# Guaranteeing Correctness in Privacy-Friendly Outsourcing by Certificate Validation

Berry Schoenmakers and Meilof Veeningen Eindhoven University of Technology, The Netherlands

{berry@win.,m.veeningen@}tue.nl

Abstract. With computation power in the cloud becoming a commodity, it is more and more convenient to outsource computations to external computation parties. Assuring confidentiality, even of inputs by mutually distrusting inputters, is possible by distributing computations between different parties using multiparty computation. Unfortunately, this typically only guarantees correctness if a limited number of computation parties are malicious. If correctness is needed when all computation parties are malicious, then one currently needs either fully homomorphic encryption or "universally verifiable" multiparty computation; both are impractical for large computations. In this paper, we show for the first time how to achieve practical privacy-friendly outsourcing with correctness guarantees, by using normal multiparty techniques to compute the result of a computation, and then using slower verifiable techniques only to verify that this result was correct. We demonstrate the feasibility of our approach in a linear programming case study.

**Keywords:** multiparty computation, verifiable computation, multiparty Fiat-Shamir heuristic, secure linear programming, certificate validation

# 1 Introduction

When outsourcing a computation, we want to be sure that the result is correct. But if the computation involves confidential inputs, possibly of multiple mutually distrusting input parties, we also want to guarantee the privacy of the inputs. Separately, privacy and correctness can each be achieved. Correctness can be achieved by replicating a computation and comparing the results [CL02] (but this only protects against uncorrelated failure); or by relying on the use of trusted hardware by the computation party [SZJVD04]. Recent cryptographic techniques achieve correctness without assuming uncorrelated failure or trusted hardware, by instead producing cryptographic proofs of correctness [PHGR13].

Achieving privacy is hard when outsourcing to a single computation party, but feasible if the computation is distributed between several parties. Indeed, having a single computation party perform arbitrary computations on encryptions requires fully homomorphic encryption, a cryptographic primitive that is still impractical for realistic applications [DM14]. But distributing computations between multiple computation parties in a privacy friendly way is possible, and getting more and more practical, using multiparty computation protocols (e.g., [BCD<sup>+</sup>09,DKL<sup>+</sup>13]). Unfortunately, such protocols typically guarantee privacy and correctness only if at least one of the computation parties is honest.

If correctness needs to be guaranteed even if all computation parties are corrupted, then recent *universally verifiable* [SV15] (or *publicly auditable* [BDO14]) multiparty computation protocols can be used. Privacy is still only guaranteed up to a certain maximum of corruptions: we cannot hope to circumvent this in outsourcing scenarios (indeed, unconditional privacy requires fully homomorphic encryption). Unfortunately, also these universally verifiable techniques are not really practical: they require several cryptographic operations per multiplication in the computation circuit, which is too much for most practical computations.

In this paper, we show that universally verifiable multiparty computation can be made practical by only applying it to part of the computation. We observe that, for many computations, it is much easier to verify an obtained result than to obtain the result in the first place. For instance, the solution to a linear optimisation problem can be easily verified, but solving it with the simplex algorithm takes many iterations. Hence, we use normal, privacy-friendly multiparty computation techniques for the complicated computation, and slower verifiable techniques only to prove correctness. In this paper, we consider a particular instantiation of this idea (but other instantiations are possible): we distribute the computation between three parties using protocols that privacy if at most one of them is passively corrupted; and combine this with verifiability to ensure that found solutions are correct. We demonstrate that for linear programming, this leads to feasible performance; in particular, verifying the solution of the outsourced computation is much faster than performing the computation between the input parties with the same correctness guarantees.

*Outline* We first present a protocol for proving and verifying that a set of encryptions satisfy some given polynomial relations (Section 2). We then show how to combine this protocol with fast, non-verifiable multiparty computation (Section 3). We show with experiments that this gives rise to practical verifiable secure linear programming (Section 4). We finish with a discussion of related and future work (Section 5). Figure 1 shows notation used in this paper.

#### 2 Proving Polynomial Relations on ElGamal Encryptions

In this section, we show how to prove correctness of computation results in terms of ElGamal encryptions. Suppose that the encrypted inputs and outputs of a computation are known as ElGamal encryptions. Suppose, also, that the decryption key for these encryptions, as well as the plaintexts and randomness, are shared between two computation parties. We show how these parties can together produce a non-interactive zero-knowledge proof that the inputs and outputs satisfy a set of polynomial relations, without learning any values.

The overall idea of our protocol is the following. For each polynomial relation  $r(x_1, \ldots, x_n) = 0$  in the in/outputs  $x_1, \ldots, x_n$ , the parties produce an encryption R of the left-hand side. This requires additions, multiplications by a constant, and multiplications of two encryptions. The first two can be computed locally

$\mathcal{I}/\mathcal{P}/\mathcal{R}$	input/computation/result parties
parties $P \ / \ \mathbf{party} \ \mathcal{P} \ \mathbf{do} \ S$	let parties $P$ /party $\mathcal{P}$ (simultaneously) perform $S$
$Enc_{pk}(x;r)$	ElGamal encryption under $pk$ of $x$ with randomness $r$
$Dec_{sk}(x)$	ElGamal decryption of $x$ with secret key (share) sk
p	prime for ElGamal (plaintext/randomness space is $\mathbb{Z}_p$ )
$\oplus, \otimes$	addition, multiplication by constant of encryptions
$send(v;\mathcal{P});recv(\mathcal{P})$	send/receive $v$ over secure and private channel (if ${\mathcal P}$ is
	implicit, then communication is between $\mathcal{P}_1, \mathcal{P}_2$ )
bcast(v)	exchange $v$ over broadcast channel
NIZKVER $(\Sigma; v; \pi; aux)$	verification of Fiat-Shamir proof (p. 5)
$a \in_R S$	sample $a$ uniformly at random from $S$
[x], [x']	own/other party's additive share of $x$
$\mathcal{H}$	cryptographic hash function

Fig. 1. Notation in algorithms and protocols

using homomorphic properties of ElGamal. Multiplications of an encryption Y by an encryption  $X_i$  of a shared plaintext  $x_i$  can be performed verifiably by letting the parties verifiably multiply their shares, and combining correctness proofs on the shares into an overall correctness proof using techniques from [SV15]. Finally, the protocol produces a proof that R encrypts zero, again by homomorphically combining the decryption proofs of the computation parties.

This protocol is based on the universally verifiable protocols from [SV15], adapted to the two-party ElGamal setting. We first review the threshold homomorphic ElGamal cryptosystem and associated proofs of correct multiplication and decryption. We then provide the protocol for proving polynomial relations.

#### 2.1 Threshold ElGamal Cryptosystem and Zero-Knowledge Proofs

First, recall the additively homomorphic ElGamal cryptosystem. In this cryptosystem, the public key is a pair (g, h) of generators of a discrete logarithm group of size p (p prime) such that  $s = \log_g h$  is unknown; the private key is s; encryption of  $m \in \mathbb{Z}_p$  with randomness  $r \in \mathbb{Z}_p$  is  $(g^r, g^m h^r)$ ; and decryption of (a, b) is  $g^m = ba^{-s}$ . This cryptosystem is indeed additively homomorphic: if (a, b) encrypts m and (a', b') encrypts m', then  $(a \cdot a', b \cdot b')$  encrypts m + m'. Moreover, if (a, b) encrypts m, then  $(a^{\alpha}, b^{\alpha})$  encrypts  $m\alpha$ ; and  $(a^{\alpha}g^r, b^{\alpha}h^r)$  is a random encryption of  $m\alpha$ . Because ElGamal decrypts to  $g^m$  and not to m, it is only possible to decrypt small values for which the discrete logarithm problem with respect to g is tractable. Suitable discrete logarithm groups include groups of points on elliptic curves, e.g., [NIS99]. ElGamal is turned into a threshold cryptosystem in which two parties together can perform decryption, by sharing the private key s as  $s = s_1 + s_2$ ; parties publish their decryption shares  $a^{-s_i}$ , recombine to  $a^{-s}$ , and compute  $g^m$  from this [Ped91]. Verification keys  $v_i = g^{s_i}$  are published that the parties use to prove correctness of decryption shares.

The correctness of decryption shares and multiplications can be proven using  $\Sigma$ -protocols. Recall that a  $\Sigma$ -protocol for a binary relation R is a three-move protocol in which a potentially malicious prover convinces a honest verifier that

he knows a witness w for statement v such that  $(v, w) \in R$ . First, the prover sends an announcement (computed using algorithm  $\Sigma$ .ann) to the verifier; the verifier responds with a uniformly random *challenge*; and the prover sends his *response* (computed using algorithm  $\Sigma$ .res), which the verifier verifies (using predicate  $\Sigma$ .ver).  $\Sigma$ -protocols are defined as follows:

**Definition 1.** Let  $R \subset V \times W$  be a binary relation with language  $L_R = \{v \in V \mid \exists w \in W : (v, w) \in R\}$ . Let  $\Sigma$  be a collection of PPT algorithms  $\Sigma$ .ann,  $\Sigma$ .res,  $\Sigma$ .sim,  $\Sigma$ .ext, and polynomial time predicate  $\Sigma$ .ver. Let C be a finite set called the challenge space.  $\Sigma$  is a  $\Sigma$ -protocol for relation R if it satisfies:

- **Completeness** If  $(a, s) \leftarrow \Sigma$ .ann(v; w),  $c \in C$ , and  $r \leftarrow \Sigma$ .res(v; w; a; s; c), then  $\Sigma$ .ver(v; a; c; r).
- **Special soundness** If  $v \in V$ ,  $c \neq c'$ , and  $\Sigma$ .ver(v; a; c; r) and  $\Sigma$ .ver(v; a; c'; r')both hold, then  $w \leftarrow \Sigma$ .ext(v; a; c; c'; r; r') satisfies  $(v, w) \in R$ .
- **Special honest-verifier zero-knowledge** If  $v \in L_R$ ,  $c \in C$ , then  $(a, r) \leftarrow \Sigma.sim(v; c)$  has the same probability distribution as (a, r) obtained by  $(a, s) \leftarrow \Sigma.ann(v; w)$ ,  $r \leftarrow \Sigma.res(v; w; a; s; c)$ . If  $v \notin L_R$ , then  $(a, r) \leftarrow \Sigma.sim(v; c)$  satisfies  $\Sigma.ver(v; a; c; r)$ .

Completeness states that a protocol between a honest prover and verifier succeeds; special soundness essentially means that a successful prover must know the witness; special honest-verifier zero-knowledge essentially means that a honest verifier does not learn anything about the witness. For many  $\Sigma$ -protocols, the announcement a can be computed from the challenge c, statement v, the response r, denoted  $\Sigma$ .rea(c, v, r). This reduces space for storing non-interactive proofs. Finally, we need that announcements are "non-trivial" [SV15] in the sense that they are random from a large space. Our  $\Sigma$ -protocols indeed satisfy this.

For our proposes, we need three  $\Sigma$ -protocols: proof of knowledge  $\Sigma_{\rm PK}$ , proof of correct multiplication  $\Sigma_{\rm CM}$ , and proof of correct decryption  $\Sigma_{\rm CD}$ . These protocols are standard; we give them in the Appendix for completeness.  $\Sigma_{\rm PK}$  ( $\Sigma$ protocol 1) proves knowledge of plaintext y and randomness r used to encrypt  $(a,b) = (g^r, h^r g^y)$ .  $\Sigma_{\rm CM}$  ( $\Sigma$ -protocol 2) proves the following: given encryptions  $(a_1,b_1)$ ,  $(a_2,b_2)$ , and  $(a_3,b_3)$ , the prover knows y,r,s such that  $a_2 = g^r$  and  $b_2 = h^r g^y$  (i.e.,  $(a_2,b_2)$  encrypts plaintext y with randomness r); and  $a_3 = a_1^y g^s$ and  $b_3 = b_1^y h^s$  (i.e.,  $(a_3,b_3)$  encrypts the product encryption, randomised with s). For decryption, recall that each party has a share  $s_i$  of the decryption key  $s = \log_g h$ , so that its decryption share of  $(a_1, b_1)$  is  $a_1^{s_i}$ . Correctness of this decryption share is proven with respect to verification keys  $h_i = g^{s_i}$  generated as part of key generation; hence,  $\Sigma_{\rm CD}$  is a standard equality proof ( $\Sigma$ -protocol 3).

#### 2.2 Homomorphisms and Combined Proofs

 $\Sigma$ -protocols can be used to obtain non-interactive zero-knowledge proofs using the well-known *Fiat-Shamir heuristic* [FS86]. Namely, a party proves knowledge of a witness for statement v by generating announcement a using  $\Sigma$ .ann; setting challenge  $c = \mathcal{H}(v||a||aux)$  with some auxiliary information aux; and computing response r with  $\Sigma$ .ver. If  $\Sigma$ .rea exists, then the proof is (c, r), which a verifier accepts if NIZKVER $(\Sigma; v; c, r; aux) := \mathcal{H}(v||\Sigma$ .rea(v; c; r)||aux) = c holds. Security holds in the random oracle model [BR93] of idealised hash functions.

If a party needs to prove multiple statements  $v_i$  at the same time, then it is possible to use the same challenge for all the proofs by computing announcements  $a_i$  and setting  $c = \mathcal{H}(v_1||a_1||v_2||a_2||\ldots||aux)$ . Moreover, in our setting where encryption plaintexts and decryption keys are shared between two parties, it is possible to combine the two proofs about these shares into one proof about the overall encryption. This is done by exploiting "homomorphic properties" of  $\Sigma_{\text{PK}}$ ,  $\Sigma_{\text{CM}}$ , and  $\Sigma_{\text{CD}}$ . For instance, let  $a_i$  be valid announcements for proving that  $Z_i$  is a correct multiplication of X and  $Y_i$ , let c be a challenge, and let  $r_i$ be the responses to this challenge for the respective announcements  $a_i$ ; hence,  $\Sigma_{\text{CM}}.\text{ver}(X, Y_i, Z_i; a_i; c; r_i)$ . Then, it is possible to combine the  $a_i$  into a and  $r_i$ into r in such a way that (a, c, r) is a proof that  $\bigoplus_i Z_i$  is a correct multiplication of X and  $\bigoplus_i Y_i$ , that is, that  $\Sigma_{\text{CM}}.\text{ver}(X, \bigoplus_i Y_i, \bigoplus_i Z_i; a; c; r)$ . This combined proof can be shown to the verifier, who now needs to verify a single proof instead of two separate proofs from the computation parties.

In general, such "homomorphic properties" need to satisfy two properties [SV15]. First, the combination of valid transcripts is a valid transcript. Second, the combination of different honest announcements with the same corrupted announcements is likely to lead to a different combined announcement:

**Definition 2.** Let  $\Sigma$  be a  $\Sigma$ -protocol for  $R \subset V \times W$ . Let  $\Phi$  be a collection of partial functions  $\Phi$ .stmt,  $\Phi$ .ann, and  $\Phi$ .resp.  $\Phi$  is a homomorphism of  $\Sigma$  if:

- **Combination** Let c be a challenge; I a set of parties; and  $\{(v_i, a_i, r_i)\}_{i \in I}$  a collection of statements, announcements, and responses for parties in I. If  $\Phi$ .stmt( $\{v_i\}_{i \in I}$ ) is defined and for all i,  $\Sigma$ .ver( $v_i, a_i, c, r_i$ ) holds, then also  $\Sigma$ .ver( $\Phi$ .stmt( $\{v_i\}_{i \in I}$ );  $\Phi$ .ann( $\{a_i\}_{i \in I}$ ); c;  $\Phi$ .resp( $\{r_i\}_{i \in I}$ )).
- **Randomness** Let c be a challenge;  $C \subsetneq I$  sets of parties;  $\{v_i\}_{i \in I}$  statements such that  $\Phi$ .stmt( $\{v_i\}_{i \in I}$ ) is defined; and  $\{a_i\}_{i \in I \cap C}$  announcements. If for all  $i \in I \setminus C$ ,  $(a_i, \_), (a'_i, \_) \leftarrow \Sigma$ .sim $(v_i, c)$ , then with overwhelming probability,  $\Phi$ .ann( $\{a_i\}_{i \in I} \neq \Phi$ .ann( $\{a_i\}_{i \in I \cap C} \cup \{a'_i\}_{i \in I \setminus C}$ ).

The above  $\Sigma$ -protocols  $\Sigma_{\rm PK}$ ,  $\Sigma_{\rm CD}$  and  $\Sigma_{\rm CM}$  have simple homomorphisms. Homomorphism  $\Phi_{\rm PK}$  for  $\Sigma_{\rm PK}$  combines proofs of knowledge for  $(a_i, b_i)$  into a proof of knowledge for  $(\prod a_i, \prod b_i)$ . Homomorphism  $\Phi_{\rm CM}$  for  $\Sigma_{\rm CM}$  combines proofs of multiplication of  $(a_1, b_1)$  with several  $(a_2^i, b_2^i)$  into a proof of multiplication of  $(a_1, b_1)$  with  $(\prod a_2^i, \prod b_2^i)$ . Homomorphism  $\Phi_{\rm CD}$  for  $\Sigma_{\rm CD}$  combines proofs of correct decryption of a to shares  $d_i$  with respect to keys  $h_i$  into a proof of correct decryption of a to  $\prod d_i$  with respect to key  $h = \prod h_i$ . Each homomorphism is defined by taking the product of the announcements and the sum of the responses.

Now, a multiparty variant of the Fiat-Shamir heuristic can be applied to obtain combined non-interactive proofs. Namely, suppose the two parties want to provide a series of proofs for statements  $v_i = \Phi.\mathsf{stmt}(\{v'_i, v''_i\})$ . They exchange announcements  $a'_i, a''_i$  for their shares  $v'_i, v''_i$  of  $v_i$ ; compute  $a_i = \Phi.\mathsf{ann}(\{a'_i, a''_i\})$ ; take challenge  $h = \mathcal{H}(v_1||v_2||\ldots||a_1||a_2||\ldots||aux)$ ; and exchange responses  $r'_i, r''_i$ . Taking  $r_i = \Phi$ .resp $(\{r'_i, r''_i\})$ , the challenge *h* along with responses  $r_i$  prove collective knowledge of witnesses corresponding to statements  $v_i$ . (For security reasons, the second party should not be able to choose  $a''_i$  based on  $a'_i$ . Therefore, the first party first provides a hash of its announcements; the second party then provides its announcements, after which the first party opens the hash.)

#### 2.3 Proving and Verifying Polynomial Relations

Protocol 1 shows our POLYPROVE protocol for producing a proof that ElGamal encryptions  $X_1, \ldots, X_n$  satisfy a given set of polynomial relations. We assume that the polynomial relations are given as equations  $x_j = 0$   $(1 \le j \le N$  for some  $N \ge n$ ), and an arithmetic circuit to compute these values  $x_j$ . This arithmetic circuit consists of gates  $x_k = v$ ,  $x_k = x_i + x_j$ ,  $x_k = x_i \cdot v$ , and  $x_k = x_i \cdot x_j$  (v any constant). For the multiplication gate, we require  $1 \le j \le n$  because we will use the additive shares of  $[x_j]$  of their plaintexts. Clearly, any set of polynomial relations can be described like this.

The first step of the protocol is to evaluate the circuit (lines 3–10) to obtain encryptions  $X_{n+1}, \ldots, X_N$ . All gates except  $x_k = x_i \cdot x_j$  can be evaluated locally; for  $x_k = x_i \cdot x_j$ , the parties use their additive shares of the plaintext of  $X_j$ to obtain shares of  $X_k$ , randomised using randomness  $[s_k]$ . Then, the parties compute announcements for the proofs of correctness of their multiplications (line 10). They use these announcements to make combined multiplication proofs as described in Section 2.2: they exchange (line 11–15) and combine (line 17) their announcements; compute one overall challenge (line 18); and compute (line 19– 20), exchange (line 21), and combine (line 23) the responses. For each equation  $x_j = 0$ , they compute decryption share  $[d_k]$  (lines 24–25) and produce a combined proof that decryption is to zero in the same way. (Note that the multiplication and decryption proofs cannot use the same challenge: for security, values  $X_k$ should be decrypted only after the multiplication proofs have been verified.) The overall proof consists of the encrypted products, the challenges, and the responses.

Algorithm 1 shows how to check if the proof produced by POLYEVAL is correct. First, all missing encryptions  $\in \{X_{n+1}, \ldots, X_N\}$  are computed, i.e., of gates that are not inputs or multiplication results (line 3–6). Then, the announcements for all multiplication (line 7) and decryption (line 8) proofs are computed. The proof is correct when these announcements hash to challenges  $h_1$ ,  $h_2$  (line 9).

# 3 Combining Computation with Certificate Validation

In this section, we present our main protocol for privacy-friendly outsourcing with correctness guarantees. In a nutshell, we compute a solution using normal multiparty computation techniques, and then produce a proof of correctness of this solution using the ElGamal-based proofs from Section 2.

The goal of our protocol is the following. We have m input parties  $\mathcal{I}_1, \ldots, \mathcal{I}_m$ , who want to perform a computation on their respective inputs  $\boldsymbol{x} = x_1, \ldots, x_m$ . The input parties do not trust each other, so their inputs should be hidden from

**Require:**  $\mathcal{G}$  arithmetic circuit for  $x_{n+1}, \ldots, x_N$  with multiplication gates  $\mathcal{M} \subset \mathcal{G}$ ; set  $\mathcal{E}$  of equations  $x_k = 0; X_1, \ldots, X_n$  ElGamal encryptions s.t.  $X_i = \mathsf{Enc}_{\mathsf{pk}}([x_i]; [r_i])$ **Ensure:**  $h_1, \{X_k, r_k\}_{k \in \mathcal{M}}, h_2, \{R_k\}_{k \in \mathcal{E}}$  prove that equations in  $\mathcal{E}$  hold for  $X_1, \ldots, X_N$ 1: protocol POLYPROVE<sup> $\mathcal{E},\mathcal{G}$ </sup>(pk; [pk]; [sk];  $X_1, \ldots, X_n; [x_1], \ldots, [x_n]; [r_1], \ldots, [r_n]$ ) 2: parties  $\{\mathcal{P}_1, \mathcal{P}_2\}$  do 3: for all gates  $\in \mathcal{G}$  do  $\triangleright$  evaluate circuit; make  $\Sigma_{\rm CM}$  announcements if (constant gate  $x_k = v$ ) then  $X_k \leftarrow \mathsf{Enc}_{\mathsf{pk}}(c; 0)$ 4: **if** (addition gate  $x_k = x_i + x_j$ ) **then**  $X_k \leftarrow X_i \oplus X_j$ 5: **if** (multiplication gate  $x_k = x_i \cdot v$ ) **then**  $X_k \leftarrow X_i \otimes v$ 6: if (multiplication gate  $x_k = x_i \cdot x_j, 1 \leq j \leq n$ ) then 7:  $[r_k] \in_r \mathbb{Z}_p; [X_k] \leftarrow (X_i \otimes [x_i]) \oplus \mathsf{Enc}_{\mathsf{pk}}(0; [r_k])$ 8: 9:  $send([X_k]); [X'_k] \leftarrow recv(); X_k \leftarrow [X_k] \oplus [X'_k]$  $([a_k], s_k) \leftarrow \Sigma_{\text{CM}}.ann(X_i, [X_j], [X_k]; [x_j], [r_j], [r_k])$ 10:▷ build combined proofs of correct multiplication party  $\mathcal{P}_1$  do 11:  $h \leftarrow \mathcal{H}(\{[a_k]\}_{k \in \mathcal{M}}); \operatorname{send}(h); \{[a'_k]\}_{k \in \mathcal{M}} \leftarrow \operatorname{recv}(); \operatorname{send}(\{[a_k]\}_{k \in \mathcal{M}})$ 12:13:party  $\mathcal{P}_2$  do  $h \leftarrow \operatorname{recv}(); \operatorname{send}(\{[a_k]\}_{k \in \mathcal{M}}); \{[a'_k]\}_{k \in \mathcal{M}} \leftarrow \operatorname{recv}()$ 14: if  $h \neq \mathcal{H}(\{[a'_k]\}_{k \in \mathcal{M}})$  then fail 15:16:parties  $\{\mathcal{P}_1, \mathcal{P}_2\}$  do for all  $k \in \mathcal{M}$  do  $a_k \leftarrow \Phi_{CM}$ .ann $([a_k], [a'_k])$ 17:18: $h_1 \leftarrow \mathcal{H}(\{a_k\}_{k \in \mathcal{M}})$ 19:for all multiplication gates  $x_k = x_i \cdot x_j, 1 \leq j \leq n$  in  $\mathcal{M}$  do  $[r_k] \leftarrow \Sigma_{CM}.\mathsf{res}(X_i, [X_j], [X_k]; [x_j], [r_j], [r_k]; [a_k]; s_k; h_1)$ 20: $\operatorname{send}(\{[r_k]\}_{k \in \mathcal{M}}); \{[r'_k]\}_{k \in \mathcal{M}} \leftarrow \operatorname{recv}()$ 21: 22:for all  $k \in \mathcal{M}$  do if  $\neg \Sigma_{CM}$ .ver $([a'_k]; h_1; [r'_k])$  then fail for all  $k \in \mathcal{M}$  do  $r_k \leftarrow \Phi_{CM}$ .resp $([r_k]; [r'_k])$ 23:for all  $(x_k = 0) \in \mathcal{E}$  do  $[d_k] \leftarrow \mathsf{Dec}_{[\mathsf{sk}]}(X_k)$ 24: $\triangleright$  decrypt equations; prove 25:for all  $(x_k = 0) \in \mathcal{E}$  do  $([A_k], S_k) \leftarrow \Sigma_{CD}$ .ann $(X_k, [d_k], [\mathsf{pk}]; [\mathsf{sk}])$ party  $\mathcal{P}_1$  do 26:27: $h \leftarrow \mathcal{H}(\{[A_k]\}_{k \in \mathcal{E}}); \operatorname{send}(h); \{[A'_k]\}_{k \in \mathcal{E}} \leftarrow \operatorname{recv}(); \operatorname{send}(\{[A_k]\}_{k \in \mathcal{E}})$ 28:party  $\mathcal{P}_2$  do 29: $h \leftarrow \operatorname{recv}(); \operatorname{send}(\{[A_k]\}_{k \in \mathcal{E}}); \{[A'_k]\}_{k \in \mathcal{E}} \leftarrow \operatorname{recv}()$ 30: if  $h \neq \mathcal{H}(\{[A'_k]\}_{k \in \mathcal{E}})$  then fail parties  $\{\mathcal{P}_1, \mathcal{P}_2\}$  do 31: 32: for all  $k \in \mathcal{E}$  do  $A_k \leftarrow \Phi_{CD}$ .ann $([A_k], [A'_k])$ 33:  $h_2 \leftarrow \mathcal{H}(\{A_k\}_{k \in \mathcal{E}})$ for all  $(x_k = 0) \in \mathcal{E}$  do  $[R_k] \leftarrow \Sigma_{CD}$ .res $(X_k, [d_k], [pk]; [sk]; [A_k]; S_k; h_2)$ 34:35:send({ $[R_k]$ }<sub> $k \in \mathcal{E}$ </sub>); { $[R'_k]$ }<sub> $k \in \mathcal{E}$ </sub>  $\leftarrow$  recv() 36: for all  $k \in \mathcal{E}$  do if  $\neg \Sigma_{CD}$ .ver $([A'_k]; h_2; [R'_k])$  then fail for all  $k \in \mathcal{E}$  do  $R_k \leftarrow \Phi_{\text{CD}}.\mathsf{resp}([R_k], [R'_k])$ 37: 38: return  $(h_1, \{X_k, r_k\}_{k \in \mathcal{M}}, h_2, \{R_k\}_{k \in \mathcal{E}})$   $\triangleright$  return mult., decryption proofs

each other, and they should not be able to adaptively choose their inputs based on those of others. The computation is given by a function (a, r) = f(x), where r is the outcome of the computation, and a is a certificate. The correctness of the computation can be efficiently verified by means of a predicate  $\phi(x, a, r)$ 

Protocol 1 POLYPROVE: Prove polynomial equations over ElGamal ciphertexts

Algorithm 1 POLYVER: Verify polynomial equations over ElGamal ciphertexts

**Require:**  $\mathcal{G}$  is an arithmetic circuit for  $x_{n+1}, \ldots, x_N$  with multiplication gates  $\mathcal{M} \subset \mathcal{G}$ ;  $\mathcal{E}$  is a set of equations  $x_k = 0$ 

**Ensure:** all equations in  $\mathcal{E}$  hold for  $X_1, \ldots, X_N$ 

1: function POLYVER<sup> $\mathcal{E},\mathcal{G}$ </sup>(pk;  $X_1, \ldots, X_n; h_1, \{X_k, r_k\}_{k \in \mathcal{M}}, h_2, \{R_k\}_{k \in \mathcal{E}}$ )

- $2: \qquad \triangleright \ {\rm determine \ encryptions \ for \ all \ gates}$
- 3: for all gates  $\in \mathcal{G}$  do
- 4: **if** (constant gate  $x_k = v$ ) **then**  $X_k \leftarrow \mathsf{Enc}_{\mathsf{pk}}(c; 0)$
- 5: **if** (addition gate  $x_k = x_i + x_j$ ) **then**  $X_k \leftarrow X_i \oplus X_j$
- 6: **if** (multiplication gate  $x_k = x_i \cdot v$ ) **then**  $X_k \leftarrow X_i \otimes v$
- 7: **for all** multiplications  $x_k = x_i \cdot x_j$  in  $\mathcal{M}$  do  $a_k \leftarrow \Sigma_{CM}$ .rea $(h_1; X_i, X_j, X_k; r_k)$
- 8: for all equations  $x_k = 0$  in  $\mathcal{E}$  do  $A_k \leftarrow \Sigma_{CD}$ .rea $(h_2; X_k, g^0, \mathsf{pk}; R_k)$
- 9: **return**  $h_1 \stackrel{?}{=} \mathcal{H}(\{a_k\}_{k \in \mathcal{M}}) \wedge h_2 \stackrel{?}{=} \mathcal{H}(\{A_k\}_{k \in \mathcal{E}})$

consisting of a set of polynomial relations. If  $(\boldsymbol{a}, \boldsymbol{r}) = f(\boldsymbol{x})$ , then  $\phi(\boldsymbol{x}, \boldsymbol{a}, \boldsymbol{r})$ , but we do not demand the converse: the outcome of the computation might not be unique, and  $\phi$  might merely check that *some* correct solution was found, not that it was produced according to algorithm f. The computation is distributed among three computation parties  $\mathcal{P}_1, \mathcal{P}_2, \mathcal{P}_3$ . They do not learn anything about the inputs under the security assumptions of the protocol used to compute f: in our case, if at most one of them is passively corrupted. A result party  $\mathcal{R}$  obtains the result (we later discuss changes when multiple parties need to get the result).

To compute (a, r) = f(x), we use passively secure multiparty computation protocols based on (2, 3)-Shamir sharing. In these protocols, private values are information-theoretically shared between three parties such that two parties are needed to recover the value. Protocols exist to, e.g., multiply, bit-decompose, compare, and open these shared values (see [dH12] for an overview); these protocols are secure against an adversary passively corrupting up to one party. Note that the computation of f involves three parties and uses Shamir shares, whereas the POLYEVAL involves two parties and uses additive shares. It is easy to switch between the two: two parties holding additive shares can Shamir-share them among all three; and two of the three parties holding Shamir shares can locally convert them to additive shares by Lagrange interpolation.

### 3.1 The VERMPC Protocol

Given a protocol to compute  $([a_1], \ldots, [a_k], [r_1], \ldots, [r_l]) \leftarrow f([x_1], \ldots, [x_m])$  and the protocol POLYPROVE  $\mathcal{E}_{\phi}, \mathcal{G}_{\phi}(X_1, \ldots; [x_1], \ldots; [r_{x,1}], \ldots)$  to prove that this result is correct, the question is how to combine them in a secure way. Our protocol VERMPC (Protocol 2) achieves this with the following steps (cf. Figure 2):

Step 1 First, the input parties announce their inputs. The input parties encrypt their respective inputs (line 3), and make a proof of knowledge of the corresponding plaintext (lines 4–5). These encryptions and proofs are broadcast. To prevent corrupted parties from adaptively choosing their input based on the inputs of others, this happens in two rounds: first, the parties provide hashes as

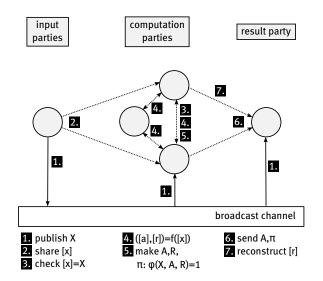


Fig. 2. Overview of the VERMPC protocol (dotted lines are private, secure channels)

a commitment to their inputs; then they open the commitments (line 6). If any party provides an incorrect input, the protocol is terminated (line 7).

Step 2 Next, the parties provide the plaintext x and randomness s of the encryption to the two computation parties who will later perform the POLYPROVE protocol, in additively secret-shared form (line 8).

Step 3 The two computation parties check if the provided sharing of the input is consistent with the encryptions that were broadcast in step 1. (Without this check, corrupted input parties could learns information about both their encrypted and their secret-shared inputs, which should not be possible.) They do this by simply encrypting their shares of the inputs using their shares of the randomness; exchanging the result; and checking correctness using the homomorphic property of the cryptosystem (lines 11–12).

Step 4 Then, the actual computation takes place (line 13). This is the only step that involves the third computation party. The two parties holding additive shares of the input Shamir-share them between all three computation parties; then the computation is performed between the three parties; and finally,  $\mathcal{P}_1$  and  $\mathcal{P}_2$  locally convert their Shamir shares to additive shares  $[a_i], [r_i]$ .

Step 5 Two of the computation parties produce the encrypted result and prove its correctness. First, they exchange encryptions of their respective additive shares of the certificate and result (line 15–20). Then, they run the POLYPROVE protocol from Section 2.3 to obtain a proof that  $\phi(X, A, R) = 1$  (line 21). The arithmetic circuit for  $\phi$  should be such that each certificate value  $A_i$  and result value  $R_i$  occurs at least once as right-hand side of a multiplication: because the

**Protocol 2** VERMPC: Verifiable computation by certificate validation

**Require:** pk/sk ElGamal public/secret keys shared between  $\mathcal{P}_1, \mathcal{P}_2; \{x_i\}_{i \in \mathcal{I}}$  inputs **Ensure:** Party  $\mathcal{R}$  returns  $r_1, \ldots, r_l$  s.t.  $\phi(x_1, \ldots, a_1, \ldots, r_1, \ldots)$  for some  $\{a_i\}$ , or  $\perp$ 1: protocol VERMPC<sup> $f,\phi$ </sup>(pk; [pk]; [sk]; { $x_i$ }<sub> $i \in \mathcal{I}$ </sub>) parties  $\mathcal{I}_1, \ldots, \mathcal{I}_m$  do 2: ⊳ step 1 3:  $r_{\mathbf{x},i} \in_R \mathbb{Z}_p; X_i \leftarrow \mathsf{Enc}_{\mathsf{pk}}(x_i; r_{\mathbf{x},i})$ 4:  $(a_i, s_i) \leftarrow \Sigma_{\text{PK}}.ann(X_i; x_i, r_{\mathbf{x},i}); c_i \leftarrow \mathcal{H}(X_i ||a_i||i)$ 5: $r_i \leftarrow \Sigma_{\text{PK}}.\text{res}(X_i; x_i, r_{\mathbf{x},i}; a_i; s_i; c_i); \pi_{\mathbf{x},i} \leftarrow (c_i, r_i)$ 6:  $h_i \leftarrow \mathcal{H}(i||X_i||\pi_{\mathbf{x},i}); \mathsf{bcast}(h_i); \mathsf{bcast}(X_i, \pi_{\mathbf{x},i})$ if  $\exists j : h_j \neq \mathcal{H}(j||X_j||\pi_{\mathbf{x},j}) \lor \neg \operatorname{NIZKVer}(\Sigma_{\operatorname{PK}};X_j;\pi_{\mathbf{x},j};j)$  then return  $\bot$ 7:  $x'_{i} \in_{R} \mathbb{Z}_{p}; r'_{x,i} \in_{R} \mathbb{Z}_{p}; \text{send}(x'_{i}, r'_{x,i}; \mathcal{P}_{1}); \text{send}(x_{i} - x'_{i}, r_{x,i} - r'_{x,i}; \mathcal{P}_{2})$ 8: parties  $\{\mathcal{P}_1, \mathcal{P}_2\}$  do 9: for all  $1 \leq i \leq m$  do 10:  $[x_i], [r_{\mathbf{x},i}] \leftarrow \mathsf{recv}(\mathcal{I}_i); [X_i] \leftarrow \mathsf{Enc}_{\mathsf{pk}}([x_i]; [r_{\mathbf{x},i}]); \mathsf{send}([X_i])$ 11:  $\triangleright$  step 3  $[X'_i] \leftarrow \mathsf{recv}(); \text{ if } X_i \neq [X_i] \oplus [X'_i] \text{ then return } \bot$ 12:parties  $\{\mathcal{P}_1, \mathcal{P}_2, \mathcal{P}_3\}$  do  $([a_1], \dots, [a_k], [r_1], \dots, [r_l]) \leftarrow f([x_1], \dots, [x_m]) \triangleright$ st 4 13:parties  $\{\mathcal{P}_1, \mathcal{P}_2\}$  do 14: $\triangleright$  step 5 15:for all  $1 \leq i \leq k$  do  $[r_{\mathbf{a},i}] \in_R \mathbb{Z}_p; [A_i] \leftarrow \mathsf{Enc}_{\mathsf{pk}}([a_i]; [r_{\mathbf{a},i}]); \operatorname{send}([A_i])$ 16: $[A'_i] \leftarrow \mathsf{recv}(); A_i \leftarrow [A_i] \oplus [A'_i]$ 17:for all  $1 \le i \le l$  do 18: $[r_{\mathbf{r},i}] \in_R \mathbb{Z}_p; [R_i] \leftarrow \mathsf{Enc}_{\mathsf{pk}}([r_i]; [r_{\mathbf{r},i}]); \operatorname{send}([R_i])$ 19: $[R'_i] \leftarrow \mathsf{recv}(); R_i \leftarrow [R_i] \oplus [R'_i]$ 20: $\pi \leftarrow \text{POLYPROVE}^{\mathcal{E}_{\phi}, \mathcal{G}_{\phi}}(\mathsf{pk}; [\mathsf{pk}]; [\mathsf{sk}]; X_1, \dots, R_l; [x_1], \dots; [r_{\mathbf{x}, 1}], \dots)$ 21: 22: $send(\{[r_i], [r_{r,i}]\}_{i=1,...,l}; \mathcal{R})$  $\triangleright$  step 6 party  $\mathcal{P}_1$  do send $(A_1, \ldots, A_k, \pi; \mathcal{R})$ 23: $\triangleright$  step 7 24:party  $\mathcal{R}$  do  $\{[r_i]^{(1)}, [r_{\mathbf{r},i}]^{(1)}\}_{i=1,...,l} \leftarrow \mathsf{recv}(\mathcal{P}_1); \, \{[r_i]^{(2)}, [r_{\mathbf{r},i}]^{(2)}\}_{i=1,...,l} \leftarrow \mathsf{recv}(\mathcal{P}_2)$ 25: $(A_1,\ldots,A_k,\pi) \leftarrow \mathsf{recv}(\mathcal{P}_1)$ 26:for all  $1 \leq i \leq m$  do if  $\neg NIZKVER(\Sigma_{PK}; X_i; \pi_{x,i}; j)$  then return  $\bot$ 27:for all  $1 \le i \le l$  do  $R_i \leftarrow \mathsf{Enc}_{\mathsf{pk}}([r_i]^{(1)} + [r_i]^{(2)}; [r_{r,i}]^{(1)} + [r_{r,i}]^{(2)})$ 28:if  $\neg \text{POLYVER}^{\mathcal{E}_{\phi}, \mathcal{G}_{\phi}}(\mathsf{pk}; X_1, \dots, R_l; \pi)$  then return  $\bot$ 29:30: return  $(r_1,\ldots,r_l)$ 

computation parties prove knowledge of these right-hand sides, this guarantees that they know the corresponding plaintexts, which our security proof requires.

Step 6 The computation parties send their additive shares of the result and the randomness of their encryption shares  $[R_i]$  to the result party (line 22).

Step 7 One of the computation parties also sends the encryptions of the certificate and computation result and their proof of correctness (line 23). The result party checks the proofs of knowledge provided by the input parties which it reads from the bulletin board (line 27); computes the encrypted result  $R_1, \ldots, R_l$  from the shared inputs (line 28); and calls POLYVER to verify correctness (line 29). If the proof checks out, then the plaintext result  $r_1, \ldots, r_l$  is the outcome of the computation (line 30).

Algorithm 2 Trusted party: verifiability by certificate validation w/threshold tRequire: Parties C corrupted,  $A \subset C$  actively corrupted

1: function IVERMPC<sup> $f,\phi$ </sup>(C, A) 2:  $\triangleright$  input phase 3: for all  $\mathcal{I}_i \in \mathcal{I} \setminus C$  do  $x_i \leftarrow \mathsf{recv}(\mathcal{I}_i)$ 4:  $\{x_i\}_{i \in \mathcal{I} \cap C} \leftarrow \mathsf{recv}(\mathcal{S})$ 5: if  $\mathcal{P} \cap A \neq \emptyset \lor |\mathcal{P} \cap C| \ge t$  then send $(\{x_i\}_{i \in \mathcal{I} \setminus C}; \mathcal{S})$  $\triangleright$  computation phase 6: 7: if  $\mathcal{P} \cap A = \emptyset$  then  $a_1, \ldots, r_l \leftarrow f(x_1, \ldots, x_m)$  else  $a_1, \ldots, r_l \leftarrow \mathsf{recv}(\mathcal{S})$ 8: if  $\exists i: x_i = \bot \lor \neg \phi(x_1, \ldots, x_m, a_1, \ldots, a_k, r_1, \ldots, r_l)$  then  $r_1, \ldots, r_l \leftarrow \bot$ 9:  $\triangleright$  result phase 10:  $send(r_1 \ldots, r_l; \mathcal{R})$ 

#### 3.2 Security Model

We prove security of our protocol using the standard formalism used for multiparty computation: the ideal/real world paradigm [Can98]. We demand that the outputs of the result party and the adversary in a protocol execution are distributed similarly to those outputs in an ideal world where the function is computed by an incorruptible trusted party. Because in the ideal world, the result party obtains the correct result and the adversary does not learn anything it should not learn, the same must be true in the real world.

More precisely, let C be a set of corrupted parties, of which A are actively corrupted. Let k be a security parameter. Let adversary  $\mathcal{A}$  be a probabilistic polynomial time Turing machine. Define real-world execution

$$\operatorname{REAL}_{\operatorname{VerMPC}^{f,\phi},\mathcal{A}}^{C,A}(k,x_1,\ldots,x_m)$$

as the distribution consisting of the output of the result party  $\mathcal{R}$  and the adversary  $\mathcal{A}$  in an execution of the protocol (see [Can98,SV15] for details). This execution consists of a secure set-up of the threshold ElGamal cryptosystem, returning public key pk shared as threshold public/private keys [pk], [sk]; followed by an execution of the protocol VERMPC<sup> $f,\phi$ </sup>(pk; [pk]; [sk]; { $x_i$ }<sub> $i \in \mathcal{I}$ </sub>) with adversary  $\mathcal{A}$ .

Similarly, the ideal-world execution given set C of corrupted parties of which A active, adversary S, security parameter k, and inputs  $x_1, \ldots, x_m$  is called

IDEAL<sup>C,A</sup><sub>IVERMPC<sup>f,\phi</sup>,S</sub>
$$(k, x_1, \ldots, x_m)$$

and is defined as the distribution consisting of the outputs of the result party  $\mathcal{R}$  and the adversary  $\mathcal{S}$  in an ideal-world protocol execution. In this ideal-world execution, all parties communicate securely with an incorruptible trusted party  $\mathcal{T}$  executing algorithm IVERMPC<sup> $f,\phi$ </sup> (Algorithm 2). Honest input parties send their inputs to  $\mathcal{T}$ ; a honest result party outputs the values it receives from  $\mathcal{T}$ ; the adversary  $\mathcal{S}$  can send arbitrary messages to  $\mathcal{T}$  and return an arbitrary value.

Algorithm IVERMPC<sup> $f,\phi$ </sup> executed by the trusted party  $\mathcal{T}$  prescribes the outcome of the computation and the information learned by the adversary. The

trusted party first obtains the inputs from the honest input parties (line 3) and from the adversary on behalf of the corrupted input parties (line 4). If there are actively corrupted computation parties, or if the number of corrupted computation parties exceeds a certain threshold (in our case, t = 2), then we no longer guarantee privacy, so we send all honest inputs to the adversary (line 5). Now, the computation takes place. If there are no active corruptions, then the certificate  $a_1, \ldots, a_k$  and function result  $r_1, \ldots, r_l$  are computed according to function f (line 7). If there are active corruptions, we can no longer guarantee that f will be correctly computed, so we ask the adversary to supply  $a_1, \ldots, r_l$  (line 7); but we do guarantee that  $\phi$  holds, so  $\mathcal{T}$  checks if  $\phi$  holds or sets the outcome to  $\perp$ (line 8). Note that, if one of the inputs is  $\perp$ , then the computation output is  $\perp$ . Hence, in this model, any input party is able to prevent the computation from giving an output. Finally, the result is sent to  $\mathcal{R}$  (line 10).

**Definition 3.** Protocol  $\Pi$  is a secure multiparty protocol with verifiability by certificate validation with threshold t if, for all probabilistic polynomial time adversaries corrupting set C of parties and actively corrupting  $A \subset C$ , there exists a probabilistic polynomial time adversary S such that for all possible inputs  $x_1, \ldots, x_m$ :

$$\operatorname{REAL}_{\Pi,\mathcal{A}}^{C,A}(k,x_1,\ldots,x_m) \approx \operatorname{IDEAL}_{\operatorname{IVerMPC}^{f,\phi},\mathcal{S}}^{C,A}(k,x_1,\ldots,x_m),$$

where  $\approx$  denotes computational indistinguishability in security parameter k.

**Theorem 1.** Protocol VERMPC is a secure multiparty protocol with verifiability by certificate validation with threshold t = 2 in the random oracle model.

We prove this theorem in Appendix A.

Because we use the Fiat-Shamir heuristic for non-interactive zero-knowledge proofs, our construction is only secure in the random oracle model [BR93]. In this model, evaluations of the hash function  $\mathcal{H}$  are modelled as queries to a "random oracle"  $\mathcal{O}$  that evaluates a perfectly random function. Although security in the random oracle model does not generally imply security in the standard model [GK03], the model is commonly used to devise simple and efficient protocols, and no security problems due to its use are known [Wee09]. In particular, our variant of the model [Wee09,SV15] assumes that the random oracle has not been used before the protocol starts: in practice, it should be instantiated with a keyed hash function, with every computation using a fresh random key.

#### 3.3 Extensions

Input range checking The multiparty computation protocols used to compute f may only guarantee correctness and privacy if their inputs x are bounded, e.g.,  $-2^k \leq x \leq 2^k$ . To guarantee that the inputs of corrupted parties lie in this range, it is possible to use statistically secure additive shares over the integers in line 8 of the protocol, i.e., by choosing  $x'_i$  at random from  $[-2^{k-1}, \ldots, 2^{k-1}]$ . The computation parties check if the shares they receive in line 11 lie in this range. Privacy of honest inputs is guaranteed if they are smaller than  $2^{k-1}$  by a statistical security parameter.

Multiple result parties and universal verifiability In our model, only one party learns the result. If multiple parties need to learn the result, then the encrypted outputs  $R_1, \ldots, R_l$  should be broadcast to ensure consistency. Note that we cannot guarantee fairness as the computation parties can always choose to send their shares of the result to some result parties but not others.

At the end of the protocol, the result party obtains not only the result; but also a non-interactive zero-knowledge proof that this result is correct. In particular, the result party can also convince third parties that the encrypted outputs  $R_1, \ldots, R_l$  are correct. In effect, this protocol achieves what is known as "universal" verifiability [dH12,SV15]. Obtaining full universal verifiability in a [SV15]-like security model requires a few changes in the way the output encryptions are constructed; we elaborate on this in a forthcoming book chapter.

Basing it on Commitments Verifiability by certificate validation can be based on Pedersen commitments instead of ElGamal encryptions. This requires a few changes; in particular, to prove that a commitments is zero, one needs to know the randomness, hence the randomness of product commitments needs to be computed in a multiparty way. Using Pedersen commitments likely leads to smaller proofs and quicker verification. Also, it is no longer needed to distribute decryption keys to the computation parties, hence a computation can be outsourced to anybody without preparation. On the other hand, when using Pedersen commitments, the correctness of the computation becomes conditional on the computation parties not knowing trapdoor  $\log_g h$ ; in the present construction, knowing this trapdoor breaks privacy but not correctness.

Load Balancing of the 2PC In the present protocol, two of the three computation parties produce the proof in line 21 while the third party does nothing. If it is important to balance the computation load, then it is possible to let let the three pairs of parties each produce one third of this proof.

### 4 Secure and Verifiable Linear Programming

To demonstrate the feasibility of our approach, we apply it to linear programming. Linear programming is a broad class of optimisation problems occurring in many applications; for instance, it was used to compute the optimal price in the Danish sugar beet auctions that were performed using multi-party computation [BCD+09]. Precisely, the problem is to minimise the output of a linear function, subject to linear constraints on its variables. One instance of this problem is called a linear program (LP); it is given by a matrix **A** and vectors **b** and **c**. The vector  $\mathbf{c} = (c_1, \ldots, c_n)$  gives the linear function  $\mathbf{c}^T \cdot \mathbf{x} = c_1 \cdot x_1 + \ldots + c_n \cdot x_n$  in variables  $\mathbf{x} = (x_1, \ldots, x_n)$  that needs to be minimised. The matrix **A** and vector **b** give the constraints  $\mathbf{A} \cdot \mathbf{x} \leq \mathbf{b}$  that need to be satisfied. **A** has *n* columns, and **A** and **b** have *m* rows, where *m* is the number of constraints. In addition to these constraints, we require  $x_i \geq 0$ . For instance, the LP

$$\mathbf{A} = \begin{pmatrix} 1 & 2 & 1 \\ 1 & -1 & 2 \end{pmatrix}, \mathbf{b} = \begin{pmatrix} 2 \\ 1 \end{pmatrix}, \mathbf{c} = \begin{pmatrix} -10 \\ 3 \\ -4 \end{pmatrix}$$

represents the problem to find  $x_1, x_2, x_3$  satisfying  $x_1 + 2x_2 + x_3 \le 2$ ,  $x_1 - x_2 + 2x_3 \le 1$ , and  $x_1, x_2, x_3 \ge 0$ , such that  $-10x_1 + 3x_2 - 4x_3$  is minimal.

To find the optimal solution of a linear program, typically an iterative algorithm called the *simplex algorithm* is used. Each iteration involves several comparisons and a Gaussian elimination step, making it quite heavy for multiparty computation. For relatively small instances, passively secure linear programming is feasible [BCD+09,Tof09,dH12]; but actively secure MPC much less so when including preprocessing (as we see later). Fortunately, given a solution  $\boldsymbol{x}$  to an LP, there is an easy way to prove that it is optimal using the optimal solution  $\boldsymbol{p}$ of the so-called dual LP "maximise  $\boldsymbol{b}^T \cdot \boldsymbol{p}$  such that  $\mathbf{A}^T \cdot \boldsymbol{p} \leq \boldsymbol{c}, \boldsymbol{p} \leq 0$ ": solutions  $\boldsymbol{x} \in \mathbb{Z}^n, \boldsymbol{p} \in \mathbb{Z}^m$  with common denominator  $q \in \mathbb{Z}$  are both optimal if [dH12]:

$$q \ge 1;$$
  $\boldsymbol{p}^T \cdot \boldsymbol{b} = \boldsymbol{c}^T \cdot \boldsymbol{x};$   $\mathbf{A} \cdot \boldsymbol{x} \le q \cdot \boldsymbol{b};$   $\boldsymbol{x} \ge 0;$   $\mathbf{A}^T \cdot \boldsymbol{p} \le q \cdot \boldsymbol{c};$   $\boldsymbol{p} \le 0.$ 

Also, the simplex algorithm for finding  $\boldsymbol{x}$  turns out to also directly give  $\boldsymbol{p}$ . To turn the above criterion into a set of polynomial equations, we add bit decompositions of  $(\boldsymbol{q} \cdot \boldsymbol{b} - \boldsymbol{A} \cdot \boldsymbol{x})_i$ ,  $\boldsymbol{x}_i$ ,  $(\boldsymbol{q} \cdot \boldsymbol{c} - \boldsymbol{A}^T \cdot \boldsymbol{p})_i$ , and  $-\boldsymbol{p}_i$  to the certificate, and prove that each bit decomposition  $b_0, b_1, \ldots$  sums up to the correct value v (with equation  $v = b_0 + 2 \cdot b_1 + \ldots$ ) and contains only bits (with equations  $b_i \cdot (1 - b_i) = 0$ ).

To test our framework, we have made a prototype implementation. We used the simplex implementation from the TUeVIFF distribution of VIFF<sup>1</sup> as a starting point, and modified it to produce the certificate of correctness, i.e., the dual solution and required bit decompositions. We implemented the VERMPC protocol from Section 3.1 using SCAPI [EFLL12]. SCAPI is a high-level cryptographic library that supports ElGamal encryption,  $\Sigma$ -protocols  $\Sigma_{PK}$  and  $\Sigma_{CD}$ , and the Fiat-Shamir heuristic; to implement VERMPC, we needed to add threshold decryption,  $\Sigma_{CD}$ , and the POLYPROVE and POLYVER protocols from Section 2.3.

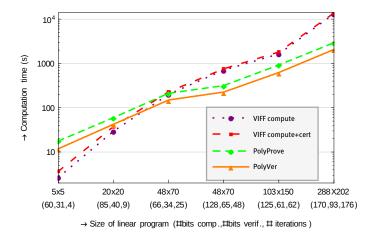
Figure 3 shows the performance of our prototype implementation on several example LPs, run on one single modern desktop PC. ElGamal uses the NIST P-224 [NIS99] elliptic curve, which is supported in SCAPI through the MIRACL library. We ran the implementation on several LPs: randomly-generated small LPs and larger LPs based on Netlib test programs<sup>2</sup>. We measured the time for VIFF to solve the LP and to compute the certificate (this depends on the LP size, number of iterations needed, and the bit length for internal computations); the time for POLYPROVE to produce a proof; and for POLYVER to verify it (this depends on the LP size and bit length for the proof).<sup>3</sup>

Our experiments show that, as the size of the linear program increases, producing and verifying proofs becomes relatively more efficient. Indeed, both the computation of the solution and the verification of its correctness scale in the size of the LP; but computation additionally scales in the number of iterations needed to reach the optimal solution; this number of iterations typically grows with the LP size. For larger LPs, verifying is faster than (passively securely) computing

<sup>&</sup>lt;sup>1</sup> Available at http://www.win.tue.nl/~berry/TUeVIFF/

<sup>&</sup>lt;sup>2</sup> See http://www.netlib.org/lp/data/; coefficients were rounded for performance

<sup>&</sup>lt;sup>3</sup> We took the minimal bitlengths needed for correctness. In practice, these are not known in advance: for VIFF, one takes a safe margin; for the proof, one can reveal and use the maximal bit length of all bit decompositions in the certificate.



**Fig. 3.** Computation times of VERMPC on various LPs (the *x*-axis shows the LP size, bit length for VIFF, bit length for the certificate, and number of iterations)

with three parties, so outsourcing both guarantees correctness and saves the verifier time. If outsourcing replaces a computation performed between more than three parties then the performance effect is even bigger because computation scales linearly in the number of parties involved. For smaller LPs, verification is slower than three-party computation; but also here, it adds correctness and its relative performance improves when more parties are involved.

Verifiability versus Active Security Correctness and privacy can alternatively be guaranteed with (n-1)-out-of-n actively secure multiparty computation. In this case, both correctness and privacy hold even if all other parties are corrupted: a much stronger guarantee than we provide. To get an idea of the performance difference between our approach and active security, we have solved several of our LP instances with an LP solver based on the state-of-the-art SPDZ protocols [DPSZ12,DKL<sup>+</sup>13]. SPDZ combines a slow preprocessing phase, in which many random values are shared between the computation parties, with a fast on-line phase with complexity comparable to passively secure protocols. Hence, after preprocessing has been performed, SPDZ can perform a computation with full privacy and correctness guarantees in about the same time as VIFF (in fact, due to a more efficient implementation, the tested implementation is even faster).

However, preprocessing is slow. No public implementation of the preprocessing phase is available, but it is possible to estimate the time it takes by measuring the amount of randomness needed for the on-line phase and combining this with the preprocessing performance figures from [DKL<sup>+</sup>13]. Even with estimates that are very generous to SPDZ, one finds that the SPDZ preprocessing time is at least 150 times more than the VIFF computation time. For instance, for the first 48-by-70 linear program, we estimate that two-party preprocessing takes at least 13 hours; VIFF computation time is 3.5 minutes and verification time is 2.5 minutes. Hence, verifiable outsourcing has favourable performance compared to using SPDZ. Moreover, SPDZ preprocessing scales linearly with the number of parties involved in the computation, including all input and result parties.

# 5 Concluding Remarks

In this paper, we have shown how to use certificate validation to obtain correctness guarantees for privacy-friendly outsourcing. We have instantiated this idea by combining passively secure three-party computation with ElGamal-based proofs. In the case of linear programming, verification time is much lower than computation time for privacy-friendly computation with correctness guarantees. For larger instances, it is even lower than computation time of privacy-friendly computation without any correctness guarantees. Hence, for computations on inputs of mutually distrusting parties, privacy-friendly outsourcing with correctness guarantees provides a compelling combination of correctness (always) and privacy (against semi-honest, non-collaborating cloud computation parties).

Verifiable multiparty computation has been considered before. [dH12] introduced universally verifiable multiparty computation, and proposed protocols based on threshold homomorphic cryptosystems. Other proposals include [BDO14,SV15]. However, these proposals make a full computation verifiable, which is unrealistic for larger problems. (Indeed, verification times are comparable to actively secure multiparty computation, which is slower than our approach by several orders of magnitude.) De Hoogh [dH12] also first suggested to use certificate validation for verifiability; we contribute the outsourcing application, provide a full security model, and achieve large speed-ups by using ElGamal.

At the same time, there has been much recent interest in verifiable (but usually not privacy-friendly) outsourcing. Without privacy, it is now sometimes possible to check correctness of an outsourced computation faster than performing the computation itself [PHGR13]. Unfortunately, efforts at achieving privacy in this line of work use costly primitives, e.g., fully homomorphic encryption with verifiable computation [FGP14]; or functional encryption with garbled circuits [GKP<sup>+</sup>13]. One recent work [ACG<sup>+</sup>14] uses multiparty techniques in the outsourcing setting; but it does not guarantee correctness if all computation parties are corrupted, and may not be faster than verifiable multiparty computation.

We see several directions for improvement of our work. We have used passively secure protocols for computation; using protocols that guarantee privacy (but not correctness) also against active attacks would offer stronger protection, possibly at a low performance cost. Our implementation can be optimised, and our alternative construction using Pedersen commitment should have smaller proofs and faster verification. Much bigger speed-ups, however, (especially for linear programming) would come from using efficient zero-knowledge proofs for specific tasks, e.g., for showing that certain values are positive. In particular, the range proofs of Boudot [Bou00] are much faster to verify than our bitwise proofs; the work of Keller et al. [KMR12] suggests ways of distributing these proofs that could be adapted to our setting. Alternatively, it may be possible to achieve even faster certificate validation by combining verifiable outsourcing techniques with the privacy guarantees of multiparty computation.

# References

- ACG<sup>+</sup>14. P. Ananth, N. Chandran, V. Goyal, B. Kanukurthi, and R. Ostrovsky. Achieving Privacy in Verifiable Computation with Multiple Servers - Without FHE and without Pre-processing. In *Proceedings of PKC '14*, 2014.
- BCD<sup>+</sup>09. P. Bogetoft, D. L. Christensen, I. Damgård, M. Geisler, T. P. Jakobsen, M. Krøigaard, J. D. Nielsen, J. B. Nielsen, K. Nielsen, J. Pagter, M. I. Schwartzbach, and T. Toft. Secure Multiparty Computation Goes Live. In Proceedings of Financial Crypto '09, 2009.
- BDO14. C. Baum, I. Damgård, and C. Orlandi. Publicly Auditable Secure Multi-Party Computation. In *Proceedings of SCN '14*, 2014.
- Bou00. F. Boudot. Efficient Proofs that a Committed Number Lies in an Interval. In *Proceedings of EUROCRYPT '00*, 2000.
- BR93. M. Bellare and P. Rogaway. Random Oracles are Practical: A Paradigm for Designing Efficient Protocols. In *Proceedings of CCS '93*, 1993.
- Can98. R. Canetti. Security and Composition of Multi-party Cryptographic Protocols. Journal of Cryptology, 13:2000, 1998.
- CL02. M. Castro and B. Liskov. Practical Byzantine fault tolerance and proactive recovery. ACM Transactions on Computer Systems, 20(4):398–461, 2002.
- dH12. S. de Hoogh. Design of large scale applications of secure multiparty computation: secure linear programming. PhD thesis, Eindhoven University of Technology, 2012.
- DKL<sup>+</sup>13. I. Damgård, M. Keller, E. Larraia, V. Pastro, P. Scholl, and N. P. Smart. Practical Covertly Secure MPC for Dishonest Majority - Or: Breaking the SPDZ Limits. In *Proceedings of ESORICS '13*, 2013.
- DM14. L. Ducas and D. Micciancio. FHEW: Bootstrapping Homomorphic Encryption in less than a second. Cryptology ePrint Archive, Report 2014/816, 2014. http://eprint.iacr.org/.
- DPSZ12. I. Damgård, V. Pastro, N. Smart, and S. Zakarias. Multiparty Computation from Somewhat Homomorphic Encryption. In *Proceedings of CRYPTO '12*. 2012.
- EFLL12. Y. Ejgenberg, M. Farbstein, M. Levy, and Y. Lindell. SCAPI: The Secure Computation Application Programming Interface. IACR Cryptology ePrint Archive, 2012:629, 2012.
- FGP14. D. Fiore, R. Gennaro, and V. Pastro. Efficiently Verifiable Computation on Encrypted Data. In *Proceedings of CCS '14*, 2014.
- FS86. A. Fiat and A. Shamir. How to Prove Yourself: Practical Solutions to Identification and Signature Problems. In *Proceedings of CRYPTO '86*, 1986.
- GK03. S. Goldwasser and Y. T. Kalai. On the (In)security of the Fiat-Shamir Paradigm. In *Proceedings of FOCS '03*, 2003.
- GKP<sup>+</sup>13. S. Goldwasser, Y. T. Kalai, R. A. Popa, V. Vaikuntanathan, and N. Zeldovich. Reusable garbled circuits and succinct functional encryption. In *Proceedings of STOC '13*, 2013.
- KMR12. M. Keller, G. L. Mikkelsen, and A. Rupp. Efficient Threshold Zero-Knowledge with Applications to User-Centric Protocols. In *Proceedings* of *ICITS* '12, 2012.
- NIS99. Recommended elliptic curves for federal government use, 1999. Available at http://csrc.nist.gov/encryption.
- Ped91. T. P. Pedersen. A Threshold Cryptosystem without a Trusted Party (Extended Abstract). In *Proceedings of EUROCRYPT '91*, 1991.

 $\Sigma$ -Protocol 1  $\Sigma_{\rm PK}$ : Proof of plaintext knowledge

<b>1:</b> $\triangleright$ Relation: $R = \{(a, b; y, r) \mid a = g^r \land b = h^r g^y\}$		
2: function $\Sigma_{\text{PK}}$ .ann $(a, b; y, r)$	▷ Announcement	
3: $u, v \in_R \mathbb{F}_q; c \leftarrow g^v; d \leftarrow h^v g^u;$ return $(c, d; u,$	,v)	
4: function $\Sigma_{\text{PK}}$ .res $(a, b; y, r; c, d; u, v; e)$	⊳ Response	
5: $k \leftarrow u + e \cdot y; l \leftarrow v + e \cdot r;$ return $(k, l)$		
6: function $\Sigma_{\text{PK}}.sim(a,b;e)$	▷ Simulator	
7: $k, l \in_R \mathbb{F}_q; a \leftarrow g^l a^{-e}; b \leftarrow h^l g^k b^{-e}; $ <b>return</b> $(a, b; e; k, l)$		
8: function $\Sigma_{\text{PK}}.ext(a,b;c,d;e;e';k,l;k',l')$	▷ Extractor	
9: $y \leftarrow (k-k')/(e-e'); r \leftarrow (l-l')/(e-e');$ return $(y,r)$		
10: function $\Sigma_{\text{PK}}$ .rea $(e; k, l)$	▷ Announcement recomputation	
11: $c \leftarrow g^l a^{-e}; d \leftarrow h^l g^k b^{-e};$ return $(c, d)$		
12: function $\Sigma_{\text{PK}}$ .ver $(a, b; c, d; e; k, l)$	▷ Verification	
13: <b>return</b> $a^e \stackrel{?}{=} g^l c^{-1} \wedge b^e \stackrel{?}{=} h^l g^k d^{-1}$		

- PHGR13. B. Parno, J. Howell, C. Gentry, and M. Raykova. Pinocchio: Nearly Practical Verifiable Computation. In *Proceedings of S&P '13*, 2013.
- SV15. B. Schoenmakers and M. Veeningen. Universally Verifiable Multiparty Computation from Threshold Homomorphic Cryptosystems. Cryptology ePrint Archive, Report 2015/058, 2015. Accepted at ACNS '15.
- SZJVD04. R. Sailer, X. Zhang, T. Jaeger, and L. Van Doorn. Design and Implementation of a TCG-based Integrity Measurement Architecture. In *Proceedings* of Usenix Security '04, volume 13, 2004.
- Tof09. T. Toft. Solving Linear Programs Using Multiparty Computation. In *Proceedings of Financial Crypto '09*, 2009.
- Wee09. H. Wee. Zero Knowledge in the Random Oracle Model, Revisited. In Proceedings of ASIACRYPT '09, 2009.

# A Security Proof

Our approach is based on the UVCDN protocol from [SV15]; we obtain Theorem 1 by adapting their security proof to our setting. Due to space constraints, we do not present the full simulators; these follow in a forthcoming book chapter. We distinguish two different cases: either at most one computation party is passively corrupted, in which case we guarantee privacy; or not.

#### A.1 The Private Case

To prove Theorem 1 in the case when at most one computation party is passively corrupted, we construct an ideal-world attacker S that simulates a real-life attacker A on the VERMPC protocol using zero inputs for the honest parties. Let B be an encryption of zero or one. We show that it is possible to define an algorithm that outputs ideal-world outcomes in case B encrypts zero, and realworld outcomes in case B encrypts one. This implies that the real-world and

 $\Sigma$ -Protocol 2  $\Sigma_{CM}$ : Proof of correct multiplication

1:  $\triangleright R = \{(a_1, b_1, a_2, b_2, a_3, b_3; y, r, s) \mid a_2 = g^r \land b_2 = h^r g^y \land a_3 = a_1^y g^s \land b_3 = b_1^y h^s\}$ 2: function  $\Sigma_{CM}$ .ann $(a_1, b_1, a_2, b_2, a_3, b_3; y, r, s)$ ▷ Announcement  $u, v, w \in_R \mathbb{F}_q; a \leftarrow g^v; b \leftarrow h^v g^u; c \leftarrow a_1^u g^w; d = b_1^u h^w$ 3: 4: **return** (a, b, c, d; u, v, w)5: function  $\Sigma_{CM}$ .res $(a_1, b_1, a_2, b_2, a_3, b_3; y, r, s; a, b, c, d; u, v, w; e)$ ▷ Response  $k \leftarrow u + e \cdot y; l \leftarrow v + e \cdot r; m \leftarrow w + e \cdot s;$  return (k, l, m)6:  $\begin{array}{l} \text{7: function } \varSigma_{\text{CM}} \text{sim}(a_1, b_1, a_2, b_2, a_3, b_3; e) \\ \text{8:} \qquad k, l, m \in_{\mathbb{R}} \mathbb{F}_q; \ a \leftarrow g^l a_2^{-e}; \ b \leftarrow h^l g^k b_2^{-e}; \ c \leftarrow a_1^k g^m a_3^{-e}; \ d \leftarrow b_1^k h^m b_3^{-e} \end{array}$  $\triangleright$  Simulator 9: return (a, b, c, d; e; k, l, m)10: function  $\Sigma_{CM}.ext(a_1, b_1, a_2, b_2, a_3, b_3; a, b, c, d; e; e'; k, l, m; k', l', m') \quad \triangleright \; Extractor$  $y \leftarrow (k - k')/(e - e'); r \leftarrow (l - l')/(e - e'); s \leftarrow (m - m')/(e - e')$ 11: 12:return (y, r, s)13: **function**  $\Sigma_{\text{PK}}$ .rea $(a_1, b_1, a_2, b_2, a_3, b_3; e; k, l, m) > \text{Announcement recomputation}$ 14:  $a \leftarrow g^l a_2^{-e}; b \leftarrow h^l g^k b_2^{-e}; c \leftarrow a_1^k g^m a_3^{-e}; d \leftarrow b_1^k h^m b_3^{-e}$ return (a, b, c, d)15:16: function  $\Sigma_{CM}$ .ver $(a_1, b_1, a_2, b_2, a_3, b_3; a, b, c, d; e; k, l, m)$  $\triangleright$  Verif. **return**  $a_2^e \stackrel{?}{=} g^l a^{-1} \wedge b_2^e \stackrel{?}{=} h^l g^k b^{-1} \wedge a_3^e \stackrel{?}{=} a_1^k g^m c^{-1} \wedge b_3^e \stackrel{?}{=} b_1^k h^m d^{-1}$ 17:

ideal-world outcomes must be computationally indistinguishable: indeed, otherwise their distinguisher could distinguish encryptions of zero and one, which contradicts the semantic security of the ElGamal cryptosystem.

The simulator S works as follows. Lines 2–7 of VERMPC are as in UVCDN, so S does the same as UVCDN's simulator. Lines 8–17 involve the passively secure multiparty computation of f, which is not part of the UVCDN protocol. S simply performs this computation using shares of zero for the honest input parties. Simulation of lines 18–20 depends on whether the result party is corrupted. If so, then S gets the result of the computation from IVERMPC<sup> $f,\phi$ </sup>, and it makes sure that  $R_i$  encrypt the result result; otherwise, it uses the zero values as before. The POLYPROVE protocol is simulated by performing real multiplications of the encryptions; but decryptions to zero. The remainder of the protocol is followed.

For security to follow, we need to argue that we can indeed simulate real/idealworld executions depending on an encrypted bit B. The argument is the same as in [SV15], except that S additionally simulates the passively secure computation of f. But the security of the protocol used to compute f implies that this simulation is statistically independent from the actual values computed with (i.e., the real inputs in the real-world execution and zero inputs in the ideal-world execution). From this, the result follows.

#### A.2 The Correct Case

In case there are multiple corrupted computation parties, or at least one is actively corrupted, our protocols do not guarantee privacy. Hence, the simulator S receives the inputs of the honest parties from IVERMPC<sup> $f,\phi$ </sup>, which it can use to run the protocol with respect to the adversary. In [SV15], this was shown for

 $\Sigma$ -Protocol 3  $\Sigma_{CD}$ : Proof of correct decryption

1: $\triangleright$ Relation: $R = \{((a, d; p; h); s) \mid d = a^s p \land h = g^s\}$	
2: function $\Sigma_{CD}$ .ann $((a, d; p; h); s)$	$\triangleright$ Announcement
3: $u \in_R \mathbb{F}_q; b \leftarrow a^u; c \leftarrow g^u; $ <b>return</b> $(b, c; u)$	
4: function $\Sigma_{CD}$ .res $((a, d; p; h); s; b, c; u; e)$	$\triangleright$ Response
5: $f = u + e \cdot s$ ; return $f$	
6: function $\Sigma_{CD}$ .sim $((a, d; p; h); e)$	▷ Simulator
7: $f \in_R \mathbb{F}_q; b \leftarrow a^f p^e d^{-e}; c \leftarrow g^f h^{-e};$ return $(b, c; e^{-e})$	e;f)
8: function $\Sigma_{CD}$ .ext $((a, d; p; h); b, c; e; e'; f; f')$	$\triangleright$ Extractor
9: $s \leftarrow (f - f')/(e - e')$ ; return s	
10: function $\Sigma_{\text{PK}}$ .rea $((a, d; p; h); e; f)$ $\triangleright$ A	Announcement recomputation
11: $b \leftarrow a^f p^e d^{-e}; c \leftarrow g^f h^{-e};$ return $(b, c)$	
12: function $\Sigma_{CD}$ .ver $((a, d; p; h); b, c; e; f)$	$\triangleright$ Verification
13: <b>return</b> $d^e p^{-e} \stackrel{?}{=} a^f b^{-1} \wedge h^e = g^f c^{-1}$	

the case when enough computation parties are corrupted to perform threshold decryption, i.e., when both  $\mathcal{P}_1$  and  $\mathcal{P}_2$  are corrupted.

If this is not the case, then the simulator cannot perform decryption, which could be a problem. However, note that in our protocol, the simulator only has to decrypt ciphertexts of which it already knows the plaintext. Namely, ciphertexts are built from the  $X_i$ ,  $A_i$ , and  $R_i$ . It has made the  $X_i$  for honest parties itself, and it has extracted the inputs of corrupted input parties from their proofs of knowledge. For the  $A_i$  and  $R_i$ , we have assumed that they occur at least once as the right-hand-side of a multiplication in the POLYPROVE circuit. Hence, the corrupted parties have had to prove knowledge of their contributions in lines 11– 23 of POLYPROVE, from which the simulator can extract their plaintexts.

So, in fact, the simulator has sufficient information to run the protocol with respect to the adversary, so the adversary's output is statistically indistinguishable in the real and ideal protocol runs. Moreover, [SV15] shows that the simulator can extract the values  $a_1, \ldots, r_l$  that it needs to provide to the simulator (which must satisfy  $\phi$  if the proofs verify, except with negligible probability), which implies that also the output by the result party is statistically indistinguishable. This concludes the argument for this case.