

Optimistic Fair Secure Computation

(Extended Abstract)

Christian Cachin and Jan Camenisch

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

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. Communication is over an asynchronous network. All our protocols are based on efficient proofs of knowledge and involve no general zero-knowledge tools. As intermediate steps we describe efficient verifiable oblivious transfer and verifiable secure function evaluation protocols, whose security is proved under the decisional Diffie-Hellman assumption.

1 Introduction

Secure computation between distrusting parties is a fundamental problem in cryptology. 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 [36, 26, 25].

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 [36]. 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 [36, 9, 15, 8].

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 [1, 2] 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 [2]). 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 large initial costs. All our zero-knowledge proofs and verifiable primitives are based on proofs of knowledge about discrete logarithms, without resorting to expensive 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" [35]), which has applications to on-line bidding and auctions.

Baum and Waidner [3] and Micali [29] have observed before that fair two-party computation is feasible in the optimistic model. They used general tools and did not focus on efficient protocols for small circuits, however.

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 [36]: 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).

However, this solution is not sufficient for fair secure computation in the optimistic model. We need to escrow some information in 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 3.4.)

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 [10] 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 [6] give a constant-round cryptographic protocol for multi-party computation. Its specialization to three parties is related to our three-party model in that it guarantees fairness against one malicious party, but T needs to be always involved.

Fair protocols for two-party computation (and extensions to multiple parties) have previously been investigated by Chaum, Damgård, and van de Graaf [13], by Beaver and Goldwasser [5], and by Goldwasser and Levin [27]. They combine oblivious circuit evaluation with gradual release techniques to obtain fairness, but without focus on particularly efficient protocols.

Feige, Kilian, and Naor [24] study an 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. In our model, T is not involved in regular computations and only used in case some party 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 $\|$.

The statistical difference between two probability distributions P_X and P_Y is denoted by $|P_X - P_Y|$. A quantity ϵ_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 is 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 $|\text{Prob}[D(X_k) = 1] - \text{Prob}[D(Y_k) = 1]|$ 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 simulatability paradigm for general secure multi-party computation [4, 30, 25, 12], although we consider only three parties. In this paradigm, the requirements on 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.

First, one has to define the real-world model and the ideal process. We assume static corruption throughout this work.

The real-world model. We consider an asynchronous three-party protocol as a collection (A, B, T) of PITM. 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, τ) to its communication output tape. The message τ is then added to \mathcal{M} , tagged with (P, R) . \mathcal{S} repeats this step arbitrarily often and 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 $P \in \{A, B\}$, but may suppress messages between A and B .) Honest parties eventually generate an output as prescribed by the protocol 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} and at most two of the parties A , B , and T . Parties controlled by the adversary are called corrupt; we assume their output is empty. The adversary itself outputs

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. we assume the adversary is deterministic. For a fixed adversary C and inputs x_A and x_B , the joint output of A , B , T , and C , denoted by $O_{ABTC}(x_A, x_B)$, is a random variable 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} sends two distinct messages to U in arbitrary order, one containing a value $b_A \in \mathcal{Y}_A \cup \{\diamond, \perp\}$ and the other one containing a value $b_B \in \mathcal{Y}_B \cup \{\diamond, \perp\}$. Messages are delivered instantly.

U is a device that computes two messages, m_A and m_B , for \bar{A} and \bar{B} , respectively. 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 is given by

$$m_A = \begin{cases} f_A(x_A, x_B) & \text{if } b_A = \diamond \text{ and } x_A \neq \perp \text{ and } x_B \neq \perp \\ \perp & \text{if } b_A = \diamond, \text{ but } x_A = \perp \text{ or } x_B = \perp \\ b_A & \text{if } b_A \neq \diamond. \end{cases}$$

m_B is computed analogously from x_A, x_B , and b_B .

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

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 a corrupted party. Corrupted parties output nothing, but the adversary outputs an arbitrary function of all information gathered in the protocol.

For a fixed (deterministic) adversary \bar{C} and inputs x_A and x_B , the output of the ideal process is the concatenation of all outputs, denoted by $O_{\bar{A}\bar{B}\bar{T}\bar{C}}(x_A, x_B)$.

In contrast to most of the literature using the simulation paradigm for secure computation, each party (including U) 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.

Simulatability. We are now ready to state the definition of fair secure computation. Seemingly separate requirements on a protocol such as correctness, privacy, and fairness are expressed via the simulatability by an ideal process. Recall that an adversary in the real world is an algorithm C that controls S and at most two of the three parties 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} in the ideal process such that for all $x_A \in \mathcal{X}_A$ and for all $x_B \in \mathcal{X}_B$, the joint distribution of all outputs of the ideal process is computationally indistinguishable from the outputs in the real world, i.e.,

$$O_{ABTC}(x_A, x_B) \stackrel{c}{\approx} O_{\bar{A}\bar{B}\bar{T}\bar{C}}(x_A, x_B).$$

A fair secure computation protocol is called optimistic if whenever all parties follow the protocol and messages between them are delivered instantly, then 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 colluding with A or B . This is unavoidable because a cheating participant of a two-party protocol may always refuse to send the last message. Protocols to defend against such misbehavior require a number of rounds of interaction that is inverse proportional to the cheating probability [36, 9].
2. Conversely, if T is corrupt, then the computation may be unfair and an honest party, say A , may not receive its output. Moreover, B and T may still decide to block A after seeing f_B and even cause A to output a value that has nothing to do with f_A . 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 $b_A \neq \diamond$. But notice that \bar{T} and \bar{B} together do not learn more about Alice's input than what follows from f_B .
3. A stronger requirement would be that T is only permitted to send \diamond or \perp , but not a substitute for A or B 's output. The current model reflects a corresponding property of our protocol because T 's actions in the resolve protocols are not verifiable. However, by making all proofs non-interactive and resorting to the random oracle model, our protocol satisfies also this stronger requirement.
4. 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 for concurrent execution of multiple protocol instances can be constructed by combining our model with techniques proposed by Asokan et al. [2]. Basically, a unique transaction identifier has to be added to all messages and techniques for concurrent composition of zero-knowledge proofs have to be used [20].

3 Proofs of Knowledge and Verifiable Encryption

This section introduces our notation for proofs of knowledge about discrete logarithms, the notion for verifiable encryption, and our escrow scheme. 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 [28]. 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 [22]. 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 probabilistic polynomial-time (PPT) algorithm that distinguishes with non-negligible probability between D and R . By a random self-reduction property [34, 31], the DDH assumption is equivalent to assuming that there is no 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 [31].

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, Damgård, and Schoenmakers [17]. The notation was first used by Camenisch and Stadler [11] 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 $y \in G$ [33]. 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 $t = g^s y^c$. 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., [20]); this allows also arbitrarily large challenges (i.e., $k' = O(k)$).

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. [17]. 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 [14]. 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 arbitrary $a \in \mathbb{Z}_q$, which 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 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 [34], key escrow, and optimistic fair exchange [2]. 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 $(s, g^s) \subset \mathbb{Z}_q \times G$. 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. [2], Camenisch and Damgård [10] 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 [10].

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 tag is encrypted under public key y . The 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 verifier's 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 $\text{VD}(\text{ElGamal}, (g, x), c, \text{string})$ is subject to the condition that a *tag* matching *string* is encrypted in c ; VD outputs the witness in this case and \perp in all other cases.

We refer to Camenisch and Damgård [10] for further details of the verifiable encryption scheme.

3.4 Escrow Schemes

A (verifiable) *escrow scheme* [2] is a protocol 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 semantically secure against adaptive chosen-ciphertext attacks [21]. S has a bit string a as private input. T 's private input is z_T , the secret key corresponding to y_T . Furthermore, there is a public input string *tag* for S and R that controls the condition under which T may resolve the escrow of a .

The operation of an escrow scheme consists of two phases. In the first phase, only S and R interact. If R accepts Phase I, then he is guaranteed to receive a 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 (and hence T needs not participate in the protocol at all) or, if this does not happen, R sends T a single request containing *tag*, to which T will reply with a .

Several escrow schemes with different tags may be run concurrently among the same participants.

The security requirements of the escrow scheme are that a malicious R cannot gain any information on a before Phase II. More precisely, for all bit strings a' , a'' , and *tag*, suppose S runs Phase I of the escrow scheme with R^* on *tag* and $a \in \{a', a''\}$ chosen at random. Subsequently R^* interacts arbitrarily with T subject only to the condition that it never submits a request containing *tag* to T ; the escrow scheme is secure if such an R^* cannot distinguish $a = a'$ from $a = a''$ with more than negligible probability.

A secure escrow scheme can be implemented easily using verifiable encryption and a cryptosystem for T that is semantically secure against chosen-ciphertext attacks. We use the Cramer-Shoup cryptosystem [18], denoted by CS , with public key y_T and private key z_T .

In Phase I, S chooses $u \in_R Z_q^*$, computes $A = g^a g_1^u$, and sends A to R . S and R also carry out $\text{PK rep}(g, g_1, A)$ and

$$\text{out} \leftarrow \text{VE}(\text{CS}, y_T, \text{tag})\{\alpha, \beta : A = g^\alpha g_1^\beta\}.$$

In Phase II, S sends a and u to R and R verifies that $A = g^a g_1^u$. If this check fails or if R did not receive a message from S , then R sends to T the message (out, tag) . T runs $\text{VD}(\text{CS}, z_T, \text{out}, \text{tag})$ and sends the output to R . In either case, R learns a .

It is easy to see that this is a secure escrow scheme using the security of CS and the properties of PK and VE .

4 Verifiable Oblivious Transfer

This section describes a variant of oblivious transfer that is needed for our fair secure computation protocol. Oblivious transfer, proposed by Rabin [32] and by Even, Goldreich, and Lempel [23], 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 .

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, van de Graaf, and Tapp [19] is a similar protocol that performs an oblivious transfer *of commitments* such that R ends up being committed to a_c ; Cramer and Damgård [16] 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 *corresponding 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. [23] and Bellare and Micali [7]. 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 $\text{VOT}(g, g_1, A_0, A_1, C)(a_0, a_1, t_0, t_1)(c, r)$

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. (The proof of the following lemma is omitted from this extended abstract.)

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

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 Section 4, 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 and r_A (similarly for B). These are the corresponding commitments used below. (Alternatively, one might assume that A and B generated and exchanged correct commitments beforehand.)

Given concrete implementations of a parties A and B , a protocol execution between A and B with inputs C_A, C_B, x_A, x_B, r_A , and r_B defines naturally the views V_A and V_B of A and B , respectively, which are families of random variables determined by the public input, A 's private input, B 's private input, and the internal random coins. Moreover, if B is deterministic then V_B is a random variable depending only on A 's coin flips.

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 \perp .*

Privacy: *We consider two cases, corresponding to cheating B and cheating A .*
 1. *Privacy for A : $\forall B^*$ there exists a probabilistic polynomial-time algorithm (PPT) 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^* ,*

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

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 brief description of our protocol and the “encrypted circuit construction”; it follows the approach to secure function evaluation developed by Yao [36], 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.

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 details of the verifiable secure function evaluation protocol and its analysis are omitted from this extended abstract.

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, where the output tokens are not directly revealed, but mutually escrowed with T first and opened later. 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 Protocol VOT from Section 4 and the secure escrow scheme based on Cramer-Shoup encryption from Section 3.4.

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 secret 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

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

If B rejects any proof, it outputs \perp and halts.

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

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

If A rejects any proof, it outputs \perp and halts.

F3. A and B invoke a modification of Protocol $VFE(g, g_1, C_A, C_B, f_B)(x_A, r_A)(x_B, r_B)$, where they replace opening the commitments of the output tokens by escrowing them with T . That is, in Step V3, A and B run Phase I of the escrow scheme for each of the values $s_{i,0}, s_{i,1}, r_{i,0}, r_{i,1}$ tagged with $C_A \| C_B \| f_B \| i$ for $i = n - n_O + 1, \dots, n$ in the circuit computing f_B . They interrupt Protocol VFE after Step V3. (Note that T has not been involved so far.)

If this fails, B simply outputs \perp and halts.

F4. B and A invoke a modification of Protocol $VFE(g, g_1, C_B, C_A, f_A)(x_B, r_B)(x_A, r_A)$, where they replace opening the commitments of the output tokens by escrowing them with T . That is, in Step V3, B and A run Phase I of the escrow scheme for each of the values $s_{i,0}, s_{i,1}, r_{i,0}, r_{i,1}$ tagged with $C_A \| C_B \| f_A \| i$ for $i = n - n_O + 1, \dots, n$ in the circuit computing f_A . They interrupt Protocol VFE after Step V3.

If this fails, A invokes Protocol `abort` with T . If T answers `abort`, then A outputs \perp and halts. If T answers `resolve`||*transcript* then A completes the VFE protocol computing f_A as read from *transcript* (continuing with Step V3), outputs O_A , and halts.

F5. A and B continue with Phase II of the escrow protocols started in Step F3. According to this, 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 answers `abort`, then B outputs \perp and halts. If T answers `resolve`||*transcript* then B completes the VFE protocol computing f_B as read from *transcript* (continuing with Step V3), outputs O_B , and halts.

Otherwise B resumes Protocol VFE started in Step F3 with Step V4 and obtains O_B .

F6. B and A continue with Phase II of the escrow protocols started in Step F4. According to this, B sends A the corresponding messages. Then B outputs O_B and halts.

A checks the messages received from B , and if a check fails or if some message does not arrive, A invokes Protocol `A-resolve` with T . If T answers `abort`, A outputs \perp and halts.

If T answers **resolve** $\|transcript$ then A completes the VFE protocol computing f_A as read from $transcript$ from Step V3, outputs O_A , and halts. Otherwise A resumes Protocol VFE started in Step F4 with Step V4, outputs O_A , and halts.

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 $(C_A\|C_B\|f, string)$, 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 $(g, g_1, f, y_T)(transcript)(z_T)$

1. B sends the message (**B-resolve**, $transcript$) to T .
2. If T 's internal state contains an entry of the form $(C_A\|C_B\|f, string)$, then T returns to B the message $string$ and halts.
3. Otherwise, B and T run Steps V1–V3 of Protocol VFE $(g, g_1, C_B, C_A, f_A)(x_B, r_B)(\emptyset)$ unmodified with B 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 B , then T adds the tuple $(C_A\|C_B\|f, \text{abort})$ to its internal state and returns to B the message **abort**.
4. Otherwise, T reads the $transcript$ sent by B and carries out its part of Phase II for the escrows of the tokens on the output wires for f_B from Step F3. T opens the escrows subject to all tags matching $C_A\|C_B\|f_B\|i$. In other words, T runs the decryption algorithm $VD(CS, z_T, \dots)$ and returns the outputs to B if all tags match, or \perp if one or more decryptions yield \perp . 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 $(g, g_1, f, y_T)(transcript)(z_T)$

1. A sends the message **(A-resolve, transcript)** to T .
2. If T 's internal state contains an entry of the form $(C_A||C_B||f, string)$, then T returns to A the message $string$ and halts.
3. Otherwise, A and T run Steps V1–V3 of Protocol 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 $(C_A||C_B||f, abort)$ to its internal state and returns to A the message **abort**.
4. Otherwise, T reads the *transcript* sent by A and carries out its part of Phase II for the escrows of the tokens on the output wires for f_A from Step F4. T opens the escrows subject to all tags matching $C_A||C_B||f_A||i$. In other words, T runs the decryption algorithm $VD(CS, z_T, \dots)$ and returns the outputs to A if all tags match, or \perp if one or more decryptions yield \perp . T computes the transcript t of Protocol A-resolve and adds $(C_A||C_B||f, resolve||t)$ 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 (ten moves). Using non-interactive proofs in the random oracle model, this could even be reduced further to three rounds (six moves).
2. A major difference between the resolve protocols here and those used for optimistic fair exchange of signatures [2] 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 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.

It can be shown that under the DDH assumption, Protocol FAIRCOMP is an optimistic fair secure computation protocol (omitted).

Acknowledgments

We thank Ran Canetti and Victor Shoup for helpful suggestions and discussions about modeling optimistic fair secure computation.

References

1. N. Asokan, M. Schunter, and M. Waidner, "Optimistic protocols for fair exchange," in *Proc. 4th ACM Conference on Computer and Communications Security*, pp. 6, 8–17, 1997.
2. N. Asokan, V. Shoup, and M. Waidner, "Optimistic fair exchange of digital signatures," *IEEE Journal on Selected Areas in Communications*, vol. 18, pp. 591–610, Apr. 2000.
3. B. Baum-Waidner and M. Waidner, "Optimistic asynchronous multi-party contract signing," Research Report RZ 3078 (#93124), IBM Research, Nov. 1998.
4. D. Beaver, "Secure multiparty protocols and zero-knowledge proof systems tolerating a faulty minority," *Journal of Cryptology*, vol. 4, no. 2, pp. 75–122, 1991.
5. D. Beaver and S. Goldwasser, "Multiparty computation with faulty majority (extended announcement)," in *Proc. 30th IEEE Symposium on Foundations of Computer Science (FOCS)*, pp. 468–473, 1989.
6. D. Beaver, S. Micali, and P. Rogaway, "The round complexity of secure protocols," in *Proc. 22nd Annual ACM Symposium on Theory of Computing (STOC)*, pp. 503–513, 1990.
7. M. Bellare and S. Micali, "Non-interactive oblivious transfer and applications," in *Advances in Cryptology: CRYPTO '89* (G. Brassard, ed.), vol. 435 of *Lecture Notes in Computer Science*, pp. 547–557, Springer, 1990.
8. M. Ben-Or, O. Goldreich, S. Micali, and R. L. Rivest, "A fair protocol for signing contracts," *IEEE Transactions on Information Theory*, vol. 36, pp. 40–46, Jan. 1990.
9. E. F. Brickell, D. Chaum, I. Damgård, and J. van de Graaf, "Gradual and verifiable release of a secret," in *Advances in Cryptology: CRYPTO '87* (C. Pomerance, ed.), vol. 293 of *Lecture Notes in Computer Science*, Springer, 1988.
10. J. Camenisch and I. Damgård, "Verifiable encryption and applications to group signatures and signature sharing," Tech. Rep. RS-98-32, BRICS, Department of Computer Science, University of Aarhus, Dec. 1998.
11. J. Camenisch and M. Stadler, "Efficient group signature schemes for large groups," in *Advances in Cryptology: CRYPTO '97* (B. Kaliski, ed.), vol. 1233 of *Lecture Notes in Computer Science*, pp. 410–424, Springer, 1997.
12. R. Canetti, "Security and composition of multi-party cryptographic protocols," *Journal of Cryptology*, vol. 13, no. 1, pp. 143–202, 2000.
13. D. Chaum, I. Damgård, and J. van de Graaf, "Multiparty computations ensuring privacy of each party's input and correctness of the result," in *Advances in Cryptology: CRYPTO '87* (C. Pomerance, ed.), vol. 293 of *Lecture Notes in Computer Science*, Springer, 1988.
14. D. Chaum and T. P. Pedersen, "Wallet databases with observers," in *Advances in Cryptology: CRYPTO '92* (E. F. Brickell, ed.), vol. 740 of *Lecture Notes in Computer Science*, pp. 89–105, Springer-Verlag, 1993.
15. R. Cleve, "Limits on the security of coin flips when half the processors are faulty," in *Proc. 18th Annual ACM Symposium on Theory of Computing (STOC)*, pp. 364–369, 1986.
16. R. Cramer and I. Damgård, "Linear zero-knowledge—a note on efficient zero-knowledge proofs and arguments," in *Proc. 29th Annual ACM Symposium on Theory of Computing (STOC)*, 1997.
17. R. Cramer, I. Damgård, and B. Schoemakers, "Proofs of partial knowledge and simplified design of witness hiding protocols," in *Advances in Cryptology: CRYPTO '94* (Y. G. Desmedt, ed.), vol. 839 of *Lecture Notes in Computer Science*, 1994.

18. R. Cramer and V. Shoup, "A practical public-key cryptosystem provably secure against adaptive chosen-ciphertext attack," in *Advances in Cryptology: CRYPTO '98* (H. Krawczyk, ed.), vol. 1462 of *Lecture Notes in Computer Science*, Springer, 1998.
19. C. Crépeau, J. van de Graaf, and A. Tapp, "Committed oblivious transfer and private multi-party computation," in *Advances in Cryptology: CRYPTO '95* (D. Coppersmith, ed.), vol. 963 of *Lecture Notes in Computer Science*, Springer, 1995.
20. I. B. Damgård, "Efficient concurrent zero-knowledge in the auxiliary string model," in *Advances in Cryptology: EUROCRYPT 2000* (B. Preneel, ed.), vol. 1087 of *Lecture Notes in Computer Science*, pp. 418–430, Springer, 2000.
21. D. Dolev, C. Dwork, and M. Naor, "Non-malleable cryptography (extended abstract)," in *Proc. 23rd Annual ACM Symposium on Theory of Computing (STOC)*, pp. 542–552, 1991.
22. T. ElGamal, "A public key cryptosystem and a signature scheme based on discrete logarithms," *IEEE Transactions on Information Theory*, vol. 31, pp. 469–472, July 1985.
23. S. Even, O. Goldreich, and A. Lempel, "A randomized protocol for signing contracts," *Communications of the ACM*, vol. 28, pp. 637–647, 1985.
24. U. Feige, J. Kilian, and M. Naor, "A minimal model for secure computation (extended abstract)," in *Proc. 26th Annual ACM Symposium on Theory of Computing (STOC)*, pp. 554–563, 1994.
25. O. Goldreich, "Secure multi-party computation." Manuscript, 1998. (Version 1.1).
26. O. Goldreich, S. Micali, and A. Wigderson, "How to play any mental game or a completeness theorem for protocols with honest majority," in *Proc. 19th Annual ACM Symposium on Theory of Computing (STOC)*, pp. 218–229, 1987.
27. S. Goldwasser and L. Levin, "Fair computation of general functions in presence of immoral majority," in *Advances in Cryptology: CRYPTO '90* (A. J. Menezes and S. A. Vanstone, eds.), vol. 537 of *Lecture Notes in Computer Science*, Springer, 1991.
28. S. Goldwasser and S. Micali, "Probabilistic encryption," *Journal of Computer and System Sciences*, vol. 28, pp. 270–299, 1984.
29. S. Micali, "Secure protocols with invisible trusted parties." Presentation at the Workshop on Multi-Party Secure Protocols, Weizmann Institute of Science, Israel, June 1998.
30. S. Micali and P. Rogaway, "Secure computation," in *Advances in Cryptology: CRYPTO '91* (J. Feigenbaum, ed.), vol. 576 of *Lecture Notes in Computer Science*, pp. 392–404, Springer, 1992.
31. M. Naor and O. Reingold, "Number-theoretic constructions of efficient pseudo-random functions," in *Proc. 38th IEEE Symposium on Foundations of Computer Science (FOCS)*, 1997.
32. M. O. Rabin, "How to exchange secrets by oblivious transfer," Tech. Rep. TR-81, Harvard University, 1981.
33. C. P. Schnorr, "Efficient signature generation by smart cards," *Journal of Cryptology*, vol. 4, pp. 161–174, 1991.
34. M. Stadler, "Publicly verifiable secret sharing," in *Advances in Cryptology: EUROCRYPT '96* (U. Maurer, ed.), vol. 1233 of *Lecture Notes in Computer Science*, pp. 190–199, Springer, 1996.
35. A. C. Yao, "Protocols for secure computation," in *Proc. 23rd IEEE Symposium on Foundations of Computer Science (FOCS)*, pp. 160–164, 1982.
36. A. C. Yao, "How to generate and exchange secrets," in *Proc. 27th IEEE Symposium on Foundations of Computer Science (FOCS)*, pp. 162–167, 1986.