

# Optimistic Fair Secure Computation

Christian Cachin      Jan Camenisch

IBM Research, Zurich Research Laboratory  
CH-8803 Rüschlikon, Switzerland  
{cca,jca}@zurich.ibm.com

August 28, 2003

## Abstract

We present an efficient and fair protocol for secure two-party computation in the optimistic model, where a partially trusted third party  $T$  is available, but not involved in normal protocol executions.  $T$  is needed only if communication is disrupted or if one of the two parties misbehaves. The protocol guarantees that although one party may terminate the protocol at any time, the computation remains fair for the other party. The two parties are linked by an asynchronous communication network only, but the link between each party and  $T$  requires minimal synchrony. All our protocols are based on efficient proofs of knowledge and involve no general zero-knowledge tools. As intermediate steps we describe efficient implementations of verifiable oblivious transfer, escrowed oblivious transfer, and verifiable secure function evaluation, which may be useful in other contexts. The security of all protocols is proved under the decisional Diffie-Hellman assumption.

## 1 Introduction

Secure computation between distrusting parties is a fundamental problem in cryptography. Suppose two parties  $A$  with input  $x$  and  $B$  with input  $y$  wish to jointly compute a function  $f(x, y)$  of their inputs without revealing anything else than the result. It is known that any function can be computed securely and with only few rounds of interaction under cryptographic assumptions [Yao86, GMW87, Gol98].

However, if the computation should also be *fair* and give a guarantee that  $A$  learns  $f(x, y)$  if and only if  $B$  learns  $f(x, y)$ , two-party protocols inevitably come at the cost of many rounds of interaction [BGMR90, Yao86]. The reason is that a malicious party could always quit the protocol early, e.g., as soon as it obtains the information it is interested in, and the other party may not get any output at all. The only way to get around this are several rounds of interaction, in which the result is revealed verifiably and gradually bit-by-bit so that a cheating party has an unfair advantage of at most one bit [Yao86, BCDvdG88].

This work presents an efficient protocol for *fair secure computation* using a third party  $T$  to ensure fairness, which is not actively involved if  $A$  and  $B$  are honest and messages are delivered without errors. This approach has been proposed for fair exchange (e.g., of digital signatures) by Asokan, Schunter, Shoup, and Waidner [ASW97, ASW00] and is known as the *optimistic model*. Its main benefits are a small, constant number of rounds of interaction between  $A$  and  $B$ , independent of the security parameter, and the minimal involvement of  $T$ . Our secure computation protocol maintains the privacy of one party's inputs even if  $T$  should collude with the other party (unlike [ASW00]). We achieve this by combining Yao's technique for securely evaluating a circuit with efficient zero-knowledge proofs.

We consider actually a more general model of fair secure computation, in which there are two functions,  $f_A(x, y)$  and  $f_B(x, y)$ , and  $A$  should learn  $f_A(x, y)$  if and only if  $B$  learns  $f_B(x, y)$ , evaluated on the *same* inputs.

A key feature of our protocol is that it works in an *asynchronous* environment such as the Internet, where messages between  $A$  and  $B$  might be lost or reordered.

Our protocol is *efficient* in the sense that its complexity is directly proportional to the size of the circuit computing  $f$  and does not involve a large initial costs. All our zero-knowledge proofs and verifiable primitives are based on proofs of knowledge about discrete logarithms, without resorting to costly general zero-knowledge proof techniques involving NP-reductions. Our solution is of practical relevance for cases where  $A$  and  $B$  want to compute  $f$  with a small circuit, for example, to evaluate the predicate  $x_A \geq x_B$  (the “millionaire’s problem” [Yao82]), which has potential applications to on-line bidding and auctions.

It has been observed before [BW98, Mic98] that fair two-party computation is feasible in the optimistic model using general tools (not focusing on efficiency, however). In Baum and Waidner’s [BW98] solution for computing a function  $f$ , the circuit is modified such that it blinds the output and encrypts unblinding information using a non-malleable cryptosystem for which  $T$  knows the private key. Clearly, this incurs an additional cost that might exceed the complexity of  $f$  by a large amount and makes the protocol impractical even for small circuits. **check that again.**

## 1.1 Overview

We build the fair secure computation protocol in several steps and use intermediate concepts and protocols that may be of independent interest.

Recall Yao’s approach to secure function evaluation [Yao86]: The circuit constructor  $A$  scrambles the bits on the wires of the circuit by replacing each with a random token, encrypting the truth tables of all gates accordingly such that two tokens together decrypt the corresponding token on the outgoing wire, and providing the cleartext interpretation for the tokens appearing in the circuit output. It sends the encrypted circuit to  $B$  (the circuit evaluator), who obtains the tokens corresponding to his input bits using one-out-of-two oblivious transfer; this ensures that he learns nothing about other tokens.  $B$  is then able to evaluate the circuit and to compute the output on his own. Note that secure function evaluation is one-sided because only  $B$  learns the output.

Our fair secure computation protocol, presented in Section 6, consists of two intertwined executions of *verifiable secure function evaluation* (VFE) on committed inputs between  $A$  and  $B$ , plus recovery involving  $T$ . Verifiable secure function evaluation is a protocol (which we define in Section 5) extending Yao’s construction that computes a given function on committed inputs of  $A$  and  $B$ .

In order to obtain the initial tokens,  $A$  and  $B$  use a *verifiable oblivious transfer* (VOT) protocol that performs a one-out-of-two oblivious transfer on committed values (as defined in Section 4.1).

However, this solution is not sufficient for fair secure computation in the optimistic model. We need to escrow certain information the VFE construction such that a third party  $T$  can open the result of the computation in case the sender refuses to continue or some of its messages are lost. (The escrow protocol is defined and described in Section ??.)

These protocols are based on proofs of knowledge about discrete logarithms and verifiable encryption. Our notation for proofs of knowledge is introduced in Section 3.2 and allows to describe modular composition of proofs. For verifiable encryption we use the methods of Camenisch and Damgård [CD98] as described in Section 3.3. Our model for optimistic fair secure two-party computation is formalized in Section 2.

## 1.2 Related Work

Beaver, Micali, and Rogaway [BMR90] give a constant-round cryptographic protocol for multi-party computation. Its specialization to three parties is somewhat related to our three-party model; it guarantees fairness against one malicious party, but requires a broadcast channel and its round complexity, although constant, is unlikely to be small.

Feige, Kilian, and Naor [FKN94] study a “minimal” extension of the multi-party secure computation models using a third party  $T$ , which receives a single message, does some computation, and outputs the function value, but does not learn anything else about the inputs. Under cryptographic assumptions, every polynomial-time computable function can be computed efficiently (i.e., in polynomial time) in their model.

One can view the optimistic model used in this paper as even “more minimal,” because  $T$  is not involved in regular computations and only used in case some party misbehaves.

In the multi-party setting, the idea of *fast-track computation* has also been introduced and gives protocols that run fast if all participants are honest, but may resort to slower and more secure methods should a cheater be detected [GRR98]. However, one difference between the optimistic model as used here and fast-track computation is that  $T$  is *a priori* assumed to be honest and  $A$  or  $B$  may fail, whereas in the multi-party setting an unknown subset of the parties misbehaves.

## 2 Optimistic Fair Secure Two-Party Computation

### 2.1 Notation

The security parameter is denoted by  $k$ . The random choice of an element  $x$  from a set  $\mathcal{X}$  with uniform distribution is denoted by  $x \in_R \mathcal{X}$ . The concatenation of strings is denoted by  $\parallel$ .

An *algorithm* is a (probabilistic) Turing machine. A *probabilistic polynomial-time* (PPT) algorithm runs in time polynomial in  $k$  except for an exponentially small fraction of its random choices.

The statistical difference between two probability distributions  $P_X$  and  $P_Y$  is denoted by  $|P_X - P_Y|$ . A quantity  $\epsilon_k$  is called *negligible* (as a function of  $k$ ) if for all  $c > 0$  there exists a constant  $k_0$  such that  $\epsilon_k < \frac{1}{k^c}$  for all  $k > k_0$ . The formal security notion is defined in terms of indistinguishability of probability ensembles indexed by  $k$ , but extension from a single random variable to an ensemble assumed implicitly. Two probability ensembles  $X = \{X_k\}$  and  $Y = \{Y_k\}$  are called *computationally indistinguishable* (written  $X \stackrel{c}{\approx} Y$ ) if for every algorithm  $D$  that runs in probabilistic polynomial time (in  $k$ ), the quantity  $|\mathbb{P}[D(X_k) = 1] - \mathbb{P}[D(Y_k) = 1]|$  is negligible.

### 2.2 Definition

The parties  $A$ ,  $B$ , and  $T$  are probabilistic interactive Turing Machines (PITM) that communicate via secure channels in an asynchronous environment. Let  $f : \mathcal{X}_A \times \mathcal{X}_B \rightarrow \mathcal{Y}_A \times \mathcal{Y}_B$  be a deterministic function with two inputs and two outputs that  $A$  and  $B$  want to evaluate, possibly using  $T$ 's help. Suppose  $f$  can be evaluated by a polynomial-sized circuit in  $k$  (the extension to probabilistic functions is straightforward and omitted). Let  $f_A : \mathcal{X}_A \times \mathcal{X}_B \rightarrow \mathcal{Y}_A$  denote the restriction of  $f$  to  $A$ 's output and let  $f_B : \mathcal{X}_A \times \mathcal{X}_B \rightarrow \mathcal{Y}_B$  denote the restriction of  $f$  to  $B$ 's output.  $A$  has private input  $x_A$  and should output  $f_A(x_A, x_B)$  and  $B$  has private input  $x_B$  and should output  $f_B(x_A, x_B)$ .

These requirements are expressed formally in terms of the simulation paradigm for modeling general multi-party computation [Bea91, MR92, Gol98, ?], although we consider only three parties. In this paradigm, the requirements of a protocol are expressed in terms of an ideal

process, where the parties have access to a universally trusted device that performs the actual computation. A protocol is considered secure if all an adversary may do in the real world can also happen in the ideal process; formally, for every real-world adversary there must exist some adversary in the ideal process such that the real protocol execution is indistinguishable from execution of the ideal process.

**The real-world model.** We consider an asynchronous three-party protocol as a collection  $(A, B, T)$  of PITM  $A$ ,  $B$ , and  $T$ . All parties are initialized with the public inputs of the protocol that includes the function  $f$ ,  $T$ 's public key  $y_T$  and possibly further parameters of the encryption schemes. The private inputs are  $x_A$  for  $A$ ,  $x_B$  for  $B$ , and  $z_T$  for  $T$ .

There is no global clock and the parties are linked by secure authenticated channels in the following sense. All communication is driven by the adversary in form of a scheduler  $\mathcal{S}$ . There exists a global set  $\mathcal{M}$  of undelivered messages tagged with  $(S, R)$  that denote sender  $S$  and receiver  $R$ .  $\mathcal{M}$  is initially empty. At each step,  $\mathcal{S}$  chooses a party  $P$ , selects some message  $M \in \mathcal{M}$  with receiver  $P$ , and activates  $P$  with  $M$  on its communication input tape. If  $\mathcal{M}$  is empty,  $P$  may also be activated with empty input.  $P$  performs some computation and eventually writes a message  $(R, \tau)$  to its communication output tape. The message  $\tau$  is then added to  $\mathcal{M}$ , tagged with  $(P, R)$ .  $\mathcal{S}$  repeats this step arbitrarily often, but it is not allowed to terminate as long as  $\mathcal{M}$  contains messages with receiver or sender equal to  $T$ . (In other words,  $\mathcal{S}$  must eventually deliver all messages between  $T$  and any other party  $P \in \{A, B\}$ .) Honest parties eventually generate an output and terminate by raising a corresponding flag; they will not process any more messages.

An adversary in the real world is an algorithm  $C$  that controls  $\mathcal{S}$  plus one of  $A$  or  $B$  and possibly also  $T$ . Parties controlled by the adversary are called corrupted; we assume their output is empty. The adversary itself outputs  $O_C$ , computed as an arbitrary function of its view, which consists of the information observed by the scheduler and all messages written to and read from communication tapes of corrupted parties. W.l.o.g. the adversary is deterministic. For a fixed adversary  $C$ , the outputs of  $A$ ,  $B$ , and  $T$  denoted by  $O_A$ ,  $O_B$ , and  $O_T$ , respectively, are random variables induced by the internal coins of the honest parties.

**The ideal process.** The ideal process consists of algorithms  $\bar{A}$ ,  $\bar{B}$ , and  $\bar{T}$ , and uses on a universally trusted party  $U$  to specify all desired properties of the real protocol.  $U$  is parametrized by  $f$ .  $\bar{A}$  has input  $x_A$ ,  $\bar{B}$  has input  $x_B$ , and  $\bar{T}$  has no input. The operation is as follows.  $\bar{A}$  sends a message in  $\mathcal{X}_A \cup \{\perp\}$  to  $U$ , and  $\bar{B}$  sends a message in  $\mathcal{X}_B \cup \{\perp\}$  to  $U$ , and  $\bar{T}$  may send two distinct messages in arbitrary order, one containing a bit  $b_A$  and the other one containing a bit  $b_B$ . All messages are delivered instantly.

Honest parties in the ideal process operate as follows.  $\bar{A}$  and  $\bar{B}$  just send their input to  $U$  and  $\bar{T}$  sends  $b_A = 1$  and  $b_B = 1$ .  $\bar{A}$  and  $\bar{B}$  then wait for an answer from  $U$ , output the received value, and terminate.  $\bar{T}$  terminates as soon as it has sent two messages to  $U$  and outputs nothing. The outputs are denoted by  $O_{\bar{A}}$ ,  $O_{\bar{B}}$ , and  $O_{\bar{T}}$ .

$U$  is a device that computes two messages, one for  $\bar{A}$  and one for  $\bar{B}$ . Each message is generated as soon as all necessary inputs have arrived. The message for  $\bar{A}$  depends on  $x_A$ ,  $x_B$ , and  $b_A$  and its value is  $f_A(x_A, x_B)$  if  $b_A = 1$  and  $x_A \neq \perp$  and  $x_B \neq \perp$ , and it is  $\perp$  otherwise. The message for  $\bar{B}$  is computed analogously: it depends on  $x_A$ ,  $x_B$ , and  $b_B$  and its value is  $f_B(x_A, x_B)$  if  $b_B = 1$  and  $x_A \neq \perp$  and  $x_B \neq \perp$ , and it is  $\perp$  otherwise.

The ideal-process adversary is an algorithm  $\bar{C}$  that controls the behavior of the corrupted parties in the ideal process. It sees the inputs of a corrupted party and may substitute them by an arbitrary value before sending the specified message to  $U$ . The adversary sees also  $U$ 's answer to the corrupted party. A corrupted party outputs nothing, but the adversary outputs  $O_{\bar{C}}$ , an arbitrary function of all information gathered in the protocol.

In contrast to most of the literature using the simulation paradigm for secure computation, each party sends a message as soon as it is ready in this asynchronous specification. This means that an adversary may also delay the message of a corrupted party until it has obtained the output of another corrupted party.

We are now ready to state the definition. Seemingly separate requirements on a fair secure computation protocol such as correctness, privacy, and fairness are expressed via the simulatability by an ideal process. Recall that an adversary in the real world in an algorithm  $C$  that controls  $S$ ,  $A$  or  $B$ , and possibly also  $T$ , and that  $C$ 's output is arbitrary.

**Definition 1.** Let  $f : \mathcal{X}_A \times \mathcal{X}_B \rightarrow \mathcal{Y}_A \times \mathcal{Y}_B$  be a function that can be evaluated by a polynomial-sized circuit. We say that a protocol  $(A, B, T)$  performs *fair secure computation* if for every real-world adversary  $C$  there exists an adversary  $\bar{C}$  for the ideal process such that the joint distribution of all outputs of the ideal process is computationally indistinguishable from the outputs in the real world, i.e.,

$$O_A O_B O_T O_C \stackrel{c}{\approx} O_{\bar{A}} O_{\bar{B}} O_{\bar{T}} O_{\bar{C}}.$$

A fair secure computation is called *optimistic* if whenever an adversary follows the protocol for honest parties and delivers messages instantly an dreliably  $T$  does not receive or send any message.

Remarks on the above definition.

1. By the design of the ideal process, fairness is only guaranteed if  $T$  is not corrupted; this is unavoidable [?]. However, a corrupt  $T$  colluding with the other party might

On the other hand, if  $T$  is corrupt, then the computation may be unfair and an honest party, say  $A$ , may not receive its output. This occurs in the ideal process if  $\bar{T}$  colluding with  $\bar{B}$  delays sending  $b_A$  until it has observed  $\bar{B}$ 's output and then decides to send  $b_A \neq 1$ .

2. Our model applies only to an isolated three-party case (as is customary in the literature on secure computation). A multi-user model that allows concurrent execution of multiple protocol instances can be constructed by combining our model with techniques proposed by Asokan et al. [ASW00]. Basically, a unique transaction identifier has to be added to all messages and all participants must have public keys and send messages encrypted with a non-malleable cryptosystem (i.e., secure against adaptive chosen-ciphertext attacks).

### 3 Proofs of Knowledge and Verifiable Encryption

This section presents our notation for proofs of knowledge about discrete logarithms and introduces our notion of verifiable encryption. It starts with a description of the underlying encryption schemes.

#### 3.1 Preliminaries

A *semantically secure public-key cryptosystem*  $(E_k, D_k)$  with security parameter  $k$  consists of a (public) probabilistic encryption algorithm  $E_k(\cdot)$  and a (secret) decryption algorithm  $D_k(\cdot)$ . The encryption algorithm  $E_k : \mathcal{M} \rightarrow \mathcal{C}$  takes a message  $m \in \mathcal{M}$  and outputs a ciphertext  $c$ ; the corresponding decryption algorithm  $D_k : \mathcal{C} \rightarrow \mathcal{M}$  computes  $m$  from  $c$ .

Semantic security asserts that an eavesdropper cannot get partial information about the plaintext from a ciphertext [GM84]. More precisely,  $(E_k, D_k)$  is a semantically secure public-key system if for two arbitrary messages  $m_0$  and  $m_1$ , the random variables representing the two encryptions  $E_k(m_0)$  and  $E_k(m_1)$  are computationally indistinguishable.

The protocols in this paper are mostly based on ElGamal encryption [ElG85]. Let  $G$  be a group of large prime order  $q$  (polynomial in  $k$ ) and let  $g \in G$  be a randomly chosen generator. An ElGamal public key is  $(g, y)$  for  $y = g^x$  with a randomly chosen  $x \in \mathbb{Z}_q$  and the corresponding secret key is  $x$ . ElGamal encryption of a message  $m \in G$  proceeds as follows:

**Algorithm** ElGamal( $g, y$ )( $m$ )

1. choose a random  $r \in \mathbb{Z}_q$ ;
2. compute and output  $(c, c') = (g^r, my^r)$ .

The decryption algorithm computes  $m = c'/c^x$  and outputs  $m$ .

Consider the two distributions over  $G^4$  with  $D_0 = (g, g^x, g^y, g^z)$  for  $x, y, z \in_R \mathbb{Z}_q$  and  $D_1 = (g, g^x, g^y, g^{xy})$  for  $x, y \in_R \mathbb{Z}_q$ . The *Decisional Diffie-Hellman (DDH) assumption* is that there exists no PPT algorithm that distinguishes with non-negligible probability between  $D$  and  $R$ . By a random self-reduction property [Sta96, NR97], the DDH assumption is equivalent to assuming that there is no efficient (PPT) algorithm that *decides with high probability* for all tuples  $(g, g^x, g^y, g^z)$  if  $z = xy \pmod q$ . It is well known that ElGamal encryption is semantically secure under the DDH assumption.

Using a hybrid argument, one can show that also the two distributions

$$M_0 = (g, g^{x_1}, \dots, g^{x_n}, g^{y_1}, \dots, g^{y_m}, g^{z_1}, \dots, g^{z_{nm}})$$

with  $x_i, y_j, z_{ij} \in_R \mathbb{Z}_q$  and

$$M_1 = (g, g^{x_1}, \dots, g^{x_n}, g^{y_1}, \dots, g^{y_m}, g^{x_1 y_1}, \dots, g^{x_n y_m})$$

with  $x_i, y_j \in_R \mathbb{Z}_q$  for  $i = 1, \dots, n$  and  $j = 1, \dots, m$  are computationally indistinguishable under the DDH Assumption. The argument is essentially the same as the one by Naor and Reingold [NR97].

### 3.2 Proofs of Knowledge about Discrete Logarithms

We introduce a notation for describing proofs of knowledge about discrete logarithms. Such three-move proofs of knowledge can be composed efficiently in parallel and in a modular way, as shown by Cramer et al. [CDS94] The notation was first used by Camenisch and Stadler [CS97] and subsumes several discrete logarithm-based proof techniques (see the references therein). Our extension allows to describe modular composition.

Let  $G$  be a group of large prime order  $q$  and let  $g, g_1 \in G$  be generators such that  $\log_g g_1$  is not known (e.g. provided by a trusted dealer).

The simplest example of such a proof is the proof of knowledge of a discrete logarithm of  $y \in G$  [Sch91]. For reference, we recall some of properties of this protocol between a prover  $P$  and verifier  $V$ . Public inputs are  $(g, y)$  and  $P$ 's private input is  $x$  such that  $y = g^x$ . First,  $P$  computes a commitment  $t = g^r$  with  $r \in_R \mathbb{Z}_q$  and sends it to  $V$ . Then  $V$  sends to  $P$  a random challenge  $c \in \{0, 1\}^{k'}$ , to which  $P$  responds with  $s = r - cx \pmod q$ , where  $k'$  is a security parameter.  $V$  accepts if and only if  $t = g^s y^c$ . This three-move protocol constitutes a proof of knowledge [FFS88, BG92] because from two accepting conversations with the same commitment  $t$  but different challenges  $c, c'$  and responses  $s, s'$ , a knowledge extractor can compute  $x = \log_g y$  as  $x = \frac{s-s'}{c'-c} \pmod q$ . The protocol is zero-knowledge for  $k' = O(\log k)$ . We denote this protocol by

$$PK \log(g, y) \\ \{\xi : y = g^\xi\}.$$

The witness(es) are conventionally written in Greek letters and only known to the prover while all other parameters are known to the verifier as well.

Unlike the simplifying description above, we assume that all proofs here are actually three-move concurrent zero-knowledge protocols, i.e., carried out using trapdoor commitments for the first message  $t$ . Such trapdoor commitments may be constructed, for example, using an additional generator  $h \in G$ , which is chosen at random by a trusted dealer or is determined in a once-and-for-all setup phase; the zero-knowledge simulator can extract the trapdoor  $\log_g h$  from this. It will allow the simulator to open a given commitment  $t$  in an arbitrary way upon receiving a challenge  $c$  because it can compute suitable  $s$  from the trapdoor, without having to rewind the verifier (for more details see, e.g., [Dam99]); this allows also arbitrarily large challenges.

This basic protocol can be extended in many ways. For example,

$$PK \text{ rep}(g, g_1, y) \\ \{\xi, \rho : y = g^\xi g_1^\rho\}$$

denotes a proof of knowledge of a representation of  $y$  with respect to  $g$  and  $g_1$ .

Proofs written in this notation may be composed in a modular way. It is known that this is sound for monotone boolean expressions from the results of Cramer et al. [CDS94]. For instance, the prover can convince the verifier that he knows the representation of at least one of  $x$  and  $y$  w.r.t. bases  $g$  and  $g_1$  with

$$PK \text{ or}(g, g_1, x, y) \\ \{\text{rep}(g, g_1, x) \vee \text{rep}(g, g_1, y)\}.$$

It is also possible to prove that two discrete logarithms (or parts of representations) are equal [CP93]. We give an example of this technique. It shows that a commitment  $z$  contains the product modulo  $q$  of the two values committed to in  $x$  and  $y$ :

$$PK \text{ mul}(g, g_1, x, y, z) \\ \{\alpha, \beta, \gamma, \delta, \varepsilon : x = g^\alpha g_1^\gamma \wedge y = g^\beta g_1^\delta \wedge z = y^\alpha g_1^\varepsilon\}.$$

This works also for  $z = g^a g_1^r$  with  $r = 0$  and is needed in Section 5.

When such proofs are combined, some optimizations are often possible, just like in assembly code that is produced by a compiler from a high-level language. An example that occurs in Section 5 below is that multiple parallel commitments to the same value are introduced, where only one of them is needed.

### 3.3 Verifiable Encryption

Verifiable encryption is an important building block here and has been used for publicly verifiable secret sharing [Sta96], key escrow, and optimistic fair exchange [ASW00]. It is a two-party protocol between a prover and encryptor  $P$  and a verifier and receiver  $V$ . Their common inputs are a public encryption key  $E$ , a public value  $v$ , and a binary relation  $\mathcal{R}$  on bit strings. As a result of the protocol,  $V$  either rejects or obtains the encryption  $c$  of some value  $s$  under  $E$  such that  $(s, v) \in \mathcal{R}$ . For instance,  $\mathcal{R}$  could be the relation  $(s, g^s) \in \mathbb{Z}_q \times G$ . The protocol should ensure that  $V$  accepts an encryption of an invalid  $s$  only with negligible probability and that  $V$  learns nothing beyond the fact that the encryption contains some  $s$  with  $(s, v) \in \mathcal{R}$ . The encryption key  $E$  typically belongs to a third party, which is not involved in the protocol at all.

Generalizing the protocol of Asokan et al. [ASW00], Camenisch and Damgård [CD98] provide a verifiable encryption scheme for all relations  $\mathcal{R}$  that have an honest-verifier zero-knowledge three-move proof of knowledge where the second message is a random challenge

and the witness can be computed from two transcripts with the same first message but different challenges. This includes most known proofs of knowledge, and in particular, all proofs about discrete logarithms from the previous section. The verifiable encryption scheme is itself a three-move proof of knowledge of the encrypted witness  $s$  and is zero-knowledge if a semantically secure encryption scheme is used [CD98].

We use a similar notation as above and denote by, e.g.,

$$VE(\text{ElGamal}, (g, y), \text{tag})\{\xi : v = g^\xi\}$$

the verifiable encryption protocol for the ElGamal scheme, whereby  $\log_g v$  along with  $\text{tag}$  is encrypted under public key  $y$ . The  $\text{tag}$ , an arbitrary bit string, is needed for the composition of such protocols, as we will see later. The ciphertext  $c$  is represented by (a function) of the transcript of this protocol, which we abbreviate by writing  $c \leftarrow VE(\text{ElGamal}, (g, y), \text{tag})\{\xi : v = g^\xi\}$ , and is stored by  $V$ .

Together with the corresponding secret key ( $x = \log_g y$  in this example), transcript  $c$  contains enough information to decrypt the witness efficiently. We assume that the corresponding decryption algorithm  $VD(\text{ElGamal}, g, x, c, \text{string})$  is subject to the condition that a  $\text{tag}$  matching  $\text{string}$  is encrypted in  $c$ ;  $VD$  outputs the witness in this case and  $\perp$  in all other cases.

We refer to Camenisch and Damgård [CD98] for further details of the verifiable encryption scheme.

## 4 Verifiable and Escrowed Oblivious Transfer

This section introduces two variants of oblivious transfer that are needed for our fair secure computation protocol. Oblivious transfer, proposed by Rabin [Rab81] and by Even et al. [EGL85], is a fundamental primitive for multi-party computation. In its basic incarnation as a one-out-of-two oblivious transfer, a sender  $S$  has two input bits  $b_0$  and  $b_1$ , and a receiver  $R$  has a bit  $c$ . As a result of the protocol  $R$  should obtain  $b_c$ , but should not learn anything about  $b_{c \oplus 1}$  whereas  $S$  should not get any information about  $c$ .

### 4.1 Verifiable Oblivious Transfer

A *verifiable oblivious transfer* (VOT) is an oblivious transfer on committed values, where the sender  $S$  has made two commitments  $A_0$  and  $A_1$ , containing two values  $a_0$  and  $a_1$ , and  $R$  has made a commitment  $C$ , containing a bit  $c$ . The requirements are that  $R$  outputs  $a_c$  without learning anything about  $a_{c \oplus 1}$  and that  $S$  does not learn anything about  $c$ . (A *committed oblivious transfer* as described by Crépeau et al. [CvdGT95] is a related protocol that performs an oblivious transfer of commitments such that  $R$  ends up being committed to  $a_c$ ; Cramer and Damgård [CD97] give an efficient implementation for this.)

Suppose the commitments  $A_0, A_1$ , and  $C$  are of the form  $B = g^b g_1^r$  for a randomly chosen  $r \in \mathbb{Z}_q$  and committed value  $b \in \mathbb{Z}_q$ . In this section, we assume that the commitments are computed correctly from the inputs  $a_0, a_1$ , and  $c$ . In other words, a *commitment oracle* receives  $a_0$  and  $a_1$  from  $S$ , chooses random  $t_0, t_1 \in \mathbb{Z}_q$ , places  $A_0 = g^{a_0} g_1^{t_0}$  and  $A_1 = g^{a_1} g_1^{t_1}$  in the public input, and returns  $t_0$  and  $t_1$  to  $S$  privately; similarly, it receives  $c$  from  $R$ , computes  $C = g^c g_1^r$  using a random  $r \in \mathbb{Z}_q$ , places  $C$  in the public input and gives  $r$  privately to  $R$ . This commitment oracle is an artificial construction for using VOT as part of a larger protocol. Alternatively, one might assume that  $S$  and  $R$  generated and exchanged the commitments beforehand, together with a proof that they are constructed correctly; this is indeed how VOT is used in Section 6 below.

The following protocol is based on verifiable encryption and the oblivious transfer constructions by Even et al. [EGL85] and Bellare and Micali [BM90]. Our notational convention for

such protocols is as follows. All inputs are written as argument lists in parentheses, grouped by the receiving party; the first list contains public inputs, the second list private inputs of the first party ( $S$ ), the third list private inputs of the second party ( $R$ ), and so on.

**Protocol VOT** $(g, g_1, A_0, A_1, C)(a_0, a_1, t_0, t_1)(c, r)$

1.  $S$  as encryptor and  $R$  as receiver engage in two verifiable encryption protocols

$$\begin{aligned} out_0 &\leftarrow VE(\text{ElGamal}, (g_1, C), \emptyset) \{ \alpha, \beta : A_0 = g^\alpha g_1^\beta \} \\ out_1 &\leftarrow VE(\text{ElGamal}, (g_1, \frac{C}{g}), \emptyset) \{ \alpha, \beta : A_1 = g^\alpha g_1^\beta \}. \end{aligned}$$

2. If  $R$  accepts both of the above protocols, he computes

$$a_c = \text{VD}(\text{ElGamal}, g_1, r, out_c, \emptyset).$$

The above protocol uses  $R$ 's commitment  $C$  directly as encryption public key and saves one round compared to the direct adoption of the Bellare-Micali scheme. The way the commitment  $C$  is constructed from  $c$  ensures that  $R$  knows  $\log_{g_1}(C/g^c) = r$  needed to decrypt  $out_c$ , but not the discrete logarithm needed to decipher the other encryption.

**Lemma 1.** *Under the DDH assumption, Protocol VOT is a secure verifiable oblivious transfer.*

*Proof.* Correctness follows from the construction. It remains to show privacy for  $S$  and  $R$ . We have to prove (1) that  $S$  gets no information about  $R$ 's bit  $c$  and (2) that if  $R$  can compute anything about  $a_{c \oplus 1}$  from the information from the protocol, then the verifiable encryption scheme is insecure.

Part (1) is clear because  $S$  sees only an unconditionally hiding commitment of  $c$ .

Part (2) more involved. The properties of the verifiable encryption protocol guarantee privacy for  $S$  if the underlying encryption scheme (here ElGamal) is semantically secure [CD98]. Consider the following game:  $R$  obtains  $g$  and  $g_1$ , calls the commitment oracle with a bit  $c$ , receives  $C$  and  $r$ , and outputs two pairs  $(a_0, a'_0) \in G^2$  and  $(a_1, a'_1) \in G^2$ . Then a message  $m_0$  is set to  $a_0$  or to  $a'_0$  with probability one half each and, independently,  $m_1$  is set to  $a_1$  or to  $a'_1$  with probability one half each. Encryptions  $\text{ElGamal}(g_1, C)(m_0)$  and  $\text{ElGamal}(g_1, C/g)(m_1)$  are computed and given to  $R$ . Finally,  $R$  outputs two elements  $d_0, d_1 \in G$ . We say that  $R$  wins the game if  $d_0 = m_0$  and  $d_1 = m_1$ . Note that an  $R$  following the protocol can win the game with probability one half by random guessing.

Next we show that if there is an  $R^*$  that wins the game with non-negligible advantage over random guessing, then there exists a distinguisher  $D$  (who also controls the commitment oracle) for which ElGamal encryption is not semantically secure. This violates the DDH assumption. Given an ElGamal public-key  $\tilde{g}, \tilde{y}$ ,  $D$  sets  $g = \tilde{g}$  and  $g_1 = \tilde{g}$  and invokes  $R^*$  on  $g, g_1$ . It simulates the commitment oracle and remembers the values  $c, C$ , and  $r$  such that  $C = \tilde{y}^c \tilde{g}^r$ . Then  $R^*$  provides  $(a_0, a'_0) \in G^2$  and  $(a_1, a'_1) \in G^2$ . We show that if  $c = 0$ , then  $D$  distinguishes encryptions of  $1/a_1$  and  $1/a'_1$ , and if  $c = 1$ , then  $D$  distinguishes encryptions of  $a_0$  and  $a'_0$  with non-negligible probability.

$D$  continues as follows:

1. If  $c = 0$ , then  $D$  is given  $(A, B)$ , an ElGamal encryption of either  $1/a_1$  or  $1/a'_1$  with public key  $(\tilde{g}, \tilde{y})$ .  $D$  encrypts one of  $a_0$  or  $a'_0$  with public key  $(g_1, C)$  for the first encryption and uses ciphertext  $(A, A^r/B)$ , supposedly with public key  $(g_1, C/g)$ , for the second encryption that it gives to  $R^*$ .

2. If  $c = 1$ , then  $D$  is given  $(A, B)$ , an ElGamal encryption of  $a_0$  or  $a'_0$  with public key  $(\tilde{g}, \tilde{y})$ .  $D$  uses ciphertext  $(A, A^r B)$ , supposedly with public key  $(g_1, C)$ , for the first encryption and encrypts one of  $a_1$  or  $a'_1$  with public key  $(g_1, C/g)$  for the second encryption that it gives to  $R^*$ .

When  $R^*$  answers with  $d_0$  and  $d_1$ ,  $D$  outputs  $1/d_0$  if  $c = 0$  and  $d_1$  if  $c = 1$ . It is straightforward to show that  $R^*$  sees the same distribution as in Protocol VOT; therefore,  $D$  distinguishes ElGamal encryptions with non-negligible probability by the assumption on  $R^*$ .  $\square$

## 4.2 Escrowed Oblivious Transfer

An *escrowed oblivious transfer* (EOT) is an extension of verifiable oblivious transfer involving three parties: a sender  $S$ , a receiver  $R$ , and a third party  $T$ , whose public key  $y_T$  of an encryption scheme is known to  $S$  and  $R$ . We require that  $T$ 's encryption scheme is non-malleable, i.e., semantically secure against adaptive chosen-ciphertext attacks [DDN91]. The private inputs of  $S$  and  $R$  are the same as for VOT above. The public input of all parties consists of  $A_0$ ,  $A_1$ ,  $C$ , and  $y_T$  and also the commitments are constructed as above.  $T$ 's private input is  $z_T$ , the secret key corresponding to  $y_T$ .

EOT is an interactive protocol that consists of two phases. In Phase I, only  $S$  and  $R$  interact. If  $R$  accepts Phase I, then he is guaranteed to receive  $a_c$  in Phase II as long as either  $S$  or  $T$  is honest. That is,  $R$  either receives a single message from  $S$  that will allow him to compute  $a_c$  (and hence  $T$  needs not participate in the protocol at all) or, if this does not happen,  $R$  sends  $T$  a single request to which  $T$  will reply with information allowing  $R$  to calculate  $a_c$ . Furthermore, there is a public input string  $tag$  that controls the condition under which  $T$  answers  $R$ 's request in Phase II. The usage of this tag will become clear later. The security requirements for EOT are that

- $R$  does not learn anything about  $a_0$  or  $a_1$  in Phase I; after Phase II,  $R$  learns  $a_c$  but nothing about  $a_{c \oplus 1}$  (even when colluding with  $T$ );
- $S$  does not learn anything about  $c$ ;
- $T$  does not learn anything about  $a_0$ ,  $a_1$ , or  $c$ .

The following realization of EOT is derived from Protocol VOT above. As encryption scheme for  $T$  we use the Cramer-Shoup scheme [CS98], denoted by CS, with public key  $y_T$  and private key  $z_T$ .

**Protocol** EOT( $g, g_1, A_0, A_1, C, y_T, tag$ )( $a_0, a_1, t_0, t_1$ )( $c, r$ )( $z_T$ )

Phase I

1.  $S$  chooses  $u_0, u_1 \in_R Z_q^*$ , computes  $y_0 = C^{u_0}$  and  $y_1 = (C/g)^{u_1}$ , and sends  $y_0, y_1$  to  $R$ .
2.  $S$  and  $R$  engage in the following three verifiable encryption protocols:

$$\begin{aligned} out_0 &\leftarrow VE(\text{ElGamal}, (g_1, y_0), \emptyset) \{ \alpha, \beta : A_0 = g^\alpha g_1^\beta \} \\ out_1 &\leftarrow VE(\text{ElGamal}, (g_1, y_1), \emptyset) \{ \alpha, \beta : A_1 = g^\alpha g_1^\beta \} \\ out_2 &\leftarrow VE(\text{CS}, y_T, tag) \{ \alpha, \beta : y_0 = C^\alpha \wedge y_1 = (C/g)^\beta \}. \end{aligned}$$

$R$  aborts if it rejects any of the verifiable encryption protocols.

Phase II

3.  $S$  sends to  $R$  the values  $u_0$  and  $u_1$ .  $R$  verifies that  $y_0 = C^{u_0}$  and  $y_1 = (C/g)^{u_1}$ .

If this check fails or if  $R$  did not receive a message from  $S$ , then  $R$  sends to  $T$  the message  $(out_2, tag)$ , and  $T$  runs  $\text{VD}(CS, z_T, out_2, tag)$  and sends the output to  $R$ .

In either case,  $R$  knows  $u_c$  such that  $u_c = \log_{C/g^c} y_c$  and obtains  $a_c$  from  $out_c$  by running  $\text{VD}(\text{ElGamal}, g_1, ru_c, out_c, \emptyset)$ .

**Lemma 2.** *Under the DDH assumption, Protocol EOT is a secure escrowed oblivious transfer protocol.*

*Proof.* To verify the correctness of the protocol, note that, as for VOT, the commitments are constructed such that  $R$  knows  $r = \log_{g_1}(C/g^c)$ ; therefore,  $\log_{g_1} y_c = ru_c$  is indeed the correct private key for decrypting  $out_c$ .

The security of Phase I (that  $R$  learns nothing about  $a_0$  or  $a_1$ ) follows from the non-malleability of  $T$ 's encryption scheme, even if  $R$  would submit arbitrary decryption requests to  $T$  with tag string  $\neq tag$ . It is also clear that  $T$  does not learn anything about  $a_0$ ,  $a_1$ , or  $c$  because it never even sees their encryptions.

$T$  is only used to guarantee the extra properties of EOT over VOT. Moreover,  $T$  has no influence except for the values  $u_0$  and  $u_1$ ; if  $R$  knows both of them, Protocol EOT reduces to VOT above. This implies that Protocol EOT with colluding  $R$  and  $T$  meets the security requirements of VOT after Phase II.  $\square$

## 5 Verifiable Secure Function Evaluation

Verifiable secure function evaluation (VFE) is an interactive protocol between a circuit constructor  $A$  and an evaluator  $B$ . Both parties have as common public input values  $C_A$  and  $C_B$ , representing commitments to their inputs.  $A$  has two private input strings: her input string  $x_A$  and a string  $r_A$  allowing her to open  $C_A$ ; likewise,  $B$  has two private input strings,  $x_B$  and  $r_B$ . Their goal is to evaluate  $f_B$  on the committed inputs such that  $B$  learns  $f_B(x_A, x_B)$ .

We assume here, as already in Sections 4.1 and 4.2, that all commitments are computed correctly from the inputs, which in turn may have been chosen in an arbitrary way. More precisely, assume  $A$  gives  $x_A$  to a commitment oracle, which computes  $C_A$  according to the specified commitment scheme using the random bits  $r_A$  and returns  $C_A, r_A$  and similarly for  $B$ . This is the notion of *corresponding* commitments used below. (Alternatively, one might assume that  $A$  and  $B$  generated and exchanged the commitments beforehand.)

A concrete protocol execution of  $A$  and  $B$  defines naturally the views  $V_A$  and  $V_B$  of  $A$  and  $B$ , respectively, which are functions of the public input,  $A$ 's private input, and  $B$ 's private input.

**Definition 2.** A *verifiable secure function evaluation protocol* for a function  $f_B : \mathcal{X}_A \times \mathcal{X}_B \rightarrow \mathcal{Y}_B$  between  $A$  and  $B$  satisfies the following requirements:

**Correctness:** If  $A$  and  $B$  are honest and follow the protocol, then  $\forall x_A \in \mathcal{X}_A, \forall x_B \in \mathcal{X}_B$  and corresponding commitments,  $B$  outputs  $f_B(x_A, x_B)$  except with negligible probability.

**Soundness:**  $\forall A^*$  and  $\forall x_A^* \in \mathcal{X}_A$  and corresponding commitments  $C_A^*$ , if the protocol starts with public inputs  $C_A^*, C_B$ , then, except with negligible probability,  $B$  outputs  $f_B(x_A^*, x_B)$  or error.

**Privacy:** We consider two cases, corresponding to cheating  $B$  and cheating  $A$ .

1. *Privacy for A:*  $\forall B^*$  there exists a PPT algorithm  $\text{SIM}_{B^*}$  such that  $\forall x_A \in \mathcal{X}_A$  and  $\forall x_B^* \in \mathcal{X}_B$  with corresponding commitments  $C_A, C_B^*$ ,

$$V_{B^*}(C_A, C_B^*, x_A, r_A, x_B^*, r_B^*) \stackrel{c}{\approx} \text{SIM}_{B^*}(C_A, C_B^*, f_B(x_A, x_B^*), x_B^*).$$

2. *Privacy for B*:  $\forall A^*$  there exists a PPT algorithm  $SIM_{A^*}$  such that  $\forall x_B \in \mathcal{X}_B$  and  $\forall x_A^* \in \mathcal{X}_A$  with corresponding commitments  $C_A^*, C_B$ ,

$$V_{A^*}(C_A^*, C_B, x_A^*, r_A^*, x_B, r_B) \stackrel{c}{\approx} SIM_{A^*}(C_A^*, C_B, x_A^*).$$

The soundness condition binds  $A$  to her committed inputs. The corresponding binding for  $B$  is part of the privacy condition for  $A$ , which ensures that  $B$  is committed to the value  $x_B$  at which he evaluates  $f_B$  before the protocol starts. This is needed to use the one-sided concept of VFE as a building block for optimistic fair secure computation below.

## 5.1 Overview of the Encrypted Circuit Construction

We give a short description of our protocol and the “encrypted circuit construction”; it follows the approach to secure function evaluation developed by Yao [Yao86], but uses public-key encryption instead of pseudo-random functions for the sake of verifiability. Suppose  $A$ ’s private input is a binary string  $x_A = (x_{A,1}, \dots, x_{A,n_A})$  and  $B$ ’s private input is a binary string  $x_B = (x_{B,1}, \dots, x_{B,n_B})$ ; assume further w.l.o.g. that  $f_B$  is represented a binary circuit consisting of NAND gates.

**Protocol**  $VFE(g, g_1, C_A, C_B, f_B)(x_A, r_A)(x_B, r_B)$

- V1.  $A$  produces an encrypted version of the circuit computing  $f_B$ . The circuit consists of gates and wires linking the gates. Except for input and output wires, each wire connects the output of one gate with the input of one or more other gate(s). For each wire,  $A$  chooses two random *tokens*  $s_0$  and  $s_1$ , representing bits 0 and 1 on this wire, and produces unconditionally hiding commitments  $u_0$  and  $u_1$  to these tokens.

For each gate,  $A$  encrypts the truth table as follows: First, the bits are replaced by (new) commitments to the tokens representing the bits. Next, for each row, a “row public key” for encryption is computed and added to the table such that the corresponding secret key can be derived from combining the two input tokens of the row. Finally, all four rows are permuted randomly.

These tables and the commitments are sent to  $B$  as an ordered list such that  $B$  knows which commitment represents token 0 or 1 etc. Moreover,  $A$  proves to  $B$  in zero-knowledge that the commitments and the encrypted gates are consistent, ensuring (1) that the tokens of the input and output wires are the same as those committed to in the truth table, (2) that the secret key for each row of a gate is derived correctly from the input tokens of the row, and (3) that each encrypted gate implements NAND.

- V2. For each row of each gate of the circuit,  $A$  and  $B$  engage in verifiable encryption of the output token under the row public key.
- V3. For each of her input bits,  $A$  sends to  $B$  the corresponding token and proves to him that this is consistent with her input  $x_A$  committed in  $C_A$ . Furthermore,  $B$  obtains the tokens representing his input bits through  $n_B$  verifiable oblivious transfers from  $A$  to  $B$  and  $A$  opens all the commitments of the output wires.
- V4. Once  $B$  has obtained all this information, he is able to evaluate the circuit gate by gate on his own.

We now describe the verifiable secure function evaluation protocol in more detail.

Suppose w.l.o.g. the circuit consists of  $n$  NAND gates  $\mathcal{G}_1, \dots, \mathcal{G}_n$  and  $n + n_A + n_B$  wires  $\mathcal{W}_1, \dots, \mathcal{W}_{n+n_A+n_B}$  and has  $n_A + n_B$  inputs and  $n_O$  outputs. Wires  $\mathcal{W}_1, \dots, \mathcal{W}_n$  are output

wires of the gates  $\mathcal{G}_1, \dots, \mathcal{G}_n$ . Wires  $\mathcal{W}_{n+1}, \dots, \mathcal{W}_{n+n_A}$  are input wires of  $A$  and  $\mathcal{W}_{n+n_A+1}, \dots, \mathcal{W}_{n+n_A+n_B}$  are input wires of  $B$ . Wires  $\mathcal{W}_{n-n_O+1}, \dots, \mathcal{W}_n$  are the output wires of the circuit; except for those, any wire is an input to at least one gate.

The commitment to  $A$ 's input  $x_A$  is  $C_A = (C_{A,1}, \dots, C_{A,n_A})$ , where for  $i = 1, \dots, n_A$ , a bit commitment

$$C_{A,i} = g^{x_{A,i}} g_1^{r_{A,i}}$$

has been constructed using a random  $r_{A,i} \in \mathbb{Z}_q$  and  $r_A = (r_{A,1}, \dots, r_{A,n_A})$  is a private input of  $A$ .

Similarly, the commitment to  $B$ 's input  $x_B$  is  $C_B = (C_{B,1}, \dots, C_{B,n_B})$ , where for  $i = 1, \dots, n_B$ , a bit commitment

$$C_{B,i} = g^{x_{B,i}} g_1^{r_{B,i}}$$

has been constructed using a random  $r_{B,i} \in \mathbb{Z}_q$  and  $r_B = (r_{B,1}, \dots, r_{B,n_B})$  is a private input of  $B$ .

The following subsections describe Steps V1–V4 in more detail. Throughout the description we assume that  $B$  outputs **error** and halts as soon as he rejects any *PK* or *VE* protocol.

## 5.2 Constructing the Committed Circuit (Step V1)

Let  $j, l : \{1, \dots, n\} \rightarrow \{1, \dots, n + n_A + n_B\}$  be such that  $j(i)$  and  $l(i)$  denote the index of the left and right input wire of gate  $\mathcal{G}_i$ .  $A$  carries out the following step to obtain an encrypted circuit:

1. For each wire  $\mathcal{W}_i$  choose  $s_{i,0}, s_{i,1} \in_R \mathbb{Z}_q$  as tokens representing 0 and 1. Next, choose  $r_{i,0}, r_{i,1} \in_R \mathbb{Z}_q$  and compute the commitments

$$\begin{aligned} u_{i,0} &= g^{s_{i,0}} g_1^{r_{i,0}} \\ u_{i,1} &= g^{s_{i,1}} g_1^{r_{i,1}}. \end{aligned}$$

2. For each gate  $\mathcal{G}_i$  construct the committed truth table  $T_i$  as follows:

- (a) Choose twelve entries of a  $4 \times 3$  matrix  $R_i$  randomly from  $\mathbb{Z}_q$ . The commitment table  $\bar{T}_i$  contains information-theoretic commitments to all input and output tokens, plus the encryption public key, derived from the input tokens, in the last column:

$$\bar{T}_i = \begin{pmatrix} g^{s_{j(i),0}} g_1^{R_i(1,1)} & g^{s_{l(i),0}} g_1^{R_i(1,2)} & g^{s_{i,1}} g_1^{R_i(1,3)} & g^{s_{j(i),0} s_{l(i),0}} \\ g^{s_{j(i),0}} g_1^{R_i(2,1)} & g^{s_{l(i),1}} g_1^{R_i(2,2)} & g^{s_{i,1}} g_1^{R_i(2,3)} & g^{s_{j(i),0} s_{l(i),1}} \\ g^{s_{j(i),1}} g_1^{R_i(3,1)} & g^{s_{l(i),0}} g_1^{R_i(3,2)} & g^{s_{i,1}} g_1^{R_i(3,3)} & g^{s_{j(i),1} s_{l(i),0}} \\ g^{s_{j(i),1}} g_1^{R_i(4,1)} & g^{s_{l(i),1}} g_1^{R_i(4,2)} & g^{s_{i,0}} g_1^{R_i(4,3)} & g^{s_{j(i),1} s_{l(i),1}} \end{pmatrix}$$

- (b) Choose a random permutation  $\pi_i : \{1, 2, 3, 4\} \rightarrow \{1, 2, 3, 4\}$  and obtain  $T_i$  by permuting the rows of  $\bar{T}_i$  accordingly:

$$T_i(m) = \bar{T}_i(\pi(m))$$

for  $m = 1, \dots, 4$ .

The list of commitments

$$\mathcal{C}_{f_B} = (u_{1,0}, u_{1,1}, \dots, u_{n+n_A+n_B,0}, u_{n+n_A+n_B,1}, T_1, \dots, T_n)$$

is sent to  $B$ . Next,  $A$  proves to  $B$  for each wire  $\mathcal{W}_i$  that the tokens committed to in  $u_{i,0}, u_{i,1}$  are different mod  $q$ :

$$\begin{aligned} & PK \text{ tokens}(g, g_1, u_{i,0}, u_{i,1}) \\ & \{ \sigma, \rho : g = (u_{i,0}/u_{i,1})^\sigma g_1^\rho \}. \end{aligned}$$

Furthermore, for each gate  $\mathcal{G}_i$   $A$  proves to  $B$  (1) that the public key attached to each row is constructed correctly and (2) that  $T_i$  indeed implements all four rows of the NAND truth-table. Let

$$W_i = (u_{i,0}, u_{i,1}, u_{j(i),0}, u_{j(i),1}, u_{l(i),0}, u_{l(i),1})$$

denote the list of commitments of the wires incident to  $\mathcal{G}_i$ . The first part is done with

$$\begin{aligned} & PK \text{ gate-keys}(g, g_1, T_i, W_i) \\ & \left\{ \text{mul}(g, g_1, T_i(1,1), T_i(1,2), T_i(1,4)) \wedge \text{mul}(g, g_1, T_i(2,1), T_i(2,2), T_i(2,4)) \right. \\ & \quad \left. \wedge \text{mul}(g, g_1, T_i(3,1), T_i(3,2), T_i(3,4)) \wedge \text{mul}(g, g_1, T_i(4,1), T_i(4,2), T_i(4,4)) \right\}; \end{aligned}$$

it shows that the key (committed to) in  $T(m,4)$  is the product of the two tokens in  $T(m,1)$  and  $T(m,2)$ . The second part is done with

$$\begin{aligned} & PK \text{ gate-nand}(g, g_1, T_i, W_i) \\ & \left\{ \text{nand}_{0,0}(g, g_1, T_i, W_i) \wedge \text{nand}_{0,1}(g, g_1, T_i, W_i) \right. \\ & \quad \left. \wedge \text{nand}_{1,0}(g, g_1, T_i, W_i) \wedge \text{nand}_{1,1}(g, g_1, T_i, W_i) \right\}, \end{aligned}$$

where the following protocol is used:

$$\begin{aligned} & PK \text{ nand}_{a,b}(g, g_1, T_i, W_i) \\ & \left\{ \alpha_1, \beta_1, \gamma_1, \mu_1, \nu_1, \rho_1, \sigma_1, \tau_1, \vartheta_1, \dots, \alpha_4, \beta_4, \gamma_4, \mu_4, \nu_4, \rho_4, \sigma_4, \tau_4, \vartheta_4 : \right. \\ & \quad (u_{j(i),a} = g^{\beta_1} g_1^{\nu_1} \wedge u_{l(i),b} = g^{\gamma_1} g_1^{\rho_1} \wedge u_{i,(a\bar{b})} = g^{\alpha_1} g_1^{\mu_1} \\ & \quad \wedge T_i(1,1) = g^{\beta_1} g_1^{\tau_1} \wedge T_i(1,2) = g^{\gamma_1} g_1^{\vartheta_1} \wedge T_i(1,3) = g^{\alpha_1} g_1^{\sigma_1}) \\ & \quad \vee (u_{j(i),a} = g^{\beta_2} g_1^{\nu_2} \wedge u_{l(i),b} = g^{\gamma_2} g_1^{\rho_2} \wedge u_{i,(a\bar{b})} = g^{\alpha_2} g_1^{\mu_2} \\ & \quad \wedge T_i(2,1) = g^{\beta_2} g_1^{\tau_2} \wedge T_i(2,2) = g^{\gamma_2} g_1^{\vartheta_2} \wedge T_i(2,3) = g^{\alpha_2} g_1^{\sigma_2}) \\ & \quad \vee (u_{j(i),a} = g^{\beta_3} g_1^{\nu_3} \wedge u_{l(i),b} = g^{\gamma_3} g_1^{\rho_3} \wedge u_{i,(a\bar{b})} = g^{\alpha_3} g_1^{\mu_3} \\ & \quad \wedge T_i(3,1) = g^{\beta_3} g_1^{\tau_3} \wedge T_i(3,2) = g^{\gamma_3} g_1^{\vartheta_3} \wedge T_i(3,3) = g^{\alpha_3} g_1^{\sigma_3}) \\ & \quad \vee (u_{j(i),a} = g^{\beta_4} g_1^{\nu_4} \wedge u_{l(i),b} = g^{\gamma_4} g_1^{\rho_4} \wedge u_{i,(a\bar{b})} = g^{\alpha_4} g_1^{\mu_4} \\ & \quad \wedge T_i(4,1) = g^{\beta_4} g_1^{\tau_4} \wedge T_i(4,2) = g^{\gamma_4} g_1^{\vartheta_4} \wedge T_i(4,3) = g^{\alpha_4} g_1^{\sigma_4}) \left. \right\}. \end{aligned}$$

$PK \text{ nand}_{a,b}$  shows that some row of the permuted encrypted truth table  $T_i$  with token commitments  $W_i$  corresponds to the row in the cleartext truth table with input bits  $a$  and  $b$  and output bit  $a \bar{b}$ .

### 5.3 Verifiably Encrypting the Gate Output Tokens (Step V2)

For each gate  $\mathcal{G}_i$ , parties  $A$  and  $B$  carry out the following four verifiable encryptions protocols:

$$\begin{aligned} v_{i,1} & \leftarrow VE \text{ (ElGamal, } (g, T_i(1,4)), \emptyset) \{ \alpha, \beta : T_i(1,3) = g^\alpha g_1^\beta \} \\ v_{i,2} & \leftarrow VE \text{ (ElGamal, } (g, T_i(2,4)), \emptyset) \{ \alpha, \beta : T_i(2,3) = g^\alpha g_1^\beta \} \\ v_{i,3} & \leftarrow VE \text{ (ElGamal, } (g, T_i(3,4)), \emptyset) \{ \alpha, \beta : T_i(3,3) = g^\alpha g_1^\beta \} \\ v_{i,4} & \leftarrow VE \text{ (ElGamal, } (g, T_i(4,4)), \emptyset) \{ \alpha, \beta : T_i(4,3) = g^\alpha g_1^\beta \} \end{aligned}$$

## 5.4 Transferring the Input and Output Tokens (Step V3)

For each input wire  $\mathcal{W}_{n+i}$  of  $A$ , she sends the token representing  $x_{A,i}$  to  $B$ ; that is,  $A$  sends

$$w_{n+i} = \begin{cases} s_{n+i,0} & \text{if } x_{A,i} = 0 \\ s_{n+i,1} & \text{if } x_{A,i} = 1 \end{cases}$$

for  $i = 1, \dots, n_A$  and carries out

$$\begin{aligned} & PK \text{ input}(g, g_1, C_{A,i}, w_{n+i}, u_{n+i,0}, u_{n+i,1}) \\ & \left\{ \varphi_1, \rho_1, \varphi_2, \rho_2 : \right. \\ & \left. (C_{A,i} = g_1^{\rho_1} \wedge u_{n+i,0}/g^{w_{n+i}} = g_1^{\varphi_1}) \vee (C_{A,i}/g = g_1^{\rho_2} \wedge u_{n+i,1}/g^{w_{n+i}} = g_1^{\varphi_2}) \right\}. \end{aligned}$$

This ensures  $B$  that  $w_{n+i}$  is the token representing  $A$ 's input  $x_{A,i}$  as committed to in  $C_{A,i}$ .

$A$  also opens the commitments of the output wires to  $B$ ; that is,  $A$  sends to  $B$  the values  $(s_{i,0}, s_{i,1}, r_{i,0}, r_{i,1})$  for  $i = n - n_O + 1, \dots, n$ .

Finally,  $A$  and  $B$  run  $n_B$  verifiable oblivious transfer protocols: for each input wire  $\mathcal{W}_{n+n_A+i}$  of  $B$ ,  $A$  offers tokens  $s_{n+n_A+i,0}$  and  $s_{n+n_A+i,1}$  committed to in  $u_{n+n_A+i,0}$  and  $u_{n+n_A+i,1}$ , and  $B$  chooses to receive the one representing the bit he committed to in  $C_{B,i}$ . That is, they engage in

$$VOT(g, g_1, u_{n+n_A+i,0}, u_{n+n_A+i,1}, C_{B,i})(s_{n+n_A+i,0}, s_{n+n_A+i,1}, r_{n+n_A+i,0}, r_{n+n_A+i,1})(x_{B,i}, r_{B,i})$$

in parallel for  $i = 1, \dots, n_B$ . Denote the values that  $B$  receives by  $w_{n+n_A+1}, \dots, w_{n+n_A+n_B}$ .

## 5.5 Evaluating the Circuit (Step V4)

If  $B$  has accepted all the proofs and verifiable encryption protocols, he is convinced that the encrypted circuit construction is correct and he has obtained all necessary information for computing the value of  $f_B$  by himself. He proceeds by evaluating the circuit gate by gate, computing a token  $w_i$  for each gate  $\mathcal{G}_i$ . Note that  $B$  already knows the input tokens  $w_{n+1}, \dots, w_{n+n_A+n_B}$ . Suppose  $\mathcal{G}_i$  has not been evaluated yet and  $B$  knows the tokens  $w_{j(i)}$  and  $w_{l(i)}$ . Then  $B$

1. computes  $\tilde{s}_i = w_{j(i)} w_{l(i)} \pmod{q}$ ;
2. finds index  $m \in \{1, \dots, 4\}$  such that  $g^{\tilde{s}_i} = T_i(m, 4)$ ; and
3. computes  $w_i = \text{VD}(\text{ElGamal}, g, \tilde{s}_i, v_{i,m}, \emptyset)$ .

Once all gates are evaluated,  $B$  also knows the tokens of the output gates  $\mathcal{G}_{n-n_O+1}, \dots, \mathcal{G}_n$ .  $B$  decodes them by letting  $o_i \in \{0, 1\}$  such that  $w_i = s_{i,o_i}$  and his output is  $O = (o_{n-n_O+1}, \dots, o_n)$ .

## 5.6 Analysis

The round complexity of the protocol is minimal: because the proofs of knowledge and verifiable encryptions have only three moves and can be composed in parallel, all steps in the verifiable secure function evaluation protocol can be arranged in three moves only. Furthermore, some steps could be simplified by omitting multiple or unnecessary commitments.

The security analysis is based on the following lemma.

**Lemma 3.** *Under the DDH assumption,  $B^*$  can decrypt at most one row of the truth table for each gate and cannot compute any further information from the other three rows.*

*Proof (Sketch).* The proof is by induction on the structure of the circuit. Consider an input gate. The properties of VOT ensure that if the sender inputs two random tokens, the receiver gets one but cannot compute further information about the other token under the DDH assumption.

Consider an arbitrary gate  $\mathcal{G}_i$  and assume the claim holds for  $\mathcal{G}_j$  and  $\mathcal{G}_l$  that feed into  $\mathcal{G}_i$ . Then  $B^*$  knows at most one of the four possible token products and this allows to decrypt one row. The semantic security of the remaining three encryptions is guaranteed under the DDH assumption; in other words,  $B^*$  cannot distinguish which tokens are encrypted in the other three rows.

Apart from the public keys  $g^{s_{j,a}, s_{l,b}}$ , the gate tables contain only information-theoretic commitments and they do not reveal any information about the permutations or the cleartext bits associated with a particular row. The tokens  $s_{j,a}, s_{l,b}$  occurring in the public keys of  $\mathcal{G}_i$  (and possibly in other gates in the same “layer” of the circuit) correspond to  $x_i, y_j$  in the distributions  $M_0, M_1$  from Section 3.1. Hence, they are indistinguishable from elements under the DDH assumption.  $\square$

**Theorem 4.** *Under the DDH assumption, Protocol VFE from Sections 5.1–5.5 is a verifiable secure function evaluation protocol.*

*Proof (Sketch).* We have to show correctness, soundness, and privacy for  $A$  and for  $B$ . Correctness is clear from the construction of the protocol. Soundness follows from the soundness of the proofs of knowledge, of the verifiable encryptions, and of the VOT protocols, which together enforce that  $B$  obtains  $f_B$  only evaluated at  $A^*$ ’s input committed to by  $C_A^*$ .

Privacy for  $A$ : this is the most interesting part because it involves showing that  $B^*$  does not learn more than what follows from  $f_B(x_A, x_B^*)$ . To this end, we describe a simulator  $SIM_{B^*}$  that has black-box access to  $B^*$ ; the simulator’s output is computationally indistinguishable from  $B^*$ ’s view in a real protocol execution. The idea behind the simulator is that  $B^*$  knows only one “computation path” through the circuit and learns nothing about the values involved except for the output gates. The simulator thus interacts with  $B^*$  for an arbitrary input  $x_A$  of  $A$ ; it only has to make sure that  $B^*$ ’s output will be  $f_B(x_A, x_B^*)$ .

More precisely,  $SIM_{B^*}$  is the following PPT algorithm. It takes as input the function value  $f_B(x_A, x_B^*)$ . From the commitment oracle it obtains  $B^*$ ’s input  $x_B^*$ , chooses an arbitrary value  $\tilde{x}_A$  for the input of  $A$  and executes the Steps 1–4 (Sections 5.2–5.5) exactly as  $A$  with the following exceptions:

1. In Step V1 (Section 5.2) for  $i = n - n_O + 1, \dots, n$ , the simulator uses the same token  $s_{i,o_i}$  in all commitments  $T_i(1, 3)$ ,  $T_i(2, 3)$ ,  $T_i(3, 3)$ , and  $T_i(4, 3)$ , where  $o_i$  is the output bit of  $f_B$  that  $SIM_{B^*}$  has been given. Consequently, the simulator has to forge the proofs **gate-nand** for these gates, which it can do by exploiting the simulatability of these protocols.
2. Analogously, in Step V2 (Section 5.3) for  $i = n - n_O + 1, \dots, n$ , the simulator encrypts the same  $s_{i,o_i}$  in all four verifiable encryptions.
3. In Step V3 (Section 5.4), the simulator behaves like  $A$  except for  $PK$  input; here it has to forge the proof of the correspondence between  $A$ ’s input commitment  $C_A$  and  $\tilde{x}_A$  chosen by the simulator, again by exploiting the simulatability of the proof.

It remains to argue that  $B^*$ ’s view when interacting with the real  $A$  and the view provided by  $SIM_{B^*}$  are computationally indistinguishable. Because the whole construction uses unconditionally hiding commitments and all proofs are zero-knowledge, the only place where there could be a difference is the encryption of the output tokens of the output gates. However, due to Lemma 3 this is not the case and we have established privacy for  $A$ .

Privacy for  $B$ : it suffices to consider VOT, which is the only place where  $B$  ever sends information to  $A$  that could compromise  $B$ ’s inputs. Protocol VOT of Section 4.1 provides even

information-theoretic privacy for  $B$  in the role of  $R$  and the proofs can be simulated by the standard techniques.  $\square$

*Remark.* The invocations of Protocol VOT at the end of Step V3 deserve special attention because of the way Protocol VFE is used in the next section. Step 1 of each VOT involves two verifiable encryptions with the circuit constructor  $A$  as prover and the circuit evaluator  $B$  as verifier. These proofs may also be verified by an independent third party  $T$ , which  $B$  trusts to act as verifier. More precisely, because the verifiable encryption public keys are also known beforehand (they are derived from the commitments), the VOT protocol may, equivalently, consist of an interaction between  $A$  and  $T$ , followed by interaction between  $T$  to  $B$ , where  $T$  sends to  $B$  the transcript of its interaction with  $A$ . Such a  $T$  may not know how to decrypt the transferred values.

## 6 Optimistic Fair Secure Computation Protocol

We are now ready to describe our protocol for optimistic fair secure two-party computation. In short, the protocol consists of two intertwined executions of the verifiable secure function evaluation protocol from the previous section, with all occurrences of VOT replaced by EOT. Recall that optimistic fair secure computation involves three parties  $A$ ,  $B$ , and  $T$ , in the asynchronous communication model of Definition 1.

In the following we use the discrete logarithm-based protocols VOT and EOT from Section 4. Furthermore, we assume that the encryption scheme of  $T$  is the Cramer-Shoup scheme [CS98].

Common inputs are a function  $f : \mathcal{X}_A \times \mathcal{X}_B \rightarrow \mathcal{Y}_A \times \mathcal{Y}_B$ ,  $T$ 's public key  $y_T$ , and generators  $g, g_1 \in G$ . The private input of  $A$  is  $x_A \in \mathcal{X}_A$ , the private input of  $B$  is  $x_B \in \mathcal{X}_B$ , and the private input of  $T$  is the private key  $z_T$  corresponding to  $y_T$ .

**Protocol FAIRCOMP** $(g, g_1, f, y_T)(x_A)(x_B)(z_T)$

F1.  $A$  chooses  $r_{A,1}, \dots, r_{A,n_A} \in_R \mathbb{Z}_q$ , computes the commitments

$$C_A = (C_{A,1}, \dots, C_{A,n_A}) = (g^{x_{A,1}} g_1^{r_{A,1}}, \dots, g^{x_{A,n_A}} g_1^{r_{A,n_A}}),$$

sends  $C_A$  to  $B$ , and runs with  $B$

$$PK \{ \alpha_1, \beta_1, \dots, \alpha_{n_A}, \beta_{n_A} : C_{A,1} = g^{\alpha_1} g_1^{\beta_1} \wedge \dots \wedge C_{A,n_A} = g^{\alpha_{n_A}} g_1^{\beta_{n_A}} \}.$$

F2.  $B$  chooses  $r_{B,1}, \dots, r_{B,n_B} \in_R \mathbb{Z}_q$ , computes the commitments

$$C_B = (C_{B,1}, \dots, C_{B,n_B}) = (g^{x_{B,1}} g_1^{r_{B,1}}, \dots, g^{x_{B,n_B}} g_1^{r_{B,n_B}}),$$

sends  $C_B$  to  $A$ , and runs with  $A$

$$PK \{ \alpha_1, \beta_1, \dots, \alpha_{n_B}, \beta_{n_B} : C_{B,1} = g^{\alpha_1} g_1^{\beta_1} \wedge \dots \wedge C_{B,n_B} = g^{\alpha_{n_B}} g_1^{\beta_{n_B}} \}.$$

F3.  $A$  and  $B$  invoke Protocol VFE $(g, g_1, C_A, C_B, f_B)(x_A, r_A)(x_B, r_B)$  with Protocol VOT replaced by Phase I of Protocol EOT using public key  $y_T$  and tag  $C_A \| C_B \| f$ ; that is, in Step V3,  $A$  and  $B$  run Phase I of

$$\text{EOT}(g, g_1, u_{i,0}^A, u_{i,1}^A, C_{B,i}, y_T, C_A \| C_B \| f \| i)(s_{i,0}^A, s_{i,1}^A, r_{i,0}^A, r_{i,1}^A)(x_{B,i}, r_{B,i})(z_T)$$

in parallel for  $i = n + n_A + 1, \dots, n + n_A + n_B$ , where  $s_{i,b}^A$ ,  $u_{i,b}^A$ , and  $r_{i,b}^A$  are the commitments  $u_{i,b}$ , tokens  $s_{i,b}$ , and random strings  $r_{i,b}$  of Protocol VFE. They interrupt Protocol VFE at this point in Step V3. (Note that  $T$  has not been involved so far.)

If this fails,  $B$  simply halts.

F4.  $B$  and  $A$  invoke Protocol  $\text{VFE}(g, g_1, C_B, C_A, f_A)(x_B, r_B)(x_A, r_A)$ , with  $A$ 's and  $B$ 's roles exchanged, and with Protocol  $\text{VOT}$  replaced by Phase I of Protocol  $\text{EOT}$  using public key  $y_T$  and tag  $C_A\|C_B\|f$ ; that is, in Step V3,  $B$  and  $A$  run Phase I of

$$\text{EOT}(g, g_1, u_{i,0}^B, u_{i,1}^B, C_{A,i}, y_T, C_A\|C_B\|f\|i)(s_{i,0}^B, s_{i,1}^B, r_{i,0}^B, r_{i,1}^B)(x_{A,i}, r_{A,i})(z_T)$$

in parallel for  $i = n + 1, \dots, n + n_A$ , where  $s_{i,b}^B$ ,  $u_{i,b}^B$ , and  $r_{i,b}^B$  are the commitments  $u_{i,b}$ , tokens  $s_{i,b}$ , and random strings  $r_{i,b}$  of Protocol  $\text{VFE}$ . They interrupt Protocol  $\text{VFE}$  at this point in Step V3.

If this fails,  $A$  invokes Protocol **abort** with  $T$  and halts.

F5.  $A$  and  $B$  continue with Phase II of the  $\text{EOT}$  protocols started in Step F3. According to Protocol  $\text{EOT}$ ,  $A$  sends  $B$  the corresponding messages,  $B$  checks their contents, and if a check fails or if some message does not arrive,  $B$  invokes Protocol **B-resolve** with  $T$ . If  $T$ 's answer is **abort**,  $B$  halts.

Otherwise  $B$  resumes Protocol  $\text{VFE}$  started in Step F3 with Step V4, and outputs  $O_B$ .

F6.  $B$  and  $A$  continue with Phase II of the  $\text{EOT}$  protocols started in Step F4. According to Protocol  $\text{EOT}$ ,  $B$  sends  $A$  the corresponding messages,  $A$  checks their contents, and if a check fails or if some message does not arrive,  $A$  invokes Protocol **A-resolve** with  $T$ . If  $T$ 's answer is **abort**,  $A$  halts.

Otherwise  $A$  resumes Protocol  $\text{VFE}$  started in Step F4 with Step V4, and outputs  $O_A$ .

We now describe the sub-protocols for aborting and resolving. They also take place in the model of Definition 1, where all parties maintain internal state (private inputs are sometimes mentioned nevertheless). In particular,  $T$  maintains a list of tuples internally and processes all abort and resolve requests atomically. Recall that the transcript of a party of a protocol consists of all messages received or sent by this party.

Protocol **abort** is a protocol between  $A$  and  $T$ ; it is invoked by  $A$  with inputs  $C_A$  and  $C_B$ .

**Protocol abort** $(g, g_1, f, y_T)(C_A, C_B)$

1.  $A$  sends the message (**abort**,  $C_A\|C_B\|f$ ) to  $T$ .
2. If  $T$ 's internal state contains an entry of the form  $(C_A\|C_B\|f, \text{string})$ , then  $T$  returns to  $A$  the message *string*.
3. Otherwise,  $T$  adds the tuple  $(C_A\|C_B\|f, \text{abort})$  to its internal state and returns to  $A$  the message **abort**.

Protocol **B-resolve** is a protocol between  $B$  and  $T$ ; it is invoked by  $B$  with input a string *transcript*, containing  $B$ 's complete transcript of Steps F1–F4 in Protocol **FAIRCOMP**, which includes also  $C_A$  and  $C_B$ .

**Protocol B-resolve** $(g, g_1, f, y_T)(\text{transcript})$

1.  $B$  sends the message (**B-resolve**, *transcript*) to  $T$ .
2. If  $T$ 's internal state contains an entry of the form  $(C_A\|C_B\|f, \text{string})$ , then  $T$  returns to  $B$  the message *string* and halts.
3. Otherwise,  $B$  and  $T$  run Steps V1–V3 of Protocol  $\text{VFE}(g, g_1, C_B, C_A, f_A)(x_B, r_B)(\emptyset)$  unmodified with  $B$  in the role of circuit constructor ( $\text{VFE-}$ ) $A$  and  $T$  in the role of circuit evaluator ( $\text{VFE-}$ ) $B$ . They stop after Step 1 in Protocol  $\text{VOT}$ , before  $T$  would have to decrypt the tokens. (Thus,  $T$ 's inputs to the protocol may be empty.)

If  $T$  rejects any of the proofs by  $B$ , then  $T$  adds the tuple  $(C_A\|C_B\|f, \text{abort})$  to its internal state and returns  $B$  the message **abort**.

4. Otherwise,  $T$  carries out its part of Phase II in the  $n_B$  EOT protocols from Step F3 subject to tags matching  $C_A\|C_B\|f\|i$  for  $i = n + n_A + 1, \dots, n + n_A + n_B$ ; in particular, this involves  $T$  sending to  $B$  the outputs of decryption algorithm  $\text{VD}(\text{CS}, z_T, \dots)$ , which either decrypts correctly if the tags match or outputs  $\perp$  otherwise.

$T$  computes the transcript  $t$  of Protocol **B-resolve** and adds  $(C_A\|C_B\|f, \text{resolve}\|t)$  to its internal state.

Protocol **A-resolve** is a protocol between  $A$  and  $T$ ; it is invoked by  $A$  with input a string *transcript*, containing her complete transcript of Steps F1–F3 in Protocol **FAIRCOMP**, which includes also  $C_A$  and  $C_B$ .

**Protocol A-resolve** $(g, g_1, f, y_T)(\text{transcript})$

1.  $A$  sends the message  $(\text{A-resolve}, \text{transcript})$  to  $T$ .
2. If  $T$ 's internal state contains an entry of the form  $(C_A\|C_B\|f, \text{string})$ , then  $T$  returns to  $A$  the message *string* and halts.
3. Otherwise,  $A$  and  $T$  run Steps V1–V3 of Protocol  $\text{VFE}(g, g_1, C_A, C_B, f_B)(x_A, r_A)(\emptyset)$  unmodified with  $A$  in the role of circuit constructor (**VFE**-) $A$  and  $T$  in the role of circuit evaluator (**VFE**-) $B$ . They stop after Step 1 in Protocol **VOT**, before  $T$  would have to decrypt the tokens. (Thus,  $T$ 's inputs to the protocol may be empty.)

If  $T$  rejects any of the proofs by  $A$ , then  $T$  adds the tuple  $(C_A\|C_B\|f, \text{abort})$  to its internal state and returns  $A$  the message **abort**.

4. Otherwise,  $T$  carries out its part of Phase II in the  $n_A$  EOT protocols from Step F4 subject to tags matching  $C_A\|C_B\|f\|i$  for  $i = n + 1, \dots, n + n_A$ ; in particular, this involves  $T$  sending to  $A$  the outputs of decryption algorithm  $\text{VD}(\text{CS}, z_T, \dots)$ , which either decrypts correctly if the tags match or outputs  $\perp$  otherwise.

$T$  computes the transcript  $t$  of Protocol **A-resolve** and adds  $(C_A\|C_B\|f, \text{resolve}\|t)$  to its internal state.

Remarks about the protocol.

1. Protocol **FAIRCOMP** as described above consists of seven rounds (14 moves). By pipelining the execution of Steps F1–F4 one can reduce this to five rounds (10 moves).
2. A major difference between the resolve protocols here and those used for optimistic fair exchange of signatures [ASW00] is that  $T$  cannot directly replace the other party here. Whereas in a fair exchange of digital signatures,  $T$  can verify that the party requesting to resolve supplies a correct signature,  $T$  has to re-run almost the complete **VFE** protocol with the requesting party here. After  $T$  has done this, the other party is able to complete **VFE** and its part of the computation from this transcript.
3.  $T$  does not have to know any secrets of the other party for re-running **VFE**. For instance, in Step 3 of Protocol **B-resolve**, when  $B$  and  $T$  run Protocol **VFE** for  $f_A$  (and  $T$  plays the role of  $A$ ),  $T$  does not have to know anything about  $A$ 's secret input  $x_A$  besides the commitments  $C_A$ ; this follows because the **VFE** protocol is stopped after Step V3 and because of a special feature of the underlying Protocol **VOT**, in which the commitments are used for encryption.

**Theorem 5.** *Under the DDH assumption, Protocol FAIRCOMP above is an optimistic fair secure computation protocol.*

*Proof (Sketch).* We have to show correctness, privacy, and fairness according to Definition 1.

**Correctness.** Correctness is obvious from the construction and the remarks above.

**Privacy.** In short, privacy follows from the privacy of VFE, the remarks regarding the replacement of VOT by EOT above, plus the fact that the simulator can extract one party's inputs from the commitments it makes.

We show privacy for  $A$  against colluding  $B$  and  $T$  in more detail. (Privacy for  $B$  follows from essentially the same argument.) Consider the differences between a standard execution of Protocol FAIRCOMP and the resolve protocols:

1. Protocol B-resolve corresponds to replacing VOT by EOT (and actually invoking  $T$ ) in Protocol VFE for computing  $f_B$ .

However, the properties of Protocol EOT (cf. Section 4.2) guarantee the privacy of the input for  $A$  (in the role of  $S$ ) even against  $B$  (in the role of  $R$ ) colluding with  $T$ ; this follows because EOT reduces to VOT in that case and then from Lemma 1.

2. In Protocol A-resolve,  $T$  plays the role of circuit evaluator  $B$  and this does not change anything about the privacy of  $A$ 's inputs.

The two observations above imply that the analysis reduces to Protocol VFE between  $A$  and the circuit evaluator  $B^*$  (including  $T^*$ ) with commitments computed as in Steps F1 and F2. Privacy for  $A$  is shown by using essentially the same simulator as in the proof of Theorem 4. The only difference is that the extractor  $EX$  rewinds  $B^*$  in Step F2, after  $B^*$  has produced the commitments  $C_B^*$ , and extracts the values  $x_{B,i}^*$  and  $r_{B,i}^*$  from the proof of knowledge. This replaces the the commitment oracle in the proof for VFE.

**Fairness.** Consider fairness for  $A$ : we have to show that if  $B^*$  can compute anything about  $A$ 's secret inputs, then  $A$  obtains  $f_A$  on some value extracted from  $B^*$ 's messages.

The properties of EOT guarantee that up to the end of Phase I,  $B^*$  does not learn anything about  $A$ 's inputs. This holds certainly up to the end of Step F4 in Protocol FAIRCOMP. At this point  $A$ 's and  $B^*$ 's inputs are committed in  $C_A$  and  $C_B^*$ .

It follows from the proof of privacy above, from the non-malleability of  $T$ 's cryptosystem, and from the fact that  $T$  is honest and verifies the tags  $C_A \| C_B^* \| f \| \dots$  that even if  $B^*$  invokes B-resolve, he obtains the function evaluated *on the committed inputs* in  $C_A$  and  $C_B^*$ . But after this point (i.e., after Step F4),  $A$  may always invoke A-resolve and obtain  $f_A$  evaluated at the *same* inputs committed to in  $C_A$  and  $C_B^*$ .

Fairness for  $B$  can be shown analogously and needs only one additional step. Suppose  $A$  invokes **abort** and  $T$  accepts this by returning **abort**; after that,  $B$  will no more receive anything useful by invoking B-resolve.

By the logic of the protocol,  $B$  will halt in Step F5 and not send his secrets to  $A$  in Phase II of EOT. And because  $T$  is honest, it will not give useful information to  $A$  in future A-resolve requests.  $\square$

## References

- [ASW97] N. Asokan, M. Schunter, and M. Waidner, *Optimistic protocols for fair exchange*, Proc. 4th ACM Conference on Computer and Communications Security, 1997, pp. 6, 8–17.

- [ASW00] N. Asokan, V. Shoup, and M. Waidner, *Optimistic fair exchange of digital signatures*, IEEE Journal on Selected Areas in Communications **18** (2000), no. 4, To appear.
- [BCDvdG88] E. F. Brickell, D. Chaum, I. Damgård, and J. van de Graaf, *Gradual and verifiable release of a secret*, Advances in Cryptology: CRYPTO '87 (C. Pomerance, ed.), Lecture Notes in Computer Science, vol. 293, Springer, 1988.
- [Bea91] D. Beaver, *Secure multiparty protocols and zero-knowledge proof systems tolerating a faulty minority*, Journal of Cryptology **4** (1991), no. 2, 75–122.
- [BG92] M. Bellare and O. Goldreich, *On defining proofs of knowledge*, Advances in Cryptology: CRYPTO '92 (E. F. Brickell, ed.), Lecture Notes in Computer Science, vol. 740, Springer-Verlag, 1992, pp. 390–420.
- [BGMR90] M. Ben-Or, O. Goldreich, S. Micali, and R. L. Rivest, *A fair protocol for signing contracts*, IEEE Transactions on Information Theory **36** (1990), no. 1, 40–46.
- [BM90] M. Bellare and S. Micali, *Non-interactive oblivious transfer and applications*, Advances in Cryptology: CRYPTO '89 (G. Brassard, ed.), Lecture Notes in Computer Science, vol. 435, Springer, 1990, pp. 547–557.
- [BMR90] D. Beaver, S. Micali, and P. Rogaway, *The round complexity of secure protocols*, Proc. 22nd Annual ACM Symposium on Theory of Computing (STOC), 1990, pp. 503–513.
- [BW98] B. Baum-Waidner and M. Waidner, *Optimistic asynchronous multi-party contract signing*, Research Report RZ 3078 (#93124), IBM Research, November 1998.
- [Can98] R. Canetti, *Security and composition of multi-party cryptographic protocols*, Report 98-18, Theory of Cryptography Library, 1998.
- [CD97] R. Cramer and I. Damgård, *Linear zero-knowledge—a note on efficient zero-knowledge proofs and arguments*, Proc. 29th Annual ACM Symposium on Theory of Computing (STOC), 1997.
- [CD98] J. Camenisch and I. Damgård, *Verifiable encryption and applications to group signatures and signature sharing*, Tech. Report RS-98-32, BRICS, Department of Computer Science, University of Aarhus, December 1998.
- [CDS94] R. Cramer, I. Damgård, and B. Schoemakers, *Proofs of partial knowledge and simplified design of witness hiding protocols*, Advances in Cryptology: CRYPTO '94 (Y. G. Desmedt, ed.), Lecture Notes in Computer Science, vol. 839, 1994.
- [CP93] D. Chaum and T. P. Pedersen, *Wallet databases with observers*, Advances in Cryptology: CRYPTO '92 (E. F. Brickell, ed.), Lecture Notes in Computer Science, vol. 740, Springer-Verlag, 1993, pp. 89–105.
- [CS97] J. Camenisch and M. Stadler, *Efficient group signature schemes for large groups*, Advances in Cryptology: CRYPTO '97 (B. Kaliski, ed.), Lecture Notes in Computer Science, vol. 1233, Springer, 1997, pp. 410–424.
- [CS98] R. Cramer and V. Shoup, *A practical public-key cryptosystem provably secure against adaptive chosen-ciphertext attack*, Advances in Cryptology: CRYPTO '98 (H. Krawczyk, ed.), Lecture Notes in Computer Science, vol. 1462, Springer, 1998.

- [CvdGT95] C. Crépeau, J. van de Graaf, and A. Tapp, *Committed oblivious transfer and private multi-party computation*, Advances in Cryptology: CRYPTO '95 (D. Coppersmith, ed.), Lecture Notes in Computer Science, vol. 963, Springer, 1995.
- [Dam99] I. Damgård, *Concurrent zero-knowledge is easy in practice*, Report 99-14, Theory of Cryptography Library, 1999, <http://philby.ucsd.edu/>.
- [DDN91] D. Dolev, C. Dwork, and M. Naor, *Non-malleable cryptography (extended abstract)*, Proc. 23rd Annual ACM Symposium on Theory of Computing (STOC), 1991, pp. 542–552.
- [EGL85] S. Even, O. Goldreich, and A. Lempel, *A randomized protocol for signing contracts*, Communications of the ACM **28** (1985), 637–647.
- [ElG85] T. ElGamal, *A public key cryptosystem and a signature scheme based on discrete logarithms*, IEEE Transactions on Information Theory **31** (1985), no. 4, 469–472.
- [FFS88] U. Feige, A. Fiat, and A. Shamir, *Zero-knowledge proofs of identity*, Journal of Cryptology **1** (1988), 77–94.
- [FKN94] U. Feige, J. Kilian, and M. Naor, *A minimal model for secure computation (extended abstract)*, Proc. 26th Annual ACM Symposium on Theory of Computing (STOC), 1994, pp. 554–563.
- [GM84] S. Goldwasser and S. Micali, *Probabilistic encryption*, Journal of Computer and System Sciences **28** (1984), 270–299.
- [GMR88] S. Goldwasser, S. Micali, and R. L. Rivest, *A digital signature scheme secure against adaptive chosen-message attacks*, SIAM Journal on Computing **17** (1988), no. 2, 281–308.
- [GMW87] O. Goldreich, S. Micali, and A. Wigderson, *How to play any mental game or a completeness theorem for protocols with honest majority*, Proc. 19th Annual ACM Symposium on Theory of Computing (STOC), 1987, pp. 218–229.
- [Gol98] O. Goldreich, *Secure multi-party computation*, Manuscript, 1998, (Version 1.1).
- [GRR98] R. Gennaro, M. O. Rabin, and T. Rabin, *Simplified VSS and fast-track multi-party computations with applications to threshold cryptography*, Proc. 17th ACM Symposium on Principles of Distributed Computing (PODC), 1998.
- [Mic98] S. Micali, *Secure protocols with invisible trusted parties*, Presentation at the Workshop on Multi-Party Secure Protocols, Weizmann Institute of Science, Israel, June 1998.
- [MR92] S. Micali and P. Rogaway, *Secure computation*, Advances in Cryptology: CRYPTO '91 (J. Feigenbaum, ed.), Lecture Notes in Computer Science, vol. 576, Springer, 1992, pp. 392–404.
- [NR97] M. Naor and O. Reingold, *Number-theoretic constructions of efficient pseudo-random functions*, Proc. 38th IEEE Symposium on Foundations of Computer Science (FOCS), 1997.
- [Rab81] M. O. Rabin, *How to exchange secrets by oblivious transfer*, Tech. Report TR-81, Harvard University, 1981.

- [Sch91] C. P. Schnorr, *Efficient signature generation by smart cards*, Journal of Cryptology 4 (1991), 161–174.
- [Sta96] M. Stadler, *Publicly verifiable secret sharing*, Advances in Cryptology: EUROCRYPT '96 (U. Maurer, ed.), Lecture Notes in Computer Science, vol. 1233, Springer, 1996, pp. 190–199.
- [Yao82] A. C. Yao, *Protocols for secure computation*, Proc. 23rd IEEE Symposium on Foundations of Computer Science (FOCS), 1982, pp. 160–164.
- [Yao86] A. C. Yao, *How to generate and exchange secrets*, Proc. 27th IEEE Symposium on Foundations of Computer Science (FOCS), 1986, pp. 162–167.