Oblivious and Fair Server-Aided Two-Party Computation

Amir Herzberg and Haya Shulman,
Department of Computer Science
Bar Ilan University
Ramat Gan, Israel 52900
Email: {amir.herzberg,haya.shulman}@gmail.com

Abstract

We show efficient, practical (server-aided) secure two-party computation protocols ensuring privacy, correctness and fairness in the presence of malicious (Byzantine) faults. Our requirements from the server are modest. To ensure privacy and correctness, we only assume *trusted execution* service, executing an initialization program provided by both parties. To further ensure fairness, we further assume a *trusted-decryption* service, providing decryption service using known public key.

Both of these trusted services are feasible in practice, and may be useful for additional tasks; both can also be distributed, with linear overhead, for redundancy. Formally, we model them as ideal functionalities, allowing proof of security using the hybrid model. Our fairness-ensuring protocol is *optimistic*, i.e., the decryption service is invoked only in case of faults. We believe that the protocols are sufficiently efficient, to allow deployment, in particular for financial applications.

Keywords: Two-party computation, fair optimistic protocols, server-aided computation.

1. Introduction

Secure computation, beginning with the seminal papers of Yao [1] and Goldreich et al. [2], investigates how to securely compute functionalities over inputs of two (or multiple) parties. Security implies *correctness*, i.e., both parties receive the correct function of the inputs, and *privacy*, i.e., even corrupt participant cannot learn more (e.g., learn secret input of the other party) than his output. Secure computation can trivially be done by a fully trusted third party, which receives the inputs, and then computes and announces the results. The goal of secure computation protocols is to achieve the same impact but *without* a trusted party, i.e., using a protocol between the parties.

Secure computation received a lot of attention during the last two decades, with numerous works, including several implementations, e.g., [3], and few real-world applications [4]. However, it is widely recognised, that (standard) secure computation mechanisms have very high computational costs, which are prohibitive for most applications; moreover, this is unlikely to change in the forseeable future.

Another challenge with (standard) secure computation is *fairness*, especially when focusing on two-party computations (or, in general, without honest majority). Cleve [5] showed that complete fairness cannot be achieved for general two-party computation. Fairness *can* be acheived for *some* non-trivial computations, e.g., see [6], however, it seems prudent to assume that fairness is not possible for most practical secure computation problems.

Like several other works, e.g., [7], our goal is to apply the theory of secure computation to the design of practical systems, in particular, for financial applications. This requires protocols that ensure secure and fair computations, with reasonable efficiency against potentially malicious, and possibly colluding, participants. Formally, we use a hybrid model, where we assume very restricted ideal functionalities; in practice, these ideal functionalities can be implemented by simple, highly-feasible services.

Such practical implementation requires modest computational resources and very limited trust; security is assured as long as the service does not collude with one of the parties. The use of these ideal functionalities (or, in practice,

weakly-trusted services), allows us to address the two main challenges: *efficiency* and *fairness*. We next briefly discuss these challenges.

Efficiency. Secure computation protocols suffer from a significant overhead, especially when considering arbitrary (byzantine) faults, as often required in practice. The basic secure function evaluation technique for two-party computation, [1], ensures security, privacy and integrity only against passive, semi-honest, adversaries that follow the steps of the protocol. This protection however does not suffice for real life systems, where malicious participants may arbitrarily deviate from the prescribed steps of the protocol, e.g., in an attempt to gain an unfair advantage. The basic protocol of [1] was extended by Goldreich et al. [2] to ensure security against malicious adversaries, e.g., those that may try to alter the agreed computation. Several later works improved efficiency, e.g., [8–10]. However, these protocols are still so computationally intensive, that they are impractical for many real life applications, and specifically for most financial applications. Indeed, there is little hope of sufficient improvements in the efficiency of secure computation protocols, to allow their use for such tasks in the forseenable future.

Fairness. Protocols for financial applications, e.g., for currency exchange, must also ensure *fairness* to the transaction performed by the parties, i.e., either both parties receive the result of the computation, e.g., a signed check, or no one does. Indeed, in the malicious model, an adversary can always abort after receiving its output and before the honest party receives output. As early as in 1986, Cleve [5] showed that fairness cannot be achieved for general computation without an honest majority. Hence, to ensure fairness *and* efficiency against malicious faults, as required for practical (financial) applications, some extra assumptions are necessary. Indeed, for fairness, most works assume that the protocol involves an additional party, usually referred to as the *Trusted Third Party (TTP)*, that provides a service ensuring fairness.

An important goal is to reduce the requirements from this service, thereby making the solutions more practical and effective, including computations requirements, communication requirements, implementation complexity requirements and trust requirements. In this work, we study the use of highly restricted third party services; formally, we model these services as simple *ideal functionalities*, allowing us to prove security of our constructions in the hybrid model.

The first third party (or ideal functionality) we define is the *Common Circuit Evaluation (CCE)*, a generalisation of the Common Reference String (CRS) functionality of [11]. The CCE service receives, from each of the two participants (Alice and Bob), a circuit (C_A, C_B), confirms it is the same $(C_A = C_B)$, and produces the result of running it on random inputs. We show that the CCE service allows an efficient implementation of two party secure computation, secure against malicious participants; additional applications are also possible. Note that this service is simple, rather efficient, and easily parallelised (for scalability and availability) as well as distributed (for security against a rogue server).

The second third party (aka ideal functionality) we define is the *decryption service*, which is used to ensure fairness to the two-party computation protocol. This service operates in two phases. During the initial phase, after being invoked by the two parties Alice and Bob, it produces a decryption key and sends to both parties. The first phase is not required in practice: the service can register its encryption key in a public repository, and the parties can retrieve it from there as needed. If one of the parties misbehaves, the decryption service is invoked to restore fairness: it decrypts a message (which it receives from one of the parties) with its secret decryption key, and again sends the result to *both parties*. Note that, like the CCE service, the trusted decryption service is also simple, efficient, and easily parallelised and distributed.

The ideal functionalities capturing the models can be realised, e.g., by weakly-trusted additional parties. Further research is required to determine whether these models are indeed the minimal necessary, or whether the same goals (efficiency, fairness) be achieved in even weaker models; furthermore, the models that we present may also be useful for other tasks.

Following many previous protocols for ensuring fairness using a TTP, our fairness-ensuring protocol is *optimistic*; specifically, the parties involve the TTP only in case of a misbehaviour by the peer or in case of faults. In this sense, our work is related to many existing works on optimistic fair exchange and similar tasks, e.g., see [12, 13], except that these works support only specific, relatively simple interactions, e.g., certified mail, contract signing, and do not hide the values exchanged from the trusted third party. In contrast, our protocol supports arbitrary computations, and does not expose the inputs (or respective outputs) of the parties to the ideal functionality that ensures fairness.

1.1. Related Work

Following to [5], which showed that fairness cannot be achieved for general computation without an honest majority, different relaxed definitions of fairness were considered.

One approach, the gradual release, see [14–22], considers a relaxed notion of fairness, where the output is revealed gradually, and a cheating party does not obtain a significant advantage over the honest party, by aborting early. In order to release the output gradually many rounds of interaction are required, which may render this approach impractical for realistic applications.

Another approach towards ensuring fairness is to use an optimistic trusted third party, which allows to restore complete fairness in case one of the parties aborts; optimistic approach was proposed mostly for specific tasks, esp. fair exchange [12, 13, 23].

In [24], Lindell considered a relaxed notion of fairness, and presented the *legally enforceable fair* secure two-party computation, using trusted third parties. An outcome of the protocol is that either both parties receive the output, or only one receives the output while the other receives a digitally signed check from the other party which can be then used at a court of law or a bank. In contrast to optimistic model, [24] provides a weaker security guarantee by allowing an adversary to breach fairness. Cachin and Camenisch, [25], presented optimistic fair secure computation protocol, with constant number of interactions. Our protocols essentially improve over this earlier work, in efficiency, see comparison and analysis in Section 6.1, provable security, and most notably, by allowing a guarantee to the computation.

1.2. Contributions

This work has both practical and theoretical contributions:

- Efficient, practical protocols for (fair) two party computation, with set-up assumptions formulating the new models which we introduce in this work. The models are implementable using a weakly-trusted 'third party'. Our protocols are secure against malicious (byzantine) faults, but with efficiency comparable to that of the existing protocols that are secure only against honest-but-curious participants.

- Introduction of two new models, the *Common Circuit Evaluation (CCE) model* where we use a CCE service to improve efficiency, and the *decryption model* which provides basic generic facility allowing resolution of conflicts (e.g., for fairness).

1.3. Organisation

In the next section, we present definitions and preliminaries. Then, in Section 3, we introduce the *Common Circuit Evaluation (CCE) model*, allowing offline set-up assumption for greater efficiency; we subsequently use this model in Section 4. In Section 5 we introduce the *decryption functionality* that upon requests validates and decrypts the input ciphertext. We later use this functionality in Section 6 in the construction of a fair two-party protocol, where we use the decryption functionality to restore fairness in case of failures or misbehaviour by one of the participants. In Section 7 we present sample financial applications based on our protocols. Finally, in Section 8, we conclude and present future research directions.

A preliminary version of this work appeared in [26].

2. Definitions and Preliminaries

In this section we present the definition for secure two-party computation, following the standard definitions presented in prior art, specifically [27–29]. We begin, in subsection 2.1, by defining fair two-party computation, for functions $f(x_1, x_2) = (f_1(x_1, x_2), f_2(x_1, x_2))$, where x_1, x_2 are the inputs from the corresponding parties P_1, P_2, with f_1, f_2 defining the corresponding outputs.

Later in the paper, we present two efficient protocols; first, in Section 4, we present a protocol that ensures two-party secure computation for the special case of functions with output only at the second party ('Bob'), i.e., $f(x_1, x_2) = (\perp, g(x_1, x_2))$, assuming a Common Circuit Evaluation (CCE) service. Then, in Section 6, we present

a protocol that ensures *fair* two party secure computation for functions f with outputs at both parties, using the protocol of Section 4 and further assuming and using a Decryption service. In subsection 2.2, we define cryptographic mechanisms used by both protocols.

2.1. Fair Two-Party Computation

The goal of a secure two-party protocol is to allow two parties P_1, P_2 to securely compute a functionality[1] $f(x_1, x_2)$ over their private inputs $(x_1, x_2 \in \{0,1\}^n$, respectively, where $n \in \mathbb{N}$ is the *security parameter*). The functionality f may have different outputs for each party, which we often denote as two single-output functionalities, one for each of the two parties (correspondingly), i.e., $f(x_1, x_2) = (f_1(x_1, x_2), f_2(x_1, x_2))$.

The security requirements are essentially *correctness*, i.e., that the output will be consistent with the inputs and with the function; and *privacy*, i.e., that neither party $P_{i|i\in\{1,2\}}$ can learn more than its output $f_i(x_1, x_2)$.

These security requirements are trivially satisfied, if the parties can use an 'ideal' third party, which would simply compute the functionality and provide each of them with the respective output, i.e., received x_1 from P_1 and x_2 from P_2, and then compute and send $f_i(x_1, x_2)$ to P_i, for $i \in \{1, 2\}$. This 'ideal' trusted third party is usually referred to as the *ideal functionality*, and denoted $\mathcal{F}_f$.

Execution in the Ideal Model

The execution in the ideal model is defined by an interaction of $\mathcal{F}_f$ with a (polytime) machine S, which acts as an adversary, however, for reasons to become apparent later, is referred to as the *simulator*, and with one of the honest parties, $P_{i|i\in\{1,2\}}$. The complete interaction is in Alg. 2.1; notice, in particular, that the adversary selects which party will interact with $\mathcal{F}_f$, and 'takes over' the other party.

1. S selects $i, j \in \{1, 2\}$ s.t. $i \neq j$
2. Let $x'_i = x_i$
3. Let S specify x'_j
4. The ideal functionality $\mathcal{F}_f$ sends $f_i(x'_1, x'_2)$ to P_i, and $f_j(x'_1, x'_2)$ to S.
5. The honest party P_i output what it received, and S outputs arbitrary strings.

Algorithm 1: The ideal execution with functionality $\mathcal{F}_f$ for computing $f(\cdot,\cdot)$, with P_1, P_2, and simulator (adversary) S.

The pair of outputs of the honest party and an adversary S in an ideal execution, where the trusted party computes f, is denoted $\text{IDEAL}_{f,S(z)}(x_1, x_2, n)$; z is an auxiliary input of the adversary S. Notice that this is a function of the function f, the simulator S, the auxiliary information z, the inputs x_1, x_2, and the security parameter n.

Execution in the Real Model

The goal of a secure computation protocol, is to achieve the same results as that of the ideal execution - but using only protocol between the two parties, without depending on any trusted third party/service, and in particular, without using an ideal functionality oracle $\mathcal{F}_f$ for computing f. We next define such execution of the protocol, where it runs against an adversary; this is called the *real model*.

In the real model, a two party protocol $\Pi_f = (\Pi_{f,1}, \Pi_{f,2})$ is executed by P_1 and P_2, without a trusted party. The adversary A controls one of the parties, obtains the inputs of that (corrupted) party, and sends messages on behalf of that party. The honest party i follows the protocol $\Pi_{f,i}$ and returns the output specified by $\Pi_{f,i}$. The adversary outputs an arbitrary function of its view. The pair of outputs of the honest party and of the adversary A in the real protocol execution is denoted $\text{REAL}_{\Pi_f,A(z)(x_1,x_2,n)}$. The details of the real model execution are presented in Alg. 2.1.

[1] A functionality $f(x_1, x_2)$ is essentially a function which has, in addition to x_1, x_2, an additional input r, selected uniformly at random from the set $\{0,1\}^n$; we can write it as the function $f(x_1, x_2; r)$.

1. A selects $i, j \in \{1,2\}$ s.t. $i \neq j$
2. A receives x_j.
3. A, impersonating as P_j, interacts with P_i, which runs $\Pi_{f,i}$ with input x_i.
4. The output of the real execution, denoted $\text{REAL}_{\Pi_f, A(z)(x_1,x_2,n)}$, is the output of P_i together with the output of A.

Algorithm 2: The real execution of protocol Π_f for computing $f(\cdot,\cdot)$, with P_1, P_2, and adversary A.

Emulation of Ideal Model in Real Model

We now define a secure two party protocol, as a protocol where the results of executions in the real model, against arbitrary polytime adversary A, are indistinguishable from the results of executions of some polytime 'simulating-adversary' S, in the ideal model (where the computation is done by the ideal functionality $\mathcal{F}_f$. Namely, a secure protocol in the real model emulates the ideal model.

Definition 2.1 (Secure Fair Two-Party Computation). *Let Π_f be a protocol and let f be a polynomial two-party functionality. Protocol Π_f is said to securely compute f if for every probabilistic polynomial-time adversarial algorithm A in the real model running with Π_f, there exists a probabilistic polynomial-time simulator S in the ideal model, such that for every $x_1, x_2, z \in \{0,1\}^n$, holds*

$$\left\{\text{REAL}_{\Pi_f, A(z)}(x_1, x_2, n)\right\}_{n\in\mathbb{N}} \overset{poly}{=} \left\{\text{IDEAL}_{f,S(z)}(x_1, x_2, n)\right\}_{n\in\mathbb{N}}$$

This definition requires that the fairness achieved in the ideal model is indistinguishable from the fairness achieved in the real protocol execution. Unfortunately, this is *impossible to achieve for arbitrary functionalities*, as well as for typical, useful functionalities, such as exchange of signatures.

One approach to deal with this impossibility, is to relax[2] or eliminate the fairness requirement, and allow the adversary to abort the protocol when only one party received output ('secure computation with abort'). Equivalently (see [29], Section 2.2), we can simply focus on functions with output at only one party, say 'Bob' (the second party), as follows.

Definition 2.2 (Secure Two-Party Computation for functions with output only at Bob). *Let Π be a protocol and let $g : \{0,1\}^n \times \{0,1\}^n \to \{0,1\}^n$ be a polynomial two-party functionality (with single output). Protocol Π is said to securely compute g with output at Bob, if Π securely computes $f(x_1, x_2) = (\perp, g(x_1, x_2))$*

Another approach to deal with the impossibility of secure two-party computation for arbitrary functionalities (with output at both parties), is to assume some trusted third party (of course, 'weaker' than just assuming $\mathcal{F}_f$). In the rest of this subsection, we take this approach, and present a definition of execution in a *hybrid model*, which is similar to the real model, except for the inclusion of one or more trusted parties/services.

We use the hybrid model with abort to present efficient secure two party computation protocols: in Section 4, restricted for functionalities with output only at Bob, and in Section 6, for functionalities with output at both parties; both protocols use simple and highly-efficient trusted services.

Execution in the Hybrid Model

In the hybrid model, the parties run a protocol Π_f between them and during the execution they also have access to an ideal functionality computing some specific functionality (of course, not f). We use the hybrid model for our functionalities: *common circuit evaluation* functionality and the *decryption* functionality.

The pair of outputs of the honest party and the adversary A, in a hybrid protocol execution of a protocol Π_f with ideal functionality $\mathcal{F}_h$, is denoted $\text{HYBRID}^h_{\Pi_f, A(z)(x_1,x_2,n)}$. The details of the hybrid model execution are presented in Alg. 2.1.

We can now present the main security definition which we use: fair two-party computation in the hybrid model, i.e., security of a server-aided two-party computation protocol Π_f (with access to an oracle for h).

[2]Indeed, the fairness requirement as we presented, is often referred to as 'complete fairness', to distinguish from weaker variants; however we do not discuss such variants in this work.

1. A selects $i, j \in \{1,2\}$ s.t. $i \neq j$
2. A receives x_j.
3. A, impersonating as P_j, interacts with P_i, which runs $\Pi_{f,i}$ with input x_i. Both A and P_i may also interact with the ideal functionality $\mathcal{F}_h$.
4. The output of the hybrid execution, denoted $\text{HYBRID}^h_{\Pi_f, A(z)(x_1,x_2,n)}$, is the output of P_i together with the output of A.

Algorithm 3: The hybrid execution of protocol Π_f for computing $f(x_1, x_2)$, with P_1, P_2, adversary A and ideal functionality $\mathcal{F}_h$.

Definition 2.3 (Fair Two-Party Computation in Hybrid Model). *Let Π_f be a protocol and let h be a polynomial two-party functionality. Protocol Π_f is said to securely compute f in the h-hybrid model if for every probabilistic polynomial-time adversarial algorithm A in the hybrid model running with Π_f, there exists a probabilistic polynomial-time simulator S in the ideal model, such that for every $x_1, p_2, z \in \{0,1\}^n$, holds*

$$\left\{\text{HYBRID}^h_{\Pi_f, A(z)}(x_1, x_2, n)\right\}_{n\in\mathbb{N}} \overset{poly}{=} \left\{\text{IDEAL}_{f,S(z)}(x_1, x_2, n)\right\}_{n\in\mathbb{N}}$$

We next define secure computation in the hybrid model, of functions with output only at Bob, as we present later in Section 4. Note that such functions can also be computed in the real model; however, our protocol (in the hybrid model) allows to significantly improve performance, and we later use it to build the protocol for functionalities with output at both parties.

Definition 2.4 (Secure Two-Party Computation for functions with output only at Bob, in Hybrid Model). *Let Π be a protocol and let $g : \{0,1\}^n \times \{0,1\}^n \to \{0,1\}^n$ be a polynomial two-party functionality (with single output), and h be a polynomial two-party functionality. Protocol Π is said to securely compute g with output at Bob in the h-hybrid model, if Π securely computes $f(x_1, x_2) = (\perp, g(x_1, x_2))$ in the h-hybrid model.*

2.2. Cryptographic Tools

Our constructions use several tools, i.e., standard cryptographic mechanisms:

Authenticated symmetric-key encryption scheme $(\mathcal{K}, \mathcal{E}, \mathcal{D})$, to ensure confidentiality and integrity of the inputs and outputs of the participants, see [30]. When applying $\mathcal{E}_K(x)$ we perform an authenticated encryption of input x using the key K.

Non-malleable public-key encryption scheme $(\mathcal{NG}, \mathcal{NE}, \mathcal{ND})$ to ensure confidentiality and non-malleability, [31].

Signature scheme $(\mathcal{G}, \mathcal{S}, \mathcal{V})$, [32], where we use $\perp$ to denote authentication failure.

Two-party (1-2) oblivious transfer which we denote by the ideal functionality $\mathcal{F}^2_{\mathsf{ot}}$, [33].

3. Common Circuit Evaluation Model

In this section we introduce the Common Circuit Evaluation (CCE) model allowing efficient protocols for secure computation in the malicious setting. The model assumes that two (or multiple[3]) parties in the protocol have access to an ideal functionality (oracle), which upon invocation provides them with output strings (one for each party) that they later use to carry out the protocol between them. The oracle is not involved in the two-party computation, and is used prior to the protocol execution between the two parties, to generate the respective output strings for the parties, which they later use to carry out the computation. The CCE oracle functionality is one of our basic set-up assumptions and it is captured with the ideal functionality $\mathcal{F}_{\mathsf{cce}}$, in Algorithm 4. In the next section we show how the strings, generated by the oracle, can assist in ensuring privacy and correctness to the computation during the protocol execution, while allowing efficiency equivalent to that of the semi-honest setting. The oracle does not learn anything about the inputs

[3] In this work we focus on the two-party computation; the CCE model can be extended to multi-party computation.

Functionality $\mathcal{F}_{\mathsf{cce}}$

Input: C_1 from party P_1 and C_2 from party P_2, input length n
Computation Phase
  $r \xleftarrow{R} \{0,1\}^n$
  if $C_1 \neq C_2$ **then**
    output $\perp$
  Evaluate C_1 on r and obtain $(p_1 \| p_2) \leftarrow C_1(r)$
end
Output: send p_1 to P_1 and p_2 to P_2

Algorithm 4: The functionality $\mathcal{F}_{\mathsf{cce}}$ for executing a 'common' circuit, which description it receives in an input from both parties. The functionality checks if the circuits supplied by both parties are equivalent, in which case it evaluates the circuit on a random string r, and obtains $p_1 \| p_2$ where p_1 is the output of P_1 and p_2 is the output of P_2, and sends the outputs to the respective parties. Otherwise, if the circuits are not the same, then it returns $\perp$.

of the parties from the computation that it performs since it does not obtain the secret inputs of the parties.

The CCE oracle receives a circuit that it should compute. This allows for a general definition, one that allows the parties to define the computation that they wish the oracle to perform. Our CCE model is a generalisation of the Common Reference String (CRS) model, [33, 34]. In the CRS model the parties are given a common public reference string that was chosen from a given distribution. Note that, similarly to CSR, the computation that the CCE oracle performs is not a function of the private inputs of the parties, and neither does it observe the output from the protocol. Furthermore, the parties are not required to identify themselves before participating in the protocol. In contrast, the string produced by the CCE consists of two parts: one public, which both parties obtain access to, and one private, which only the party that initiates the protocol obtains.

More precisely, the $\mathcal{F}_{CCE}$ oracle computes a universal function which receives in an input a description of a function (or rather a description of a circuit C implementing a function that the parties agreed upon) from the parties and evaluates that function on a random string. We capture the CCE model with the ideal functionality $\mathcal{F}_{\mathsf{cce}}$: $\mathcal{F}_{\mathsf{cce}}$ receives a description of two circuits, C_1 and C_2, (computing an agreed upon function) from both participants, executes that circuit on a random string r, obtains two outputs p_1, p_2 and returns each output to the corresponding party. The $\mathcal{F}_{\mathsf{cce}}$ ideal functionality is not involved during the computation performed by the parties.

4. Two-Party Protocol in Common Circuit Evaluation Model

In this section we consider functionalities with output only at Bob (the circuit evaluator). Let $g : \{0,1\}^n \times \{0,1\}^n \to \{0,1\}^n$ be such a two-party functionality, and let a, b be the inputs of Alice and Bob respectively. We construct a two-party protocol, in Algorithm 6, for evaluation of inputs of Alice and Bob on a known function g using the $\mathcal{F}_{CCE}$ functionality presented in Algorithm 4.

During the preprocessing phase, Alice and Bob send a description of a circuit, which they agreed on, to the third party, and receive an output. For the purpose of our construction the output that $\mathcal{F}_{CCE}$ produces is an encoding of a garbled circuit, see description in Section 4.1, which is then used by Alice and Bob to evaluate the function $g : \{0,1\}^n \times \{0,1\}^n \to \{0,1\}^n$ on their respective inputs.

When Alice and Bob have the inputs they run the two-party computation protocol, whereby Alice transfers the strings representing her input to Bob and runs an $\mathcal{F}_{\mathsf{ot}}^2$ functionality (capturing the oblivious transfer protocol, [33]), for strings representing Bob's input bits; Bob evaluates the circuit on both inputs, concluding the protocol. After evaluating the g on a and b, Bob obtains $g(a, b)$, while Alice learns nothing at all.

The preprocessing phase ensures that the circuit was correctly constructed and prevents cheating by either party.

In Section 4.1 we present a description of the circuit that Alice and Bob send to the $\mathcal{F}_{CCE}$ functionality, which allows for garbled circuit pre-generation, and in Section 4.2 we construct the two-party protocol in the preprocessing model.

4.1. Garbled Circuit Pre-Generation

To allow for evaluation of arbitrary functions we define a garbler to be a circuit that receives in an input a description of a circuit and generates a garbled circuit from it. The garbler is illustrated in Algorithm 5. It receives in an

input a circuit C_g that the parties agreed on; C_g is parametrised by a function $g : \{0,1\}^n \times \{0,1\}^n \to \{0,1\}^n$. The circuit generates a signature key pair (sk_{CCE}, vk_{CCE}), then constructs and garbles a circuit C, and signs the circuit with the freshly generated signature key sk_{CCE}. We use part of the string r (which is selected at random by $\mathcal{F}_{\mathsf{cce}}$) for garbled circuit generation, and another part for keys generation and signatures. Then C will be used by the parties during the protocol execution.

The circuit C_g, in Algorithm 5, consists of gates; in the first level each gate has two input wires, one for input bit of Alice and another for input bit of Bob. $\mathcal{F}_{\mathsf{cce}}$ generates a garbled circuit C as follows: it first modifies the circuit C_g to a circuit C'_g where each input wire of Bob is replaced with a XOR-gate with s input wires; Bob later uses this redundancy, to thwart the attempts by a malicious Alice to expose his secret inputs, by providing Bob with incorrect random strings for his input values (during the oblivious transfer protocol); see [8] for details. Next, random strings, corresponding to each input bit of Alice and Bob, are generated, and the gates of the circuit are replaced with garbled gates, i.e., boolean tables incorporating the outputs from a gate for all possible combinations of inputs of Alice and Bob.

Eventually, after running the garbler (i.e., the circuit in Algorithm 5) on the input circuit C_g, $\mathcal{F}_{\mathsf{cce}}$ sends the random input strings (corresponding to all possible inputs) to Alice, and the garbled gates and output decryption tables to Bob. Note that it is possible to send the entire output (the signed garbled circuit) to Alice and the output of Bob can be set to $\perp$ (i.e., no private output), and Alice in turn will forward to Bob the signed garbled tables and output decryption tables. For efficiency (and simplicity) we let $\mathcal{F}_{\mathsf{cce}}$ send the garbled tables and output decryption tables directly to Bob (this is only a simplifying assumption since these tables constitute an output that is not secret, i.e., known to both Alice and Bob).

4.2. Two-Party Protocol Construction

In Algorithm 6 we construct a two-party protocol by applying the $\mathcal{F}_{\mathsf{cce}}$ functionality during the preprocessing phase. The resulting two-party protocol illustrates one of the applications of the preprocessing model. We also use this mechanism in the subsequent section as a building block in our fair two-party computation protocol. The procedure that generates the garbled circuit using a third party[4] during the preprocessing phase can be of independent interest to enhance efficiency (see Section 4.2.2 for efficiency comparison of techniques employed in malicious model) of two-party protocols in malicious setting. Assuming preprocessing phase allows a simpler and much more efficient protocol (cf. to [8, 35–37]).

Input: C_g, s, n

1. Generate signature key-pair: $(vk_{CCE}, sk_{CCE}) \leftarrow \mathcal{G}(1^s)$
2. Let C_g be a circuit that computes g
3. Construct C'_g from C_g by replacing each input wire of Bob with a XOR-gate of s new input wires of Bob
4. Garble the circuit C'_g to obtain C. The garbled circuit consists of:
   (a) Random strings corresponding to all possible input bits of Alice: $\overline{\mathcal{K}_A} = ((\mathcal{K}_A^0[1], \mathcal{K}_A^1[1]), ..., (\mathcal{K}_A^0[n], \mathcal{K}_A^1[n]))$
   (b) Random strings corresponding to all possible input bits of Bob: $\overline{\mathcal{K}_B} = ((\mathcal{K}_B^0[1], \mathcal{K}_B^1[1]), ..., (\mathcal{K}_B^0[sn], \mathcal{K}_B^1[sn]))$
   (c) Garbled boolean tables $\overline{\mathcal{T}_G}$ for each garbled gate G of the circuit
   (d) Output decryption tables $\overline{\mathcal{T}_D}$ mapping output strings to bits
5. Sign the random input strings $\overline{\mathcal{K}_A}$ of Alice: $\overline{\sigma}_A = ((\sigma_A^0[1], \sigma_A^1[1]), ..., (\sigma_A^0[n], \sigma_A^1[n]))$ where $\forall i, j : \sigma_A^j[i] = \mathcal{S}_{sk_{pre}}(\mathcal{K}_A^j[i], i)$
6. Sign the random input strings $\overline{\mathcal{K}_B}$ of Bob: $\overline{\sigma}_B = ((\sigma_B^0[1], \sigma_B^1[1]), ..., (\sigma_B^0[sn], \sigma_B^1[sn]))$ where $\forall i, j : \sigma_B^j[i] = \mathcal{S}_{sk_{pre}}(\mathcal{K}_B^j[i], i, j)$

Output: $((\overline{\mathcal{K}_A}, \overline{\sigma}_A), (\overline{\mathcal{K}_B}, \overline{\sigma}_B), vk_{CCE})$ to Alice and $(\overline{\mathcal{T}_G}, \overline{\mathcal{T}_D}, vk_{CCE})$ to Bob.

Algorithm 5: The description of a garbler circuit, that receives in an input a circuit C_g, that both parties agreed upon, and generates a garbled circuit C.

4.2.1. Security Analysis

We analyse Π_g in a hybrid model where there is a trusted party computing $\mathcal{F}_{\mathsf{cce}}$ and $\mathcal{F}_{\mathsf{ot}}^2$. The simulator S interacts with the ideal functionality $\mathcal{F}_g$ and uses the adversary A in a black-box manner, simulating for A the real protocol execution and emulating the ideal functionalities $\mathcal{F}_{\mathsf{cce}}$ and $\mathcal{F}_{\mathsf{ot}}^2$.

[4]Especially when the third party is unavoidable, e.g., to achieve fairness in general computation, the third party can also be used to run the preprocessing phase.

Input: security parameter s, number of bits n
Output: $y_B = g(a, b)$
Offline Generation Phase
  Alice and Bob send C_1, C_2 (respectively) to $\mathcal{F}_{\mathsf{cce}}$
  Alice receives $((\overline{\mathcal{K}_A}, \overline{\sigma}_A), (\overline{\mathcal{K}_B}, \overline{\sigma}_B), vk_{CCE})$
  Bob receives $(\overline{\mathcal{T}_G}, \overline{\mathcal{T}_D}, vk_{CCE})$ (see Algorithm 5)
end
Computation Phase
  Alice receives $\overline{a} = [a_i]_{i=1}^n$
  Bob receives $\overline{b} = [b_i]_{i=1}^n$
  Input Encoding
    $encodeInput([b_i]_{i=1}^n)$ {
    $b' = \emptyset$
    for $i \leftarrow 1$ ***to*** n **do**
      Let $b_1^i, ..., b_s^i \in_R \{0,1\}$ s.t. $b_i = b_1^i \oplus ... \oplus b_s^i$
      $b' \leftarrow (b' \| b_1^i, ..., b_s^i)$
    return b' }
    //`in n iterations` $b' = [b'_i]_{i=1}^{n \cdot s} = (b_1^1, ..., b_s^1, ..., b_1^n, ..., b_s^n)$
  end
  Alice: sends to Bob: $((\mathcal{K}_A^{a[1]}[1], \sigma_A^{a[1]}[1]), ..., (\mathcal{K}_A^{a[n]}[n], \sigma_A^{a[n]}[n]))$
  Bob:
    if $\exists(\mathcal{K}_A^{a[i]}[i], \sigma_A^{a[i]}[i])$, *s.t.*, $\mathcal{V}_{vk_{CCE}}(\mathcal{K}_A^{a[i]}[i], i, \sigma_A^{a[i]}[i]) = \mathsf{false}$ **then**
      output $\perp$ and halt
    for $i \leftarrow 1$ ***to*** $n \cdot s$ **do**
      run with Alice $\mathcal{F}_{\mathsf{ot}}^2((\mathcal{K}_B^0[i], \sigma_B^0[i]), (\mathcal{K}_B^1[i], \sigma_B^1[i]), b'_i)$
      //`run oblivious transfer, Alice provides` $(\mathcal{K}_B^0[i], \sigma_B^0[i]), (\mathcal{K}_B^1[i], \sigma_B^1[i])$ `and Bob` b'_i
      receive $(\mathcal{K}_B^{b'[i]}[i], \sigma_B^{b'[i]}[i])$
      if $\mathcal{V}_{vk_{CCE}}(\mathcal{K}_B^{b'[i]}[i], \sigma_B^{b'[i]}[i]) == \mathsf{false}$ **then**
        output $\perp$ and halt
    $(y_B = (y_B[1], ..., y_B[n])) \leftarrow C((\mathcal{K}_A^{a[1]}[1], ..., \mathcal{K}_A^{a[n]}[n]), (\mathcal{K}_B^{b'[1]}[1], ..., \mathcal{K}_B^{b'[sn]}[sn]))$ (below)
  end
end
Circuit Evaluation
  $C((\mathcal{K}_A^{a[1]}[1], ..., \mathcal{K}_A^{a[n]}[n]), (\mathcal{K}_B^{b'[1]}[1], ..., \mathcal{K}_B^{b'[sn]}[sn]))$ {
    $(\mathcal{K}_Y^{y[1]}[1], ..., \mathcal{K}_Y^{y[n]}[n]) \leftarrow \overline{\mathcal{T}_G}((\mathcal{K}_A^{a[1]}[1], ..., \mathcal{K}_A^{a[n]}[n]), (\mathcal{K}_B^{b'[1]}[1], ..., \mathcal{K}_B^{b'[sn]}[sn]))$
  return $\omega \leftarrow \overline{\mathcal{T}_D}(\mathcal{K}_Y^{y[1]}[1], ..., \mathcal{K}_Y^{y[n]}[n])$ }
end
$//C(a, b) = \overline{\mathcal{T}_G}(\overline{\mathcal{T}_D}(\overline{\mathcal{K}}_A^a, \overline{\mathcal{K}}_B^b))$

Algorithm 6: Secure Two Party Protocol Π_g^E in the $(\mathcal{F}_{\mathsf{cce}}, \mathcal{F}_{\mathsf{ot}}^2)$-hybrid model, for computing $g(a, b) = y_B$, where $g : \{0,1\}^n \times \{0,1\}^n \rightarrow \{0,1\}^n$.

Claim 4.1. *Let $g : \{0,1\}^n \times \{0,1\}^n \rightarrow \{0,1\}^n$ be a polynomial time two-party functionality. Assume that the signature scheme $(\mathcal{G}, \mathcal{S}, \mathcal{V})$ is existentially unforgeable under a chosen-message attack. Then protocol Π_g securely realises a two-party functionality with output at Bob only (according to Definition 2.4), in the presence of malicious adversaries in the $(\mathcal{F}_{\mathsf{cce}}, \mathcal{F}_{\mathsf{ot}}^2)$-hybrid model.*

Proof. We analyse Π_g in a $(\mathcal{F}_{\mathsf{cce}}, \mathcal{F}_{\mathsf{ot}}^2)$-hybrid model, and show that the execution of Π_g is computationally indistinguishable from computation of g in the ideal model. We prove the Claim 4.1 in Propositions 4.2 and 4.3 for cases where the adversary controlls Alice or Bob, respectively. □

In our construction we assume an ideal functionality $\mathcal{F}_g$. However, the third party is not required to be honest, and is only required to correctly construct the garbled circuit based on the inputs from Alice and Bob, to ensure correctness and privacy when a malicious behaviour of the participants is considered. The privacy of Bob's input is ensured against a malicious third party that colludes with Alice. However, the privacy of Alice is not ensured in case of collusion between the third party and Bob.

Proposition 4.2 (Security Against Malicious Alice). *For every polynomial time adversary A corrupting Alice and running with Π_g with abort in a hybrid model with access to $\mathcal{F}_{\mathsf{cce}}$ and $\mathcal{F}_{\mathsf{ot}}^2$, there exists a probabilistic polynomial-time simulator S corrupting Alice and running in the ideal model with access to an ideal functionality $\mathcal{F}_g$, such that for every $a, b, z \in \{0,1\}^*$ holds:*

$$\left\{\text{IDEAL}_{g,S(a,C_1,z,1^n)}(a,b,n)\right\}_{n\in\mathbb{N}} \overset{poly}{=} \left\{\text{HYBRID}^{\mathcal{F}_{CCE},\mathcal{F}_{\text{ot}}}_{\Pi_g,A(a,C_1,z,1^n)}(a,b,n)\right\}_{n\in\mathbb{N}}$$

Proof. Let A be a malicious static (hybrid model) adversary controlling Alice, and running the protocol in Algorithm 6. We construct an ideal model simulator S which has access to Alice and to the trusted party computing $\mathcal{F}_g$, and can simulate the view of the execution of the protocol to the adversary. In Algorithm 7 we construct a simulator S given a black-box access to A. The view of A in a simulation with S is identical to its view in an $(\mathcal{F}_{\text{cce}},\mathcal{F}^2_{\text{ot}})$-hybrid execution of Π_g with a honest Bob. The joint distribution of A's view and Bob's output in a hybrid execution is identical to the joint distribution of S and Bob's output in an ideal model. In addition, there is only a negligible probability for the adversary to forge the signature, thus the output distribution of the simulator and the honest party in the ideal model is identical to that of the adversary and the honest party in the real protocol execution. □

$S(a, C_1, 1^n)$

- $C_1^A \overset{\mathcal{F}_{CCE}}{\longleftarrow} A(a, C_1, 1^n)$
- **if** $C_1^A = \perp \vee C_1^A \neq C_1$ **then**
  - send $\perp$ to A as its response from $\mathcal{F}_{\text{cce}}$
  - output whatever A outputs and halt
- **else**
  - simulate functionality $\mathcal{F}_{\text{cce}}$ for A:
    1. choose a key pair $(vk, sk) \leftarrow \mathcal{G}(1^n)$
    2. construct C_1' from C_1, by replacing each input wire of Bob with a XOR-gate consisting of s input wires of Bob
    3. garble the resulting circuit C_1' and obtain C, consisting of:
       (a) Random strings corresponding to all possible input bits of Alice: $\bar{\mathcal{K}}_A = ((\mathcal{K}_A^0[0], \mathcal{K}_A^1[0]), ..., (\mathcal{K}_A^0[n], \mathcal{K}_A^1[n]))$
       (b) Random strings corresponding to all possible input bits of Bob: $\bar{\mathcal{K}}_B = ((\mathcal{K}_B^0[0], \mathcal{K}_B^1[0]), ..., (\mathcal{K}_B^0[n], \mathcal{K}_B^1[n]))$
       (c) Garbled boolean tables $\bar{\mathcal{T}}_G$ for each garbled gate G of the circuit C
       (d) Output decryption tables $\bar{\mathcal{T}}_D$ mapping output strings to bits
    4. sign the random input strings $\bar{\mathcal{K}}_B$ of Bob: $\bar{\sigma} = \mathcal{S}_{sk}(\bar{\mathcal{K}}_B)$, where $\bar{\sigma} = ((\sigma_0^0, \sigma_0^1), ..., (\sigma_n^0, \sigma_n^1))$
    5. send $(\bar{\mathcal{K}}_A, (\bar{\mathcal{K}}_B, \bar{\sigma}), vk)$ to A as its output from $\mathcal{F}_{\text{cce}}$;
  - A sends $\bar{\mathcal{K}}'_A$, intended for Bob and $(\bar{\mathcal{K}'}_B, \bar{\sigma}')$ for ideal functionality $\mathcal{F}^2_{\text{ot}}$
  - **if** $((\bar{\mathcal{K}}'_A \neq \bar{\mathcal{K}}_A) \vee ((\bar{\mathcal{K}'}_B, \bar{\sigma}') \neq (\bar{\mathcal{K}}_B, \bar{\sigma})))$ **then**
    - send input $\perp$ to the trusted party computing $\mathcal{F}_{\text{cce}}$ as Alice's input
    - send $\perp$ to A as its input from $\mathcal{F}^2_{\text{ot}}$
    - output whatever A outputs and halt
- A outputs its view and halts, S outputs the same and halts

end

Algorithm 7: Simulator S, simulating the view of Alice.

Proposition 4.3 (Security Against Malicious Bob). *For every polynomial time adversary A corrupting Bob and running with Π_g with abort in a hybrid model with access to $\mathcal{F}_{\text{cce}}$ and $\mathcal{F}^2_{ot}$, there exists a probabilistic polynomial-time simulator S corrupting Bob and running in the ideal model with access to an ideal functionality computing $\mathcal{F}_g$, such that for every $a, b, z \in \{0,1\}^*$ holds:*

$$\left\{\text{IDEAL}_{g,S(a,C_1,z,1^n)}(a,b,n)\right\}_{n\in\mathbb{N}} \overset{poly}{=} \left\{\text{HYBRID}^{\mathcal{F}_{CCE},\mathcal{F}_{\text{ot}}}_{\Pi_g,A(a,C_1,z,1^n)}(a,b,n)\right\}_{n\in\mathbb{N}}$$

Proof. Let A be a malicious static adversary with Alice and Bob running the protocol in Algorithm 6. We construct an ideal model simulator S which has access to Bob and to the trusted party computing $\mathcal{F}_g$, and can simulate the view of the execution of the protocol. Assume that Bob is corrupted by a hybrid model adversary A. In Algorithm 8 we construct a simulator S given a black-box access to A. The security is based on the fact that the 1-2 oblivious transfer functionality $\mathcal{F}^2_{\text{ot}}$ is secure and as a result Bob learns only a single set of random strings, corresponding to its input. The view of A is identical to its view in a $(\mathcal{F}_{\text{cce}},\mathcal{F}^2_{\text{ot}})$-hybrid execution of protocol Π_g with a honest Alice. In addition, the joint distribution of A and Alice's output in a hybrid execution of the protocol is identical to that of S and Alice's output in an ideal execution. □

$S(b, C_2, 1^n)$

$C_2^B \xleftarrow{\text{CCE}} A(b, C_2, 1^n)$

if $C_2^B = \perp \vee C_2^B \neq C_2$ **then**

send $\perp$ to A as its input from $\mathcal{F}_{\text{cce}}$

output whatever A outputs and halt

else

simulate functionality $\mathcal{F}_{\text{cce}}$ for A:

1. choose a key pair $(vk, sk) \leftarrow \mathcal{G}(1^n)$
2. construct C_2' from C_2, by replacing each input wire of Bob with a XOR-gate consisting of s input wires of Bob
3. garble the resulting circuit and obtain C, consisting of:
   (a) Random strings corresponding to all possible input bits of Alice: $\bar{\mathcal{K}_A} = ((\mathcal{K}_A^0[0], \mathcal{K}_A^1[0]), ..., (\mathcal{K}_A^0[n], \mathcal{K}_A^1[n]))$
   (b) Random strings corresponding to all possible input bits of Bob: $\bar{\mathcal{K}_B} = ((\mathcal{K}_B^0[0], \mathcal{K}_B^1[0]), ..., (\mathcal{K}_B^0[n], \mathcal{K}_B^1[n]))$
   (c) Garbled boolean tables $\bar{\mathcal{T}_G}$ for each garbled gate G of the circuit C
   (d) Output decryption tables $\bar{\mathcal{T}_D}$ mapping output strings to bits
4. sign the random input strings $\bar{\mathcal{K}_B}$ of Bob: $\bar{\sigma} = \mathcal{S}_{sk_T}(\bar{\mathcal{K}_B})$, where $\bar{\sigma} = ((\sigma_0^0, \sigma_0^1), ..., (\sigma_n^0, \sigma_n^1))$
5. send $(\bar{\mathcal{T}_G}, \bar{\mathcal{T}_D}, vk)$ to A as its output from $\mathcal{F}_{\text{cce}}$
6. **for** $i \leftarrow 1$ ***to*** $|b|$ **do**
   (a) run $\mathcal{F}_{\text{ot}}^2((\mathcal{K}_B^0[i], \sigma_i^0), (\mathcal{K}_B^1[i], \sigma_i^1), b_i)$, providing $(\mathcal{K}_B^0[i], \sigma_i^0), (\mathcal{K}_B^1[i], \sigma_i^1)$ and A provides b_i
   (b) A receives $(\mathcal{K}_B^{b_i}[i], \sigma_i^{b_i})$
7. output whatever A outputs and halt

end

Algorithm 8: Simulator S, simulating the view of Bob.

4.2.2. Efficiency Discussion

Secure function evaluation based on garbled circuits, [1], allows to perform a two-party computation in a secure manner, i.e., ensuring privacy, correctness and inputs independence (see [38]). The computation is constant-round but ensures security only against semi-honest adversaries. When considering malicious adversaries, which is the typical model in practice, additional security concerns arise, that are not addressed by the basic secure function evaluation protocol, [1, 38].

Any two-party protocol can be transformed into a secure protocol in the malicious setting, e.g.,[2], but the resulting protocol is not constant round. Subsequently, constant round protocols were presented, e.g., see [2, 25, 32, 39–41]. However, these protocols are based on zero knowledge proofs, which renders them inefficient for practical purposes. Also protocols that do not employ zero knowledge were constructed, e.g., [42], however their round complexity is linear in the depth of the circuit.

In [35] the authors apply the cut-and-choose approach to Yao's protocol, which reduces the probability of evaluating an incorrect circuit, and the efficiency is correlated to the cheating probability; specifically, their protocol has a communication overhead of $O(s|C| + sn^2)$ (where n is the number of input bits to the circuit C and s is the statistical security parameter). Then [36] improved the communication complexity of [35] to $O(s|C|)$ using expanders. However as [8] observed, the protocol in [35] is susceptible to 'input corruption' attack; [8] also present a protocol with roughly the same communication complexity as [35], of $O(s|C| + s^2n)$ (this protocol was implemented in [43]). Another improvement to two-party computation in malicious setting was made by [9] using homomorphic encryption; they present a protocol in the common reference string (CRS) model, that has a constant number of rounds, and has an $O(|C|)$ public-key operations (cf. $O(s|C| + s^2n)$ in [8]), and computational complexity of $O(|C|)$ (as opposed to $O(n)$ in [8]). Subsequently, the work of [37], also followed the cut-and choose approach in a different manner and improved the complexity to $O(\frac{s|C|}{log(|C|)})$. Efficiency improvements were also designed for multi-party computation, [44], by optimising AES encryption; their ideas can be applied when implementing the encrypion in our protocols. A new multi-party protocol to securely evaluate reactive arithmetic circuits, offering security against an active adversary in the universally composable security framework, was proposed by [45]; the protocol is based on a design of an efficient 'cut-and-choose' technique. Techniques reducing the size of garbled tables, thus improving computation and communication complexity, were proposed in [46]; the design of the gates rely on a 'free-XOR' technique. [47] present a framework for secure function evaluation using 'privately programmable blocks'.

Our protocol, in Algorithm 6, is computationally efficient as it uses public key operations only for signing (by C_g)

and verifying (by Bob) the strings supplied by Alice to Bob, and for oblivious transfer (for every input bit of Bob). The communication and computational overhead is $O(|C|)$ (roughly as that of the original Yao's protocol, see [1, 38]). Our protocol is efficient in that it has only a constant number of rounds and uses only one oblivious transfer operation per each input bit. This is in contrast to the complexity of [8], which due to the cut-and-choose incur a multiplicative increase by a factor of s (the statistical security parameter) and results in communication complexity of $O(s|C| + s^2 n)$.

5. The Decryption Model

The decryption model provides the parties in the two-party protocol with access to the decryption oracle that upon invocation with a ciphertext, validates and decrypts the ciphertext, and returns the result to Alice and Bob.

The decryption model is captured with the $\mathcal{F}_{\mathsf{Decryption}}$ functionality, Algorithm 9. In contrast to $\mathcal{F}_{\mathsf{cce}}$ (Algorithm 4), even if $\mathcal{F}_{\mathsf{Decryption}}$ is corrupt, the implication is on fairness only, but not on privacy or correctness. We use $\mathcal{F}_{\mathsf{Decryption}}$ ideal functionality in the fair protocol which we present in Section 6. However, we believe that the $\mathcal{F}_{\mathsf{Decryption}}$ could be useful for other tasks too, and not only to ensure fairness. This functionality can be implemented using simple, secure (stateless) hardware, or via a set of servers (using distributed/proactive decryption), [48].

6. Fair Two-Party Protocol with Decryption

The standard definition of two-party computation [32] allows Alice and Bob to securely evaluate a function over their private inputs; however, a corrupted party can abort the protocol execution prematurely after it receives its output, while preventing the honest party from receiving output. In many scenarios both parties should receive output, which requires an additional property of *fairness*. Specifically, Alice receives her output if and only if Bob receives his, or no party receives the output. Fairness is especially important for financially oriented tasks, e.g., exchange of signed checks, or currency exchange. In our construction Alice receives her output first (the case where Bob receives output first is symmetric), and should send to Bob his output. If Alice does not send the output to Bob, Bob contacts the third party, an decryption service, and receives his output.

In Algorithm 12 we construct a protocol Π_f that realises the complete fairness functionality $\mathcal{F}_f$ presented in Algorithm 2.1; the protocol Π_f uses as a building block a secure (with abort) two-party protocol in the malicious setting, e.g., the one we presented in Section 4. Concretely, protocol Π_f computes functionality $f(a, b) = (f_A(a, b), f_B(a, b))$, providing output at both Alice and Bob while ensuring complete fairness, i.e., either neither party receives output or both participants do, Definition 2.3.

The protocol in Algorithm 12 uses a weakly trusted (oblivious) third party that is involved only for resolution in case one of the parties misbehaves. We call this third party the decryption and capture it with the ideal functionality $\mathcal{F}_{\mathsf{Decryption}}$ in Algorithm 9. The fair protocol, in Algorithm 12, also uses as a building block the $\mathcal{F}_{\mathsf{cce}}$ ideal functionality. As a result, the functionality, $\mathcal{F}_{\mathsf{cce}}$, ensures correctness and privacy, and the optimistic ideal functionality $\mathcal{F}_{\mathsf{Decryption}}$, involved during the evaluation phase in case of malicious behaviour, ensures fairness of the computation. The third parties do not learn anything about the inputs or the result of the computation.

We now informally describe the protocol presented in Algorithm 12. When the protocol is initiated the decryption functionality $\mathcal{F}_{\mathsf{Decryption}}$, generates a key pair $(dk_R, ek_R) \leftarrow \mathcal{NG}(1^n)$, and this key ek_R is part of the function g. Alice and Bob obtain the public encryption key ek_R of $\mathcal{F}_{\mathsf{Decryption}}$ which defines the function that they agreed to compute. We use, as a module, the ideal two-party computation functionality with output at Bob, as implemented by the protocol Π_g in the $(\mathcal{F}_{\mathsf{cce}}, \mathcal{F}_{\mathsf{ot}}^2)$-hybrid model, in Section 4 (Algorithm 6). The parties then run a protocol Π_g between them, to compute the function g_{ek_R} (presented in Equation 1). The functionality $\mathcal{F}_{\mathsf{cce}}$ generates a garbled circuit that computes g such that part of the output is encrypted with the key ek_R. We take the function g for Π_g (that provides output at Bob only) to be the function computing the following:

$$g_{ek_R}((a||K_A), (b||K_B)) = \mathcal{NE}_{ek_R}(c_A||c_B)||(\mathcal{E}_{K_A}(f_A(a, b), c_B)) \tag{1}$$

where $c_A = \mathcal{E}_{K_A}(f_A(a, b))$ and $c_B = \mathcal{E}_{K_B}(f_B(a, b))$. At the execution, Alice has input a and Bob has input b; they both generate secret keys, K_A and K_B respectively, for symmetric authenticated encryption $(\mathcal{K}, \mathcal{E}, \mathcal{D})$, that will protect their corresponding outputs; then they run a protocol Π_g and provide their inputs, $(a||K_A)$ and $(b||K_B)$ respectively. The protocol Π_g evaluates the function over the inputs and generates output at Bob. The output consists of two parts: one

encrypted with Alice's key and another encrypted with the key ek_R of the $\mathcal{F}_{\text{Decryption}}$ (containing both the output of Bob and of Alice). The output part of the $\mathcal{F}_{\text{Decryption}}$ is encrypted with a non-malleable encryption scheme $(\mathcal{NG}, \mathcal{NE}, \mathcal{ND})$ and is used in case of malicious behaviour, for resolution (non-malleability is required to ensure that the output cannot be maliciously altered in a meaningful way). If Alice does not respond, Bob contacts the $\mathcal{F}_{\text{Decryption}}$ with the part of the output encrypted with the key ek_R. The $\mathcal{F}_{\text{Decryption}}$ validates, decrypts and sends to Alice her output, and to Bob his (restoring fairness). Upon receipt of an output from Bob, Alice validates and decrypts her part of the output and Bob's output encrypted with his secret key. Alice then sends this part to Bob, who validates and decrypts the result, which concludes the protocol. We first informally analyse the security of the protocol in situations when one of the

Functionality $\mathcal{F}_{\text{Decryption}}$

generate encryption key-pair: $(dk_R, ek_R) \leftarrow \mathcal{NG}(1^n)$
send the encryption key: (decryption, ek_R) to Alice and Bob
Computation Phase
  receive c
  $(y_A, y_B) \leftarrow \mathcal{ND}_{dk_R}(c)$
end
Output: send y_A to Alice
  send y_B to Bob

Algorithm 9: The decryption functionality $\mathcal{F}_{\text{Decryption}}$

parties aborts the execution, and then provide a formal security analysis.

1. Either Alice or Bob abort prior to performing the computation. In this case, no one learns anything and fairness is preserved.
2. Bob performs the computation and then aborts and does not send the output to Alice. Since the output is encrypted with the secret key K_A of Alice, Bob cannot recover his output either, thus fairness is ensured.
3. Alice receives the output from Bob, recovers her output, and aborts. Bob will contact the decryption functionality, which will send the output both to Alice and Bob. Since both parties receive the output fairness is restored.

Claim 6.1. *Let $f : \{0,1\}^n \times \{0,1\}^n \to \{0,1\}^n \times \{0,1\}^n$ be a polynomial two-party functionality, let $(\mathcal{K}, \mathcal{E}, \mathcal{D})$ be a secure symmetric authenticated encryption scheme, and let $(\mathcal{NG}, \mathcal{NE}, \mathcal{ND})$ be a secure non-malleable encryption scheme. Then, the protocol Π_f securely realises a two-party functionality $\mathcal{F}_f$ with complete fairness (Definition 2.3) in the presence of malicious static adversaries in the ($\mathcal{F}_{\text{Decryption}}$,$\mathcal{F}_{CCE}$)-hybrid model.*

Proof. We analyse Π_f in a ($\mathcal{F}_{\text{Decryption}}$,$\mathcal{F}_{CCE}$)-hybrid model, and show that the execution of Π_f is computationally indistinguishable from computation of f in the ideal model with complete fairness. Our claim (and its proof) relies on the composability theorem, [33], and allows to prove security in the $\mathcal{F}_{CCE}$-hybrid model, instead of assuming $\mathcal{F}_g$. We prove Claim 6.1 in Propositions 6.2 and 6.3 respectively. □

Proposition 6.2 (Security Against Malicious Alice). *For every non-uniform polynomial time adversary A corrupting Alice and running Π_f with complete fairness in a hybrid model with access to $\mathcal{F}_{\text{Decryption}}$ and $\mathcal{F}_g$, there exists a non-uniform polynomial time simulator S corrupting Alice and running in the ideal model with access to an ideal functionality $\mathcal{F}_f$, such that for every $a, b, z \in \{0,1\}^*$ holds:*

$$\left\{\text{IDEAL}_{f,S(z)}(a,b,n)\right\}_{n\in\mathbb{N}} \overset{poly}{=} \left\{\text{HYBRID}^{\mathcal{F}_{Decrypt},\mathcal{F}_g}_{\Pi_f,A(z)}(a,b,n)\right\}_{n\in\mathbb{N}}$$

Proof. We construct an ideal model simulator which has access to Alice and to the universally trusted party, and can simulate the view of the execution of the protocol. Assume that Alice is corrupted by a hybrid model adversary A. In Algorithm 10 we construct a simulator S given a black-box access to A.

The view of A in a simulation with S is identical to its view in an ($\mathcal{F}_{\text{Decryption}}$,$\mathcal{F}_g$)-hybrid execution of Π_f with a honest Bob. The joint distribution of A's view and Bob's output in a hybrid execution is identical to the joint distribution of S and Bob's output in an ideal model. □

$S(a, 1^n)$

1. generate $(dk, ek) \leftarrow \mathcal{K}(1^n)$ and select a random key $K_S \in \{0,1\}^n$
2. invoke A with input a, n
3. send A the public key ek
4. obtain $(a' \| K_A)$ from A
5. send a' to $\mathcal{F}_f$ and obtain y'_A
6. select $s_B \in \{0,1\}^*$ at random and compute $\mathcal{E}_{K_A}(y'_A, \mathcal{E}_{K_S}(s_B))$ and hand it to A
7. **if** A sends abort **then** send abort to trusted party
   else if A sends $c_B == \mathcal{E}_{K_S}(s_B)$send fair to trusted party
8. S outputs whatever A outputs

end

Algorithm 10: The simulator S running in ideal model with trusted party computing $\mathcal{F}_f$, and simulating the view of Alice.

Proposition 6.3 (Security Against Malicious Bob). *For every non-uniform polynomial time adversary A corrupting Bob and running Π_f with complete fairness in a hybrid model with access to $\mathcal{F}_{Decrypt}$ and $\mathcal{F}_g$, there exists a non-uniform polynomial time simulator S corrupting Bob and running in the ideal model with access to an ideal functionality $\mathcal{F}_f$, such that for every $a, b, z \in \{0,1\}^*$ holds:*

$$\left\{\text{IDEAL}_{f,S(z)}(a,b,n)\right\}_{n\in\mathbb{N}} \overset{poly}{=} \left\{\text{HYBRID}^{\mathcal{F}_{Decrypt},\mathcal{F}_g}_{\Pi_f,A(z)}(a,b,n)\right\}_{n\in\mathbb{N}}$$

Proof. We construct an ideal model simulator which has access to Bob and to the universally trusted party, and can simulate the view of the execution of the protocol. Assume that Bob is corrupted by a hybrid model adversary A. In Algorithm 11 we construct a simulator S given a black-box access to A.

The view of A in a simulation with S is identical to its view in an $(\mathcal{F}_{\text{Decryption}},\mathcal{F}_g)$-hybrid execution of Π_f with a honest Alice. The joint distribution of A's view and Alice's output in a hybrid execution is identical to the joint distribution of S and Alice's output in an ideal model. □

$S(b, 1^n)$

1. generate $(dk, ek) \leftarrow \mathcal{K}(1^n)$ and select a random key $K_S \in \{0,1\}^n$
2. invoke A with input b, n
3. send A the public key ek
4. obtain $(b' \| K_B)$ from A
5. send b' to $\mathcal{F}_f$ and obtain y'_B
6. select $s_A \in_R \{0,1\}^*$ and compute $\mathcal{NE}_{ek}(c_A, c_B), \mathcal{E}_{K_S}(s_A \| \mathcal{E}_{K_B}(y'_B))$
7. send $\mathcal{NE}_{ek}(c_A, c_B), \mathcal{E}_{K_S}(s_A \| \mathcal{E}_{K_B}(y'_B))$ to A
8. on input ω from A, check
   if $\mathcal{D}_{K_S}(\omega) = (s_A \| \mathcal{E}_{K_B}(y'_B))$ **then** send $\mathcal{E}_{K_B}(y'_B)$ to A
   else if $(\omega = \omega_1 \| \omega_2) \vee \mathcal{ND}_{dk}(\omega_1 \| \omega_2) = (c_A \| c_B)$ **then** send c_B to A
   else send $\perp$ to trusted party
9. S outputs whatever A outputs

end

Algorithm 11: The simulator S running in ideal model with trusted party computing $\mathcal{F}_f$, and simulating the view of Bob.

6.1. Efficiency Discussion

In [25], the authors designed an efficient optimistic fair protocol using proofs of knowledge. The number of rounds in their protocol is constant, and does not depend on the security parameter. Yet their protocol incurs a significant efficiency degradation, since the zero-knowledge proofs are required for every gate of the circuit, resulting in $O(s|C|)$ communication and computational complexity. Furthermore, the protocol of [25] seems to be susceptible to 'inputs corruption' attack, whereby Alice corrupts one of the inputs to oblivious transfer protocol, and based on the behaviour of Bob learns the corresponding value of his input bit. In our protocol, when the parties are honest and follow the

steps of the protocol (which is the typical case), the computation complexity is roughly as that of the Yao's original protocol (see Section 4.2.2 for discussion). When one of the parties misbehaves, the protocol requires an additional round, to send the encrypted result to the $\mathcal{F}_{\mathsf{Decryption}}$ and to receive a decrypted response back.

Input: security parameters n, s, $a = [a_i]_{i=1}^n$ from Alice, $b = [b_i]_{i=1}^n$ from Bob
Output: $y = (y_A, y_B)$
Alice and Bob obtain (decryption, ek_R) (each)
Computation Phase
Alice and Bob do:
generate secret keys K_A and K_B respectively
run a protocol Π_g (in Algorithm 6 realising $\mathcal{F}_g$) computing g_{ek_R} (g_{ek_R} is constructed from f as in Equation 1) with key ek_R of $\mathcal{F}_{\mathsf{Decryption}}$, on inputs $(a||K_A, b||K_B)$
Alice provides $(a||K_A)$ and Bob provides $(b||K_B)$

Bob:
onReceive($\alpha||\beta$) from the protocol Π_g
send(β) to Alice
onReceive(c_B) from $\mathcal{F}_{Decrypt}$
if $(\mathcal{D}_{K_B}(c_B) \neq \perp)$ **then**
recover and output $\mathcal{D}_{K_B}(c_B)$
onWakeup()
send $\mathcal{NE}_{ek_R}(c_A, c_B)$ to $\mathcal{F}_{\mathsf{Decryption}}$
end

Alice:
onReceive(β) from Bob
let $(y_A, c_B) \leftarrow \mathcal{D}_{K_A}(\beta)$
if $(y_A \neq bot)$ **then**
send(c_B) to Bob
recover and output $y_A = f_A(a, b)$
onReceive(β) from $\mathcal{F}_{Decrypt}$
recover and output $\mathcal{D}_{K_A}(\beta)$
end

end

Algorithm 12: Secure Two Party Protocol Π_f that realises $\mathcal{F}_f$ according to Definition 2.3, in the $(\mathcal{F}_{\mathsf{Decryption}}, \mathcal{F}_g)$-hybrid model for computing $f(a, b) = (f_A(a), f_B(b))$, where $f : \{0,1\}^n \times \{0,1\}^n \rightarrow \{0,1\}^n \times \{0,1\}^n$; and $g_{ek_R}((a||K_A), (b||K_B)) = \mathcal{NE}_{ek_R}(c_A||c_B)||(\mathcal{E}_{K_A}(f_A(a, b), c_B))$, $c_A = \mathcal{E}_{K_A}(f_A(a, b))$ and $c_B = \mathcal{E}_{K_B}(f_B(a, b))$.

7. Applications

In this work we investigated applications of secure computation, and constructed efficient and secure two-party protocols, based on assumptions of weakly-trusted, simple and efficient services (or, formally, on corresponding ideal functionalities).

We believe that these protocols could be particularly useful for financial applications, which typically involve third parties (that often cannot be avoided, and are often trusted much more than needed by our protocols). We suggest to use these 'already present' third parties, to produce efficient and practical systems. We next present two specific applications: *stocks trade* and *currency exchange*:

7.1. Stocks Trade

Consider the following sample application for stocks trade, utilising the protocols presented in this work. Alice wishes to sell IBM stocks and Bob wants to buy them. The challenge here is to agree on a rate, yet without exposing each other's private inputs. Exposing the input of the other party will allow to adjust the price offered by the other party.

STOCKS TRADE VIA SERVER-AIDED TWO-PARTY COMPUTATION. Alice contacts a third party (which they both trust to some extent) e.g., a broker, to obtain a program for this specific transaction. Both Alice and Bob provide some secret policy, as their respective inputs, and possibly their secret signature keys (which are essential since the algorithm produces signed orders); then the algorithm, constructed by the third party, will produce corresponding signed orders if there is a match between the policies, or $\perp$ if there is no match. Concretely, the protocol satisfies the following requirements: (1) it does not expose the policies (nor the secret inputs, e.g., secret keys); (2) it ensures correctness of computation (since the program, implementing the algorithm, was supplied and signed by a third party); (3) ensures fairness (if, say, Alice aborts after receiving her signed order, Bob can contact the resolver, e.g., a broker, or a clearing house, to recover his order).

7.2. Currency Exchange

Alice and Bob wish to sign an agreement for currency exchange, e.g., to exchange € for $. The goal is to sign the agreement without exposing their respective private inputs, e.g., if a malicious party learns the policy or the rates of the honest party, it can cheat by adjusting its own input accordingly.

CURRENCY EXCHANGE VIA SERVER-AIDED TWO-PARTY COMPUTATION. Alice contacts a third party, which they both trust, e.g., a broker, to obtain a program implementing the agreement. The agreement receives the X€ of Alice, and Y$ of Bob, and outputs two signed checks, for Alice and Bob respectively. Since the program (implementing the agreement) was supplied and signed by a third party, Alice and Bob are assured that it is correct. Also fairness is assured, since if Alice decides to cheat and aborts after receiving her check, Bob can contact a resolver, that will recover and send the checks to both parties.

8. Conclusion

Two-party computation received a lot of attention during the last two decades, with numerous works, and although it was shown to be practical (see [3]), there are no applications or systems utilising it. In this work we present financially oriented protocols, facilitating two-party computation as a basic building block, which can fit well in the field of secure ecommerce. Such financial applications are often required to ensure fairness to the transactions performed by the parties, as well as guaranteed compensation, in case of failures. Other critical properties of financial protocols is ensuring privacy to the inputs of the participants, correctness of the transaction, and efficiency.

Our protocols assume weakly trusted (oblivious) third parties, e.g., involved only in case of misbehaviour or failures, that cannot observe neither the inputs of the parties to the transaction, nor the compensation granted in case of failures. Hiding the inputs and outputs from the third parties is critical to financial applications. We stress that the success of electronic financial applications may depend on the ability to construct efficient protocols with rigorous security guarantees. We believe that applying server-aided two-party computation to produce practical, efficient and secure protocols, is an important challenge of the research on two-party computation. Specifically, we suggest carrying this research forward and encourage improving over the efficiency of our protocols, and further reducing the trust assumption in third parties. In addition, providing efficient real life implementations for specific tasks is a significant goal that would utilise the potential of the Internet to allow arbitrary parties to perform commerce, with automated, trustworthy dispute-resolution and compensation mechanisms.

Acknowledgments

We are grateful to anonymous referees for their comments and feedback on the earlier write up of this manuscript. This work was supported by grants from the Israeli Ministry of Science (MoS) and the Binational Science Foundation (BSF).

References

[1] A. C. Yao, How to generate and exchange secrets, in: Proc. 27th IEEE Symp. on Foundations of Comp. Science, IEEE, Toronto, 1986, pp. 162–167.

[2] O. Goldreich, S. Micali, A. Wigderson, How to play any mental game or a completeness theorem for protocols with honest majority, in: 19th ACM Symposium on the Theory of Computing, 1987, pp. 218–229.

[3] B. Pinkas, T. Schneider, N. Smart, S. Williams, Secure two-party computation is practical, Advances in Cryptology–ASIACRYPT 2009 (2009) 250–267.

[4] P. Bogetoft, D. L. Christensen, I. Damgård, M. Geisler, T. P. Jakobsen, M. Krøigaard, J. D. Nielsen, J. B. Nielsen, K. Nielsen, J. Pagter, M. I. Schwartzbach, T. Toft, Secure multiparty computation goes live, in: R. Dingledine, P. Golle (Eds.), Financial Cryptography, Vol. 5628 of Lecture Notes in Computer Science, Springer, 2009, pp. 325–343.
URL http://dx.doi.org/10.1007/978-3-642-03549-4

[5] R. Cleve, Limits on the security of coin flips when half the processors are faulty, in: Proceedings of the eighteenth annual ACM symposium on Theory of computing, ACM, 1986, pp. 364–369.

[6] D. Gordon, C. Hazay, J. Katz, Y. Lindell, Complete fairness in secure two-party computation, in: Proceedings of the 40th annual ACM symposium on Theory of computing, ACM, 2008, pp. 413–422.

[7] O. Catrina, F. Kerschbaum, Fostering the uptake of secure multiparty computation in E-commerce, in: ARES, IEEE Computer Society, 2008, pp. 693–700.
URL http://doi.ieeecomputersociety.org/10.1109/ARES.2008.49

[8] Y. Lindell, B. Pinkas, An efficient protocol for secure two-party computation in the presence of malicious adversaries, Advances in Cryptology-EUROCRYPT 2007 (2007) 52–78.
[9] S. Jarecki, V. Shmatikov, Efficient two-party secure computation on committed inputs, Advances in Cryptology-EUROCRYPT 2007 (2007) 97–114.
[10] S. Choi, A. Elbaz, A. Juels, T. Malkin, M. Yung, Two-party computing with encrypted data, in: Proceedings of the Advances in Crypotology 13th international conference on Theory and application of cryptology and information security, Asiacrypt, Springer-Verlag, 2007, pp. 298–314.
[11] R. Canetti, M. Fischlin, Universally composable commitments, in: J. Kilian (Ed.), Advances in Cryptology – CRYPTO ' 2001, Vol. 2139 of Lecture Notes in Computer Science, International Association for Cryptologic Research, Springer-Verlag, Berlin Germany, 2001, pp. 19–40. URL http://link.springer.de//link/service/series/0558/papers/2139/21390019.pdf
[12] N. Asokan, M. Schunter, M. Waidner, Optimistic protocols for fair exchange, in: Proceedings of the 4th ACM conference on Computer and communications security, ACM, 1997, p. 17.
[13] N. Asokan, V. Shoup, M. Waidner, Optimistic fair exchange of digital signatures, Advances in Cryptology EUROCRYPT 1998 (1998) 591–606.
[14] E. F. Brickell, D. Chaum, I. B. Damgård, J. van de Graaf, Gradual and verifiable release of a secret, in: C. Pomerance (Ed.), Proc. CRYPTO 87, Springer-Verlag, 1988, pp. 156–166, lecture Notes in Computer Science No. 293.
[15] D. Beaver, S. Goldwasser, Multiparty Computation with Faulty Majority, in: Advances in Cryptology - Crypto 1989 Proceedings, Springer, 1990, pp. 589–590.
[16] R. Cleve, Controlled gradual disclosure schemes for random bits and their applications, in: Proceedings on Advances in cryptology, Springer-Verlag New York, Inc., 1989, pp. 573–588.
[17] M. Ben-Or, O. Goldreich, S. Micali, R. Rivest, A fair protocol for signing contracts, Automata, Languages and Programming (1985) 43–52.
[18] S. Goldwasser, L. Levin, Fair computation of general functions in presence of immoral majority, Advances in Cryptology-CRYPT0'90 (1990) 77–93.
[19] I. Damgård, Practical and provably secure release of a secret and exchange of signatures, Journal of Cryptology 8 (4) (1995) 201–222.
[20] B. Pinkas, Fair secure two-party computation, Advances in Cryptology-Eurocrypt 2003 (2003) 647–647.
[21] J. Garay, P. MacKenzie, M. Prabhakaran, K. Yang, Resource fairness and composability of cryptographic protocols, Theory of Cryptography (2006) 404–428.
[22] D. Gordon, J. Katz, Partial fairness in secure two-party computation, Tech. rep., Cryptology ePrint Archive, Report 2008/206, 2008 (2008).
[23] S. Micali, Simple and fast optimistic protocols for fair electronic exchange, in: Proceedings of the twenty-second annual symposium on Principles of distributed computing, ACM, 2003, p. 19.
[24] A. Lindell, Legally-enforceable fairness in secure two-party computation, in: Proceedings of the 2008 The Cryptopgraphers' Track at the RSA conference on Topics in cryptology, Springer-Verlag, 2008, pp. 121–137.
[25] C. Cachin, J. Camenisch, Optimistic fair secure computation, in: M. Bellare (Ed.), Advances in Cryptology – CRYPTO ' 2000, Vol. 1880 of Lecture Notes in Computer Science, International Association for Cryptologic Research, Springer-Verlag, Berlin Germany, 2000, pp. 93–111.
[26] A. Herzberg, H. Shulman, Oblivious and Fair Server-Aided Two-Party Computation, in: International Conference on Availability, Reliability and Security (ARES), 2012.
[27] O. Goldreich, The Foundations of Cryptography, Volume 2 (Basic Applications), Cambridge University Press, 2004.
[28] R. Canetti, Security and composition of multiparty cryptographic protocols, Journal of Cryptology 13 (1) (2000) 143–202.
[29] Y. Lindell, B. Pinkas, An efficient protocol for secure two-party computation in the presence of malicious adversaries, Advances in Cryptology-EUROCRYPT 2007 (2007) 52–78.
[30] M. Bellare, C. Namprempre, Authenticated encryption: Relations among notions and analysis of the generic composition paradigm, in: ASIACRYPT '00: Proceedings of the 6th International Conference on the Theory and Application of Cryptology and Information Security, Springer-Verlag, London, UK, 2000, pp. 531–545.
[31] M. Bellare, A. Sahai, Non-malleable encryption: Equivalence between two notions, and an indistinguishability-based characterization, in: Advances in cryptology-CRYPTO'99, Springer, 1999, pp. 78–78.
[32] O. Goldreich, Foundations of Cryptography: Volume 2, Basic Applications, Cambridge University Press New York, NY, USA, 2004.
[33] R. Canetti, Universally composable security: A new paradigm for cryptographic protocols, in: Foundations of Computer Science, 2001. Proceedings. 42nd IEEE Symposium on, IEEE, 2001, pp. 136–145.
[34] M. Blum, P. Feldman, S. Micali, Non-interactive zero-knowledge and its applications, in: Proceedings of the twentieth annual ACM symposium on Theory of computing, ACM, 1988, pp. 103–112.
[35] P. Mohassel, M. Franklin, Efficiency tradeoffs for malicious two-party computation, Public Key Cryptography-PKC 2006 (2006) 458–473.
[36] D. Woodruff, Revisiting the efficiency of malicious two-party computation, Advances in Cryptology-EUROCRYPT 2007 (2007) 79–96.
[37] J. Nielsen, C. Orlandi, LEGO for Two-Party Secure Computation, in: Proceedings of the 6th Theory of Cryptography Conference on Theory of Cryptography, Springer-Verlag, 2009, pp. 368–386.
[38] Y. Lindell, B. Pinkas, A Proof of Yao Protocol for Secure Two-Party Computation, in: Electronic Colloquium on Computational Complexity, Vol. 11, 2004, p. 063.
[39] J. Katz, R. Ostrovsky, Round-optimal secure two-party computation, in: Advances in Cryptology–CRYPTO 2004, Springer, 2004, pp. 3–34.
[40] O. Goldreich, S. Micali, A. Wigderson, Proofs that yield nothing but their validity or all languages in NP have zero-knowledge proof systems, Journal of the ACM (JACM) 38 (3) (1991) 690–728.
[41] J. Kilian, Founding crytpography on oblivious transfer, in: Proceedings of the twentieth annual ACM symposium on Theory of computing, ACM, 1988, pp. 20–31.
[42] J. Garay, Efficient and universally composable committed oblivious transfer and applications, Theory of Cryptography (2004) 297–316.
[43] D. Malkhi, N. Nisan, B. Pinkas, Y. Sella, Fairplay - secure two-party computation system, in: Proceedings of the 13th USENIX Security Symposium, USENIX, 2004, pp. 287–302.
URL http://www.usenix.org/publications/library/proceedings/sec04/tech/malkhi.html

[44] I. Damgard, M. Keller, Secure multiparty aes, Financial Cryptography and Data Security (2010) 367–374.
[45] I. Damgard, C. Orlandi, Multiparty computation for dishonest majority: From passive to active security at low cost, Advances in Cryptology-CRYPTO 2010 (2010) 558–576.
[46] V. Kolesnikov, A. Sadeghi, T. Schneider, Improved garbled circuit building blocks and applications to auctions and computing minima, Cryptology and Network Security (2009) 1–20.
[47] A. Paus, A. Sadeghi, T. Schneider, Practical secure evaluation of semi-private functions, in: Applied Cryptography and Network Security, Springer, 2009, pp. 89–106.
[48] A. Herzberg, M. Jakobsson, S. Jarecki, H. Krawczyk, M. Yung, Proactive public key and signature systems, in: Proceedings of the 4th ACM conference on Computer and communications security, ACM, 1997, pp. 100–110.