

# Verifiable Computation in Multiparty Protocols with Honest Majority

Peeter Laud<sup>1</sup> and Alisa Pankova<sup>1,2,3</sup>

<sup>1</sup> Cybernetica AS, Estonia

<sup>2</sup> Software Technologies and Applications Competence Centre (STACC), Estonia

<sup>3</sup> University of Tartu, Estonia

**Abstract.** We present a generic method for turning passively secure protocols into protocols secure against covert attacks. The method adds a post-execution verification phase to the protocol that allows a misbehaving party to escape detection only with negligible probability. The execution phase, after which the computed protocol result is already available for parties, has only negligible overhead added by our method. The checks, based on linear probabilistically checkable proofs, are done in zero-knowledge, thereby preserving the privacy guarantees of the original protocol. Our method is inspired by recent results in verifiable computation, adapting them to multiparty setting and significantly lowering their computational costs for the provers.

**Keywords:** Secure multiparty computation, Verifiable computation, Linear PCP.

## 1 Introduction

Any multiparty computation can be performed in a manner that the participants only learn their own outputs and nothing else [24]. While the generic construction is expensive in computation and communication, the result has sparked research activities in secure multiparty computation (SMC), with results that are impressive both performance-wise [9, 11, 17, 20], as well as in the variety of concrete problems that have been tackled [10, 14, 16, 21]. From the start, two kinds of adversaries — passive and active — have been considered in the construction of SMC protocols, with highest performance and the greatest variety achieved for protocol sets secure only against passive adversaries.

Verifiable computation (VC) [22] allows a weak client to outsource a computation to a more powerful server that accompanies the computed result with a proof of correct computation, the verification of which by the client is cheaper than performing the computation itself. VC could be used to strengthen protocols secure against passive adversaries — after executing the protocol, the parties could prove to each other that they have correctly followed the protocol. If the majority of the parties are honest (an assumption which is made also by the most efficient SMC protocol sets secure against passive adversaries), then the resulting protocol would satisfy a strong version of *covert security* [2], where any

deviations from the protocol are guaranteed to be discovered and reported. Unfortunately, existing approaches to VC have a large computational overhead for the server/prover. Typically, if the computation is represented as an arithmetic circuit  $C$ , the prover has to perform  $\Omega(|C|)$  public-key operations in order to ensure its good behaviour, as well as to protect its privacy.

In this paper we show that in the multiparty context, with an honest majority, these public-key operations are not necessary. Instead, the verifications can be done in distributed manner, in a way that provides the same security properties. For this, we apply the ideas of existing VC approaches based on linear probabilistically checkable proofs (PCPs) [25], and combine them with linear secret sharing, which we use also for commitments. We end up with a protocol transformation that makes the executions of any protocol (and not just SMC protocols) verifiable afterwards. Our transformation commits the randomness (this takes place offline), inputs, and the communication of the participants. The commitments are cheap, being based on digital signatures and not adding a significant overhead to the execution phase. The results of the protocol are available after the execution. The verification can take place at any time after the execution; dedicated high-bandwidth high-latency communication channels can be potentially used for it. The verification itself is succinct. The proof is generated in  $O(|C| \log |C|)$  field operations, but the computation is local. The generation of challenges costs  $O(1)$  in communication and  $O(|C|)$  in local computation.

We present our protocol transformation as a functionality in the universal composability (UC) framework. After reviewing related work in Sec. 2, we describe the ideal functionality in Sec. 3 and its implementation in Sec. 5. Before the latter, we give an overview of the existing building blocks we use in Sec. 4. The computational overhead of our transformation is estimated in Sec. 6.

Besides increasing the security of SMC protocols, our transformation can be used to add verifiability to other protocols. In Sec. 7 we demonstrate how a verifiable secret sharing (VSS) scheme can be constructed. We compare it with state-of-the-art VSS schemes and find that despite much higher genericity, our construction enjoys similar complexity.

## 2 Related Work

The property brought by our protocol transformation is similar to security against *covert* adversaries [2] that are prevented from deviating from the prescribed protocol by a non-negligible chance of getting caught. A similar transformation, applicable to protocols of certain structure, was introduced by Damgård et al. [18]. Compared to their transformation, ours is more general, has lower overhead in the execution phase, and is guaranteed to catch the deviating parties. Our transformation can handle protocols, where some of the results are made available to the computing parties already before the end of the protocol; this may significantly lower the protocol's complexity [10]. A good property of their construction is its black-box nature, which our transformation does not have. Hence different transformations may be preferable in different situations.

There have been many works dedicated to short verifications of solutions to NP-complete problems. Probabilistically checkable proofs [1] allow to verify a possibly long proof by querying a small number of its bits. Micali [31] has presented *computationally sound proofs*, where the verification is not perfect, and the proof can be forged, but it is computationally hard to do. Kilian [26] proposed interactive probabilistically checkable proofs using bit commitments. A certain class of *linear* probabilistically checkable proofs [25], allows to make argument systems much simpler and more general.

In computation verification, the prover has to prove that, given valuations of certain wires of a circuit, there exists a correct valuation of all the other wires such that the computation is correct with respect to the given circuit. Verifiable computation can in general be based not only on the PCP theorem. In [22], Yao’s garbled circuits [37] are executed using fully homomorphic encryption. Quadratic span programs for boolean circuits and quadratic arithmetic programs for arithmetic circuits without PCP have first been proposed in [23], later extended to PCP by [6], and further optimized and improved in [5, 29, 32]. Particular implementations of verifiable computations have been done for example in [5, 32, 35].

The goal of our transformation is to provide security against a certain form of active attackers. SMC protocols secure against active attackers have been known for a long time [15, 24]. SPDZ [19, 20] is probably the SMC protocol set secure against active adversaries with currently the best online performance, achieved through extensive offline precomputations. Similarly to several other protocol sets, SPDZ provides only a minimum amount of protocols to cooperatively evaluate an arithmetic circuit. We note that very recently, a form of post-execution verifiability has been proposed for SPDZ [4].

### 3 Ideal Functionality

We use the universal composability (UC) framework [13] to specify our verifiable execution functionality. We have  $n$  parties (indexed by  $[n] = \{1, \dots, n\}$ ), where  $\mathcal{C} \subseteq [n]$  are corrupted for  $|\mathcal{C}| = t < n/2$  (we denote  $\mathcal{H} = [n] \setminus \mathcal{C}$ ). The protocol has  $r$  rounds, where the computations of the party  $P_i$  on the  $\ell$ -th round are given by an arithmetic circuit  $C_{ij}^\ell$  over a field  $\mathbb{F}$ , computing the  $\ell$ -th round messages  $\mathbf{m}_{ij}^\ell$  to all parties  $j \in [n]$  from the input  $\mathbf{x}_i$ , randomness  $\mathbf{r}_i$  and the messages  $P_i$  has received before (all values  $\mathbf{x}_i, \mathbf{r}_i, \mathbf{m}_{ij}^\ell$  are vectors over  $\mathbb{F}$ ). We define that the messages received during the  $r$ -th round comprise the *output of the protocol*. The ideal functionality  $\mathcal{F}_{vmPC}$ , running in parallel with the environment  $\mathcal{Z}$  and the adversary  $\mathcal{A}_S$ , is given in Fig. 1.

We see that  $\mathcal{M}$  is the set of parties actually deviating from the protocol. Our verifiability property is very strong — they *all* will be reported to *all* honest parties. Even if only *some* rounds of the protocol are computed, all the parties that deviated from the protocol in completed rounds will be detected. Also, no honest parties (in  $\mathcal{H}$ ) can be falsely blamed. We also note that if  $\mathcal{M} = \emptyset$ , then  $\mathcal{A}_S$  does not learn anything that a semi-honest adversary could not learn.

**In the beginning**,  $\mathcal{F}_{vmpc}$  gets from  $\mathcal{Z}$  for each party  $P_i$  the message (circuits,  $i$ ,  $(C_{ij}^\ell)_{i,j,\ell=1,1,1}^{n,n,r}$ ) and forwards them all to  $\mathcal{A}_S$ . For each  $i \in \mathcal{H}$  [resp  $i \in \mathcal{C}$ ],  $\mathcal{F}_{vmpc}$  gets (input,  $\mathbf{x}_i$ ) from  $\mathcal{Z}$  [resp.  $\mathcal{A}_S$ ]. For each  $i \in [n]$ ,  $\mathcal{F}_{vmpc}$  randomly generates  $\mathbf{r}_i$ . For each  $i \in \mathcal{C}$ , it sends (randomness,  $i$ ,  $\mathbf{r}_i$ ) to  $\mathcal{A}_S$ .

**For each round**  $\ell \in [r]$ ,  $i \in \mathcal{H}$  and  $j \in [n]$ ,  $\mathcal{F}_{vmpc}$  uses  $C_{ij}^\ell$  to compute the message  $\mathbf{m}_{ij}^\ell$ . For all  $j \in \mathcal{C}$ , it sends  $\mathbf{m}_{ij}^\ell$  to  $\mathcal{A}_S$ . For each  $j \in \mathcal{C}$  and  $i \in \mathcal{H}$ , it receives  $\mathbf{m}_{ji}^\ell$  from  $\mathcal{A}_S$ .

**After  $r$  rounds**,  $\mathcal{F}_{vmpc}$  sends (output,  $\mathbf{m}_{1i}^r, \dots, \mathbf{m}_{ni}^r$ ) to each party  $P_i$  with  $i \in \mathcal{H}$ . Let  $r' = r$  and  $\mathcal{B}_0 = \emptyset$ .

Alternatively, **at any time** before outputs are delivered to parties,  $\mathcal{A}_S$  may send (stop,  $\mathcal{B}_0$ ) to  $\mathcal{F}_{vmpc}$ , with  $\mathcal{B}_0 \subseteq \mathcal{C}$ . In this case the outputs are not sent. Let  $r' \in \{0, \dots, r-1\}$  be the last completed round.

**After  $r'$  rounds**,  $\mathcal{A}_S$  sends to  $\mathcal{F}_{vmpc}$  the messages  $\mathbf{m}_{ij}^\ell$  for  $\ell \in [r']$  and  $i, j \in \mathcal{C}$ .  $\mathcal{F}_{vmpc}$  defines  $\mathcal{M} = \mathcal{B}_0 \cup \{i \in \mathcal{C} \mid \exists j \in [n], \ell \in [r'] : \mathbf{m}_{ij}^\ell \neq C_{ij}^\ell(\mathbf{x}_i, \mathbf{r}_i, \mathbf{m}_{1i}^1, \dots, \mathbf{m}_{ni}^{\ell-1})\}$ .

**Finally**, for each  $i \in \mathcal{H}$ ,  $\mathcal{A}_S$  sends (blame,  $i$ ,  $\mathcal{B}_i$ ) to  $\mathcal{F}_{vmpc}$ , with  $\mathcal{M} \subseteq \mathcal{B}_i \subseteq \mathcal{C}$ .  $\mathcal{F}_{vmpc}$  forwards this message to  $P_i$ .

**Fig. 1.** The ideal functionality for verifiable computations

## 4 Building Blocks

Throughout this paper, bold letters  $\mathbf{x}$  denote vectors, where  $x_i$  denotes the  $i$ -th coordinate of  $\mathbf{x}$ . Concatenation of  $\mathbf{x}$  and  $\mathbf{y}$  is denoted by  $(\mathbf{x} \parallel \mathbf{y})$ , and their scalar product by  $\langle \mathbf{x}, \mathbf{y} \rangle$ , which is defined (only if  $|\mathbf{x}| = |\mathbf{y}|$ ) as  $\langle \mathbf{x}, \mathbf{y} \rangle = \sum_{i=1}^{|\mathbf{x}|} x_i y_i$ .

Our implementation uses a number of previously defined subprotocols and algorithm sets.

**Message Transmission.** For message transmission between parties, we use functionality  $\mathcal{F}_{tr}$  [18], which allows one to prove to third parties which messages one received during the protocol, and to further transfer such revealed messages. Our definition of  $\mathcal{F}_{tr}$  differs from Damgård et al.'s [18]  $\mathcal{F}_{transmit}$  by supporting the forwarding of received messages, as well as broadcasting as a part of the outer protocol. The definition of the ideal functionality of  $\mathcal{F}_{tr}$  is shown in Fig. 2. The real implementation of the transmission functionality is built on top of signatures. This makes the implementation very efficient, as hash trees allow several messages (sent in the same round) to be signed with almost the same computation effort as a single one [30], and signatures can be verified in batches [12]. An implementation of  $\mathcal{F}_{tr}$  is given in the full version of this paper [28].

**Shamir's Secret Sharing.** For commitments, we use  $(n, t)$  Shamir secret sharing [36], where any  $t$  parties are able to recover the secret, but less than  $t$  are not. By sharing a vector  $\mathbf{x}$  over  $\mathbb{F}$  into vectors  $\mathbf{x}^1, \dots, \mathbf{x}^n$  we mean that each  $i$ -th entry  $x_i \in \mathbb{F}$  of  $\mathbf{x}$  is shared into the  $i$ -th entries  $x_i^1 \in \mathbb{F}, \dots, x_i^n \in \mathbb{F}$  of  $\mathbf{x}^1, \dots, \mathbf{x}^n$ . In this way, for each  $T = \{i_1, \dots, i_t\} \subseteq [n]$ , the entries can be restored as  $x_i = \sum_{j=1}^t b_{Tj} x_{i_j}^{i_j}$  for certain constants  $b_{Tj}$ , and hence  $\mathbf{x} = \sum_{j=1}^t b_{Tj} \mathbf{x}^{i_j}$ . The linearity extends to scalar products: if a vector  $\boldsymbol{\pi}$  is shared to  $\boldsymbol{\pi}^1, \dots, \boldsymbol{\pi}^n$ , then for any vector  $\mathbf{q}$  and  $T = \{i_1, \dots, i_t\}$ , we have  $\sum_{j=1}^t b_{Tj} \langle \boldsymbol{\pi}^{i_j}, \mathbf{q} \rangle = \langle \boldsymbol{\pi}, \mathbf{q} \rangle$ .

$\mathcal{F}_{tr}$  works with unique message identifiers  $mid$ , encoding a sender  $s(mid) \in [n]$ , a receiver  $r(mid) \in [n]$ , and a party  $f(mid) \in [n]$  to whom the message should be forwarded by the receiver (if no forwarding is foreseen then  $f(mid) = r(mid)$ ).

**Secure transmit:** Receiving  $(\text{transmit}, mid, m)$  from  $P_{s(mid)}$  and  $(\text{transmit}, mid)$  from all (other) honest parties, store  $(mid, m, r(mid))$ , mark it as undelivered, and output  $(mid, |m|)$  to the adversary. If the input of  $P_{s(mid)}$  is invalid (or there is no input), and  $P_{r(mid)}$  is honest, then output  $(\text{corrupt}, s(mid))$  to all parties.

**Secure broadcast:** Receiving  $(\text{broadcast}, mid, m)$  from  $P_{s(mid)}$  and  $(\text{broadcast}, mid)$  from all honest parties, store  $(mid, m, bc)$ , mark it as undelivered, output  $(mid, |m|)$  to the adversary. If the input of  $P_{s(mid)}$  is invalid, output  $(\text{corrupt}, s(mid))$  to all parties.

**Synchronous delivery:** At the end of each round, for each undelivered  $(mid, m, r)$  send  $(mid, m)$  to  $P_r$ ; mark  $(mid, m, r)$  as delivered. For each undelivered  $(mid, m, bc)$  send  $(mid, m)$  to each party and the adversary; mark  $(mid, m, bc)$  as delivered.

**Forward received message:** On input  $(\text{forward}, mid)$  from  $P_{r(mid)}$  after  $(mid, m)$  has been delivered to  $P_{r(mid)}$ , and receiving  $(\text{forward}, mid)$  from all honest parties, store  $(mid, m, f(mid))$ , mark as undelivered, output  $(mid, |m|)$  to the adversary. If the input of  $P_{r(mid)}$  is invalid, and  $P_{f(mid)}$  is honest, output  $(\text{corrupt}, r(mid))$  to all parties.

**Publish received message:** On input  $(\text{publish}, mid)$  from the party  $P_{f(mid)}$  which at any point received  $(mid, m)$ , output  $(mid, m)$  to each party, and also to the adversary.

**Do not commit corrupt to corrupt:** If for some  $mid$  both  $P_{s(mid)}$ ,  $P_{r(mid)}$  are corrupt, then on input  $(\text{forward}, mid)$  the adversary can ask  $\mathcal{F}_{tr}$  to output  $(mid, m')$  to  $P_{f(mid)}$  for any  $m'$ . If additionally  $P_{f(mid)}$  is corrupt, then the adversary can ask  $\mathcal{F}_{tr}$  to output  $(mid, m')$  to all honest parties.

Fig. 2. Ideal functionality  $\mathcal{F}_{tr}$

We note that sharing a value  $x$  as  $x^1 = \dots = x^k = x$  is valid, i.e.  $x$  can be restored from  $x^{i_1}, \dots, x^{i_t}$  by forming the same linear combination. In our implementation of the verifiable computation functionality, we use such “sharing” for values that end up public due to the adversary’s actions.

**Linear PCP** This primitive forms the basis of our verification. Before giving its definition, let us formally state when a protocol is statistically privacy-preserving.

**Definition 1 ( $\delta$ -private protocol [8]).** Let  $\Pi$  be a multiparty protocol that takes input  $\mathbf{x}$  from honest parties and  $\mathbf{y}$  from adversarially controlled parties. The protocol  $\Pi$  is  $\delta$ -private against a class of adversaries  $\mathcal{A}$  if there exists a simulator  $\text{Sim}$ , such that for all adversaries  $A \in \mathcal{A}$  and inputs  $\mathbf{x}, \mathbf{y}$ ,  $|\Pr[A^{\Pi(\mathbf{x}, \mathbf{y})}(\mathbf{y}) = 1] - \Pr[A^{\text{Sim}(\mathbf{y})}(\mathbf{y}) = 1]| \leq \delta$ .

**Definition 2 (Linear Probabilistically Checkable Proof (LPCP) [6]).** Let  $\mathbb{F}$  be a finite field,  $v, \omega \in \mathbb{N}$ ,  $R \subseteq \mathbb{F}^v \times \mathbb{F}^\omega$ . Let  $P$  and  $Q$  be probabilistic algorithms, and  $D$  a deterministic algorithm. The pair  $(P, V)$ , where  $V = (Q, D)$  is a  $d$ -query  $\delta$ -statistical HVZK linear PCP for  $R$  with knowledge error  $\varepsilon$  and query length  $m$ , if the following holds.

**Syntax.** On input  $\mathbf{v} \in \mathbb{F}^v$  and  $\mathbf{w} \in \mathbb{F}^\omega$ , algorithm  $P$  computes  $\boldsymbol{\pi} \in \mathbb{F}^m$ . The algorithm  $Q$  randomly generates  $d$  vectors  $\mathbf{q}_1, \dots, \mathbf{q}_d \in \mathbb{F}^m$  and some state

information  $\mathbf{u}$ . On input  $\mathbf{v}$ ,  $\mathbf{u}$ , as well as  $a_1, \dots, a_d \in \mathbb{F}$ , the algorithm  $D$  accepts or rejects. Let  $V^\pi(\mathbf{v})$  denote the execution of  $Q$  followed by the execution of  $V$  on  $\mathbf{v}$ , the output  $\mathbf{u}$  of  $Q$ , and  $a_1, \dots, a_d$ , where  $a_i = \langle \pi, \mathbf{q}_i \rangle$ .

**Completeness.** For every  $(\mathbf{v}, \mathbf{w}) \in R$ , the output of  $P(\mathbf{v}, \mathbf{w})$  is a vector  $\pi \in \mathbb{F}^m$  such that  $V^\pi(\mathbf{v})$  accepts with probability 1.

**Knowledge.** There exists a knowledge extractor  $E$  such that for every vector  $\pi^* \in \mathbb{F}^m$ , if

$\Pr[V^{\pi^*}(\mathbf{v}) \text{ accepts}] \geq \varepsilon$  then  $E(\pi^*, \mathbf{v})$  outputs  $\mathbf{w}$  such that  $(\mathbf{v}, \mathbf{w}) \in R$ .

**Honest Verifier Zero Knowledge.** The protocol between an honest prover executing  $\pi \leftarrow P(\mathbf{v}, \mathbf{w})$  and adversarial verifier executing  $V^\pi(\mathbf{v})$  with common input  $\mathbf{v}$  and prover's input  $\mathbf{w}$  is  $\delta$ -private for the class of passive adversaries.

Similarly to different approaches to verifiable computation [5, 6, 23, 29, 32], in our work we let the relation  $R$  to correspond to the circuit  $C$  executed by the party whose observance of the protocol is being verified. In this correspondence,  $\mathbf{v}$  is the tuple of all inputs, outputs, and used random values of that party. The vector  $\mathbf{w}$  extends  $\mathbf{v}$  with the results of all intermediate computations by that party. Differently from existing approaches,  $\mathbf{v}$  itself is private. Hence it is unclear how the decision algorithm  $D$  can be executed on it. Hence we do not use  $D$  as a black box, but build our solution on top of a particular LPCP [5].

The LPCP algorithms used by Ben-Sasson et al. [5] are statistical HVZK. Namely, the values  $\langle \pi, \mathbf{q}_i \rangle$  do not reveal any private information about  $\pi$ , unless the random seed  $\tau \in \mathbb{F}$  for  $Q$  is chosen in a bad way, which happens with negligible probability for a sufficiently large field. In [5],  $Q$  generates 5 challenges  $\mathbf{q}_1, \dots, \mathbf{q}_5$  and the state information  $\mathbf{u}$  with length  $|\mathbf{v}|+2$ . Given the query results  $a_i = \langle \pi, \mathbf{q}_i \rangle$  for  $i \in \{1, \dots, 5\}$  and the state information  $\mathbf{u} = (u_0, u_1, \dots, u_{|\mathbf{v}|+1})$ , the following two checks have to pass:

$$a_1 a_2 - a_3 - a_4 u_{|\mathbf{v}|+1} = 0, \tag{*}$$

$$a_5 - \langle (1 \parallel \mathbf{v}), (u_0, u_1, \dots, u_{|\mathbf{v}|}) \rangle = 0. \tag{**}$$

Here (\*) is used to show the existence of  $\mathbf{w}$ , and (\*\*) shows that a certain segment of  $\pi$  equals  $(1 \parallel \mathbf{v})$  [5]. Throughout this work, we reorder the entries of  $\pi$  compared to [5] and write  $\pi = (\mathbf{p} \parallel 1 \parallel \mathbf{v})$  where  $\mathbf{p}$  represents all the other entries of  $\pi$ , as defined in [5]. The challenges  $\mathbf{q}_1, \dots, \mathbf{q}_5$  are reordered in the same way.

This linear interactive proof can be converted to a zero-knowledge succinct non-interactive argument of knowledge [6]. Unfortunately, it requires homomorphic encryption, and the number of encryptions is linear in the size of the circuit. We show that the availability of honest majority allows the proof to be completed without public-key encryptions.

The multiparty setting introduces a further difference from [5]: the vector  $\mathbf{v}$  can no longer be considered public, as it contains a party's private values. We thus have to strengthen the HVZK requirement in Def. 2, making  $\mathbf{v}$  private to the prover. The LPCP constructions of [5] do not satisfy this strengthened HVZK requirement, but their authors show that this requirement would be satisfied if  $a_5$  were not present. In the following, we propose a construction where just the first check (\*) is sufficient, so only  $a_1, \dots, a_4$  have to be published. We prove that the second check (\*\*) will be passed implicitly. We show the following.

**Theorem 1.** *Given a  $\delta$ -statistical HVZK instance of the LPCP of Ben-Sasson et al. [5] with knowledge error  $\varepsilon$ , any  $n$ -party  $r$ -round protocol  $\Pi$  can be transformed into an  $n$ -party  $(r + 8)$ -round protocol  $\Xi$  in the  $\mathcal{F}_{tr}$ -hybrid model, which computes the same functionality as  $\Pi$  and achieves covert security against adversaries statically corrupting at most  $t < n/2$  parties, where the cheating of any party is detected with probability at least  $(1 - \varepsilon)$ . If  $\Pi$  is  $\delta'$ -private against passive adversaries statically corrupting at most  $t$  parties, then  $\Xi$  is  $(\delta' + \delta)$ -private against cover adversaries. Under active attacks by at most  $t$  parties, the number of rounds of the protocol may at most double.*

Theorem 1 is proved by the construction of the real functionality in the next section, as well as the simulator presented in [28]. In the construction, we use the following algorithms implicitly defined by Ben-Sasson et al. [5]:

- *witness*( $C, \mathbf{v}$ ): if  $\mathbf{v}$  corresponds to a valid computation of  $C$ , returns a witness  $\mathbf{w}$  such that  $(\mathbf{v}, \mathbf{w}) \in \mathcal{R}_C$ .
- *proof*( $C, \mathbf{v}, \mathbf{w}$ ): if  $(\mathbf{v}, \mathbf{w}) \in \mathcal{R}_C$ , it constructs a corresponding proof  $\mathbf{p}$ .
- *challenge*( $C, \tau$ ): returns  $\mathbf{q}_1, \dots, \mathbf{q}_5, \mathbf{u}$  that correspond to  $\tau$ , such that:
  - for any valid proof  $\boldsymbol{\pi} = (\mathbf{p} \| 1 \| \mathbf{v})$ , where  $\mathbf{p}$  is generated by *proof*( $C, \mathbf{v}, \mathbf{w}$ ) for  $(\mathbf{v}, \mathbf{w}) \in \mathcal{R}_C$ , the checks (\*) and (\*\*) succeed with probability 1;
  - for any proof  $\boldsymbol{\pi}^*$  generated without knowing  $\tau$ , or such  $\mathbf{w}$  that  $(\mathbf{v}, \mathbf{w}) \in \mathcal{R}_C$ , either (\*) or (\*\*) fails, except with negligible probability  $\varepsilon$ .

## 5 Real Functionality

The protocol  $\Pi_{vmc}$  implementing  $\mathcal{F}_{vmc}$  consists of  $n$  machines  $M_1, \dots, M_n$  doing the work of parties  $P_1, \dots, P_n$ , and the functionality  $\mathcal{F}_{tr}$ . The internal state of each  $M_i$  contains a bit-vector  $mlc_i$  of length  $n$  where  $M_i$  marks which other parties are acting maliciously. The protocol  $\Pi_{vmc}$  runs in five phases: initialization, execution, message commitment, verification, and accusation.

In the initialization phase, the inputs  $\mathbf{x}_i$  and the randomness  $\mathbf{r}_i$  are committed. It is ensured that the randomness indeed comes from uniform distribution. This phase is given in Fig.3. If at any time (**corrupt**,  $j$ ) comes from  $\mathcal{F}_{tr}$ , each (uncorrupted)  $M_i$  writes  $mlc_i[j] := 1$  (for each message (**corrupt**,  $j$ )) and goes to the accusation phase.

In the execution phase, the parties run the original protocol as before, just using  $\mathcal{F}_{tr}$  to exchange the messages. This is given in Fig.4. If at any time at some round  $\ell$  the message (**corrupt**,  $j$ ) comes from  $\mathcal{F}_{tr}$  (all uncorrupted machines receive it at the same time), the execution is cut short, no outputs are produced and the protocol continues with the commitment phase.

In the message commitment phase, all the  $n$  parties finally commit their sent messages  $\mathbf{c}_{ij}^\ell$  for each round  $\ell \in [r']$  by sharing them to  $\mathbf{c}_{ij}^{\ell 1}, \dots, \mathbf{c}_{ij}^{\ell n}$  according to  $(n, t + 1)$  Shamir scheme. This phase is given in Fig. 5. Let  $\mathbf{v}_{ij}^\ell = (\mathbf{x}_i \| \mathbf{r}_i \| \mathbf{c}_{1i}^1 \| \dots \| \mathbf{c}_{ni}^{\ell-1} \| \mathbf{c}_{ij}^\ell)$  be the vector of inputs and outputs to the circuit  $C_{ij}^\ell$  that  $M_i$  uses to compute the  $\ell$ -th message to  $M_j$ . If the check performed by  $M_j$  fails, then  $M_j$  has received from  $M_i$  enough messages to prove its corruptness

**Circuits:**  $M_i$  gets from  $\mathcal{Z}$  the message (circuits,  $i$ ,  $(C_{ij}^\ell)_{i,j,\ell=1,1,1}^{n,n,r}$ ) and sends it to  $\mathcal{A}$ .

**Randomness generation and commitment:** Let  $\mathcal{R} = [t+1]$ . For all  $i \in \mathcal{R}$ ,  $j \in [n]$ ,  $M_i$  generates  $\mathbf{r}_{ij}$  for  $M_j$ .  $M_i$  shares  $\mathbf{r}_{ij}$  to  $n$  vectors  $\mathbf{r}_{ij}^1, \dots, \mathbf{r}_{ij}^n$  according to  $(n, t+1)$  Shamir scheme. For  $j \in [n]$ ,  $M_i$  sends (transmit, (r\_share,  $i, j, k$ ),  $\mathbf{r}_{ij}^k$ ) to  $\mathcal{F}_{tr}$  for  $M_k$ .

**Randomness approval:** For each  $j \in [n] \setminus \{k\}$ ,  $i \in \mathcal{R}$ ,  $M_k$  sends (forward, (r\_share,  $i, j, k$ )) to  $\mathcal{F}_{tr}$  for  $M_j$ . Upon receiving ((r\_share,  $i, j, k$ ),  $\mathbf{r}_{ij}^k$ ) for all  $k \in [n]$ ,  $i \in \mathcal{R}$ ,  $M_j$  checks if the shares comprise a valid  $(n, t+1)$  Shamir sharing.  $M_j$  sets  $\mathbf{r}_i = \sum_{i \in \mathcal{R}} \mathbf{r}_{ij}$ .

**Input commitments:**  $M_i$  with  $i \in \mathcal{H}$  [resp.  $i \in \mathcal{C}$ ] gets from  $\mathcal{Z}$  [resp.  $\mathcal{A}$ ] the input  $\mathbf{x}_i$  and shares it to  $n$  vectors  $\mathbf{x}_i^1, \dots, \mathbf{x}_i^n$  according to  $(n, t+1)$  Shamir scheme. For each  $k \in [n] \setminus \{i\}$ ,  $M_i$  sends to  $\mathcal{F}_{tr}$  (transmit, (x\_share,  $i, k$ ),  $\mathbf{x}_i^k$ ) for  $M_k$ .

**At any time:** if (corrupt,  $j$ ) comes from  $\mathcal{F}_{tr}$ ,  $M_i$  writes  $mlc_i[j] := 1$  and goes to the accusation phase.

**Fig. 3.** The real functionality: initialization phase

**For each round  $\ell$**  the machine  $M_i$  computes  $\mathbf{c}_{ij}^\ell = C_{ij}^\ell(\mathbf{x}_i, \mathbf{r}_i, \mathbf{c}_{1i}^1, \dots, \mathbf{c}_{ni}^{\ell-1})$  for each  $j \in [n]$  and sends to  $\mathcal{F}_{tr}$  the message (transmit, (message,  $\ell, i, j$ ),  $\mathbf{c}_{ij}^\ell$ ) for  $M_j$ .

**After  $r$  rounds,** uncorrupted  $M_i$  sends (output,  $\mathbf{c}_{1i}^r, \dots, \mathbf{c}_{ni}^r$ ) to  $\mathcal{Z}$  and sets  $r' := r$ .

**At any time:** if (corrupt,  $j$ ) comes from  $\mathcal{F}_{tr}$ , each (uncorrupted)  $M_i$  writes  $mlc_i[j] := 1$  sets  $r' := \ell - 1$  and goes to the message commitment phase.

**Fig. 4.** The real functionality: execution phase

to others (but Fig. 5 presents an alternative, by publicly agreeing on  $\mathbf{c}_{ij}^\ell$ ). After this phase,  $M_i$  has shared  $\mathbf{v}_{ij}^\ell$  among all  $n$  parties. Let  $\mathbf{v}_{ij}^{\ell k}$  be the share of  $\mathbf{v}_{ij}^\ell$  given to machine  $M_k$ .

Each  $M_i$  generates  $\mathbf{w}_{ij}^\ell = \text{witness}(C_{ij}^\ell, \mathbf{v}_{ij}^\ell)$ , a proof  $\mathbf{p}_{ij}^\ell = \text{proof}(C_{ij}^\ell, \mathbf{v}_{ij}^\ell, \mathbf{w}_{ij}^\ell)$ , and  $\boldsymbol{\pi}_{ij}^\ell = (\mathbf{p}_{ij}^\ell, \|1\| \mathbf{v}_{ij}^\ell)$  in the verification phase, as explained in Sec. 4. The vector  $\mathbf{p}_{ij}^\ell$  is shared to  $\mathbf{p}_{ij}^{\ell 1}, \dots, \mathbf{p}_{ij}^{\ell n}$  according to  $(n, t+1)$  Shamir scheme.

All parties agree on a random  $\tau$ , with  $M_i$  broadcasting  $\tau_i$  and  $\tau$  being their sum. A party refusing to participate is ignored. The communication must be synchronous, with no party  $P_i$  learning the values  $\tau_j$  from others before he has sent his own  $\tau_i$ . Note that  $\mathcal{F}_{tr}$  already provides this synchronicity. If it were not available, then standard tools (commitments) could be used to achieve fairness.

**Message sharing:** As a sender,  $M_i$  shares  $\mathbf{c}_{ij}^\ell$  to  $\mathbf{c}_{ij}^{\ell 1}, \dots, \mathbf{c}_{ij}^{\ell n}$  according to  $(n, t+1)$  Shamir scheme. For each  $k \in [n] \setminus \{i\}$ ,  $M_i$  sends to  $\mathcal{F}_{tr}$  the messages (transmit, (c\_share,  $\ell, i, j, k$ ),  $\mathbf{c}_{ij}^{\ell k}$ ) for  $M_j$ .

**Message commitment:** upon receiving ((c\_share,  $\ell, i, j, k$ ),  $\mathbf{c}_{ij}^{\ell k}$ ) from  $\mathcal{F}_{tr}$  for all  $k \in [n]$ , the machine  $M_j$  checks if the shares correspond to  $\mathbf{c}_{ij}^\ell$  it has already received. If they do not,  $M_j$  sends (publish, (message,  $\ell, i, j$ )) to  $\mathcal{F}_{tr}$ , so now everyone sees the values that it has actually received from  $M_i$ , and each (uncorrupted)  $M_k$  should now use  $\mathbf{c}_{ij}^{\ell k} := \mathbf{c}_{ij}^\ell$ . If the check succeeds, then  $M_i$  sends to  $\mathcal{F}_{tr}$  (forward, (c\_share,  $\ell, i, j, k$ )) for  $M_k$  for all  $k \in [n] \setminus \{i\}$ .

**Fig. 5.** The real functionality: message commitment phase



All (honest) parties generate  $\mathbf{q}_{1ij}^\ell, \dots, \mathbf{q}_{4ij}^\ell, \mathbf{q}_{5ij}^\ell, \mathbf{u}_{ij}^\ell = \text{challenge}(C_{ij}^\ell, \tau)$  for  $\ell \in [r']$ ,  $i \in [n]$ ,  $j \in [n]$ . In the rest of the protocol, only  $\mathbf{q}_{1ij}^\ell, \dots, \mathbf{q}_{4ij}^\ell$ , and  $(\mathbf{u}_{ij}^\ell)_{|v|+1}$  will be actually used.

As a verifier, each  $M_k$  computes  $\boldsymbol{\pi}_{ij}^{\ell k} = (\mathbf{p}_{ij}^{\ell k} \| \mathbf{1} \| \mathbf{v}_{ij}^{\ell k}) = (\mathbf{p}_{ij}^{\ell k} \| \mathbf{1} \| \mathbf{x}_i^k \| \sum_{j \in \mathcal{R}} \mathbf{r}_{ji}^k \| \mathbf{c}_{1i}^{1k} \| \dots \| \mathbf{c}_{ni}^{\ell-1,k} \| \mathbf{c}_{ij}^{\ell k})$ , and then computes and publishes the values  $\langle \boldsymbol{\pi}_{ij}^{\ell k}, \mathbf{q}_{1ij}^\ell \rangle, \dots, \langle \boldsymbol{\pi}_{ij}^{\ell k}, \mathbf{q}_{4ij}^\ell \rangle$ .  $M_i$  checks these values and complains about  $M_k$  that has incorrectly computed them. An uncorrupted  $M_k$  may disprove the complaint by publishing the proof and message shares that it received. Due to the linearity of scalar product and the fact that all the vectors have been shared according to the same  $(n, t + 1)$  Shamir sharing, if the  $n$  scalar product shares correspond to a valid  $(n, t + 1)$  Shamir sharing, the shared value is uniquely defined by any  $t + 1$  shares, and hence by the shares of some  $t + 1$  parties that are all from  $\mathcal{H}$ . Hence  $M_i$  is obliged to use the values it has committed before. The verification phase for  $C_{ij}^\ell$  for fixed  $\ell \in [r']$ ,  $i \in [n]$ ,  $j \in [n]$  is given in Fig.6. For different  $C_{ij}^\ell$ , all the verifications can be done in parallel.

As described, the probability of cheating successfully in our scheme is proportional to  $1/|\mathbb{F}|$ . In order to exponentially decrease it, we may run  $s$  instances of the verification phase in parallel, since by that time  $\mathbf{v}_{ij}^\ell$  are already committed. This will not break HVZK assumption if fresh randomness is used in  $\mathbf{p}_{ij}^\ell$ .

During the message commitment and the verification phases, if at any time  $(\text{corrupt}, j)$  comes from  $\mathcal{F}_{tr}$ , the proof for  $P_j$  ends with failure, and all uncorrupted machines  $M_i$  write  $\text{mlc}_i[j] := 1$ .

Finally, each party outputs the set of parties that it considers malicious. This short phase is given in Fig. 7.

## 6 Efficiency

In this section we estimate the overheads caused by our protocol transformation. The numbers are based on the dominating complexities of the algorithms of linear PCP of [5]. We omit local addition and concatenation of vectors since it is cheap. The preprocessing phase of [5] is done offline, and can be re-used, so we do not estimate the complexity here. It can be done with practical overhead [5].

Let  $n$  be the number of parties,  $t < n/2$  the number of corrupt parties,  $r$  the number of rounds,  $N_g$  the number of gates,  $N_w$  the number of wires,  $N_x$  the number of inputs (elements of  $\mathbb{F}$ ),  $N_r$  the number of random elements of  $\mathbb{F}$ ,  $N_c$  the number of communicated elements of  $\mathbb{F}$ , and  $N_i = N_w - N_x - N_r - N_c$  the number of intermediate wires in the circuit; then  $|v| = N_x + N_r + N_c$ .

Let  $\overline{S}(n, k)$  denote the number of field operations used in sharing one field element according to Shamir scheme with threshold  $k$ , which is at most  $nk$  multiplications. We use  $\overline{S}^{-1}(n, k)$  to denote the complexity of verifying if the shares comprise a valid sharing and recovering the secret, which is also at most  $nk$  multiplications. Compared to the original protocol, for each  $M_i$  the proposed solution has the following computation/communication overheads.

**Initialization:** Do Shamir sharing of one vector of length  $N_x$  in  $N_x \cdot \overline{S}(n, t+1)$  field operations. Transmit  $t + 1$  vectors of length  $N_r$  and one vector of length  $N_x$

**Remaining proof commitment:** As the prover,  $M_i$  obtains  $w_{ij}^\ell$  and  $\pi_{ij}^\ell = (\mathbf{p}_{ij}^\ell \| 1 \| \mathbf{v}_{ij}^\ell)$  using the algorithms *witness* and *proof*.  $M_i$  shares  $\mathbf{p}_{ij}^\ell$  to  $\mathbf{p}_{ij}^{\ell 1}, \dots, \mathbf{p}_{ij}^{\ell r}$  according to  $(n, t + 1)$  Shamir scheme. For each  $k \in [n] \setminus \{i\}$ , it sends to  $\mathcal{F}_{tr}$  (transmit, ( $\mathbf{p\_share}, \ell, i, j, k$ ),  $\mathbf{p}_{ij}^{\ell k}$ ) for  $M_k$ .

**Challenge generation:** Each  $M_k$  generates random  $\tau_k \leftarrow \mathbb{F}$  and sends to  $\mathcal{F}_{tr}$  the message (broadcast, ( $\mathbf{challenge\_share}, \ell, i, j, k$ ),  $\tau_k$ ). If some party refuses to participate its share will just be omitted. The challenge randomness is  $\tau = \tau_1 + \dots + \tau_n$ . Machine  $M_k$  generates  $\mathbf{q}_{1ij}^\ell, \dots, \mathbf{q}_{4ij}^\ell, \mathbf{q}_{5ij}^\ell, \mathbf{u}_{ij}^\ell = \mathbf{challenge}(C_{ij}^\ell, \tau)$ , then computes  $\pi_{ij}^{\ell k} = (\mathbf{p}_{ij}^{\ell k} \| 1 \| \mathbf{v}_{ij}^{\ell k}) = (\mathbf{p}_{ij}^{\ell k} \| 1 \| \mathbf{x}_i^k \| \sum_{j \in \mathcal{R}} \mathbf{r}_{ji}^k \| \mathbf{c}_{1i}^{1k} \| \dots \| \mathbf{c}_{ni}^{\ell-1, k} \| \mathbf{c}_{ij}^{\ell k})$ , and finally computes and broadcasts  $\langle \pi_{ij}^{\ell k}, \mathbf{q}_{1ij}^\ell \rangle, \dots, \langle \pi_{ij}^{\ell k}, \mathbf{q}_{4ij}^\ell \rangle$ .

**Scalar product verification:** Each  $M_i$  verifies the published  $\langle \pi_{ij}^{\ell k}, \mathbf{q}_{sij}^\ell \rangle$  for  $s \in \{1, \dots, 4\}$ . If  $M_i$  finds that  $M_k$  has computed the scalar products correctly, it sends to  $\mathcal{F}_{tr}$  the message (broadcast, ( $\mathbf{complain}, \ell, i, j, k$ ), 0). If some  $M_k$  has provided a wrong value,  $M_i$  sends to  $\mathcal{F}_{tr}$  (broadcast, ( $\mathbf{complain}, \ell, i, j, k$ ),  $(1, sh_{sij}^{\ell k})$ ), where  $sh_{sij}^{\ell k}$  is  $M_i$ 's own version of  $\langle \pi_{ij}^{\ell k}, \mathbf{q}_{sij}^\ell \rangle$ . Everyone waits for a disproof from  $M_k$ . An uncorrupted  $M_k$  sends to  $\mathcal{F}_{tr}$  the messages (publish, *mid*) for  $mid \in \{(\mathbf{x\_share}, i, k), (\mathbf{r\_share}, 1, i, k), \dots, (\mathbf{r\_share}, |\mathcal{R}|, i, k), (\mathbf{p\_share}, \ell, i, j, k), (\mathbf{c\_share}, 1, 1, i, k), \dots, (\mathbf{c\_share}, r', n, i, k), (\mathbf{c\_share}, \ell, i, j, k)\}$ . Now everyone may construct  $\pi_{ij}^{\ell k}$  and verify whether the version provided by  $M_i$  or  $M_k$  is correct.

**Final verification:** Given  $\langle \pi_{ij}^{\ell k}, \mathbf{q}_{sij}^\ell \rangle$  for all  $k \in [n]$ ,  $s \in \{1, \dots, 4\}$ , each machine  $M_i$  checks if they indeed correspond to valid  $(n, t + 1)$  Shamir sharing, and then locally restores  $a_{sij}^\ell = \langle \pi_{ij}^{\ell k}, \mathbf{q}_{sij}^\ell \rangle$  for  $s \in \{1, \dots, 4\}$ , and checks (\*). If the check succeeds, then  $M_v$  accepts the proof of  $M_i$  for  $C_{ij}^\ell$ . Otherwise it immediately sets  $mlc_v[i] := 1$ .

**Fig. 6.** The real functionality: verification phase

**Finally**, each party  $M_i$  sends to  $\mathcal{Z}$  the message ( $\mathbf{blame}, i, \{j \mid mlc_i[j] = 1\}$ ).

**Fig. 7.** The real functionality: accusation phase

to each other party. Do  $t + 1$  recoverings in  $(t + 1) \cdot N_r \cdot \overline{S}^{-1}(n, t + 1)$ . The parties that generate randomness do  $n \cdot N_r \cdot \overline{S}(n, t + 1)$  more field operations to compute  $n$  more sharings, and transmit  $n$  more vectors of length  $N_r$  to each other party.

**Execution:** No computation/communication overheads, except those caused by the use of the message transmission functionality.

**Message Commitment:** Share all the communication in  $rn(n - 1) \cdot N_c \cdot \overline{S}(n, t + 1)$  operations. Send to each other party  $rn$  vectors of length  $N_c$ . Do  $r(n - 1)$  recoverings in  $r(n - 1) \cdot N_c \cdot \overline{S}^{-1}(n, t + 1)$  operations.

**Verification:** Compute the proof  $\mathbf{p}$  of length  $(4 + N_g + N_i)$  in  $18N_g + 3 \cdot \overline{FFT}(N_g) + \log N_g + 1$  field operations [5], where  $\overline{FFT}(N)$  denotes the complexity of the Fast Fourier Transform which is  $c \cdot N \cdot \log N$  for a small constant  $c$ . Share  $\mathbf{p}$  in  $(4 + N_g + N_i) \cdot \overline{S}(n, t + 1)$  operations. Send a vector of length  $(4 + N_g + N_i)$  to every other party. Broadcast one field element (the  $\tau$ ). Generate the 4 challenges and the state information in  $14 \cdot N_g + \log(N_g)$  field operations [5]. Compute and broadcast 4 scalar products of vectors of length  $(5 + N_w + N_g)$

(the shares of  $\langle(\mathbf{p}\|1\|\mathbf{v}), \mathbf{q}_s\rangle$ ). Compute 4 certain linear combinations of  $t$  scalar products and do 2 multiplications in  $\mathbb{F}$  (the products in  $a_1a_2 - a_3 - a_4u$ ).

Assuming  $N_w \approx 2 \cdot N_g$ , for the whole verification phase, this adds up to  $\approx rn(2 \cdot \overline{S}(n, t+1)N_g + 3\overline{FFT}(2N_g) + 26nN_g)$  field operations, the transmission of  $\approx 4rn^2N_g$  elements of  $\mathbb{F}$ , and the broadcast of  $4rn^2$  elements of  $\mathbb{F}$  per party.

If there are complaints, then at most  $rn$  vectors of length  $N_c$  should be published in the message commitment phase, and at most  $rn$  vectors of length  $(4 + N_g + N_i)$  ( $\mathbf{p}$  shares),  $rn^2$  vectors of length  $N_c$  (communication shares),  $n \cdot (t+1)$  vectors of length  $N_r$  (randomness shares) and  $n$  vectors of length  $N_x$  (input shares) in the verification phase (per complaining party).

As long as there are no complaints, the only overheads that  $\mathcal{F}_{tr}$  causes is that each message is signed, and each signature is verified.

The knowledge error of the linear PCP of [5] is  $\varepsilon = 2N_g/\mathbb{F}$ , and the zero knowledge is  $\delta$ -statistical for  $\delta = N_g/\mathbb{F}$ . Hence desired error and the circuit size define the field size. If we do not want to use too large fields, then the proof can be parallelized as proposed in the end of Sec. 5.

## 7 Example: Verifiable Shamir Secret Sharing

In this section we show how our solution can be applied to [36], yielding a verifiable secret sharing (VSS) protocol. Any secret sharing scheme has two phases — sharing and reconstruction — to which the construction presented in this paper adds the verification phase.

To apply our construction, we have to define the arithmetic circuits used in [36]. For  $i \in \{1, \dots, n\}$  let  $C_i$  be a circuit taking  $s, r_1, \dots, r_t \in \mathbb{F}$  as inputs and returning  $s + \sum_{j=1}^t r_j i^j$ . If  $s$  is the secret to be shared, then  $C_i$  is the circuit used by the dealer (who is one of the parties  $P_1, \dots, P_n$ ) to generate the share for the  $i$ -th party using the randomness  $(r_1, \dots, r_t)$ . It computes a linear function, and has no multiplication gates. According to the LPCP construction that we use, each circuit should end with a multiplication. Hence we append a multiplication gate to it, the other argument of which is 1. Let  $C$  be the union of all  $C_i$ , it is a circuit with  $1 + t$  inputs and  $n$  outputs.

In the reconstruction phase, the parties just send the shares they've received to each other. A circuit computing the messages of this phase is trivial — it just copies its input to output. We note that  $\mathcal{F}_{tr}$  already provides the necessary publishing functionality for that. Hence we're not going to blindly follow our VMPC construction, but use this opportunity to optimize the protocol. In effect, this amounts to only verifying the sharing phase of the VSS protocol, and relying on  $\mathcal{F}_{tr}$  to guarantee the proper behaviour of parties during the reconstruction. The whole protocol is depicted in Fig. 8.

A couple of points are noteworthy there. First, the reconstruction and verification phases can take place in any order. In particular, verification could be seen as a part of the sharing, making a 3-round protocol (in optimistic case). Second, the activities of the dealer in the sharing phase have a dual role in terms of the VMPC construction. They form both the *input commitment* step in Fig. 3, as well as the execution step for actual sharing.

**Preprocessing.** Parties run the *Randomness generation and commitment* and *Randomness approval* steps of Fig. 3, causing the dealer to learn  $r_1, \dots, r_t$ . Each  $r_i$  is shared as  $r_{i1}, \dots, r_{in}$  between  $P_1, \dots, P_n$ .

**Sharing.** Dealer computes the shares  $s_1, \dots, s_n$  of the secret  $s$ , using the randomness  $r_1, \dots, r_t$  [36], and uses  $\mathcal{F}_{tr}$  to send them to parties  $P_1, \dots, P_n$ .

**Reconstruction.** All parties use the publish-functionality of  $\mathcal{F}_{tr}$  to make their shares known to all parties. The parties reconstruct  $s$  as in [36].

**Verification.** The dealer shares each  $s_i$ , obtaining  $s_{i1}, \dots, s_{in}$ . It transmits them all to  $P_i$ , which verifies that they are a valid sharing of  $s_i$  and then forwards each  $s_{ij}$  to  $P_j$ . [*Message commitment*]

The dealer computes  $\mathbf{w} = \text{witness}(C, s, r_1, \dots, r_t)$  and  $\mathbf{p} = \text{proof}(C, (s, r_1, \dots, r_t), \mathbf{w})$ . It shares  $\mathbf{p}$  as  $\mathbf{p}_1, \dots, \mathbf{p}_n$  and transmits  $\mathbf{p}_j$  to  $P_j$ . [*Proof commitment*]

Each party  $P_i$  generates a random  $\tau_i \in \mathbb{F}$  and broadcasts it. Let  $\tau = \tau_1 + \dots + \tau_n$ . Each party constructs  $\mathbf{q}_1, \dots, \mathbf{q}_4, \mathbf{q}_5, \mathbf{u} = \text{challenge}(C, \tau)$ . [*Challenge generation*]

Each party  $P_i$  computes  $a_{ji} = \langle (\mathbf{p}_i \| 1 \| s_i \| r_{1i} \| \dots \| r_{ti} \| s_{1i} \| \dots \| s_{ni}), \mathbf{q}_j \rangle$  for  $j \in \{1, 2, 3, 4\}$  and broadcasts them. The dealer may complain, in which case  $\mathbf{p}_i, s_i, r_{1i}, \dots, r_{ti}, s_{1i}, \dots, s_{ni}$  are made public and all parties repeat the computation of  $a_{ji}$ . [*Scalar product verification*]

Each party reconstructs  $a_1, \dots, a_4$  and verifies the LPCP equation (\*).

**Fig. 8.** LPCP-based VSS

Ignoring the randomness generation phase (which takes place offline), the communication complexity of our VSS protocol is the following. In sharing phase,  $(n-1)$  values (elements of  $\mathbb{F}$ ) are transmitted by the dealer and in the reconstruction phase, each party broadcasts a value. These coincide with the complexity numbers for non-verified secret sharing. In the verification phase, in order to commit to the messages, the dealer transmits a total of  $n(n-1)$  values to different parties. The same number of values are forwarded. According to Sec. 6, the proof  $\mathbf{p}$  contains  $t+n+4$  elements of  $\mathbb{F}$ . The proof is shared between parties, causing  $(n-1)(t+n+4)$  elements of  $\mathbb{F}$  to be transmitted. The rest of the verification phase takes place over the broadcast channel. In the optimistic case, each party broadcasts a value in the challenge generation and four values in the challenge verification phase. Hence the total cost of the verification phase is  $(n-1)(3n+t+4)$  point-to-point transmissions and  $5n$  broadcasts of  $\mathbb{F}$  elements.

We have evaluated the communication costs in terms of  $\mathcal{F}_{tr}$  invocations, and have avoided estimating the cost of implementing  $\mathcal{F}_{tr}$ . This allows us to have more meaningful comparisons with other VSS protocols. We will compare our solution to the 4-round statistical VSS of [27], the 3-round VSS of [33], and the 2-round VSS of [3] (see Table 1). These protocols have different security models and different optimization goals, therefore also selecting different methods for securing communication between parties. The number of field elements thus communicated is likely the best indicator of complexity.

**The 4-Round Statistical VSS of [27].** This information-theoretically secure protocol uses an *information checking protocol (ICP)* for transmission, which is a modified version of *ICP* introduced in [34]. The broadcast channel is also used.

**Table 1.** Comparing the Efficiency of VSS Protocols ( $tr$  transmissions,  $bc$  broadcasts)

	Rounds	Sharing	Reconstruction	Verification
Ours	7	$(n - 1) \cdot tr$	$n \cdot bc$	$(3n + t + 4)(n - 1) \cdot tr + 5n \cdot bc$
[27]	4	$3n^2 \cdot tr$	$O(n^2) \cdot tr$	0
[33]	3	$2n \cdot tr + (n + 1) \cdot bc$	$2n \cdot bc$	0
[3]	2	$4n^2 \cdot tr + 5n^2 \cdot bc$	$n^2 \cdot bc$	0

In the protocol, the dealer constructs a symmetric bivariate polynomial  $F(x, y)$  with  $F(0, 0) = s$ , and gives  $f_i(x) = F(i, x)$  to party  $P_i$ . Conflicts are then resolved, leaving the honest parties with a polynomial  $F^H(x, y)$  that allows the reconstruction of  $s$ . The distribution takes  $3n^2$  transmissions of field elements using the *ICP* functionality, while the conflict resolution requires  $4n^2$  broadcasts (in the optimistic case). The reconstruction phase requires each honest party  $P_i$  to send its polynomial  $f_i$  to all other parties using the *ICP* functionality, which again takes  $O(n^2)$  transmissions.

**The 3-Round VSS of [33].** Pedersen’s VSS is an example of a computationally secure VSS. The transmission functionality of this protocol is based on homomorphic commitments. Although the goal of commitments is also to ensure message delivery and make further revealing possible, they are much more powerful than  $\mathcal{F}_{tr}$  and *ICP*, so direct comparison is impossible. In the following, let  $Comm(m, d)$  denote the commitment of the message  $m$  with the witness  $d$ . We note that the existence of a suitable  $Comm$  is a much stronger computational assumption than the existence of a signature scheme sufficient to implement  $\mathcal{F}_{tr}$ .

To share  $s$ , the dealer broadcasts a commitment  $Comm(s, r)$  for a random  $r$ . It shares both  $s$  and  $r$ , using Shamir’s secret sharing with polynomials  $f$  and  $g$ , respectively. It also broadcasts commitments to the coefficients of  $f$ , using the coefficients of  $g$  as witnesses. This takes  $2n$  transmissions of field elements, and  $(n + 1)$  broadcasts (in the optimistic case). Due to the homomorphic properties of  $Comm$ , the correctness of any share can be verified without further communication. The reconstruction requires the shares of  $s$  and  $r$  to be broadcast; i.e. there are 2 broadcasts from each party.

**The 2-Round VSS of [3].** This protocol also uses commitments that do not have to be homomorphic. This is still different from  $\mathcal{F}_{tr}$  and *ICP*: commitments can ensure that the same message has been transmitted to distinct parties.

The protocol is again based on the use of a symmetric bivariate polynomial  $F(x, y)$  with  $F(0, 0) = s$  by the dealer. The dealer commits to all values  $F(x, y)$ , where  $1 \leq x, y \leq n$  and opens the polynomial  $F(i, x)$  for the  $i$ -th party. The reduction in rounds has been achieved through extra messages committed and sent to the dealer by the receiving parties. These messages can help in conflict resolution. In the optimistic case, the sharing protocol requires  $4n^2$  transmissions of field elements and  $5n^2$  broadcasts. The reconstruction protocol is similar to [27], with each value of  $F(x, y)$  having to be broadcast by one of the parties.

We see that the LPCP-based approach performs reasonably well in verifiable Shamir sharing. The protocols from the related works have less rounds, and

the 3-round protocol of [33] has also clearly less communication. However, for a full comparison we would also have to take into account the local computation, since operations on homomorphic commitments are more expensive. Also, the commitments may be based on more stringent computational assumptions than the signature-based communication primitives we are using. We have shown that the LPCP-based approach is at least comparable to similar VSS schemes. Its low usage of the broadcast functionality is definitely of interest.

## 8 Conclusions and Further Work

We have proposed a scheme transforming passively secure protocols to covertly secure ones, where a malicious party can skip detection only with negligible probability. The protocol transformation proposed here is particularly attractive to be implemented on top of some existing, highly efficient, passively secure SMC framework. The framework would retain its efficiency, as the time from starting a computation to obtaining the result at the end of the execution phase would not increase. Also, the overheads of verification, proportional to the number of parties, would be rather small due to the small number of *computing parties* in all typical SMC deployments (the number of *input* and *result parties* [7] may be large, but they can be handled separately).

The implementation would allow us to study certain trade-offs. Sec. 6 shows that the proof generation is still slightly superlinear in the size of circuits, due to the complexity of FFT. Shamir's secret sharing would allow the parties to commit to some intermediate values in their circuits, thereby replacing a single circuit with several smaller ones, and decreasing the computation time at the expense of communication. The usefulness of such modifications, and the best choice of intermediate values to be committed, would probably depend to large extent on the actual circuits.

Note that the verifications could be done after each round. This would give us security against active adversaries in a quite cheap manner, but would incur the overhead of the verification phase during the runtime of the actual protocol. The implementation will allow us to evaluate the usefulness of such transformation.

**Acknowledgements.** This work has been supported by Estonian Research Council through grant IUT27-1, ERDF through EXCS and STACC, ESF through the ICT doctoral school, and EU FP7 through grant no. 284731 (UaESMC).

## References

1. Arora, S., Safra, S.: Probabilistic Checking of Proofs: A New Characterization of NP. *J. ACM* 45(1), 70–122 (1998)
2. Aumann, Y., Lindell, Y.: Security against covert adversaries: Efficient protocols for realistic adversaries. *J. Cryptology* 23(2), 281–343 (2010)
3. Backes, M., Kate, A., Patra, A.: Computational verifiable secret sharing revisited. In: Lee, D.H., Wang, X. (eds.) ASIACRYPT 2011. LNCS, vol. 7073, pp. 590–609. Springer, Heidelberg (2011)

4. Baum, C., Damgård, I., Orlandi, C.: Publicly Auditable Secure Multi-Party Computation. In: Abdalla, M., De Prisco, R. (eds.) SCN 2014. LNCS, vol. 8642, pp. 175–196. Springer, Heidelberg (2014)
5. Ben-Sasson, E., Chiesa, A., Genkin, D., Tromer, E., Virza, M.: SNARKs for C: Verifying Program Executions Succinctly and in Zero Knowledge. In: Canetti, R., Garay, J.A. (eds.) CRYPTO 2013, Part II. LNCS, vol. 8043, pp. 90–108. Springer, Heidelberg (2013)
6. Bitansky, N., Chiesa, A., Ishai, Y., Ostrovsky, R., Paneth, O.: Succinct non-interactive arguments via linear interactive proofs. In: Sahai, A. (ed.) TCC 2013. LNCS, vol. 7785, pp. 315–333. Springer, Heidelberg (2013)
7. Bogdanov, D., Kamm, L., Laur, S., Pruulmann-Vengerfeldt, P.: Secure multi-party data analysis: end user validation and practical experiments. Cryptology ePrint Archive, Report 2013/826 (2013)
8. Bogdanov, D., Laud, P., Laur, S., Pullonen, P.: From Input Private to Universally Composable Secure Multi-party Computation Primitives. In: Proceedings of the 27th IEEE Computer Security Foundations Symposium, pp. 184–198. IEEE (2014)
9. Bogdanov, D., Niitsoo, M., Toft, T., Willemsen, J.: High-performance secure multi-party computation for data mining applications. *Int. J. Inf. Sec.* 11(6), 403–418 (2012)
10. Brickell, J., Shmatikov, V.: Privacy-preserving graph algorithms in the semi-honest model. In: Roy, B. (ed.) ASIACRYPT 2005. LNCS, vol. 3788, pp. 236–252. Springer, Heidelberg (2005)
11. Burkhart, M., Strasser, M., Many, D., Dimitropoulos, X.: SEPIA: Privacy-preserving aggregation of multi-domain network events and statistics. In: USENIX Security Symposium, Washington, DC, USA, pp. 223–239 (2010)
12. Camenisch, J., Hohenberger, S., Pedersen, M.Ø.: Batch verification of short signatures. In: Naor, M. (ed.) EUROCRYPT 2007. LNCS, vol. 4515, pp. 246–263. Springer, Heidelberg (2007)
13. Canetti, R.: Universally composable security: A new paradigm for cryptographic protocols. In: FOCS, pp. 136–145. IEEE Computer Society (2001)
14. Catrina, O., de Hoogh, S.: Secure multiparty linear programming using fixed-point arithmetic. In: Gritzalis, D., Preneel, B., Theoharidou, M. (eds.) ESORICS 2010. LNCS, vol. 6345, pp. 134–150. Springer, Heidelberg (2010)
15. Cramer, R., Damgård, I., Nielsen, J.B.: Multiparty computation from threshold homomorphic encryption. In: Pfitzmann, B. (ed.) EUROCRYPT 2001. LNCS, vol. 2045, pp. 280–299. Springer, Heidelberg (2001)
16. Damgård, I., Fitzi, M., Kiltz, E., Nielsen, J.B., Toft, T.: Unconditionally secure constant-rounds multi-party computation for equality, comparison, bits and exponentiation. In: Halevi, S., Rabin, T. (eds.) TCC 2006. LNCS, vol. 3876, pp. 285–304. Springer, Heidelberg (2006)
17. Damgård, I., Geisler, M., Krøigaard, M., Nielsen, J.B.: Asynchronous Multiparty Computation: Theory and Implementation. In: Jarecki, S., Tsudik, G. (eds.) PKC 2009. LNCS, vol. 5443, pp. 160–179. Springer, Heidelberg (2009)
18. Damgård, I., Geisler, M., Nielsen, J.B.: From passive to covert security at low cost. In: Micciancio, D. (ed.) TCC 2010. LNCS, vol. 5978, pp. 128–145. Springer, Heidelberg (2010)
19. Damgård, I., Keller, M., Larraia, E., Pastro, V., Scholl, P., Smart, N.P.: Practical Covertly Secure MPC for Dishonest Majority – Or: Breaking the SPDZ Limits. In: Crampton, J., Jajodia, S., Mayes, K. (eds.) ESORICS 2013. LNCS, vol. 8134, pp. 1–18. Springer, Heidelberg (2013)

20. Damgård, I., Pastro, V., Smart, N., Zakarias, S.: Multiparty computation from somewhat homomorphic encryption. In: Safavi-Naini, R., Canetti, R. (eds.) CRYPTO 2012. LNCS, vol. 7417, pp. 643–662. Springer, Heidelberg (2012)
21. Franklin, M., Gondree, M., Mohassel, P.: Communication-efficient private protocols for longest common subsequence. In: Fischlin, M. (ed.) CT-RSA 2009. LNCS, vol. 5473, pp. 265–278. Springer, Heidelberg (2009)
22. Gennaro, R., Gentry, C., Parno, B.: Non-interactive verifiable computing: Outsourcing computation to untrusted workers. In: Rabin, T. (ed.) CRYPTO 2010. LNCS, vol. 6223, pp. 465–482. Springer, Heidelberg (2010)
23. Gennaro, R., Gentry, C., Parno, B., Raykova, M.: Quadratic Span Programs and Succinct NIZKs without PCPs. In: Johansson, T., Nguyen, P.Q. (eds.) EUROCRYPT 2013. LNCS, vol. 7881, pp. 626–645. Springer, Heidelberg (2013)
24. Goldreich, O., Micali, S., Wigderson, A.: How to Play any Mental Game or A Completeness Theorem for Protocols with Honest Majority. In: STOC, pp. 218–229. ACM (1987)
25. Ishai, Y., Kushilevitz, E., Ostrovsky, R.: Efficient arguments without short PCPs. In: Twenty-Second Annual IEEE Conference on Computational Complexity, CCC 2007, pp. 278–291. IEEE (2007)
26. Kilian, J.: A note on efficient zero-knowledge proofs and arguments (extended abstract). In: Proceedings of the Twenty-fourth Annual ACM Symposium on Theory of Computing, STOC 1992, pp. 723–732. ACM, New York (1992), <http://doi.acm.org/10.1145/129712.129782>
27. Kumaresan, R., Patra, A., Rangan, C.P.: The round complexity of verifiable secret sharing: The statistical case. In: Abe, M. (ed.) ASIACRYPT 2010. LNCS, vol. 6477, pp. 431–447. Springer, Heidelberg (2010)
28. Laud, P., Pankova, A.: Verifiable Computation in Multiparty Protocols with Honest Majority. Cryptology ePrint Archive, report 2014/060 (2014)
29. Lipmaa, H.: Succinct non-interactive zero knowledge arguments from span programs and linear error-correcting codes. Cryptology ePrint Archive, report 2013/121 (2013)
30. Merkle, R.C.: Secrecy, authentication, and public key systems. Ph.D. thesis, Stanford University (1979)
31. Micali, S.: CS Proofs (Extended Abstract). In: FOCS, pp. 436–453. IEEE Computer Society (1994)
32. Parno, B., Howell, J., Gentry, C., Raykova, M.: Pinocchio: Nearly practical verifiable computation. In: IEEE Symposium on Security and Privacy, pp. 238–252. IEEE Computer Society (2013)
33. Pedersen, T.P.: Non-interactive and information-theoretic secure verifiable secret sharing. In: Feigenbaum, J. (ed.) CRYPTO 1991. LNCS, vol. 576, pp. 129–140. Springer, Heidelberg (1992)
34. Rabin, T., Ben-Or, M.: Verifiable secret sharing and multiparty protocols with honest majority (extended abstract). In: Johnson, D.S. (ed.) STOC, pp. 73–85. ACM (1989)
35. Setty, S.T.V., Vu, V., Panpalia, N., Braun, B., Blumberg, A.J., Walfish, M.: Taking proof-based verified computation a few steps closer to practicality. In: USENIX Security Symposium (2012)
36. Shamir, A.: How to share a secret. *Commun. ACM* 22(11), 612–613 (1979)
37. Yao, A.: Protocols for secure computations. In: Proceedings of the 23rd Annual Symposium on Foundations of Computer Science, pp. 160–164 (1982)