = b_{ij}$, $C_j(\omega^i) = c_{ij}$. The coefficients of these polynomials can be computed by interpolation, for example using Fast Fourier Transform, and they are of degree $m-1$ since they are defined on m points. Let $\mathbf{A}(x) := (A_0(x), \ldots, A_{n-1}(x))$, $\mathbf{B}(x) := (B_0(x), \ldots, B_{n-1}(x))$, $\mathbf{C}(x) := (C_0(x), \ldots, C_{n-1}(x))$ denote the vectors of corresponding polynomials.

Let $Z_S(x) = \prod_{s \in S}(x - s)$ be an m-degree polynomial over $\mathbb{F}$. In this way, $Z_S(x)$ has exactly m roots which are the elements of S.

The set S and the coefficients of $\mathbf{A}(x)$, $\mathbf{B}(x)$, $\mathbf{C}(x)$, $Z_S(x)$ are published.

Linear PCP Prover $P(\mathbf{v}, \mathbf{w})$: Let $\mathbf{v} \in \mathbb{F}^k$, $\mathbf{w} \in \mathbb{F}^n$. The prover works as follows:

- Let $\delta_A, \delta_B, \delta_C \in \mathbb{F}$ be random field elements.
- Let $A(x)$, $B(x)$, $C(x)$ be polynomials of degree m such that:

$$A(x) := \langle \mathbf{w}, \mathbf{A}(x) \rangle + \delta_A Z_S(x) \ ;$$

$$B(x) := \langle \mathbf{w}, \mathbf{B}(x) \rangle + \delta_B Z_S(x) \ ;$$

$$C(x) := \langle \mathbf{w}, \mathbf{C}(x) \rangle + \delta_C Z_S(x) \ ;$$

  where the degree m comes from the fact that the degree of each polynomial in $\mathbf{A}(x)$, $\mathbf{B}(x)$, $\mathbf{C}(x)$ is $m-1$, and the degree of $Z_S(x)$ is m.
- Let $\mathbf{h} = (h_0, \ldots, h_m)$ be the coefficients of the polynomial

$$H(x) := \frac{A(x)B(x) - C(x)}{Z_S(x)} \ .$$

The algorithm returns the proof $\pi = ((\delta_A, \delta_B, \delta_C)||\mathbf{w}||\mathbf{h})$. It can be done locally by the prover in time $O(|C| \log |C|)$, and the details can be seen in [BSCG+13].

Linear PCP Verifier $V = (Q_{LPCP}, D_{LPCP}(\mathbf{v}, \mathbf{u}, \mathbf{a}))$: The work of the verifier is split into two parts: the query algorithm Q_{LPCP} and the decision algorithm D_{LPCP}.

- Q_{LPCP}: First of all, a random element $\tau \in \mathbb{F}$ is generated. Then the following queries $\mathbf{q}_i \in \mathbb{F}^{3+m+(n+1)=4+m+n}$ are computed:
  1. $\mathbf{q}_1 = ((Z_S(\tau), 0, 0)||\mathbf{A}(\tau)||(0, 0, \ldots, 0))$;
  2. $\mathbf{q}_2 = ((0, Z_S(\tau), 0)||\mathbf{B}(\tau)||(0, 0, \ldots, 0))$;
  3. $\mathbf{q}_3 = ((0, 0, Z_S(\tau))||\mathbf{C}(\tau)||(0, 0, \ldots, 0))$;
  4. $\mathbf{q}_4 = ((0, 0, 0)||(0, 0, \ldots, 0, 0, \ldots, 0)||(1, \tau, \ldots, \tau^m))$;
  5. $\mathbf{q}_5 = ((0, 0, 0)||(1, \tau, \ldots, \tau^k, 0, \ldots, 0)||(0, 0, \ldots, 0))$.

  The state information is $\mathbf{u} := (1, \tau, \tau^2, \ldots, \tau^k, Z_S(\tau))$. The query results are $a_i = \langle \pi, \mathbf{q}_i \rangle$ for $i \in \{1, \ldots, 5\}$. Everything can be computed in $O(|C|)$, the details can be seen in [BSCG+13].
- $D_{LPCP}(\mathbf{v}, \mathbf{u}, \mathbf{a})$: Let $\mathbf{u} = (u_1, \ldots, u_{k+2})$, $\mathbf{a} = (a_1, \ldots, a_5)$. The algorithm accepts iff:

1. $a_1 a_2 - a_3 - a_4 u_{k+2} = 0$,
2. $a_5 - u_1 - \langle \mathbf{v}, (u_2, \dots, u_{k+1}) \rangle = 0$.

Now our goal is to perform this verification in the end of the computation. It is necessary to compute 5 scalar products $\langle \pi, \mathbf{q}_1 \rangle, \dots, \langle \pi, \mathbf{q}_5 \rangle$, where π cannot depend on any of the $\mathbf{q}_k$, and at the same time π cannot be revealed to any other party. The values $\langle \pi, \mathbf{q}_k \rangle$ themselves may be revealed since it has been proven in [BSCG$^+$13] that they are already statistical zero-knowledge on the assumption that all the vectors $\mathbf{q}_k$ have been chosen according to the rules. In general, the proposed linear interactive proof can be converted to a zero-knowledge succinct non-interactive argument of knowledge, as is shown for example in [BCI$^+$13]. The problem is that it requires homomorphic encryption, and the number of encryptions has to be linear in the size of the circuit. Additionally, according to the previous description, the only way to check if $\mathbf{w}$ indeed contains $(\mathbf{x}||\mathbf{r}||\mathbf{c})$ that correspond to the committed shares is to put $(\mathbf{x}||\mathbf{r}||\mathbf{c})$ into $\mathbf{v}$, but we would not like to make these values public. We propose a solution that uses the honest majority assumption instead of homomorphic encryption, and that is more suitable to our settings.

4 The Proposed Protocol

In this section we describe our protocol in details. First, we list the complexity overheads, based on the particular PCP that has been presented in the previous section. Then we define the ideal functionality that we would like to get (which is very similar to the one proposed in [DGN10]), and describe the behaviour of each party in the real protocol. We prove by simulation that our real functionality is as secure as the ideal functionality.

4.1 Properties

In our settings, we have n parties M_i. Compared to the original protocol, for each M_i the proposed solution has the following computational overheads.

- Let $p = \binom{n-2}{t-1}$. This is the number of t-sets in which one party participates as a verifier. If everyone is honest, then in order to verify M_j's honestness, M_i has to send the following messages:
  - In the initial protocol, send two signatures and $tp + p$ vectors of length $O(|C|)$ to each of the $n-1$ parties (tp for the randomness, and p for the inputs).
  - During the protocol execution, in addition to the original protocol communication, send $r \cdot (1 + \cdot (n-1)) = rn$ signatures to each of the $n-1$ receiver parties, where r is the number of rounds. Each receiver M_j produces rn more signatures of the same values. All these signatures are sent by each receiver M_j to the $n-1$ remaining parties (including M_i).
  - After the protocol execution, compute locally the auxiliary values for the proof in $O(|C| \log |C|)$ steps, as shown in [BSCG$^+$13]. Send to each of the other $n-1$ parties in parallel $2(n-1)$ signatures and $2p(n-1)$ vectors of length $O(|C|)$: $p(n-1)$ for intermediate variables (for each proof separately), and $p(n-1)$ for communication, each set of p vectors signed with one signature.

  As a verifier, in the verification process each M_i has to do the following:
  - Locally generate, sign and multicast a random element of $\mathbb{F}$.
  - Locally generate p state informations $\mathbf{u}$ and $5p$ challenge vectors $\mathbf{q}_k$ of length $O(|C|)$ (according to the arithmetic circuit). This can be done in $O(|C|)$ steps, as shown in [BSCG$^+$13].
  - Locally sum up and concatenate pt vectors of length $O(|C|)$ that correspond to the randomness: for each of the p verifier sets, sum up all the t vectors.
  - Locally concatenate 4 vectors of total result length $O(|C|)$: the shares of the input, randomness, communication, and the intermediate values. This is done p times, for each verification set.
  - Locally compute $5p$ scalar products of vectors of length $O(|C|)$ and broadcast them (5 for each proof).
  - In the end, compute a constant number of local operations based on these scalar products: 2 multiplications, 3 additions, 1 scalar product of length $O(|\mathbf{v}|)$ for the part of the input $\mathbf{v}$ whose

value is public (which is in general just the constant 1), all operations in $\mathbb{F}$. Everything is done p times, for each proof.

- If something goes wrong with the proof of M_j's honestness, then in the worst case each sent message has to be sent in such a way that it is possible to prove afterwards what has been sent to whom. The $\mathcal{F}_{transmit}$ functionality from [DGN10] requires each message to be broadcasted to all $n-1$ parties, and then this message should be delivered by each of the $n-2$ remaining parties to the receiver. No additional signatures are needed since we have already considered all of them in the case where everyone acts honestly.

According to the Linear PCP description from [BSCG+13], a dishonest prover may cheat with probability $\frac{2m}{|\mathbb{F}|}$ where m is the number of multiplication gates in the circuit. This means that either the field should be large enough, or the verification should be repeated k times, so that $\left(\frac{2m}{|\mathbb{F}|}\right)^k$ is negligible. All the k verifications can be done in parallel, by generating k sets of challenges instead of one, thus do not increasing the number of rounds at all, and increasing the communication in total by $p(n-1)k$ field elements and $p(n-1)k$ proof vector shares.

4.2 Notation

The circuit in general has the following variables:

- Input vectors: $\mathbf{x}_i$ for the input of each M_i.
- Randomness vectors: $\mathbf{r}_i$ for the randomness of each M_i.
- Public vectors: $\mathbf{v}_i$ for the input of each M_i. This is the part of M_i's input that may have to be public for some reason.
- Output vectors: $\mathbf{y}_i$ which is an output issued by M_i (not needed in verification unless involved in some published value).
- Intermediate gates: $\mathbf{z}_i$ computed by M_i.
- Communication values: $\mathbf{c}_{ij}$ for a total vector of all the values that have been sent from M_i to M_j.

Each party M_j is verified by each subset of t other parties. The idea is that each M_j has to prove that it knows appropriate $\mathbf{z}_j$ such that the computation is consistent with the signatures of the shares of inputs $\mathbf{x}_j$, randomness $\mathbf{r}_j$, and the communicated values $\mathbf{c}_{jk}$ and $\mathbf{c}_{kj}$ for $k \in \{1, \dots, n\} \setminus \{j\}$.

4.3 Universal Composability

For each of the 5 queries (according to the PCP construction), we need to compute one scalar product of the form $\langle \pi, \mathbf{q}_k \rangle$ where π should be provided by the prover M_j. Here the prover is not permitted to see $\mathbf{q}_k$, and the verifier is not permitted to see π, although the result $\langle \pi, \mathbf{q}_k \rangle$ can by assumption be seen by anyone. Additionally, the vector π cannot be arbitrary, some of its parts should correspond to the committed randomness, input, and the communication variables. In general, this proof can be done by using homomorphic encryption of each coordinate of $\mathbf{q}_k$, but that would be very slow, and additionally we could lose some zero-knowledge assumptions from commitment verification.

We define an ideal functionality that we would like to have, and the functionality of each M_i in our protocol (all of them are symmetric). We prove that our real functionality is as secure as this ideal functionality by simulation.

Existing Ideal Functionalities

We will use some existing ideal functionalities as components.

- $\mathcal{F}_{ppp}$ implements a point-to-point secure channel between any two parties (and the adversary).
- $\mathcal{F}_{bc}$ implements broadcast channel between subsets of parties.
- $\mathcal{F}_{sign}$ allows signature generation and verification.

- $\mathcal{F}_{transmit}$ allows to prove to third parties which messages one received during the protocol, and to further transfer such revealed messages. We use the definition similar [DGN10], which works with message identifiers *mid*, encoding a sender $s(mid) \in \{1, \ldots, n\}$ and a receiver $r(mid) \in \{1, \ldots, n\}$, and assuming that no *mid* is used twice. Since in our case the signatures and the broadcasts are treated separately, our definition is formally a bit different but its essence remains the same. The functionality works as follows.
  - When receiving $(\mathbf{transmit}, mid, m)$ from an honest $M_{s(mid)}$, $\mathcal{F}_{transmit}$ sends (mid, m) to $M_{r(mid)}$.
  - On input $(\mathbf{reveal}, mid, T)$ from a party M_j which at any point has either sent or received $(\mathbf{transmit}, mid, m)$, if at least one of $M_{s(mid)}$ and $M_{r(mid)}$ is honest, output (mid, m) to the set of parties T.
  - On input $(\mathbf{reveal}, mid, T)$ from a party M_j, which at any point has either sent or received $(\mathbf{transmit}, mid, m)$, if all the parties $M_{s(mid)}$ and $M_{r(mid)}$ are dishonest, ask m' from the adversary and output (mid, m') to T.

  The accusations are based on the result of $(\mathbf{reveal}, mid, T)$, and they are defined inside the protocol. Differently from [DGN10], we do not send the accusations immediately in the $\mathcal{F}_{transmit}$ definition.

  The real implementation of this simplified $\mathcal{F}_{transmit}$ is very simple. The sender signs the *mid* and the contents. If the receiver has not received a message from the sender, or has received a message of wrong form (for example, without the signature), it sends a complaint to everyone. Now the sender has another attempt, but now its message has to be broadcast to all the other parties, so everyone may verify its correctness and send the message further to the receiver, or immediately accuse the sender. In this way, 3 additional rounds are introduced in case of a problem. Since all the messages are signed by the sender, the contents of the transmitted message can be revealed also later. Straightforwardly, this can be done only by the receiver (since only the sender signs the messages), but it is sufficient in our settings, since the sender may still reveal the message by publicly forcing the receiver to present the signatures, and if the receiver refuses to do it, it is claimed guilty. This implementation of $\mathcal{F}_{transmit}$ can be easily extended to broadcasting the message to several parties, just using broadcast channel and defining multiple receivers in $r(mid)$.

- $\mathcal{F}_{ver}$ implements all the algorithms of Linear PCP for verifiable computation (for a certain pre-agreed field $\mathbb{F}$ in which all the arithmetic circuits are implemented). This functionality is accessed by each party individually, and it does not involve any communication. $\mathcal{F}_{ver}$ reacts to the following commands.
  - $(\mathbf{proof}, \mathbf{v}, \mathbf{z}, \mathbf{x}, \mathbf{r}, \mathbf{c}, C_j)$: Given the arithmetic circuit and the set of all the public/intermediate/input/randomness/communication variables, in constructs a proof $(\pi||\mathbf{d})$ where $\mathbf{d} = (\mathbf{x}||\mathbf{r}||\mathbf{c})$ corresponds to committed values and π to everything else.
  - $(\mathbf{challenge}, \tau, C_j)$: Given a randomness τ and a circuit C_j, it returns challenge vectors $\mathbf{q_1}, \ldots, \mathbf{q_5}$ such that for any valid proof $(\pi||\mathbf{d})$ generated by $(\mathbf{proof}, \mathbf{v}, \mathbf{z}, \mathbf{x}, \mathbf{r}, \mathbf{c}, C_j)$ for $\mathbf{d} = (\mathbf{x}||\mathbf{r}||\mathbf{c})$, the answer to the query $(\mathbf{verify}, C_j, \tau, \mathbf{v}, \langle(\pi||\mathbf{d}), \mathbf{q_1}\rangle, \ldots, \langle(\pi||\mathbf{d}), \mathbf{q_5}\rangle)$ is true with probability 1.
  - $(\mathbf{verify}, C_j, \tau, \mathbf{v}, a_1, \ldots, a_5)$: Given a randomness τ and some values $\mathbf{v}$ that should be public, check if $a_1, \ldots, a_5$ indeed correspond to a valid proof with respect to τ. A false proof may be accepted with probability $\frac{2m}{|\mathbb{F}|}$, where m is the number of multiplication gates in the circuit.

Transition Function for $\mathcal{F}_{ideal}$

The ideal functionality is secure against an ideal adversary $\mathcal{A}_S$. Let $\mathcal{M} = \{1, \ldots, n\}$ be the set of all parties.

- In the beginning, $\mathcal{F}_{ideal}$ gets from the environment all the arithmetic circuits $(\mathbf{init}, C_1^1, \ldots, C_n^r)$, where C_i^k corresponds to the computation of M_i on the k-th round. All C_i^k are sent to $\mathcal{A}_S$.
- If $\mathcal{F}_{ideal}$ receives $(\mathbf{corrupt}, i)$ from $\mathcal{A}_S$, it sets $\mathsf{evil}_i := \mathsf{true}$.
- A_S may send $(\mathbf{stop}, i_1, \ldots, i_\ell)$ to $\mathcal{F}_{ideal}$ for any corrupted parties $i_1, \ldots, i_\ell$. $\mathcal{F}_{ideal}$ sets $malicious[i_\ell] := 1$ for all i_ℓ. This models straightforward cheating in the initialization phase with the input and the randomness signatures.

- First, $\mathcal{F}_{ideal}$ computes all the messages of the honest parties of the next round by itself, based on the messages received in the previous rounds. If $\mathsf{evil_j} == \mathsf{true}$, then for any message $\mathbf{m}_k^{ij}$, $\mathcal{F}_{ideal}$ does the following:
  - Sends $\mathbf{m}_{ij}^k$ to $\mathcal{A}_S$.
  - Waits for $\mathbf{m}^{*k+1}_{ji_1}, \ldots, \mathbf{m}^{*k+1}_{ji_{|\mathcal{R}|}}$ from $\mathcal{A}_S$ for all the *honest* receivers $\mathcal{R}$. The communication between dishonest parties is not important at the moment. It should just be consistent in the proofs of the sender and the receiver, and that check will be performed in the end.
  - At any round, instead of $\mathbf{m}^{*k}_{ij}$, A_S may send $(\mathbf{stop}, i_1, \ldots, i_\ell)$ to $\mathcal{F}_{ideal}$ for any corrupted parties $i_1, \ldots, i_s$. $\mathcal{F}_{ideal}$ sets $malicious[i_\ell] := 1$ for all i_ℓ. This models straightforward cheating in the initial protocol that will be immediately detected.
- Upon receiving $(\mathbf{run}, \mathbf{x}_1, \ldots, \mathbf{x}_n)$ from the environment, $\mathcal{F}_{ideal}$ computes $C_i^1(\mathbf{x}_i)$ for all honest $i \in \mathcal{M}$, getting the vectors of values $\mathbf{m}_{ij}^1$ that will be used in the next round. On each round, it asks the adversary for the next values, as shown above (initially, it gets from the adversary the inputs $\mathbf{x}_i^*$ of malicious parties). All the $\mathbf{m}_{ij}^{k+1}$ that are computed by the functionality itself are computed based on the messages $\mathbf{m}^{*k}_{ij}$ that have been sent by the adversary and hence may be malicious. This ensures that no honest party will be accused just because someone else has sent to it wrong values.
- After the protocol has finished, $\mathcal{F}_{ideal}$ queries from the adversary all the remaining communication $\mathbf{m}^{*k}_{ji_1}, \ldots, \mathbf{m}^{*k}_{ji_{|\mathcal{R}|}}$ that corresponds to communication between dishonest parties. Then it may compute all the missing $\mathbf{m}_{k+1}^{ji}$ values. If some $\mathbf{m}^{*k+1}_{ji} \neq \mathbf{m}_{ji}^{k+1}$, set $malicious[j] := 1$. Here we assume that even if the computation requires some non-deterministic input, the communicated values are finally deterministic, and the non-determenism is just auxiliary. If is possible to make the security stronger by forcing $\mathcal{A}_S$ to decide on all the $\mathbf{m}^{*k+1}_{ji_1}, \ldots, \mathbf{m}^{*k+1}_{ji_{|\mathcal{R}|}}$ already during the computation, but this will require more communication, since all the shares of $\mathbf{c}_{ij}^\ell$ will have to be distributed already during the protocol execution.
- $\mathcal{F}_{ideal}$ also queries from the adversary a set of messages of the form $(\mathbf{corrupt}, i, j)$ for some i such that $\mathsf{evil_i} == \mathsf{true}$ and some j such that $\mathsf{evil_j} == \mathsf{false}$ (it may choose whether to send them or not). For all i such that $malicious[i] == 1$, the messages should be definitely sent to all j such that $\mathsf{evil_j} == \mathsf{false}$.
- After all the $(\mathbf{corrupt}, i, j)$ messages have been distributed, $\mathcal{A}_S$ sends to $\mathcal{F}_{ideal}$ the outputs $\mathbf{y_j^*}$ of malicious parties M_j. For the honest parties M_i, $\mathcal{F}_{ideal}$ outputs the real output $\mathbf{y_i}$ iff no $malicious[j] := 1$ has ever happened for any $j \in \mathcal{M}$, and otherwise it outputs $(\mathbf{output}, j_1, \ldots, j_k)$ such that for each j_ℓ the messages $(\mathbf{corrupt}, j_\ell, i)$ have been sent for at least t parties i.

Transition Function for M_i

We assume that each M_i maintains the current round in its state, so that in each round it will react just to those calls that are related to the current round. We assume that the number of rounds in the initial protocol is r.

- In the beginning, each M_i gets from the environment all the parts of the entire arithmetic circuit: $(\mathbf{init}, C_1^1, \ldots, C_n^r)$, where C_j^k corresponds to the computation of party M_j on the k-th round. All the C_j^k are also sent to $\mathcal{A}$.
- If any party M_i receives $(\mathbf{rand}, i)$ from environment, it sets $\mathsf{rand} := \mathsf{true}$. This means that given party will participate in randomness generation. The environment defines exactly t such parties.
- If any party M_i receives $(\mathbf{corrupt}, i)$ from $\mathcal{A}$, it sets $\mathsf{evil} := \mathsf{true}$.
- If $\mathsf{evil} == \mathsf{true}$, then upon receiving any message X, M_i does the following:
  - Sends X to $\mathcal{A}$.
  - Waits from $\mathcal{A}$ which messages should be sent to the receivers further, and in which way they should be shared and signed.
- Upon receiving $(\mathbf{preprocess}, \mathbf{x}_i)$ from the environment, the following happens.

- M_i represents its input $\mathbf{x}_i$ as $\binom{n-1}{t}$ distinct sums of the form $\mathbf{x}_i = \sum_{k \in T} \mathbf{x}_{ikT}$ for each subset T of $\mathcal{M} \setminus \{i\}$ of size t. Let $T_1^k, \ldots, T_p^k$ be all the sets to which M_k belongs. For each $k \neq i$, M_i generates a signature

$$sx_{ik} = Sign_{sk_i}(\textbf{input\_shares}, i, k, \mathbf{x}_{ikT_1^k}, \ldots, \mathbf{x}_{ikT_p^k}) \ ,$$

  and sends $(\textbf{transmit}, (\textbf{input\_shares}, i, k), (\mathbf{x}_{ikT_1^k}, \ldots, \mathbf{x}_{ikT_p^k}, sx_{ik}))$ to $\mathcal{F}_{transmit}$.
- If $\mathsf{rand} == \mathsf{true}$, M_i generates the randomness $\mathbf{r}_{ij}$ for each M_j. M_i represents its $\mathbf{r}_{ij}$ as $\binom{n-1}{t}$ distinct sums of the form $\mathbf{r}_{ij} = \sum_{k \in T} \mathbf{r}_{ijkT}$. For each $k \neq i$, M_i generates a signature

$$sr_{ijk} = Sign_{sk_i}(\textbf{rand\_shares}, i, j, k, \mathbf{r}_{ijkT_1^k}, \ldots, \mathbf{r}_{ijkT_p^k}) \ .$$

  Then M_i sends $(\textbf{transmit}, (\textbf{rand}, i, j), (\mathbf{r}_{ij1T_1^1}, \ldots, \mathbf{r}_{ijnT_p^n}, sr_{ij1}, \ldots, sr_{ijn}))$ to $\mathcal{F}_{transmit}$.
- Upon receiving all the $(\textbf{transmit}, (\textbf{rand}, i, j), (\mathbf{r}_{ij1T_1^1}, \ldots, \mathbf{r}_{ijnT_p^n}, sr_{ij1}, \ldots, sr_{ijn}))$, M_j checks if all the shares indeed correspond to the signatures, and if the shares that correspond to different sets T indeed all sum up to the same value. If everything is correct, M_j in turn creates the signatures $Sign_{sk_j}(sr_{ijk})$. For each k, it sends

$$(\textbf{transmit}, (\textbf{rand\_shares}, i, j, k), (\mathbf{r}_{ijkT_1^k}, \ldots, \mathbf{r}_{ijkT_p^k}, Sign_{sk_j}(sr_{ijk})))$$

  to $\mathcal{F}_{transmit}$.
  If something is wrong, $(\textbf{reveal}, (\textbf{rand}, i, j), \mathcal{M})$ is sent to $\mathcal{F}_{transmit}$, everyone checks the signatures, and the protocol immediately stops since one malicious party is detected (either M_i or M_j). Each honest party writes $malicious[i] := 1$ or $malicious[j] := 1$ for each cheated party, and goes to the last step of the protocol (the outputs).
- Upon receiving all $(\textbf{transmit}, (\textbf{input\_shares}, i, k), (\mathbf{x}_{ikT_1^k}, \ldots, \mathbf{x}_{ikT_p^k}, sx_{ik}))$ and $(\textbf{transmit},$ $(\textbf{rand\_shares}, i, j, k), (\mathbf{r}_{ijkT_1^k}, \ldots, \mathbf{r}_{ijkT_p^k}, Sign_{sk_j}(sr_{ijk})))$ from $\mathcal{F}_{transmit}$, M_k checks if the signatures are valid and correspond to the shares. If somthing is wrong, $(\textbf{reveal}, mid, \mathcal{M})$ is sent to $\mathcal{F}_{transmit}$ for each wrong message mid, everyone checks the signatures, and the protocol immediately stops since one malicious party is detected (either the sender M_i was inded wrong or M_k accused it without reason). Each honest party writes $malicious[i] := 1$ or $malicious[k] := 1$ for each cheated party, and goes to the last step of the protocol.

• Upon receiving $(\textbf{run}, i)$ from the environment, M_i computes $C_i^1(\mathbf{x}_i, \mathbf{r}_i)$, getting the vectors of values $\mathbf{c}_{ij}^1$ that should be sent to M_j (for each receiver M_j). We will now generalize the behaviour of M_i to an arbitrary round ℓ. Let $\mathcal{R}$ be the set of all the receivers of M_i for the round ℓ. Let $\mathbf{c}_{ij}^\ell$ be vector of values sent by M_i to M_j on the round ℓ. These values are handled similarly to the randomness values, with the difference that the shares are not distributed yet, just their signatures.
  - Generate the shares $\mathbf{c}_{ij}^\ell = \sum_{k \in T} \mathbf{c}_{ijkT}^\ell$.
  - Generate the signature $sc_{ij}^\ell = Sign_{sk_i}(\mathbf{c}_{ij}^\ell)$.
  - For each $k \neq i$, M_i generates a signature

$$sc_{ijk}^\ell = Sign_{sk_i}(\textbf{message\_shares}, \ell, i, j, k, \mathbf{c}_{ijkT_1^k}^\ell, \ldots, \mathbf{c}_{ijkT_p^k}^\ell) \ .$$

  - M_i sends $(\textbf{transmit}, (\textbf{message}, \ell, i, j), (\mathbf{c}_{ij}^\ell, sc_{ij}^\ell, sc_{ij1}^\ell, \ldots, sc_{ijn}^\ell))$ to $\mathcal{F}_{transmit}$
  - Upon receiving all the $(\textbf{transmit}, (\textbf{message}, \ell, i, j), (\mathbf{c}_{ij}^\ell, sc_{ij}^\ell, sc_{ij1}^\ell, \ldots, sc_{ijn}^\ell))$ for the current round ℓ, M_j checks if $\mathbf{c}_{ij}^\ell$ indeed corresponds to the signature sc_{ij}^ℓ. If everything is correct, M_j in turn creates the signatures $Sign_{sk_j}(sc_{ijk}^\ell)$. For each k, it sends

$$(\textbf{transmit}, (\textbf{message\_shares}, \ell, i, j, k), Sign_{sk_j}(sc_{ijk}^\ell))$$

    to $\mathcal{F}_{transmit}$.
  - If something is wrong inside some message, the corresponding mid is revealed. At the moment no one can check to which shares the signatures $Sign_{sk_j}(sc_{ijk}^\ell))$ actually correspond. This check will be done later.

- During the verification phase, M_i acts at once as a prover, and as a verifier in each of the p subsets for each of the other $n-1$ parties. First, each subset T agrees on its own $\mathbf{q}_{1Tj}, \ldots, \mathbf{q}_{5Tj}$ for each prover M_j. For each subset T that is verifying honestness of some party M_j, $M_i \in T$ generates random τ_{Tij} and sends all $(\mathbf{bc}, T, (i, j, T, \tau_{Tij}, Sign_{sk_i}(\tau_{Tij})))$ to $\mathcal{F}_{bc}$. If some party sends an incorrect signature, or refuses to participate, any verifier of T is allowed to broadcast $(\mathbf{bc}, \mathcal{M}, (\mathbf{end}, j, T))$. Since either the signer or the sender is guilty, every honest party may now believe that continuing this T-set proof is senseless since it contains at least one malicious verifier, and the prover M_j is considered to have passed it.
- Upon receiving all $(\mathbf{bc}, T, (k, j, T, \tau_{Tkj}, Sign_{sk_k}(\tau_{Tkj})))$ from $\mathcal{F}_{bc}$, M_i sums up all the received τ_{Tkj} with its own τ_{Tij}. For each T, j, this will give a unique randomness $\tau_{T,j}$ (since at least one party is honest, it is indeed distributed uniformly and randomly). M_i sends $(\mathbf{challenge}, \tau_{T,j}, C_j)$ to $\mathcal{F}_{ver}$ and gets back $\mathbf{q}_{1Tj}, \ldots, \mathbf{q}_{5Tj}$.
- As a prover, each M_i sends $(\mathbf{proof}, \mathbf{v}_i, \mathbf{z}_i, \mathbf{x}_i, \mathbf{r}_i, \mathbf{c}_i, C_i)$ to $\mathcal{F}_{ver}$ and receives back a vector $(\pi_i || \mathbf{d}_i)$ with $\mathbf{d}_i = (\mathbf{x}_i || \mathbf{r}_i || \mathbf{c}_i)$ that contains all the necessary proof of C_j computation. For the prover's security, it is preferable to use new randomness in π_i in different proofs, so M_i generates a distinct π_{iT} for each subset T of the verifiers. M_i generates the shares such that $\pi_{iT} = \sum_{k \in T} \mathbf{p}_{ikT}$. It generates the signatures $sp_{ik} = Sign_{sk_i}(\mathbf{proof\_share\_1}, i, k, \mathbf{p}_{ikT_1^k}, \ldots, \mathbf{p}_{ikT_p^k})$. For each $k \in T$, M_i sends $(\mathbf{transmit}, (\mathbf{proof\_share\_1}, i, k), (\mathbf{p}_{ikT_1^k}, \ldots, \mathbf{p}_{ikT_p^k}, sp_{ik}))$ to $\mathcal{F}_{transmit}$.
- As a sender, M_i has to finally distribute amongst the corresponding verifiers all the shares $\mathbf{c}_{ijkT} = (\mathbf{c}^1_{ijkT} || \ldots || \mathbf{c}^r_{ijkT})$ for all the messages it has sent in the initial protocol. Here M_i has to broadcast them in such a way that one copy is sent to the receiver M_j, so that M_j may verify if the shares indeed correspond to $Sign_{sk_i}(\mathbf{c}^1_{ij}), \ldots, Sign_{sk_i}(\mathbf{c}^r_{ij})$ that it has already received during the computation. It generates a signature $sc_{ijk} = Sign_{sk_i}(sc^1_{ijk}, \ldots, sc^r_{ijk})$ from the same signatures sc^ℓ_{ijk} that he has already sent on the round ℓ. M_i sends $(\mathbf{transmit}, (\mathbf{proof\_share\_2}, i, j, k), (\mathbf{c}_{ijkT_1^k}, \ldots, \mathbf{c}_{ijkT_p^k}, sc_{ijk}))$ to $\mathcal{F}_{transmit}$, which knows that the message with such a mid should be broadcast to both j and k at once. Such a message is transmitted for each $k \neq i$.
- Upon receiving all $(\mathbf{transmit}, (\mathbf{proof\_share\_2}, i, j, k), (\mathbf{c}_{ijkT_1^k}, \ldots, \mathbf{c}_{ijkT_p^k}, sc_{ijk}))$ from $\mathcal{F}_{transmit}$, the corresponding receiver M_j checks if the shares correspond to the signatures that it has collected during the initial protocol computation. If at least one of the shares is wrong, then M_j sends $(\mathbf{reveal}, mid, \mathcal{M})$ for all the shares provided by M_i (not only the incorrect ones), and additionally broadcasts $(\mathbf{reveal}, (\mathbf{message}, \ell, i, j), \mathcal{M})$ for the corresponding round ℓ, so that everyone may compute the incorrect segment of $\mathbf{c}_{ij}$ from the shares and verify the signatures. All the parties may now decide whether the sender or the receiver is wrong.
  - If the receiver M_j is guilty:
    * The proof of M_j ends with failure, and each honest party sets $malicious[j] := 1$.
    * The proof of M_i continues as before, on the assumption that the committed communication is the shared one.
  - If the sender M_i is guilty:
    * The proof of M_i ends with failure, and each honest party sets $malicious[i] := 1$.
    * The verification for M_j continues on the assumption that $Sign_{sk_i}(\mathbf{c_{ij}})$ are the committed values, since it is the sender who has provided wrong signatures.
- Upon receiving all the

$$(\mathbf{transmit}, (\mathbf{proof\_share\_1}, i, k), (\mathbf{p}_{ikT_1^k}, \ldots, \mathbf{p}_{ikT_p^k}, sp_{ik}))$$

  and all the

$$(\mathbf{transmit}, (\mathbf{proof\_share\_2}, i, j, k), (\mathbf{c}_{ijkT_1^k}, \ldots, \mathbf{c}_{ijkT_p^k}, sc_{ijk}))$$

  from $\mathcal{F}_{transmit}$ from all the provers and all the senders, M_i verifies if all the shares are consistent with the signatures it has collected during the initial protocol computation. If something is wrong, $(\mathbf{reveal}, mid, \mathcal{M})$ is sent for all the inconsistent shares and signatures, so all the parties can check them. Each sender of the wrong messages M_i is malicious, and its proof ends with $malicious[i] := 1$. If some receiver M_j does not complain about wrong shares, then it is also malicious, and its proof also ends with $malicious[j] := 1$ (this check can be actually performed already on the next round, then the other parties will not have to wait until M_j complains).

Let $\mathbf{d}_{ikT}$ denote the concatenation of all the shares of $\mathbf{x}_i$, $\mathbf{r}_i$, and $\mathbf{c}_i$ that are intended for the verifier M_k and the prover M_i, when it participates in the proof set T. More precisely, since each $\mathbf{r}_i = \mathbf{r}_{1i} + \ldots + \mathbf{r}_{ti}$, we may more formally define $\mathbf{d}_{ikT} = (\mathbf{x}_{ikT}||(\mathbf{r}_{1ikT} + \ldots + \mathbf{r}_{tikT})||\mathbf{c}_{ijkT})$. They are distributed in such a way that $\sum_{k \in T} \mathbf{d}_{ikT} = \mathbf{d}_i$.

If all the signatures have been correct, and no receiver has complained that the shares are wrong, M_k sends $(\mathbf{bc}, \mathcal{M}, (\mathbf{product\_share}, i, j, T, \langle(\mathbf{p}_{ikT}||\mathbf{d}_{ikT}), \mathbf{q}_{1Ti}\rangle, \ldots, \langle(\mathbf{p}_{ikT}||\mathbf{d}_{ikT}), \mathbf{q}_{5Ti}\rangle))$ to $\mathcal{F}_{bc}$.

- Upon receiving all the scalar products for all parties in T from $\mathcal{F}_{bc}$ for the prover M_j, each party M_i in T broadcasts $(\mathbf{bc}, \mathcal{M}, (\mathbf{publish\_challenge}, k, T, \tau_{T1j}, \ldots, \tau_{Tpj}, Sign_{sk_1}(\tau_{T1j}), \ldots, Sign_{sk_p}(\tau_{Tpj})))$.
- Upon receiving all **publish_challenge** messages, M_i selects the one in which all the signatures are correct (at least one should be since at least one party is honest). If there are several valid signatures, then the signer of multiple values M_k is definitely malicious and should be punished, so the protocol ends with $(\mathbf{bc}, \mathcal{M}, (\mathbf{end}, i, T))$, and the honest parties assume that M_i has passed the test since one of the verifiers was dishonest, so each honest party sets $malicious[k] := 1$ for each multiple signer M_k. After the correct τ is found, M_i verifies all the previously published scalar products. For any incorrect scalar product computed by some M_k, it sends $(\mathbf{reveal}, (\mathbf{proof\_share\_1}, j, i), \mathcal{M})$ and $(\mathbf{reveal}, (\mathbf{proof\_share\_2}, j, i, k), \mathcal{M})$ to $\mathcal{F}_{transmit}$, so now everyone in M may verify the signatures and compute the corresponding scalar products by itself.
- Each party M_i in $\mathcal{M}$ sums up the appropriate accepted scalar products (it may compute the scalar products for the published shares by itself), gets $a_1, \ldots, a_5$, and sends $(\mathbf{verify}, C_j, \tau, \mathbf{v}, a_1, \ldots, a_5)$ to $\mathcal{F}_{ver}$. If the answer is 1, then M_i accepts the proof of M_j for the given T. Otherwise it immediately sets $malicious[j] := 1$.
- After the protocol has finished for all T, j proofs, each M_i sees for which parties it has set $malicious[j] = 1$. If there is at least one party that is malicious, an honest M_i outputs $(\mathbf{output}, j_1, \ldots, j_k)$ for all parties j_ℓ that are malicious.

Transition Function for the Simulator S

In this subsection we prove that the real functionality is as secure as the ideal functionality. The S is located between $\mathcal{F}_{ideal}$ and $\mathcal{A}$, and it tries to convince $\mathcal{A}$ that it is a real functionality. At the same time for $\mathcal{F}_{ideal}$ it must be like if it communicated with an ideal adversary $\mathcal{A}_S$ who is not completely evil.

Since S simulates the communication between $\mathcal{A}$ and all the M_i, it should know their communication keys for $\mathcal{F}_{ppp}$ for all the M_i.

- In the beginning, $\mathcal{F}_{ideal}$ gets from the environment all the arithmetic circuits $(\mathbf{init}, C_1^1, \ldots, C_p^r)$, where C_i^k corresponds to the computation of party M_i the k-th round. All C_i^k are sent to S. The S just delivers them to $\mathcal{A}$.
- If S receives $(\mathbf{corrupt}, i)$ from $\mathcal{A}$ (that was intended for M_i), it sends $(\mathbf{corrupt}, i)$ to $\mathcal{F}_{ideal}$. $\mathcal{F}_{ideal}$ sets $\mathsf{evil}_i := \mathsf{true}$. The inputs of malicious parties are delivered by S to $\mathcal{A}$.
- In the real protocol, the parties should commit the input shares and generate the randomness. The $\mathcal{A}$ may propose its own inputs and signatures for dishonest parties. S checks by itself the signatures, as if it was an honest party. If anything is wrong, S sends $(\mathbf{stop}, i_1, \ldots, i_\ell)$ to $\mathcal{F}_{ideal}$, for all the detected parties. If the receiver is malicious, then the adversary decides whether anything should be revealed or not.
- From the name of honest parties, S should generate the shares of the randomness and the inputs by itself. Since the adversary gets up to $t-1$ shares of each value, and $t-1$ look completely random unless the last one is obtained, S may generate completely random shares, and they will not be inconsistent with the randomness and the inputs that should have been provided for the honest parties by $\mathcal{F}_{ideal}$. S signs all the shares by itself.
- For the dishonest parties M_i, $\mathcal{F}_{ideal}$ generates $\mathbf{r}_i$ by itself and shows it to the adversary. Here $\mathbf{r}_i$ should indeed be random. S needs to enforce the same $\mathbf{r}_i$ to be used in the real model. $\mathcal{A}$ may generate up to $t-1$ shares for the parties that it controls, but at least one share is generated by an honest party. From the name of the remaining honest parties M_j, S generates randomness $\mathbf{r}_{ij}$

in such a way that the sum of all the t shares equals $\mathbf{r}_i$ provided by the ideal functionality. If $\mathcal{A}$ has provided inconsistent shares for some $\mathbf{r}_{ki}$, the shares of $\mathbf{r}_{ji}$ should nevertheless be consistent. Hence S has to achieve $\mathbf{r}_i$ only for at least one all-honest t-set, and the others do not matter. Let that honest t-set be denoted H. S will now assume that H holds all the committments, just to avoid confusion using several all-honest t-sets at once.

If we do not want to use the random oracle assumption, recovering $\mathbf{r}_{ki}$ from the shares is possible only if S receives the shares, not just the signatures (and in H all the shares indeed correspond to the signatures, so there are no contradictions). Since committing the inputs is just a preprocessing phase, the shares of $\mathbf{r}_i$ are distributed immediately.

- $\mathcal{F}_{ideal}$ starts running the protocol, and it immediately waits for the input $\mathbf{x}^*{}_i$ that should be queried from $\mathcal{A}_S$. S should take H and define $\mathbf{x}^*{}_i$ to be the vector committed to H as shares. This will be considered the proper committed input. If the shares of $\mathbf{x}^*{}_i$ are not distributed already in the beginning, there is no way for S to recover $\mathbf{x}^*{}_i$ from the signatures, it would only be possible in the random oracle model. Hence if we want to avoid this assumption, the shares should be distributed immediately in the preprocessing phase.
- At some moment $\mathcal{F}_{ideal}$ reaches the place where some ${\mathbf{m}^*}^k_{ji}$ should be queried.
  - First, $\mathcal{F}_{ideal}$ computes all the messages of the next round by itself, based on the messages received in the previous rounds. If $\mathsf{evil_j} == \mathsf{true}$, then for any message $\mathbf{m}^k_{ij}$, $\mathcal{F}_{ideal}$ does the following:
    - Sends $\mathbf{m}^k_{ij}$ to S. For the honest parties, S composes all the necessary shares of this message by itself, and signs them from the name of M_i. It delivers the message shares to $\mathcal{A}$ through $\mathcal{F}_{ppp}$, pretending to be M_i.
    - $\mathcal{F}_{ideal}$ waits for ${\mathbf{m}^*}^{k+1}_{ji_1}, \ldots, {\mathbf{m}^*}^{k+1}_{ji_{|\mathcal{R}|}}$ from S for all the receivers $\mathcal{R}$. At the same time, $\mathcal{A}$ knows that in the real functionality M_i should wait for the shares and the signatures. It sends all the shares and the signatures to S, and it gives the orders to the malicious parties, which values they should sign by themselves. For each receiver i_ℓ, the signatures are not supposed to be valid and correspond to the message itself. From the name of honest receivers, S sends $(\mathbf{reveal}, mid, \mathcal{M})$ for all the messages that the honest parties would indeed reject, and since rejection is related just to checking the signatures of the malicious parties, S is able to do it itself. S sends $(\mathbf{stop}, i_1, \ldots, i_\ell)$ to $\mathcal{F}_{ideal}$, for all the detected parties of that round. If both the sender and the receiver are malicious, then the adversary decides whether the message should be accepted.
      Now S has to decide what the ${\mathbf{m}^*}^{k+1}_{ij}$ should be, since $\mathcal{F}_{ideal}$ is waiiting for it from $\mathcal{A}_S$.
      - Since each honest party uses $\mathbf{c}^k_{ij}$ it has received, and this value corresponds to the signature that each honest party presents to defend itself, S sets ${\mathbf{m}^*}^{k+1}_{ij} = \mathbf{c}^k_{ij}$ for all honest receivers M_j.
      - For the dishonest receivers, S still takes ${\mathbf{m}^*}^{k+1}_{ij} = \mathbf{c}^k_{ij}$ for all the honest senders since then $\mathbf{c}^k_{ij}$ indeed corresponds to both the shared and the non-shared committment.
      - If both the sender and the receiver are dishonest, then then $\mathcal{F}_{ideal}$ does not wait for ${\mathbf{m}^*}^{k+1}_{ij}$ since it does not need it in the next computation (the outputs of M_j in the next round are anyway defined by the adversary). $\mathcal{F}_{ideal}$ asks for these values later.

      Hence for the communications where at least one party is honest, ${\mathbf{m}^*}^{k+1}_{ij} = \mathbf{c}^k_{ij}$ is sent to $\mathcal{F}_{ideal}$.
- After the simulation of the initial protocol has finished, $\mathcal{F}_{ideal}$ queries from the adversary a set of messages of the form $(\mathbf{corrupt}, i, j)$ for some i such that $\mathsf{evil_i} == \mathsf{true}$ and some j such that $\mathsf{evil_j} == \mathsf{false}$. It also waits for the final decision on ${\mathbf{m}^*}^1_{ij}, \ldots, {\mathbf{m}^*}^r_{ij}$ for malicious parties
  - As a verifier, $\mathcal{A}$ chooses the share τ_{Tij} for each corrupted i, for each other j who acts as a prover, for each verifier subset T. It sends these values to S. Now S may broadcast all $(\mathbf{bc}, T, (k, j, T, \tau_{Tkj}, Sign_{sk_i}(\tau_{Tij})))$, simulating the random shares of the remaining honest parties by itself. If $\mathcal{A}$ says that some M_k refuses to participate or presents incorrect signatures for M_j, S broadcasts $(\mathbf{bc}, \mathcal{M}, (\mathbf{end}, j, T))$ from the names of all honest parties in T, assuming that the prover M_j is honest. If there are no problems, S may send all $(\mathbf{challenge}, \tau_{Tj}, C_j)$ to $\mathcal{F}_{ver}$ and get back all $\mathbf{q}_{1Tj}, \ldots, \mathbf{q}_{5Tj}$.

– As a prover, for each T, $k \in T$, $\mathcal{A}$ selects

$$(\mathbf{transmit}, (\mathbf{proof\_share\_1}, i, k), (\mathbf{p}_{ikT_1^k}, \dots, \mathbf{p}_{ikT_p^k}, sp_{ik}))$$

for M_k from the name of malicious M_i. As a sender, it selects

$$(\mathbf{transmit}, (\mathbf{proof\_share\_2}, i, j, k), (\mathbf{c}_{ijkT_1^k}, \dots, \mathbf{c}_{ijkT_p^k}, sc_{ijk}))\ .$$

S waits until all of them come, or until timeout (since $\mathcal{A}$ may force some M_i to refuse to participate in verification). Then, from the name of each honest party, S checks if the shares correspond to the signatures. If all the signatures are correct, S sends

$$(\mathbf{bc}, \mathcal{M}, (\mathbf{product\_share}, i, k, T, \langle(\mathbf{p}_{ikT}||\mathbf{d}_{ikT}), \mathbf{q}_{1Ti}\rangle, \dots, \langle(\mathbf{p}_{ikT}||\mathbf{d}_{ik1}), \mathbf{q}_{5Ti}\rangle))$$

to $\mathcal{F}_{bc}$, where $\mathbf{d}_{ik1}$ is constructed from the received values, as in description of M_i. If some signature is wrong, S sends $(\mathbf{reveal}, mid, \mathcal{M})$ to $\mathcal{F}_{transmit}$ for a corresponding mid. From each honest M_i, S broadcasts $(\mathbf{bc}, \mathcal{M}, (\mathbf{end}, j, T))$, and stores $(\mathbf{corrupt}, j, i)$. For a malicious receiver, the adversary decides whether it reveals the message or not.

– From the side of the honest provers, the verifier should generate all the shares of the proof by itself. All the inner communication shares between honest parties, and all the input/randomness shares that have never been seen by S, are generated randomly and signed by S from the names of corresponding honest parties. Since in any T there are at most $t-1$ dishonest parties, the adversary sees only up to $t-1$ shares which look completely random unless the last t-th share is obtained, and hence there are no inconsistencies with any real proof that S has never seen.

– The $\mathcal{A}$ decides on the scalar products for dishonest parties. The S just broadcasts these values from the names of the corresponding parties. If $\mathcal{A}$ decides that some party M_j refuses to broadcast, S simulates the end of the set T proof, writing $(\mathbf{corrupt}, j, k)$ for all honest parties M_k.

– From the name of each honest party M_i, S has to generate scalar products by itself. The problem is that the sum of final shared scalar products should be equal to the real $\langle(\pi_{iT}||\mathbf{d}_{iT}), \mathbf{q}_{kTi}\rangle$. If $(\pi_{iT}||\mathbf{d}_{iT})$ belongs to an honest verifier, then it cannot be generated by S itself. However, it is known that, for an honest verifier, $\langle(\pi_{iT}||\mathbf{d}_{iT}), \mathbf{q}_k\rangle$ is statistical HVZK (the details can be seen in [BSCG+13]). Since S knows τ, it has access to a trapdoor, and it has enough information about how to generate $a_1, \dots, a_5$ in such a way that the final proof on certain combinations of a_k would succeed, and their distribution is the same as for real proof. Hence S just generates some random a_k that satisfy this proof, and claims that $a_k = \langle(\pi_{iT}||\mathbf{d}_{iT}), \mathbf{q}_{kTi}\rangle$. Let the shares of the evil parties in the corresponding T-set be $\mathbf{s}_1, \dots, \mathbf{s}_\ell$ for $\ell \le t-1$. Their sum must now be equal to some vector $\mathbf{s}$ such that $r_1 + \langle\mathbf{s}, \mathbf{q}_{1Ti}\rangle = a_1, \dots, r_5 + \langle\mathbf{s}, \mathbf{q}_{5Ti}\rangle = a_5$ where r_i are the sums of scalar products of the honest parties. S just has to distribute all $\mathbf{s}_i$ shares uniformly amongst the evil parties, and compute r_k according to a_k. Since in the real functionality the prover is not supposed to use the same randomness in π several times, π_{iT} is being generated as a completely new random value in each proof separately.

– If the protocol has not ended yet for some T, j, there is at least one party that has broadcasted all the necessary scalar product shares, since by assumption at least one party is honest. However, some of the published scalar product shares may nevertheless be malicious. Now the shares of τ_{Tj} and their signatures have to be broadcasted. S does that from the side of all the honest parties, and it waits until $\mathcal{A}$ decides on something for dishonest parties. Then S checks the signatures for inconsistencies from the name of each honest party, and broadcasts $(\mathbf{bc}, \mathcal{M}, (\mathbf{end}, j, T))$ in the case if something is wrong, setting $(\mathbf{corrupt}, \ell, k)$ for all the honest parties M_k for all M_ℓ that provided multiple signatures.

  ∗ If the prover has not been corrupted, then S verifies all the scalar products published by corrupted parties by itself. If some signature is wrong, it sends $(\mathbf{reveal}, mid, \mathcal{M})$ to $\mathcal{F}_{transmit}$ for the corresponding messages. Since a wrong scalar product (or a wrong published $\mathbf{c}_{ijkT}$ share) comes from a corrupted party, the communication shares and the signatures are known by S. As in the real functionality, S simulates the end of T-set proof, storing $(\mathbf{corrupt}, j, k)$ for all honest parties M_k.

  * If the prover M_i has been corrupted, S waits until $\mathcal{A}$ announces which products should be claimed wrong. Again, the values from $(\mathbf{reveal}, mid, \mathcal{M})$ should be published, but S knows them since now they are already known by the malicious prover. S simulates the end of T-set proof, storing $(\mathbf{corrupt}, i, k)$ for all honest parties M_k.

  - From the name of each honest receiver M_i, S has to send a complaint if the revealed communication shares do not correspond to the committed $\mathbf{c_{ij}}$. For the dishonest receivers, the adversary decides which complaints have to be sent. Now the communication $\mathbf{m^*}_{ij}^{k}$ between the honest parties has finally been discovered. If $\mathcal{A}$ decided to reveal $\mathbf{c}_{ij}$, then $(\mathbf{m^*}_{ij}^{1}|| \dots ||\mathbf{m^*}_{ij}^{r}) = \mathbf{c}_{ij}$. Otherwise $(\mathbf{m^*}_{ij}^{1}|| \dots ||\mathbf{m^*}_{ij}^{r})$ is equal to the sum of the shares of H.
  - After all the scalar products have been verified, S makes each honest verifier M_i act exactly like in the real protocol. It computes $a_1, \dots, a_5$ from the shares, and then the certain combinations of these values, and checks if they match, storing $(\mathbf{corrupt}, j, i)$ for each honest party M_i if the proof fails. The decisions of malicious parties are made by the adversary.
  - For each honest verifier M_i, S stores the corresponding counters of how many tests there have been in which each prover M_j succeeds. If there is some party M_j that has not succeeded in all the proofs, it stores $(\mathbf{corrupt}, j, i)$. The adversary sends the decisions of dishonest parties.

- $\mathcal{F}_{ideal}$ is still waiting from the adversary a set of messages of the form $(\mathbf{corrupt}, i, j)$ for some i such that $\mathsf{evil_i} == \mathsf{true}$ and some j such that $\mathsf{evil_j} == \mathsf{false}$ Now S should decide on that. Since all the $(\mathbf{corrupt}, j, i)$ have already been generated, S just delivers them to $\mathcal{F}_{ideal}$. During the simulation, S chose to claim corrupt only in the following cases:
  1. A party may be claimed corrupt in the initial protocol, if its sent shares do not correspond to its signatures. This can be done only by malicious parties.
  2. During the verification process, S corrupts only malicious parties, according to the protocol description.
  3. Additionally, we need to ensure that S has not accused any honest parties in the final check. Since a real prover M_j would never accept the scalar product shares computed by evils unless they are computed correctly, the scalar products that have reached the end of the proof are indeed $a_1, \dots, a_5$, and they have been chosen by S in such a way that the test passes.
  4. The remaining place where honest users could be claimed malicious by $\mathcal{F}_{ideal}$ itself is the inconsistency of $\mathbf{m}_{ij}^{k}$ and $\mathbf{m^*}_{ij}^{k}$, but S delivers malicious messages only from corrupted parties.

- For all i such that $malicious[i] == 1$, the messages should be definitely sent to all j such that $\mathsf{evil_j} == \mathsf{false}$. It is sufficient to show that if an inconsistency of $\mathbf{m}_{ij}^{k}$ and $\mathbf{m^*}_{ij}^{k}$ happens in $\mathcal{F}_{ideal}$, then M_i does not pass the test of H.

  Since passing the test without actually finishing the verification happens only either if some of the verifiers acts dishonestly, or the protocol succeeds up to the final proof, the only way for M_j to pass the test for H is to make $a_1, \dots, a_5$ accepted in the end. Hence it must have succeeded in generating $a_1, \dots, a_5$ that correspond to $a_1 = \langle \mathbf{s}_{11}, \mathbf{q}_1 \rangle + \dots + \langle \mathbf{s}_{1t}, \mathbf{q}_1 \rangle, \dots, a_5 = \langle \mathbf{s}_{51}, \mathbf{q}_5 \rangle + \dots + \langle \mathbf{s}_{5t}, \mathbf{q}_5 \rangle$. Since all the verifiers in that subset are honest, none of them could provide any information about any $\mathbf{q}_k$ to M_j. Since all the verifiers are honest, they use the shares distributed in the initial protocol consistently, and hence $\mathbf{s}_{1k} = \dots = \mathbf{s}_{5k} =: (\mathbf{p}_{ikH} || \mathbf{d}_{ikH})$ for each verifier k, so we have $a_1 = \langle (\mathbf{p}_{i1H} || \mathbf{d}_{i1H}), \mathbf{q}_1 \rangle + \dots + \langle (\mathbf{p}_{itH} || \mathbf{d}_{itH}), \mathbf{q}_1 \rangle, \dots, a_5 = \langle (\mathbf{p}_{i1H} || \mathbf{d}_{i1H}), \mathbf{q}_5 \rangle + \dots + \langle (\mathbf{p}_{itH} || \mathbf{d}_{itH}), \mathbf{q}_5 \rangle$ for valid challenges $\mathbf{q}_k$ that have not been seen by M_j before generating $(\mathbf{p}_1 || \mathbf{d}_1), \dots, (\mathbf{p}_t || \mathbf{d}_t)$. Denoting $\mathbf{p} := \mathbf{p}_{i1H} + \dots + \mathbf{p}_{itH}$ and $\mathbf{d} := \mathbf{d}_{i1H} + \dots + \mathbf{d}_{itH}$, we get $a_1 = \langle (\mathbf{p} || \mathbf{d}), \mathbf{q}_1 \rangle, \dots, a_5 = \langle (\mathbf{p} || \mathbf{d}), \mathbf{q}_5 \rangle$. Since the honest parties would not accept communicated value shares that do not correspond to the signatures, the scalar products are actually of the form $\langle (\pi || \mathbf{x} || \mathbf{r} || \mathbf{c}), \mathbf{q}_1 \rangle, \dots, \langle (\pi || \mathbf{x} || \mathbf{r} || \mathbf{c}), \mathbf{q}_5 \rangle$, where $\mathbf{x}$, $\mathbf{r}$, $\mathbf{c}$ indeed correspond to the values committed to the all-honest parties.

  Even if the other t-sets have received different commitments, it is important that these values have been committed to an all-honest-party set before the computation, according to the protocol. The vector $\mathbf{r}$ is indeed random since all-honest-parties have checked carefully all the signatures of the t generators of $\mathbf{r}$, and at least one of them was definitely honest, and hence the vector is indeed random. If the proof succeeds, then $\mathbf{p}$ is a valid proof (according to the PCP description). This implies proving that M_i performed its communication correctly with respect to the committed values. Here the situation with $\mathbf{c}$ is not so clear since it has been committed in two ways. We need to show that satisfying any of these two committed values by M_i implies all $\mathbf{m^*}_{ij}^{k} = \mathbf{m}_{ij}^{k}$, where $\mathbf{m}_{ij}^{k}$ is computed by $\mathcal{F}_{ideal}$ from the previous round.

  - Since S has chosen $\mathbf{m}^{*k-1}_{ij} = \mathbf{c}^{k-1}_{ij}$ where $\mathbf{c}^{k-1}_{ij}$ is accepted by H (it does not matter now which of the two committments it was), the value $\mathbf{m}^{ij}_k$ equals to the value computed from $\mathbf{c}^{k-1}_{ij}$, the inputs $\mathbf{x}_j$, and the randmoness $\mathbf{r}_j$ which have been committed to H.
  - Since the proof has succeeded, the computation is correct with respect to $\mathbf{c}^{k-1}_{ij}$, the inputs $\mathbf{x}_j$, and the randmoness $\mathbf{r}_j$ committed to H. Hence the output of this computation $\mathbf{m}^{*k}_{ij}$ is the same value $\mathbf{m}^k_{ij}$ that $\mathcal{F}_{ideal}$ would compute.

- After all the $(\mathbf{corrupt}, i, j)$ messages have been distributed, S waits from $\mathcal{A}$ the final outputs of the dishonest parties. It delivers them to $\mathcal{F}_{ideal}$. For the honest parties, $\mathcal{F}_{ideal}$ outputs real output iff no $malicious[j] := 1$ has ever happened in $\mathcal{F}_{ideal}$, and otherwise it outputs $(\mathbf{output}, j_1, \ldots, j_k)$ such that for each j_ℓ the messages $(\mathbf{corrupt}, j_\ell, i)$ has been sent. This is what $\mathcal{A}$ awaits from the output.

5 Using the Proposed Protocol in Secure Multiparty Computation Platforms

In this section we discuss how the proposed verification could be used in Secure Multiparty Computation Platforms. More precisely, here we should consider the case where in addition to *computing* parties (that participate in the protocol) we may have *input* parties (that provide the inputs, sharing them in some way amongst the computing parties) and the *result* parties (that receive the final output). In our protocol, the computing parties do commit the inputs before the computation starts, but we must ensure that these are indeed the same inputs that have been provided by the input parties.

5.1 Treating Inputs/Outputs as Communication

As a simpler solution, we may just handle the input and the output similarly to communication. Hence the following enhancements are made.

1. Let the number of input parties be N. In the beginning, each input party P_i generates the shares $\mathbf{x}_{i1}, \ldots, \mathbf{x}_{in}$ (according to an arbitrary sharing scheme) from all the computing parties $M_1, \ldots, M_n$, as it would do without the verification. Each M_j should now use the input vector $\mathbf{x}_j = (\mathbf{x}_{1j} || \ldots || \mathbf{x}_{Nj})$, where each $\mathbf{x}_{ij}$ is provided by an input party P_i. Now, for each $\mathbf{x}_{ij}$, P_i generates by itself all the $\binom{n-1}{t}$ shares $\mathbf{x}_{ijkT_\ell}$ such that $\sum_{k \in T_j} \mathbf{x}_{ijkT_\ell} = \mathbf{x}_{ij}$, signs all these shares, and sends them to M_j. As before, M_j should also sign all of these shares before redistributing them amongst all the verifier t-sets. The verifiers should now check both signatures, similarly to how it was done to communication.
2. In the end, each receiver party R_i gets the shares $\mathbf{y}_{i1}, \ldots, \mathbf{y}_{in}$ from all the computing parties $M_1, \ldots, M_n$. Now each M_j has to generate $\binom{n-1}{t}$ sums $\sum_{k \in T_j} \mathbf{y}_{ijkT_\ell} = \mathbf{y}_{ij}$, exactly in the same way as it would do with an ordinary communication value. M_j sends the shares and their signatures to P_i, and P_i redistributes them amongst the t-sets. In the verification process, they check both signatures, similarly to how it was done to communication.

Since the parties P_i and R_i do not participate in the computation, they do not have to participate in the verification. However, they will still be punished if they provide multiple signatures for the same value.

5.2 Possible Issues

The main drawback of the previous proposition is the numerous amount of signatures that the computing parties may have to check. While in the initial scheme each party M_j has to provide just one share $\mathbf{x}_{jkT_\ell}$ for each party M_k in each T_ℓ, now it has to provide N shares, where N is the number of input parties, and all their signatures have to be checked (for M_j it is still sufficient to use just one signature, but it does not help much). Depending on the settings, N can be very large. In the worst case, each input party provides only one bit, and hence $N \in O(|C|)$. However, each computing party would have to verify the source of all the inputs anyway. For P_i, sending $t \cdot \binom{n-1}{t}$ shares instead of one is not worse since all the values used by the same P_i may have the same signature. The problem comes when M_j wants to redistribute the shares and the signatures to all T-sets, since each receiver will again have to check all N of them. Fortunately, this happens only in the beginning and in the end of the protocol.

Additionally, depending on the performed computation, the covert security may just not work with the input parties, especially in some anonymous statistical projects. Any participant may cheat without reason and complain afterwards. In our scheme, the verification of input share signatures is done already in the beginning, an hence the computing parties will not spend their time on clearly malicious parties whose shares do not correspond to their signatures. The problem still remains with the output, since the malicious output party R_i may sign wrong values just for fun, without fear of being detected. However, since such cheating would require just one additional broadcast (revealing the signatures to everyone), this is not too much different from the case if R_i has not complained. In any case, even if no one complains, it may still be some kind of attack where the input party is completely honest, but it just performs the computation without needing. In relation to statistic projects, some input parties may also produce unrealistic artificial inputs that harm the result in general, so some outlier detection would be needed in the beginning.

5.3 Deviations from the Initial Settings

In real Secure Multiparty Computation Platforms, it may happen that the number of input parties is initially unknown. For example, in the case of some statistical computation, the input parties may come and submit their inputs during the execution, and hence the shape of the computational circuit may be even unknown in the beginning, since the input length is undefined. Nevertheless, the proposed techniques still work. The coming input parties may commit the inputs as they come. In the end of the computation, the structure of the circuit will be known anyway. The formal proofs would have to be adjusted, but now we need to define a new ideal functionality that allows the adversary to introduce new input parties during the computation.

6 Conclusions and Future Work

In this paper we have proposed a scheme that allows to verify the computation of each party in a passively secure protocol, thus converting passive security to covert security. Each malicious party will be detected with probability close to 1, depending on the parameters of selected field.

While our verification is being done only after the entire computation has ended, it might be interesting to do something more similar to the active security model. Namely, we could require each party to prove the correctness after each round. If implemented straightforwardly, repeating our verification algorithm on each round, it multiplies the verification complexity by the number of rounds (actually, a bit less since in the beginning the vectors will be of smaller length). Doing it more cleverly, we could make use of the proofs of the previous rounds, making the next proof steps reliable on the proofs of the previous steps. The ideas can be taken for example from [CT10].

References

[AL10] Yonatan Aumann and Yehuda Lindell. Security against covert adversaries: Efficient protocols for realistic adversaries. *J. Cryptology*, 23(2):281–343, 2010.

[AO12] Gilad Asharov and Claudio Orlandi. Calling out cheaters: Covert security with public verifiability. In Xiaoyun Wang and Kazue Sako, editors, *ASIACRYPT*, volume 7658 of *Lecture Notes in Computer Science*, pages 681–698. Springer, 2012.

[BCI$^+$13] Nir Bitansky, Alessandro Chiesa, Yuval Ishai, Rafail Ostrovsky, and Omer Paneth. Succinct non-interactive arguments via linear interactive proofs. In *TCC*, pages 315–333, 2013.

[BSCG$^+$13] Eli Ben-Sasson, Alessandro Chiesa, Daniel Genkin, Eran Tromer, and Madars Virza. Snarks for c: Verifying program executions succinctly and in zero knowledge. In *CRYPTO (2)*, pages 90–108, 2013.

[Can01] Ran Canetti. Universally composable security: A new paradigm for cryptographic protocols. In *FOCS*, pages 136–145. IEEE Computer Society, 2001.

[CT10] Alessandro Chiesa and Eran Tromer. Proof-carrying data and hearsay arguments from signature cards. In Andrew Chi-Chih Yao, editor, *ICS*, pages 310–331. Tsinghua University Press, 2010.

[DGN10] Ivan Damgård, Martin Geisler, and Jesper Buus Nielsen. From passive to covert security at low cost. In Daniele Micciancio, editor, *TCC*, volume 5978 of *Lecture Notes in Computer Science*, pages 128–145. Springer, 2010.

[GGPR13] Rosario Gennaro, Craig Gentry, Bryan Parno, and Mariana Raykova. Quadratic span programs and succinct nizks without pcps. In Thomas Johansson and Phong Q. Nguyen, editors, *EUROCRYPT*, volume 7881 of *Lecture Notes in Computer Science*, pages 626–645. Springer, 2013.

[Lin13] Yehuda Lindell. Fast cut-and-choose based protocols for malicious and covert adversaries. In Ran Canetti and Juan A. Garay, editors, *CRYPTO (2)*, volume 8043 of *Lecture Notes in Computer Science*, pages 1–17. Springer, 2013.

[Lip13] Helger Lipmaa. Succinct non-interactive zero knowledge arguments from span programs and linear error-correcting codes. *IACR Cryptology ePrint Archive*, 2013:121, 2013.