Secure *Guaranteed* Computation

Amir Herzberg* and Haya Shulman†

Bar Ilan University
Department of Computer Science
Ramat Gan, 52900, Israel

Abstract

We introduce *secure guaranteed two-party computation*, where parties commit in advance to compute a function over their private inputs, by providing some (validated) *compensation*, such that if a party fails to provide an appropriate input during protocol execution, then the peer receives the compensation. Enforcement of the guarantees requires a *trusted enforcement authority* (TEA); however, the protocol protects confidentiality even from the TEA. Our secure guaranteed computation protocol is *optimistic*, i.e., the TEA is involved only if a party fails to participate (correctly). Secure guaranteed computation has direct practical applications, such as sensitive trading of financial products, and could also be used as a building block to motivate parties to complete protocols, e.g., ensuring unbiased coin tossing.

The guarantee process can be either symmetric (both parties provide guarantees) or asymmetric (e.g., only server provides guarantee to client). Symmetric guarantees should also be *fair*, i.e., one party cannot obtain guarantee of the other party without supplying a guarantee as well.

The protocol guaranteeing output, which we present uses in a modular manner a new protocol, that we construct, for *optimistic fair secure computation*, which may be of independent interest, as it is simpler and more efficient than previously known protocols.

Keywords: Two-party computation, fairness, guaranteed services, non-repudiation, optimistic protocols, cryptographic protocols.

1 Introduction

This work investigates the combination of two important areas of research related to secure distributed systems: the beautiful theory of *secure computation*, and the applied area of *guaranteed/non-repudiable network services*. As we explain, this combination is natural and interesting; furthermore, it has important practical applications, as well as theoretical significance.

Secure computation, beginning with the seminal papers of Yao [34] and Goldreich et al. [17], investigates how to securely compute functionalities over inputs of two (or multiple) parties, under different circumstances and in the presence of different adversaries. We focus on the case of two-party computation. 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 trusted third party; the goal of secure computation protocols is to achieve the same impact without a trusted party, i.e., using a protocol between the two parties.

Current work on secure computation does not *guarantee output*. Namely, a corrupted party may decide not to participate, in which case the computation cannot complete. In many applications, esp. financial applications, such as currency exchange or equity trading, the honest party should receive some agreed-upon output even in case of early abort or failure to conclude the protocol by the other party. Preferably, this output should be the result of the function over the inputs of the participants, i.e., correct output of the computation; but if the other party failed to participate, then the honest party should receive some agreed-upon *compensation*, such as payment or evidence that it provided inputs. This property is called *guaranteed output delivery*, and is the focus of this paper.

Guaranteed output delivery is the goal of the applied area of *guaranteed/non-repudiable network services*, see e.g., [22, 1]. As network services become more and more important, failure to provide services can become a serious

*Amir.Herzberg@gmail.com
†Haya.Shulman@gmail.com

concern. Several works and systems try to ensure services in face of unintentional failures and congestion, as well as (intentional) denial-of-service attacks. A more difficult issue is how to guarantee services in spite of *intentional delivery failures* by one of the parties, e.g., a service provider, or to provide *non-repudiation evidences*, e.g., evidence of submission (to customer submitting order to a service provider). For example, suppose a customer bought, say from his broker, an option to buy or sell some shares (or other financial product) at a fixed price; and suppose the customer sends an order to execute the option, close to its expiration time. A failure to process the order by the broker - or a communication failure - could result in a significant loss to the customer (and possibly an illegitimate gain to the broker). Guaranteed output delivery would ensure that either the order is processed, or the customer receives some pre-agreed compensation from the broker, or at least an evidence attesting to the submission.

Enforcing (guaranteeing) compensation, or providing an evidence, requires an additional (weakly or strongly) trusted third party. We focus on enforcing (guaranteeing) compensation, and therefore refer to this third party as the Trusted Enforcement Authority (TEA). The TEA does not provide inputs, but helps to compensate the honest participant (if the other participant fails to cooperate correctly). Note that a trusted third party is also required for ensuring fairness and providing evidences of delivery.

Existing research and systems focus on providing fairness and/or evidences, rather than a direct compensation; furthermore, they support only specific, relatively simple interactions, e.g., certified mail, contract signing and fair exchange [2, 3]. Furthermore, existing techniques do not hide the values exchanged from the trusted third party. Such confidentiality can be very important, e.g., the exposure of trading positions or orders can provide an unfair advantage, possibly harming the customer. This limits the use of sensitive transactions, required to allow arbitrary parties to perform commerce, with automated, trustworthy dispute-resolution and compensation mechanisms.

In this paper, we propose secure *guaranteed* (two-party) computation, i.e., a protocol ensuring that two parties can compute any function over their private inputs securely. The protocol ensures that parties have an *incentive* to complete the protocol. This incentive is accomplished by running the computation in two phases: the *guarantee phase* and the *execution phase*. In the guarantee phase, a party guarantees to participate in the (following) execution phase, by providing some *compensation* whose exposure would compensate the second party (and penalise the first party). In the following execution phase, if a party does not participate correctly, the TEA exposes the compensation of that party (as provided in the first phase). We believe that this can be a powerful facilitator for secure electronic commerce. In particular, the solution facilitates the usage of the guaranteed computation protocol as a building block in design of complex incentive-based protocols, ensuring security goals against rational adversaries.

The application discussed above focused on asymmetric guarantee: only the broker guaranteed service to the customer. We extend this setting and present a protocol, in Section 4, that supports (also) symmetric guarantee, where both peers initially 'deposit' some 'compensation' at the TEA, and can later participate in the transaction. This protocol also ensures *fairness*, i.e., Bob receives Alice's guarantee if and only if Alice receives Bob's guarantee.

Our secure guaranteed computation protocol makes use of two sub-protocols, that may be of independent interest. The first is a simple and efficient protocol for *optimistic fair secure two-party computation*, in Section 5, which we use as a module in our guaranteed secure two-party computation protocol, to ensure that the guarantee process is *fair*, i.e., one party cannot obtain guarantee by other party without providing a guarantee as well. As other protocols for optimistic fair secure computation, our protocol also involves a third party, however this party is both very simple and only weakly trusted, i.e., even if it is rogue, the implication would be on fairness only, but not on correctness or privacy. The protocol is *optimistic* in the sense that the third party is involved only if one of the two parties fails to complete the protocol properly. Note that a third party is necessary to support fairness for computation of arbitrary functionalities (although it may be avoided for some specific functionalities, see [21]). Optimistic fair secure computation protocols were presented before [8], however, our protocol is more efficient (also, the protocol in [8] was not proven secure yet). The second sub-protocol is an efficient two-party computation protocol in the malicious setting, in Section 3.2, that allows output at Bob only. This protocol uses an offline third party for the preprocessing phase, and improves efficiency over the existing protocols, see Section 3.2.1 for analysis and comparison.

Contributions. Our central contribution is the definition and realisation of *secure guaranteed computation*, providing compensation to the honest party if the other party does not participate correctly. The guarantee is provided by the TEA (trusted enforcement authority), however, even if the $\mathcal{F}_{\mathsf{tea}}$ is malicious, it can not learn the inputs or the outputs of the parties and will not be able to generate an incorrect result without the parties detecting this. In addition, we define and present an (optimistic) realisation of Δ-*delayed fair computation*, where a malicious party can delay the output of the honest party by at most a factor of Δ; this generalises the known 'fairness with abort' definition.

1.1 Related Work

There are many works, beginning with Yao [34], investigating two-party secure computation. Yao's work showed that any two-party function can be securely evaluated, while ensuring privacy and correctness, by using garbled circuits, but only against passive adversaries, i.e., when honest or semi-honest behaviour of the participants is assumed. This was extended by Goldreich et al. [17] to ensure security against malicious adversaries, and several works improved efficiency [8, 26, 23, 10].

In malicious model, an adversary can always abort after receiving its output and before the honest party receives output. Cleve [11] showed that fairness cannot be achieved for general computation without an honest majority. Hence, different approaches towards achieving fairness for general computations were considered. One approach, the gradual release, see [7, 4, 12, 6, 19, 13, 31, 15, 20], 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 is to use a trusted third party, preferably, with limited trust and/or limited involvement. This approach is highly efficient compared to the gradual release of secrets and allows to restore complete fairness in case one of the parties aborts. In particular, *optimistic* protocols involve the third party only in case one of the parties misbehaves. 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.

Optimistic protocols were mostly proposed for specific tasks, esp. fair exchange [2, 3, 28]. Cachin and Camenisch, [8], 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 5, provable security, and most notably, by allowing a guarantee to the computation.

Organisation

In Section 2 we present the model and specifications. In Section 3, we present a basic building block used by our protocols: an efficient, practical two-party computation protocol, with an offline third party. In Section 4 we define ideal functionality for guaranteed fair secure computation, and present a protocol realising it. In Section 5 we present an optimistic fair secure computation protocol. Eventually, in Section 6 we conclude our results and present future research directions.

2 Specifications

In this section we present the specifications of the tasks of fairness and guaranteed computation in two-party protocols. These specifications are captured by the ideal functionalities, in subsequent sections. We initiate with the description of the computation model, that we assume in this work, and with the cryptographic tools.

2.1 Preliminaries: Model and Tools

We use the universal composability framework, which ensures that security of the protocols is maintained under a composition with arbitrary other protocols in the system [9]. The functionality expected from the protocol is captured by a universally trusted party, that performs the computation on behalf of the participants. The algorithm run by the trusted party is called an ideal functionality. The protocol is secure if real protocol execution can be emulated by the ideal functionality. In real protocol execution, the parties run the protocol and the adversary controls the communication channels and the corrupted parties. We consider static corruptions, i.e., corrupted party is fixed prior to protocol execution; and assume malicious and semi-honest adversaries. Malicious adversary can arbitrarily deviate from the protocol, while a semi-honest adversary follows the prescribed steps of the protocol, but may try to infer additional information based on its view, and all intermediate steps of the protocol.

We assume synchronous communication model with bounded delay Δ_C. We also assume ideal channels between ideal functionalities and participants in the protocol, i.e., the messages are never lost and are delivered within the assumed delay bound.

Our constructions use several tools, i.e., standard cryptographic mechanisms: *authenticated symmetric-key encryption scheme* $(\mathcal{K}, \mathcal{E}, \mathcal{D})$ [5, 26] to ensure confidentiality and integrity of the inputs and outputs of the participants; *Non-malleable public-key encryption scheme* $(\mathcal{NG}, \mathcal{NE}, \mathcal{ND})$ [14] to ensure fairness; *Signature scheme* $(\mathcal{G}, \mathcal{S}, \mathcal{V})$, where we use $\perp$ to denote authentication failure; *Two-party (1-2) oblivious transfer* which we denote by the ideal functionality $\mathcal{F}_{\mathsf{ot}}^{2}$; *Certification Authority* which we denote by the ideal functionality $\mathcal{F}_{\mathsf{ca}}$.

2.2 Fair Two-Party Computation Functionality $\mathcal{F}^{f}_{\Delta\text{-delayed-fairness}}$

The standard definition of two-party computation [16] 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. In Algorithm 1 below, we present the ideal functionality $\mathcal{F}^{f}_{\Delta\text{-delayed-fairness}}$' that captures the notion of Δ-delayed fairness which we believe to be a suitable model in many realistic and practical scenarios. In Δ-delayed fairness, either both parties receive the output or no one does, and a corrupt party can delay the output of the honest party by at most a factor of Δ. Our definition of Δ-delayed fairness implies the standard definition with abort. Specifically, any protocol that securely computes a functionality with Δ-delayed fairness, also securely computes the functionality of fairness with abort[1]. In our definition Alice receives her output first (the case where Bob receives output first is symmetric), and should send to Bob his output. If Alice responds with fair the ideal functionality sends the output to Bob. If Alice responds with unfair or does not respond, the functionality waits the remaining time for the maximal delay of Δ, e.g., $\Delta = 4\Delta_C$, and sends the output to Bob.

```
Input: a from Alice, b from Bob, n
Computation Phase
  if a == ⊥ ∨ b == ⊥ then
    send ⊥ to Alice and to Bob, and halt
  else
    send y_A = f_A(a, b) to Alice
    sleep('wait for response', Δ)
    onReceive(fair)
      stopTimer('wait for response')
      send(y_B = f_B(a, b)) to Bob
    onWakeup('wait for response')
      send(y_B = f_B(a, b)) to Bob
end
```

Algorithm 1: The ideal functionality $\mathcal{F}^{f}_{\Delta\text{-delayed-fairness}}$ for computing a function $f(a,b) = (f_A(a,b), f_B(a,b))$ in Δ-delayed fairness model, running with Alice and Bob, and an adversary S.

2.3 Guaranteed Two-Party Computation Functionality $\mathcal{F}^{v,g}_{\Delta-\text{guaranteed-computation}}$

Fairness alone may not suffice for some applications, since, a participant may decide to abort the protocol without receiving an output, or may provide an invalid input to the computation. Such an outcome may not be acceptable in many applications, e.g., if a party may be harmed if its partner fails to complete the computation. Parties should be able to agree to participate in a computation in advance, possibly before they have all inputs to that computation, by exchanging between them *guarantees*, e.g., by signing a contract together. The *guarantee phase* should ensure fairness, i.e., prevent a malicious party from aborting after it receives its guarantee, so that the other party will not receive a guarantee. In addition, the guarantees should be validated to prevent the malicious party from providing an invalid guarantee, e.g., one that expired. Fairness does not encompass these requirements. In Algorithm 2 we define the functionality $\mathcal{F}^{v,g}_{\Delta-\text{guaranteed-computation}}$ that ensures guaranteed output delivery, by penalising a party that failed to participate; then in Section 4 we present a protocol realising it.

The guaranteed two-party computation functionality $\mathcal{F}^{v,g}_{\Delta-\text{guaranteed-computation}}$, in Algorithm 2, consists of two-phases: during the first phase the parties provide their respective guarantees, a_1 and b_1, to the ideal functionality. The functionality validates the guarantees using the validation function v_1, which is part of the pair

[1] Proof available from authors.

$v = (v_1, v_2)$. The validation function v is a parameter of $\mathcal{F}^{v,g}_{\Delta-\mathsf{guaranteed\text{-}computation}}$. If (a_1, b_1) are valid, the functionality sends them to Alice and Bob, and proceeds to second phase. During the second phase, the parties provide their inputs a_2 and b_2, which are again validated (using v_2) against the guarantees from the first phase (along with the time when each input was received). If one of the inputs is invalid, the party with the valid input is compensated (by receiving the guarantee of the other). If validation succeeds, the functionality evaluates the inputs over the agreed function g, and sends the result to Bob (since he is the first to receive the output). If Bob is malicious he can delay the output of Alice by at most a factor of $4\Delta_C$, by responding with unfair (or not responding at all). Otherwise, the functionality sends the output to Alice without additional delay. During the second phase the functionality may not receive inputs from both parties at the same time. Thus upon input from one party, it records the time, and waits for input from the other party; if no input from the other party arrives within the interval defined in the validation function, the functionality validates the input that it received (along with the guarantee of the other party) and if valid, recovers the guarantee and grants it to the party which participated correctly.

Input: n
Guarantee Phase
  Input: a_1 from Alice, b_1 from Bob
  Let $x[\mathsf{Alice}] = a_1,\ x[\mathsf{Bob}] = b_1$ (save inputs, use as compensation if needed at computation phase)
  Let $(y_A^1, y_B^1) \leftarrow v_1(a_1, b_1)$
  if $y_A^1 == \perp \vee y_B^1 == \perp$ **then**
    send $\perp$ to Alice and Bob and halt (invalid inputs)
  else
    send $y_A^1,\ y_B^1$ to Alice and Bob respectively
end
Computation Phase
  onReceive(ϕ) from party $\psi \in \{\mathsf{Alice}, \mathsf{Bob}\}$
    Let $\psi' = \{\mathsf{Alice}, \mathsf{Bob}\} - \psi$
    $t_\psi \leftarrow getTime()$
    sleep('wait for 2^{nd} input',Δ)
    onWakeup('wait for 2^{nd} input') (2^{nd} input is too late)
    if $(v_2(\psi, \phi, \perp, a_1, b_1, t_\psi, getTime()) == \psi')$ **then**
      send $x[\psi']$ to ψ (ψ input is Ok, so send it compensation from guarantee phase)
  onReceive(ϕ') from party ψ'
    stopTimer('wait for 2^{nd} input')
    Let $\psi'' = v_2(\perp, \phi, \phi', a_1, b_1, t_\psi, getTime())$ (v_2 outputs 'bad' party or $\perp$ if both inputs Ok)
    if$(\psi'' \in \{\mathsf{Alice}, \mathsf{Bob}\})$ **then**
      send $x[\psi'']$ to $\{\mathsf{Alice}, \mathsf{Bob}\} - \psi''$ and halt (send compensation from guarantee phase)
    $(y_A^2, y_B^2) \leftarrow g(a_2, b_2)$
    send (y_B^2) to Bob
    sleep('wait for response', Δ)
    onReceive(fair)
      stopTimer('wait for response')
      send (y_A^2) to Alice
    onWakeup('wait for response')
      send (y_A^2) to Alice
end

Algorithm 2: The ideal functionality $\mathcal{F}^{v,g}_{\Delta-\mathsf{guaranteed\text{-}computation}}$ for computing $g = (g_A, g_B)$ with guaranteed output delivery, where $v = (v_1, v_2)$ is the pair of inputs validation functions (one for each phase).

3 Pre-Generation of Garbled Circuits by Third Party

Secure function evaluation based on garbled circuits, see [34], allows to perform a two-party computation in a secure manner, i.e., ensuring privacy, correctness and inputs independence (see proof in [25]). Specifically, during the generation phase, Alice (the originator) constructs the garbled circuit, and then during protocol execution, Alice transfers the circuit along with the encodings of the inputs, to Bob, that evaluates the circuit and obtains the result. The basic protocol based on Yao's garbled circuits, ensures security only against semi-honest adversaries, i.e., adversaries that follow the steps of the protocol, but may try to infer additional information from the inputs-outputs. When considering malicious adversaries, additional security concerns arise. In particular, Alice may attempt to expose the secret input of Bob by providing incorrect encodings of his input bits, and based on Bob's reaction (abort

or successful completion of protocol) will learn his input. Alternately, Alice may provide an incorrect circuit, i.e., one that computes a different function, e.g., exposing the input of Bob. Although any two-party protocol can be securely computed in the malicious setting, e.g., see [17, 16], they are inefficient for practical purposes, and a series of works [29, 33, 26, 30] attempt to improve on the efficiency, by reducing the computation and the communication complexity, as well as the number of rounds required by the two-party protocol.

A basic tool that underlies our protocols, in Sections 4 and 5, is a procedure, that generates the garbled circuit. The procedure generating the circuit can be run by a two-party computation protocol between the parties, or by a third party. In Section 3.1 (Algorithm 3) we present the ideal functionality $\mathcal{F}^e_{\mathsf{offline}}$, that captures the task of garbled circuit generation. We then show how to apply this functionality, in Section 3.2 (Algorithm 4), during the offline preprocessing phase, to construct a two-party computation protocol secure in the malicious setting, by separating the offline preprocessing and the computation phases, using a weakly trusted third party for the preprocessing phase. The resulting two-party protocol in Algorithm 4, illustrates one of the applications of the circuit generation procedure. We also use this mechanism in subsequent chapters as a building block in our fair and committed two-party computation protocols. The procedure generating the garbled circuit using a weakly trusted third party[2] during the preprocessing phase can be of independent interest to enhance efficiency (see Section 3.2.1 for efficiency comparison of techniques employed in malicious model) of two-party protocols in malicious setting.

3.1 Offline Functionality

The Offline Party functionality $\mathcal{F}^e_{\mathsf{offline}}$, receives security parameter s, and is parametrised by a function $e : \{0,1\}^n \times \{0,1\}^n \rightarrow \{0,1\}^n$ (with encoding C as a circuit). Upon inputs $\mathsf{ID_A}, \mathsf{ID_B}$ from Alice (the originator) and Bob (the circuit evaluator) respectively, $\mathcal{F}^e_{\mathsf{offline}}$ generates a garbled circuit C that computes e. Then it modifies the circuit C to a circuit $\mathbf{C}$ 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 input his values (during the oblivious transfer protocol); see [26] for details. Next, $\mathcal{F}^e_{\mathsf{offline}}$ garbles the circuit $\mathbf{C}$, by selecting random encodings for each possible value of each of Alice's and Bob's input and output bits, and sends the random input strings (corresponding to all possible inputs) to Alice, and the garbled gates and output decryption tables to Bob. The functionality $\mathcal{F}^e_{\mathsf{offline}}$, in Algorithm 3, ensures privacy and correctness against malicious Alice, i.e., the originator.

Input: $\mathsf{ID_A}$ from Alice, or $\mathsf{ID_B}$ from Bob, security parameter s, number of bits n
Registration Phase
  generate signature key-pair: $(vk_{off}, sk_{off}) \leftarrow \mathcal{G}(1^s)$
  register the verification key: (register, offline, vk_T) to $\mathcal{F}_{\mathsf{ca}}$
end
Computation Phase
  1. Let C be a circuit that computes e
  2. Construct $\mathbf{C}$ from C, by replacing each input wire of Bob with a XOR-gate of s new input wires of Bob
  3. Garble the resulting circuit $\mathbf{C}$ and obtain $\mathcal{C}$, consisting of:
    a. Random strings corresponding to all possible input bits of Alice: $\bar{\mathcal{K}_A} = ((\mathcal{K}^0_A[1], \mathcal{K}^1_A[1]), ..., (\mathcal{K}^0_A[n], \mathcal{K}^1_A[n]))$
    b. Random strings corresponding to all possible input bits of Bob: $\bar{\mathcal{K}_B} = ((\mathcal{K}^0_B[1], \mathcal{K}^1_B[1]), ..., (\mathcal{K}^0_B[sn], \mathcal{K}^1_B[sn]))$
    c. Garbled boolean tables $\bar{\mathcal{T}_G}$ for each garbled gate G of the circuit $\mathcal{C}$
    d. Output decryption tables $\bar{\mathcal{T}_D}$ mapping output strings to bits
  4. Sign the random input strings $\bar{\mathcal{K}_A}$ of Alice: $\bar{\sigma}_A = ((\sigma^0_A[1], \sigma^1_A[1]), ..., (\sigma^0_A[n], \sigma^1_A[n]))$ where $\forall i, j,\ \sigma^j_A[i] = \mathcal{S}_{sk_{off}}(\mathcal{K}^j_A[i], i)$
  5. Sign the random input strings $\bar{\mathcal{K}_B}$ of Bob: $\bar{\sigma}_B = ((\sigma^0_B[1], \sigma^1_B[1]), ..., (\sigma^0_B[sn], \sigma^1_B[sn]))$ where $\forall i, j,\ \sigma^j_B[i] = \mathcal{S}_{sk_{off}}(\mathcal{K}^j_B[i], i, j)$
end
Output: send $(\bar{\mathcal{K}_A}, \bar{\sigma}_A), (\bar{\mathcal{K}_B}, \bar{\sigma}_B)$ to Alice
  send $\bar{\mathcal{T}_G}, \bar{\mathcal{T}_D}$ to Bob

Algorithm 3: The functionality $\mathcal{F}^e_{\mathsf{offline}}$ for generating a garbled circuit $\mathcal{C}$ that computes e, in order to ensure integrity of computation and prevent exposure of the input of Bob.

[2] 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.

3.2 Secure Two-Party Protocol Against Malicious Adversaries

Two-party computation involves two parties, Alice and Bob, that wish to evaluate a common function on their private inputs, while ensuring privacy of inputs and integrity of computation (correctness) In this section we consider functionalities with output only at Bob (the circuit evaluator). Let $e : \{0,1\}^n \times \{0,1\}^n \to \{0,1\}^n$ be a two-party functionality, and let a, b be the inputs of Alice and Bob respectively. Then, after evaluating the functionality e on a and b, Bob obtains $e(a, b)$, while Alice learns nothing at all.

We present an implementation, in Algorithm 4, of Yao's protocol using an offline third party, in Algorithm 3, for the preprocessing phase. The protocol allows for output at Bob only and securely realises two-party computation against static malicious adversaries with security with abort (see [24, 20, 21] for standard definition of security with abort). The preprocessing phase ensures that the circuit was correctly constructed and prevents cheating by either party, essentially avoiding the computation and communication overhead, which are required when assuming malicious behaviour of the participants. Next, at the execution phase, Alice sends the strings representing her input to Bob, and runs an oblivious transfer protocol with Bob for his input bits. Once Bob obtains all the inputs, he evaluates the function, and obtains the result of the computation, thus concluding the protocol.

Other instantiations of $\mathcal{F}^e_{\mathsf{offline}}$ can be used: e.g., to reduce the trust in Offline Party, the generation of the circuit can be distributed among n offline parties, that would run a multi-party protocol, and as long as at least t ($t < n$) out of n parties are honest, the circuit is correctly generated. Alternately, we can avoid the use of third parties entirely; the $\mathcal{F}^e_{\mathsf{offline}}$ functionality can be implemented with a secure two-party computation protocol; e.g., [30, 26].

Claim 1 *Let $e : \{0,1\}^n \times \{0,1\}^n \to \{0,1\}^n$ be a polynomial time two-party functionality. Assume that the signature scheme $(\mathcal{G}, \mathcal{S}, \mathcal{V})$ is existentially unforgeable under chosen-message attack. Then protocol Π^E_e securely realises a two-party functionality with abort, with output at* Bob *only, in the presence of malicious static adversaries in the ($\mathcal{F}^e_{offline}$,$\mathcal{F}^2_{ot}$,$\mathcal{F}_{ca}$)-hybrid model with* abort.

Proof: see Appendix, Section A.1, Propositions 4 and 5.

3.2.1 Efficiency Analysis and Comparison

The classical way of producing two-party protocols secure against malicious adversaries, see [16], is based on running a zero-knowledge protocol, see [17, 18], which renders them inefficient for practical purposes. In [29] 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 $\mathcal{O}(s|C| + sn^2)$ (where n is the number of input bits to the circuit C and s is the statistical security parameter). Then [33] improved the communication complexity of [29] to $\mathcal{O}(s|C|)$ using expanders. However as [26] observed the protocol in [29] is susceptible to 'input corruption' attack (see [26] for details); [26] also present a protocol with roughly the same communication complexity as [29], of $\mathcal{O}(s|C| + s^2n)$ (this protocol was also implemented in [27]). Another improvement to two-party computation in malicious setting was made by [23] using homomorphic encryption; their protocol has a constant number of rounds, and has a communication complexity of $\mathcal{O}(|C|)$ (cf. $\mathcal{O}(s|C| + s^2n)$ in [26]) and computational complexity of $\mathcal{O}(|C|)$ (as opposed to $\mathcal{O}(n)$ in [26]). Subsequently, the work of [30], also followed the cut-and choose approach in a different manner and improved the complexity to $\mathcal{O}(\frac{s|C|}{log(|C|)})$.

Our protocol, in Algorithm 4, is computationally efficient as it uses public key operations only for signing (by $\mathcal{F}^e_{\mathsf{offline}}$) 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 $\mathcal{O}(|C|)$ (roughly as that of the original Yao's protocol). The parties obtain the generated circuit from $\mathcal{F}^e_{\mathsf{offline}}$ during the offline preprocessing phase; then during the execution Alice sends to Bob strings corresponding to her input, and they run an oblivious transfer protocol only for every input bit of Bob. Note that we added (to the original construction of Yao's protocol) the signatures on input strings of Bob by the $\mathcal{F}^e_{\mathsf{offline}}$ during the preprocessing phase, and a verification thereof later on by Bob. Thus the resulting protocol is of similar computational and communication complexity as the construction of Yao's garbled circuit, [34, 25]. Our protocol is efficient in that it has only a constant number of rounds and uses one oblivious transfer per input bit only. This is in contrast to the complexity of [26], 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 $\mathcal{O}(s|C| + s^2n)$. Note that our protocol allows general two-party computation, and efficiency can be further improved by adjusting our protocols to specific tasks.

Input: security parameter s, number of bits n
Output: $y_B = e(a, b)$
Offline Generation Phase
  Alice receives $\bar{a} = [a_i]_{i=1}^n$
  Bob receives $\bar{b} = [b_i]_{i=1}^n$
  $(b_1^1, ..., b_s^1, ..., b_1^n, ..., b_s^n) \leftarrow encodeInput(\bar{b})$ (see below)
  Alice and Bob send $\mathsf{ID_A}$, $\mathsf{ID_B}$ (respectively) to $\mathcal{F}^e_{\mathsf{offline}}$
  Alice receives $(\bar{\mathcal{K}_A}, \bar{\sigma}_A), (\bar{\mathcal{K}_B}, \bar{\sigma}_B)$
  Bob receives $\bar{\mathcal{T}_G}, \bar{\mathcal{T}_D}$
end
Computation Phase
  Alice: sends to Bob: $((\mathcal{K}_A^{a[1]}[1], \sigma_A^{a[1]}[1]), ..., (\mathcal{K}_A^{a[n]}[n], \sigma_A^{a[n]}[n]))$, $(\forall i), \mathcal{K}_A^{a[i]}[i] \in \bar{\mathcal{K}}_A, \sigma_A^{a[i]}[i] \in \bar{\sigma}_A$
  Bob:
    send (retrieve, offline) to $\mathcal{F}_{\mathsf{ca}}$ and obtain vk_{off}
    if $\exists(\mathcal{K}_A^{a[i]}[i], \sigma_A^{a[i]}[i])$, *s.t.*, $\mathcal{V}_{vkvk_{off}}(\mathcal{K}_A^{a[i]}[i], i, \sigma_A^{a[i]}[i]) =$ false **then**
      output $\perp$ and halt
    for $i \leftarrow 1$ ***to*** $n \cdot s$ **do**
      run with Alice $\mathcal{F}^2_{\mathsf{ot}}((\mathcal{K}_B^0[i], \sigma_B^0[i]), (\mathcal{K}_B^1[i], \sigma_B^1[i]), b'_i)$
      //`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 $\mathcal{V}_{vk_{off}}(\mathcal{K}_B^{b'[i]}[i], \sigma_B^{b'[i]}[i]) ==$ false **then**
        output $\perp$ and halt
    $(y_B = (y_B[1], ..., y_B[n])) \leftarrow \mathcal{C}(\bar{\mathcal{K}_\mathcal{A}}, \bar{\mathcal{K}_\mathcal{B}})$ (see below)
  end
end
Circuit Evaluation
  $\mathcal{C}((\mathcal{K}_A^{a[1]}[1], ..., \mathcal{K}_A^{a[n]}[n]), (\mathcal{K}_B^{b'[1]}[1], ..., \mathcal{K}_B^{b'[sn]}[sn]))$ {
    $(\mathcal{K}_Y^{y[1]}[1], ..., \mathcal{K}_Y^{y[n]}[n]) \leftarrow \bar{\mathcal{T}_G}((\mathcal{K}_A^{a[1]}[1], ..., \mathcal{K}_A^{a[n]}[n]), (\mathcal{K}_B^{b'[1]}[1], ..., \mathcal{K}_B^{b'[sn]}[sn]))$
  return $\omega \leftarrow \bar{\mathcal{T}_D}(\mathcal{K}_Y^{y[1]}[1], ..., \mathcal{K}_Y^{y[n]}[n])$ }
end
$//C(a, b) = \bar{\mathcal{T}_G}(\bar{\mathcal{T}_D}(\bar{\mathcal{K}_A^a}, \bar{\mathcal{K}_B^b}))$
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. $b_i = b_1^i \oplus ... \oplus b_s^i$
    $b' = b' || b_1^i, ..., b_s^i$
  return b' }//`after` n `iterations` $b' = [b'_i]_{i=1}^{n \cdot s} = (b_1^1, ..., b_s^1, ..., b_1^n, ..., b_s^n)$
end

Algorithm 4: Secure Two Party Protocol Π_e^E in the $(\mathcal{F}^e_{\mathsf{offline}}, \mathcal{F}^2_{\mathsf{ot}}, \mathcal{F}_{\mathsf{ca}})$-hybrid model, for computing $e(a, b) = y_B$, where $e : \{0, 1\}^n \times \{0, 1\}^n \rightarrow \{0, 1\}^n$.

4 Guaranteed Two-Party Computation

Our central result is the guaranteed two-party computation, with compensation of the honest participant in case of malicious behaviour of the peer. We defined the symmetric guarantee functionality, ensuring guaranteed output delivery, in Algorithm 2, and in Algorithm 6 we construct a protocol realising it. Guaranteed two-party computation, in Algorithm 6, is a two-phase protocol, s.t., during the first phase the parties provide their guarantees to participate in protocol execution, and in second phase, they evaluate a function over their inputs (which are also validated). During the first phase, we use a fair two-party protocol (that implements an ideal functionality defined in Algorithm 1) to ensure that either both parties receive the guarantees of each other or no party does. Then during the computation phase we use $\mathcal{F}^e_{\mathsf{offline}}$ functionality (in Algorithm 3) to construct the garbled circuit. The resulting protocol $\Pi^G_{(g,v)}$ guarantees output at both parties. Both the guarantees and the inputs are validated using a pair of validation predicates[3] $v = (v_1, v_2)$ (for inputs to first and second phases). During the first phase, the parties provide their respective guarantees a_1 and b_1, which are validated by running a two party protocol that ensures fairness, such that each party receives the guarantee of the other; the fair two-party protocol computes

[3]The expiration and validation of time are part of the validation predicate.

function f defined as follows:

$$f((a_1, K_A), (b_1, K_B)) = \begin{cases} \perp & if\ v_1(a_1, b_1) \in \{\mathsf{Alice}, \mathsf{Bob}\} \quad (invalid\ inputs) \\ (c_T^B || y) || (c_T^A || y) & otherwise \qquad\qquad (valid\ inputs) \end{cases} \tag{1}$$

where $c_T^A = \mathcal{NE}_{ek_T}(\mathcal{E}_{K_A}(b_1))$, $c_T^B = \mathcal{NE}_{ek_T}(\mathcal{E}_{K_B}(a_1))$ and $y = v_1(a_1, b_1)$. If the guarantee of one of the parties is not valid, v_1 outputs the idendity of that party, in which case f outputs $\perp$. Both parties terminate execution if the output from the guarantee phase is $\perp$. Once the guarantee phase completed, the parties have each other's committments, and can engage in computation of the second phase. For simplicity, we embed the validation predicate v_2 and the function g that the parties wish to compute over their inputs during the computation phase, into function e (which we define next); specifically, prior to evaluation of $g = (g_A, g_B)$, if validation by v_2 fails, e produces $\perp$.

$$e((a_2, K_A, sk_A), b_2) = \begin{cases} \perp & if\ v_2(a_2, b_2) \in \{\mathsf{Alice}, \mathsf{Bob}\} \quad (invalid\ inputs) \\ ((\mathcal{E}_{K_A}(g_A(a_2, b_2)), \sigma_A) || g_B(a_2, b_2)) & otherwise \qquad\qquad (valid\ inputs) \end{cases} \tag{2}$$

At this stage each party holds the guarantee by the other, and can contact the trusted enforcement authority functionality $\mathcal{F}_{\mathsf{tea}}$ (in Algorithm 5) in case a malicious party fails to participate, or provides an incorrect input to the computation. The $\mathcal{F}_{\mathsf{tea}}$ attempts to complete the protocol with the other party on behalf of the party originating the resolution. In case of failure, the $\mathcal{F}_{\mathsf{tea}}$ opens the guarantee and sends it to the originating party. Otherwise, it concludes the protocol, and returns the result of the computation to the originating party.

Input: n, s, Δ_C

generates $(ek_T, dk_T) \xleftarrow{R} \mathcal{K}(1^n)$ and sends (register, TEA, ek_T) to $\mathcal{F}_{\mathsf{ca}}$
send (retrieve, offline) to $\mathcal{F}_{\mathsf{ca}}$ and obtains vk_{off}
send (retrieve, Alice) to $\mathcal{F}_{\mathsf{ca}}$ and obtains vk_A
Computation Phase
  $y_B = \perp$
  onReceive($\mathcal{NE}_{ek_T}(\mathcal{E}_{K_B}(a_1))$)
    send ('garbled inputs') to Alice
    sleep('response from Alice', $2\Delta_C$)
  onWakeup('response from Alice')
    $y_B \leftarrow \mathcal{ND}_{dk_T}(\mathcal{E}_{K_B}(a_1))$
    send (y_B) to Bob
  onReceive((($\bar{\mathcal{K}}_A^a, \bar{\sigma}_A^a$), ($\bar{\mathcal{K}}_B, \bar{\sigma}$), $\mathcal{NE}_{ek_T}(\mathcal{E}_{K_A}(b_1))$)) from Alice
  if $\forall i : (\mathcal{V}_{vk_{off}}(\mathcal{K}_A^{a[i]}[i], \sigma_A^{a[i]}[i]) \neq \perp) \vee (\mathcal{V}_{vk_{off}}(\mathcal{K}_B^{b[i]}[i], \sigma_B^{b[i]}[i]) \neq \perp)$ **then**
    send ($\bar{\mathcal{K}}_A^a, \bar{\sigma}_A^a$) to Bob
    for $i \leftarrow 1$ ***to*** $n \cdot s$ **do**
      run $\mathcal{F}_{\mathsf{ot}}^2((\mathcal{K}_B^0[i], \sigma_B^0[i]), (\mathcal{K}_B^1[i], \sigma_B^1[i]), b_i)$ with Bob, Bob receives $(\mathcal{K}_B^{b[i]}[i], \sigma_B^{b[i]}[i])$
    sleep('time to Bob',$2\Delta_C$)
    onReceive(c_A, σ_A) from Bob
      if $(\mathcal{V}_{vk_A}(c_A, \sigma_A) \neq \perp)$ **then**
        stopTimer('time to Bob')
        send (c_A, σ_A) to Alice
    onWakeup('time to Bob')
      send $\mathcal{E}_{K_A}(b_1)$ to Alice
  else
    if $y_B \neq \perp$ **then**
      send (y_B) to Bob
end

Algorithm 5: The ideal functionality $\mathcal{F}_{\mathsf{tea}}$

Claim 2 *Let $g = (g_A, g_B)$ be a polynomial two-party functionality, and let $(\mathcal{K}, \mathcal{E}, \mathcal{D})$ be a secure shared key encryption scheme, let $(\mathcal{NG}, \mathcal{NE}, \mathcal{DE})$ be a non-malleable encryption scheme, and $(\mathcal{G}, \mathcal{S}, \mathcal{V})$ is an existentially unforgeable under chosen-message attack signature scheme. Then protocol $\Pi_{(v,g)}^G$ securely realises $\mathcal{F}_{\Delta-guaranteed\text{-}computation}^{v,g}$ in the presence of malicious static adversaries in the ($\mathcal{F}_{offline}^e$,$\mathcal{F}_{ca}$,$\mathcal{F}_{tea}$,$\mathcal{F}_{\Delta\text{-}delayed\text{-}fairness}^f$)-hybrid model, with $\Delta = 6\Delta_C$.*

Proof: see Appendix, Section A.3, Propositions 8 and 9.

Input: n, s, Δ_C
Guarantee Phase
  Input: a_1 from Alice, b_1 from Bob
  Alice and Bob do:
    send (retrieve, tea) to $\mathcal{F}_{\mathsf{ca}}$ and both obtain ek_T
    run Π_f^F (in Algorithm 7 realising ideal functionality $\mathcal{F}_f$ in Algorithm 1) computing f (f is constructed from v_1 as in Equation 1) with key ek_T of the TEA, on inputs (a_1, b_1) to generate and validate guarantees
  Bob receives $(\mathcal{NE}_{ek_T}(\mathcal{E}_{K_B}(a_1))||v_1(a_1, b_1))$, Alice receives $(\mathcal{NE}_{ek_T}(\mathcal{E}_{K_A}(b_1))||v_1(a_1, b_1))$
  if $(v_1(a_1, b_1) == \perp)$ **then**
    Alice and Bob output $\perp$ and halt
end
Computation Phase
  Input: a_2 from Alice, b_2 from Bob
  Bob encodes b_2 as $(b_1^1, ..., b_s^1, ..., b_1^n, ..., b_s^n)$: $[b'_i]_{i=1}^{n \cdot s} \leftarrow encodeInput(b_2)$ (see implementation in procedure Encode Input in Algorithm 4)
  Alice and Bob run functionality $\mathcal{F}^e_{\mathsf{offline}}$ (in Algorithm 3) to generate garbled circuit computing function e (e is constructed from g and v_2 as in Equation 2)
  Bob sends (retrieve, offline) to $\mathcal{F}_{\mathsf{ca}}$ and obtains vk_{off}
  Alice generates signature key-pair: $(sk_A, vk_A) \leftarrow \mathcal{G}(1^n)$, and registers: (register, Alice, vk_A) with $\mathcal{F}_{\mathsf{ca}}$
  Alice sends to Bob her encoded input $a = (a_2, K_A, sk_A)$: $((\mathcal{K}_A^{a[1]}[1], \sigma_A^{a[1]}[1]), ..., (\mathcal{K}_A^{a[n]}[n], \sigma_A^{a[n]}[n]))$
  if $\exists(\mathcal{K}_A^{a[i]}[i], \sigma_A^{a[i]}[i])$, *s.t.*, $\mathcal{V}_{vk_{off}}(\mathcal{K}_A^{a[i]}[i], i, \sigma_A^{a[i]}[i]) == \perp$ **then** Bob sends $\mathcal{NE}_{ek_T}(\mathcal{E}_{K_B}(a_1))$ to $\mathcal{F}_{\mathsf{tea}}$
  for $i \leftarrow 1$ ***to*** $s \cdot n$ **do**
    Alice and Bob run $\mathcal{F}^2_{\mathsf{ot}}((\mathcal{K}_B^0[i], \sigma_B^0[i]), (\mathcal{K}_B^1[i], \sigma_B^1[i]), b_i)$, Bob receives $(\mathcal{K}_B^{b[i]}[i], \sigma_B^{b[i]}[i])$
    if $\mathcal{V}_{vk_{off}}(\mathcal{K}_B^{b[i]}[i], \sigma_B^{b[i]}[i]) == \perp$ **then** Bob sends $\mathcal{NE}_{ek_T}(\mathcal{E}_{K_B}(a_1))$ to $\mathcal{F}_{\mathsf{tea}}$

Bob:
$((\mathcal{E}_{K_A}(y_A), \sigma_A)||y_B) \leftarrow \mathcal{C}(\bar{\mathcal{K}_A}, \bar{\mathcal{K}_B})$
(procedure **C**ircuit Evaluation C in Algorithm 4)
if $y_B == \perp$ **then** send $\mathcal{NE}_{ek_T}(\mathcal{E}_{K_B}(a_1))$ to $\mathcal{F}_{\mathsf{tea}}$
else output y_B, send $(\mathcal{E}_{K_A}(y_A), \sigma_A)$ to Alice
onReceive($\bar{\mathcal{K}_A^a}, \bar{\sigma^a}$) from $\mathcal{F}_{\mathsf{tea}}$
  run $\mathcal{F}^2_{\mathsf{ot}}$ with $\mathcal{F}_{\mathsf{tea}}$
  obtain $\forall i$, $(\mathcal{K}_B^{b[i]}[i], \sigma_B^{b[i]}[i])$
  $((\mathcal{E}_{K_A}(y_A), \sigma_A)||y_B) \leftarrow \mathcal{C}(\bar{\mathcal{K}_A}, \bar{\mathcal{K}_B})$
  send $(\mathcal{E}_{K_A}(y_A), \sigma_A)$ to $\mathcal{F}_{\mathsf{tea}}$
onReceive($\mathcal{E}_{K_B}(a_1)$) from $\mathcal{F}_{\mathsf{tea}}$
  recover and output a_1

Alice:
sleep('response from Bob',$2\Delta_C$)
onReceive($\mathcal{E}_{K_A}(y_A), \sigma_A$)
  if $(\mathcal{V}_{vk_A}(\mathcal{E}_{K_A}(y_A), \sigma_A) \neq \perp)$ **then**
    stopTimer('response from Bob')
    recover and output y_A
onWakeup('response from Bob')
  send $((\bar{\mathcal{K}_A}, \bar{\sigma}_A), (\bar{\mathcal{K}_B}, \bar{\sigma}_B), \mathcal{NE}_{ek_T}(\mathcal{E}_{K_A}(b_1)))$ to $\mathcal{F}_{\mathsf{tea}}$
  onReceive($\mathcal{E}_{K_A}(b_1)$) from $\mathcal{F}_{\mathsf{tea}}$
    recover and output b_1
onReceive('garbled inputs')
  send $((\bar{\mathcal{K}_A}, \bar{\sigma}_A), (\bar{\mathcal{K}_B}, \bar{\sigma}_B), \mathcal{NE}_{ek_T}(\mathcal{E}_{K_A}(b_1)))$ to $\mathcal{F}_{\mathsf{tea}}$

end

Algorithm 6: Guaranteed fair secure two-party protocol $\Pi^G_{(v,g)}$ in the ($\mathcal{F}^e_{\mathsf{offline}}$,$\mathcal{F}_{\mathsf{ca}}$,$\mathcal{F}_{\mathsf{tea}}$,$\mathcal{F}^f_{\Delta\text{-delayed-fairness}}$)-hybrid model for computing $g = (g_a, g_B)$, with $\Delta = 6\Delta_C$.

5 Fair Two-Party Protocol Against Malicious Adversaries

A protocol is said to be fair if either both parties receive an output or noone does. Specifically, Alice receives her output if and only if Bob receives his, or no party receives the output.

In Algorithm 7 we construct a protocol Π_f^F that realises the $\mathcal{F}^f_{\Delta\text{-delayed-fairness}}$ functionality presented in Algorithm 1. Concretely, protocol Π_f^F computes functionality $f(a, b) = (f_A(a, b), f_B(a, b))$, providing output at both Alice and Bob while ensuring Δ-delayed fairness, i.e., either no one receives output or both participants do, such that honest party's output will be delayed by at most a factor of Δ. The protocol in Algorithm 7 uses a weakly trusted (oblivious) third party, captured by the ideal functionality $\mathcal{F}_{\mathsf{Resolve}}$ in Algorith 8, involved only for resolution in case one of the parties misbehaves. We believe that the model based on the separation between the offline generation and evaluation phases, is suitable for protocols that are to be run by ad-hoc parties in order to execute a variety of transactions over the Internet, while ensuring privacy, correctness and fairness[4]. Specifically, the (offline) third party, $\mathcal{F}^e_{\mathsf{offline}}$, that is used during the generation phase, ensures correctness and privacy, and the optimistic third party, 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.

[4]We note that having one party perform all the functionalities is also possible, albeit, less appealing in practice, due to required trust (by everyone) in one such entity.

To construct Π_f^F we use as a module the ideal functionality for protocol Π_e^E, in Section 3 (Algorithm 4), that allows to compute securely any functionality with output only at Bob. The resolver, $\mathcal{F}_{\mathsf{Resolve}}$, generates a key pair $(dk_R, ek_R) \leftarrow \mathcal{NG}(1^n)$ (see Algorithm 8), and this key is part of the function e. The functionality $\mathcal{F}^e_{\mathsf{offline}}$ generates a garbled circuit that compute e such that part of the output is encrypted with the key ek_R. We take the function e for Π_e^E (that provides output at Bob only) to be the function computing the following:

$$e((a, K_A), (b, K_B)) = \mathcal{NE}_{ek_R}(c_A||c_B)||(\mathcal{E}_{K_A}(f_A(a, b), c_B)) \tag{3}$$

where $c_A = \mathcal{E}_{K_A}(f_A(a, b))$ and $c_B = \mathcal{E}_{K_B}(f_B(a, b))$. When the protocol, in Algorithm 7, is initiated, Alice and Bob retrieve the public encryption key ek_R of the resolver $\mathcal{F}_{\mathsf{Resolve}}$ which defines the function that they agree to compute. 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_e^E and provide their inputs, $(a||K_A)$ and $(b||K_B)$ respectively. The protocol Π_e^E 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 $\mathcal{F}_{\mathsf{Resolve}}$ (containing both the output of Bob and of Alice). The output part of the $\mathcal{F}_{\mathsf{Resolve}}$ is encrypted with a non-malleable encryption scheme $(\mathcal{NG}, \mathcal{NE}, \mathcal{ND})$ (see [14] for details) 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). Since Bob performed the computation, he is assured that the output is constructed correctly. Bob sends the output (encrypted with Alice's key) to Alice. If Alice does not respond, Bob contacts the resolver with the part of the output encrypted with the key of the resolver. The $\mathcal{F}_{\mathsf{Resolve}}$ 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.

Input: security parameters n, s, maximal communication delay Δ_C, $a = [a_i]_{i=1}^n$ from Alice, $b = [b_i]_{i=1}^n$ from Bob
Output: $y = (y_A, y_B)$
Alice and Bob send (retrieve, resolve) to $\mathcal{F}_{\mathsf{ca}}$ and obtain ek_R (each)
Computation Phase
Alice and Bob do:
generate secret keys K_A and K_B respectively
run a protocol Π^E (in Algorithm 4 realising $\mathcal{F}_e$) computing e (e is constructed from f as in Equation 3) with key ek_R of the resolver, on inputs $(a||K_A, b||K_B)$
Alice provides $(a||K_A)$ and Bob provides $(b||K_B)$

Bob:
onReceive(y_B)
if($y_B == \mathcal{E}_{K_A}(f_A(a, b), c_B)||\mathcal{NE}_{ek_R}(c_A, c_B)$) **then**
send($\mathcal{E}_{K_A}(f_A(a, b), c_B)$) to Alice
sleep('time to Alice', $2\Delta_C$)
else output $\perp$ and halt
onReceive(c_B)
if ($c_B ==$ valid) **then**
stopTimer('time to Alice')
recover and output $y_B = f_B(a, b)$
onWakeup('time to Alice')
send $\mathcal{NE}_{ek_R}(c_A, c_B)$ to $\mathcal{F}_{\mathsf{Resolve}}$
onReceive(c_B)
stopTimer('time to $\mathcal{F}_{\mathsf{Resolve}}$')
recover and output $y_B = f_B(a, b)$
end

Alice:
onReceive(c)
if ($c ==$ valid) **then**
recover and output $y_A = f_A(a, b)$
if ($c == \mathcal{E}_{K_A}(f_A(a, b), c_B)$) **then**
send(c_B) to Bob
end

end

Algorithm 7: Secure Two Party Protocol Π_f^F that realises $\mathcal{F}^f_{\Delta\text{-delayed-fairness}}$ with $\Delta = 4\Delta_C$, in the $(\mathcal{F}_{\mathsf{Resolve}}, \mathcal{F}_{\mathsf{ca}}, \mathcal{F}_{\mathsf{e}})$-hybrid model for computing $f(a, b) = (f_A(a), f_B(b))$, where $f : \{0,1\}^m \times \{0,1\}^m \to \{0,1\}^m \times \{0,1\}^m$; and $e_{ek_R}((a, K_A), (b, K_B)) = \mathcal{NE}_{ek_R}(c_A||c_B)||(\mathcal{E}_{K_A}(f_A(a, b), c_B))$, where $c_A = \mathcal{E}_{K_A}(f_A(a, b))$ and $c_B = \mathcal{E}_{K_B}(f_B(a, b))$.

Claim 3 *Let $f : \{0,1\}^m \times \{0,1\}^m \to \{0,1\}^m \times \{0,1\}^m$ 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^F securely realises $\mathcal{F}^f_{\Delta\text{-delayed-fairness}}$ in the presence of malicious static adversaries in the $(\mathcal{F}_{\mathsf{Resolve}}, \mathcal{F}_{\mathsf{ca}}, \mathcal{F}_e)$-hybrid model, with $\Delta = 4\Delta_C$.*

Input: n
generate encryption key-pair: $(dk_R, ek_R) \leftarrow \mathcal{NG}(1^n)$
register the encryption key: (register, resolver, ek_R) to $\mathcal{F}_{\mathsf{ca}}$
Computation Phase
  receive c
  set $y_A = \perp,\ y_B = \perp$
  if $\mathcal{ND}_{dk_R}(c) \neq \perp$ **then**
    $(y_A, y_B) \leftarrow \mathcal{ND}_{dk_R}(c)$
end
Output: send y_A to Alice
  send y_B to Bob

Algorithm 8: The ideal functionality $\mathcal{F}_{\mathsf{Resolve}}$

Proof: see Appendix, Section A.2, Propositions 6 and 7.

Efficiency Analysis and Comparison

There are two central approaches to achieving fairness, the *gradual release* of secrets and the *optimistic model*. The gradual release imposes high communication complexity (even in case the parties are honest). A two-paty protocol, for general functions, based on gradual release (with the number of rounds proportional to the security parameter) was presented in [31]. The optimistic model relies on a third party that is involved in case of misbehaviour to restore fairness. In [8], 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 $\mathcal{O}(s|C|)$ communication and computational complexity. Furthermore, the protocol of [8] 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 addition, the work of [8] lacks a full proof of security. We are not aware of other works that provide fairness for general computations.

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 3.2.1 for discussion and analysis). When one of the parties misbehaves, the protocol requires an additional round, to send the encrypted result to the resolver and to receive a decrypted response back.

6 Conclusions

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 [32]), there are no real life 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.

We provide new notions of security aimed at addressing the requirements of the potential applications of secure ecommerce, and present protocols based on these definitions (i.e., of fairness and guaranteed output delivery). 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. Our protocols can employ only one third party to provide all the required security guarantees. However, we suggest to distribute the tasks between different third parties, e.g., the offline preprocessing functionality and the functionality involved for resolution, which we believe to be more applicable to real life scenarios. We stress that the success of electronic financial applications may depend on the ability to construct protocols that provide rigorous security guarantees that are necessary and that are sufficiently efficient. We believe that applying 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.

References

[1] Information technology – security techniques – evaluation criteria for it security – part 2: Security functional components, 2008.

[2] N. Asokan, M. Schunter, and M. Waidner. Optimistic protocols for fair exchange. In *Proceedings of the 4th ACM conference on Computer and communications security*, page 17. ACM, 1997.

[3] N. Asokan, V. Shoup, and M. Waidner. Optimistic fair exchange of digital signatures. *Advances in Cryptology EUROCRYPT 1998*, pages 591–606, 1998.

[4] D. Beaver and S. Goldwasser. Multiparty Computation with Faulty Majority. In *Advances in Cryptology - Crypto 1989 Proceedings*, pages 589–590. Springer, 1990.

[5] M. Bellare and 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*, pages 531–545, London, UK, 2000. Springer-Verlag. ISBN 3-540-41404-5.

[6] M. Ben-Or, O. Goldreich, S. Micali, and R. Rivest. A fair protocol for signing contracts. *Automata, Languages and Programming*, pages 43–52, 1985.

[7] E. F. Brickell, D. Chaum, I. B. Damgård, and J. van de Graaf. Gradual and verifiable release of a secret. In Carl Pomerance, editor, *Proc. CRYPTO 87*, pages 156–166. Springer-Verlag, 1988. Lecture Notes in Computer Science No. 293.

[8] C. Cachin and J. Camenisch. Optimistic fair secure computation. In Mihir Bellare, editor, *Advances in Cryptology – CRYPTO ' 2000*, volume 1880 of *Lecture Notes in Computer Science*, pages 93–111. International Association for Cryptologic Research, Springer-Verlag, Berlin Germany, 2000.

[9] R. Canetti. Universally Composable Security: A New Paradigm for Cryptographic Protocols. In *Proceedings of the 42nd IEEE symposium on Foundations of Computer Science*, page 136. IEEE Computer Society, 2001.

[10] S.G. Choi, A. Elbaz, A. Juels, T. Malkin, and 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*, pages 298–314. Springer-Verlag, 2007.

[11] 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*, pages 364–369. ACM, 1986.

[12] R. Cleve. Controlled gradual disclosure schemes for random bits and their applications. In *Proceedings on Advances in cryptology*, pages 573–588. Springer-Verlag New York, Inc., 1989.

[13] I.B. Damgård. Practical and provably secure release of a secret and exchange of signatures. *Journal of Cryptology*, 8 (4):201–222, 1995.

[14] D. Dolev, C. Dwork, and M. Naor. Non-malleable cryptography. In *Proceedings of the twenty-third annual ACM symposium on Theory of computing*, pages 542–552. ACM, 1991.

[15] J. Garay, P. MacKenzie, M. Prabhakaran, and K. Yang. Resource fairness and composability of cryptographic protocols. *Theory of Cryptography*, pages 404–428, 2006.

[16] O. Goldreich. *Foundations of Cryptography: Volume 2, Basic Applications*. Cambridge University Press New York, NY, USA, 2004.

[17] O. Goldreich, S. Micali, and 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*, pages 218–229, 1987.

[18] O. Goldreich, S. Micali, and 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):690–728, 1991.

[19] S. Goldwasser and L. Levin. Fair computation of general functions in presence of immoral majority. *Advances in Cryptology-CRYPT0'90*, pages 77–93, 1990.

[20] D. Gordon and J. Katz. Partial fairness in secure two-party computation. Technical report, Cryptology ePrint Archive, Report 2008/206, 2008, 2008.

[21] D.S. Gordon, H. Carmit, J. Katz, and Y. Lindell. Complete fairness in secure two-party computation. In *Proceedings of the 40th annual ACM symposium on Theory of computing*, pages 413–422. ACM, 2008.

[22] ISO JTC 1/SC 27. ISO/IEC 13888-1:2009 information technology – security techniques – non-repudiation, 2009.

[23] S. Jarecki and V. Shmatikov. Efficient two-party secure computation on committed inputs. *Advances in Cryptology-EUROCRYPT 2007*, pages 97–114, 2007.

[24] A.Y. 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*, pages 121–137. Springer-Verlag, 2008.

[25] Y. Lindell and B. Pinkas. A Proof of Yao Protocol for Secure Two-Party Computation. In *Electronic Colloquium on Computational Complexity*, volume 11, page 063, 2004.

[26] Y. Lindell and B. Pinkas. An efficient protocol for secure two-party computation in the presence of malicious adversaries. *Advances in Cryptology-EUROCRYPT 2007*, pages 52–78, 2007.

[27] D. Malkhi, N. Nisan, B. Pinkas, and Y. Sella. Fairplay - secure two-party computation system. In *Proceedings of the 13th USENIX Security Symposium*, pages 287–302. USENIX, 2004. URL `http://www.usenix.org/publications/library/proceedings/sec04/tech/malkhi.html`.

[28] S. Micali. Simple and fast optimistic protocols for fair electronic exchange. In *Proceedings of the twenty-second annual symposium on Principles of distributed computing*, page 19. ACM, 2003.

[29] P. Mohassel and M. Franklin. Efficiency tradeoffs for malicious two-party computation. *Public Key Cryptography-PKC 2006*, pages 458–473, 2006.

[30] J.B. Nielsen and C. Orlandi. LEGO for Two-Party Secure Computation. In *Proceedings of the 6th Theory of Cryptography Conference on Theory of Cryptography*, pages 368–386. Springer-Verlag, 2009.

[31] B. Pinkas. Fair secure two-party computation. *Advances in Cryptology-Eurocrypt 2003*, pages 647–647, 2003.

[32] B. Pinkas, T. Schneider, N. Smart, and S. Williams. Secure two-party computation is practical. *Advances in Cryptology–ASIACRYPT 2009*, pages 250–267, 2009.

[33] D. Woodruff. Revisiting the efficiency of malicious two-party computation. *Advances in Cryptology-EUROCRYPT 2007*, pages 79–96, 2007.

[34] A. C. Yao. How to generate and exchange secrets. In *Proc. 27th IEEE Symp. on Foundations of Comp. Science*, pages 162–167, Toronto, 1986. IEEE.

A Security Proofs

A.1 Security Analysis of Protocol Π_e^E (Section 3.2)

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

Proposition 4 (Security Against Malicious Alice**)** *For every polynomial time adversary A corrupting* Alice *and running with Π_f with abort in a hybrid model with access to $\mathcal{F}_{\mathsf{offline}}^e$, $\mathcal{F}_{ot}^2$ and $\mathcal{F}_{ca}$, there exists a probabilistic 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:*

$$\left\{\text{IDEAL}_{f,S(z)}(a,b,n)\right\}_{n\in\mathbb{N}} = \left\{\text{HYBRID}^{\mathcal{F}_{\mathsf{offline}},\mathcal{F}_{\mathsf{ca}},\mathcal{F}_{\mathsf{ot}}}_{\Pi_f,A(z)}(a,b,n)\right\}_{n\in\mathbb{N}}$$

Proof Let A be a malicious static adversary with Alice and Bob running the protocol in Algorithm 4. We construct an ideal model simulator S which has access to Alice and to the trusted party computing $\mathcal{F}_e$, and can simulate the view of the execution of the protocol. Assume that Alice is corrupted by a hybrid model adversary A. In Algorithm 9 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 $(\mathcal{F}^e_{\mathsf{offline}},\mathcal{F}_{\mathsf{ca}},\mathcal{F}^2_{\mathsf{ot}})$-hybrid execution of Π_e 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 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.

$S(a, \mathsf{ID_A}, 1^n)$
- $\mathsf{ID_{A'}} \xleftarrow{\text{OfflineParty}} A(a, \mathsf{ID_A}, 1^n)$
- **if** $\mathsf{ID_{A'}} = \perp \vee \mathsf{ID_{A'}} \neq \mathsf{ID_A}$ **then**
  - send $\perp$ to the trusted party computing $\mathcal{F}_f$ as Alice's input
  - send $\perp$ to A as its input from $\mathcal{F}^e_{\mathsf{offline}}$
  - output whatever A outputs and halt
- **else**
  - simulate functionality $\mathcal{F}^e_{\mathsf{offline}}$ for A:
  1. choose a key pair $(vk, sk) \leftarrow \mathcal{G}(1^n)$
  2. construct a circuit C computing f'_A
  3. construct $\mathbf{C}$ from C, by replacing each input wire of Bob with a XOR-gate consisting of s input wires of Bob
  4. garble the resulting circuit C and obtain $\mathcal{C}$, consisting of:
     - a. Random strings corresponding to all possible input bits of Alice: $\bar{\mathcal{K}_A} = ((\mathcal{K}^0_A[0], \mathcal{K}^1_A[0]), ..., (\mathcal{K}^0_A[n], \mathcal{K}^1_A[n]))$
     - b. Random strings corresponding to all possible input bits of Bob: $\bar{\mathcal{K}_B} = ((\mathcal{K}^0_B[0], \mathcal{K}^1_B[0]), ..., (\mathcal{K}^0_B[n], \mathcal{K}^1_B[n]))$
     - c. Garbled boolean tables $\bar{\mathcal{T}_G}$ for each garbled gate G of the circuit $\mathcal{C}$

     Output decryption tables $\bar{\mathcal{T}_D}$ mapping output strings to bits
  5. sign the random input strings $\bar{\mathcal{K}_B}$ of Bob: $\bar{\sigma} = \mathcal{S}_{sk_T}(\bar{\mathcal{K}_B})$, where $\bar{\sigma} = ((\sigma^0_0, \sigma^1_0), ..., (\sigma^0_n, \sigma^1_n))$
  6. send $\bar{\mathcal{K}_A}, (\bar{\mathcal{K}_B}, \bar{\sigma})$ to A as its output from $\mathcal{F}^e_{\mathsf{offline}}$

  A sends $\bar{\mathcal{K}'_A}$, intended for Bob and $(\bar{\mathcal{K}'_B}, \bar{\sigma}')$ for ideal functionality $\mathcal{F}^2_{\mathsf{ot}}$

  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 $\mathcal{F}_f$ as Alice's input
  - send $\perp$ to A as its input from $\mathcal{F}^2_{\mathsf{ot}}$
  - output whatever A outputs and halt
- A outputs its view and halts, S outputs the same and halts

end

Algorithm 9: Simulator S, simulating the view of Alice.

Proposition 5 (Security Against Malicious Bob) *For every polynomial time adversary A corrupting* Bob *and running with Π_f with abort in a hybrid model with access to $\mathcal{F}^e_{\mathsf{offline}}$,$\mathcal{F}^2_{\mathsf{ot}}$ and $\mathcal{F}_{\mathsf{ca}}$, there exists a probabilistic polynomial-time simulator S corrupting* Bob *and running in the ideal model with access to an ideal functionality computing $\mathcal{F}_f$, such that for every $a, b, z \in \{0,1\}^*$ holds:*

$$\left\{\text{IDEAL}_{f,S(z)}(a,b,n)\right\}_{n\in\mathbb{N}} = \left\{\text{HYBRID}^{\mathcal{F}_{\mathsf{offline}},\mathcal{F}_{\mathsf{ca}},\mathcal{F}_{\mathsf{ot}}}_{\Pi_f,A(z)}(a,b,n)\right\}_{n\in\mathbb{N}}$$

Proof Let A be a malicious static adversary with Alice and Bob running the protocol in Algorithm 4. We construct an ideal model simulator S which has access to Bob and to the trusted party computing $\mathcal{F}_f$, and can simulate the view of the execution of the protocol. Assume that Bob is corrupted by a hybrid model adversary A. In Algorithm 10 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 $\mathcal{F}^2_{\mathsf{ot}}$ 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 $(\mathcal{F}^e_{\mathsf{offline}},\mathcal{F}^2_{\mathsf{ot}},\mathcal{F}_{\mathsf{ca}})$-hybrid execution of protocol Π_f 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.

A.2 Security Analysis of Protocol Π^F_f (Section 5)

Proof We analyse Π^F_f in a $(\mathcal{F}_{\mathsf{Resolve}},\mathcal{F}_{\mathsf{ca}},\mathcal{F}_e)$-hybrid model, and show that the execution of Π^F_f is computationally indistinguishable from computation of f in the ideal model with Δ-delayed fairness. We prove the Claim 3 in Propositions 6 and 7 respectively.

$S(b, \mathsf{ID_B}, 1^n)$
  $\mathsf{ID_{B'}} \overset{\text{OfflineParty}}{\longleftarrow} A(b, \mathsf{ID_B}, 1^n)$
  if $\mathsf{ID_{B'}} = \perp \vee \mathsf{ID_{B'}} \neq \mathsf{ID_B}$ **then**
    send $\perp$ to the trusted party computing $\mathcal{F}_f$ as Bob's input
    send $\perp$ to A as its input from $\mathcal{F}^e_{\mathsf{offline}}$
    output whatever A outputs and halt
  else
    simulate functionality $\mathcal{F}^e_{\mathsf{offline}}$ for A:
    1. choose a key pair $(vk, sk) \leftarrow \mathcal{G}(1^n)$
    3. when A sends (retrieve, $\mathcal{F}^e_{\mathsf{offline}}$) to $\mathcal{F}_{\mathsf{ca}}$, respond with (retrieve $\mathcal{F}^e_{\mathsf{offline}}$, vk):
    4. construct a circuit C computing f'_A
    5. construct $\mathbf{C}$ from C, by replacing each input wire of Bob with a XOR-gate consisting of s input wires of Bob
    6. garble the resulting circuit C and obtain $\mathcal{C}$, consisting of:
      a. Random strings corresponding to all possible input bits of Alice: $\bar{\mathcal{K}_A} = ((\mathcal{K}^0_A[0], \mathcal{K}^1_A[0]), ..., (\mathcal{K}^0_A[n], \mathcal{K}^1_A[n]))$
      b. Random strings corresponding to all possible input bits of Bob: $\bar{\mathcal{K}_B} = ((\mathcal{K}^0_B[0], \mathcal{K}^1_B[0]), ..., (\mathcal{K}^0_B[n], \mathcal{K}^1_B[n]))$
      c. Garbled boolean tables $\bar{\mathcal{T}_G}$ for each garbled gate G of the circuit $\mathcal{C}$
      Output decryption tables $\bar{\mathcal{T}_D}$ mapping output strings to bits
    7. sign the random input strings $\bar{\mathcal{K}_B}$ of Bob: $\bar{\sigma} = \mathcal{S}_{sk_T}(\bar{\mathcal{K}_B})$, where $\bar{\sigma} = ((\sigma^0_0, \sigma^1_0), ..., (\sigma^0_n, \sigma^1_n))$
    8. send $\bar{\mathcal{T}_G}, \bar{\mathcal{T}_D}$ to A as its output from $\mathcal{F}^e_{\mathsf{offline}}$
    for $i \leftarrow 1$ ***to*** $|b|$ **do**
      run $\mathcal{F}^2_{\mathsf{ot}}((\mathcal{K}^0_B[i], \sigma^0_i), (\mathcal{K}^1_B[i], \sigma^1_i), b_i)$, providing $(\mathcal{K}^0_B[i], \sigma^0_i), (\mathcal{K}^1_B[i], \sigma^1_i)$ and A provides b_i
      A receives $(\mathcal{K}^{b_i}_B[i], \sigma^{b_i}_i)$
      output whatever A outputs and halt
end

Algorithm 10: Simulator S, simulating the view of Bob.

Proposition 6 (Security Against Malicious Alice) *For every non-uniform polynomial time adversary A corrupting* Alice *and running Π_g with* abort *in a hybrid model with access to $\mathcal{F}_{Resolve}$, $\mathcal{F}_{ca}$ and $\mathcal{F}_e$, there exists a non-uniform polynomial time simulator S corrupting* Alice *and running in the ideal model with access to an ideal functionality $\mathcal{F}^f_{\Delta\text{-delayed-fairness}}$, such that for every $a, b, z \in \{0,1\}^*$ holds:*

$$\left\{\text{IDEAL}_{f,S(z)}(a,b,n)\right\}_{n\in\mathbb{N}} = \left\{\text{HYBRID}^{\mathcal{F}_{Resolve},\mathcal{F}_{\mathsf{ca}},\mathcal{F}_e}_{\Pi_f, A(z)}(a,b,n)\right\}_{n\in\mathbb{N}}$$

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 11 we construct a simulator S given a black-box access to A.

S generates $(dk, ek) \leftarrow \mathcal{K}(1^n)$ and selects a random key $K_S \in \{0,1\}^n$
S invokes A with input $a, \mathsf{ID_A}, n$
When A sends (retrieve,resolve) for $\mathcal{F}_{\mathsf{ca}}$, S responds with (retrieve,resolve,ek)
S obtains A's inputs (a', K_A, ek') for the trusted party $\mathcal{F}^f_{\Delta\text{-delayed-fairness}}$
if $a' \neq a \vee ek' \neq ek$ **then**
  send $\perp$ to $\mathcal{F}^f_{\Delta\text{-delayed-fairness}}$
  send $\perp$ to A
  output whatever A outputs and halt
else
  S sends a to the trusted party computing $\mathcal{F}^f_{\Delta\text{-delayed-fairness}}$, and receives back y_A
  S chooses a random string $s_B \in \{0,1\}^n$, computes $\mathcal{E}_{K_A}(y_A, \mathcal{E}_{K_S}(s_B))$, and hands the encrypted result to A
  if *after* $2\Delta_C$ *no response arrives from* A **then**
    send unfair to trusted party.
  else
    A sends c_B
    if $c_B == \mathcal{E}_{K_S}(s_B)$ **then**
      send fair to trusted party
S outputs whatever A outputs.

Algorithm 11: The simulator S running in ideal model with trusted party computing $\mathcal{F}^f_{\Delta\text{-delayed-fairness}}$, and simulating the view of Alice.

The view of A in a simulation with S is identical to its view in an $(\mathcal{F}_{\mathsf{Resolve}},\mathcal{F}_{\mathsf{ca}},\mathcal{F}_e)$-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.

Proposition 7 (Security Against Malicious Bob) *For every non-uniform polynomial time adversary A corrupting* Alice *and running Π_g with* abort *in a hybrid model with access to $\mathcal{F}^e_{\mathsf{offline}}$ and $\mathcal{F}_{ca}$, there exists a non-uniform polynomial time simulator S corrupting* Alice *and running in the ideal model with access to an ideal functionality $\mathcal{F}^f_{\Delta\text{-delayed-fairness}}$, such that for every $a,b,z\in\{0,1\}^*$ holds:*

$$\left\{\text{IDEAL}_{f,S(z)}(a,b,n)\right\}_{n\in\mathbb{N}}=\left\{\text{HYBRID}^{\mathcal{F}_{Resolve},\mathcal{F}_{\mathsf{ca}},\mathcal{F}_e}_{\Pi_f,A(z)}(a,b,n)\right\}_{n\in\mathbb{N}}$$

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 12 we construct a simulator S given a black-box access to A.

S generates $(dk,ek)\leftarrow\mathcal{K}(1^n)$ and selects a random key $K_S\in\{0,1\}^n$
S invokes A with input $b,\mathsf{ID_B},n$
When A sends (retrieve,resolve) for $\mathcal{F}_{\mathsf{ca}}$, S responds with (retrieve,resolve,ek)
S obtains A's inputs (b',K_B,ek') for the trusted party $\mathcal{F}^f_{\Delta\text{-delayed-fairness}}$
if $b'\neq b\vee ek'\neq ek$ **then**
  send $\perp$ to $\mathcal{F}^f_{\Delta\text{-delayed-fairness}}$
  send $\perp$ to A
  output whatever A outputs and halt
else
  S sends b to the trusted party computing $\mathcal{F}^f_{\Delta\text{-delayed-fairness}}$, and receives y_B
  encrypts y_B with K_B
  S chooses a random string $s_A\in\{0,1\}^n$, computes $c_A=\mathcal{E}_{K_S}(s_A,\mathcal{E}_{K_B}(y_B))$, and $c=\mathcal{NE}_{ek}(c_A,c_B)$ and hands the encrypted result $c_A||c$ to A
  When A sends c'_A, S checks **if** $c'_A=\mathcal{E}_{K_S}(s_A,\mathcal{E}_{K_B}(y_B))$ **then**
    decrypts and sends $\mathcal{E}_{K_B}(y_B)$ to A
  else
    send $\perp$ to trusted party
S outputs whatever A outputs.

Algorithm 12: The simulator S running in ideal model with trusted party computing $\mathcal{F}^f_{\Delta\text{-delayed-fairness}}$, and simulating the view of Bob.

The view of A in a simulation with S is identical to its view in an $(\mathcal{F}_{\mathsf{Resolve}},\mathcal{F}_{\mathsf{ca}},\mathcal{F}_e)$-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.

A.3 Security Analysis of Protocol $\Pi^G_{(v,g)}$ (Section 4)

Proof We analyse $\Pi^G_{(v,g)}$ in a $(\mathcal{F}^e_{\mathsf{offline}},\mathcal{F}_{\mathsf{ca}},\mathcal{F}_{\mathsf{tea}},\mathcal{F}^f_{\Delta\text{-delayed-fairness}})$-hybrid model, and show that the execution of $\Pi^G_{(v,g)}$ is computationally indistinguishable to computation of (v,g) in the ideal model with Guaranteed Output Delivery. We prove Claim 2 in Propositions 8 and 9 respectively.

Proposition 8 (Security Against Malicious Alice) *For every non-uniform polynomial time adversary A corrupting* Alice *and running $\Pi^G_{(v,g)}$ in a hybrid model with access to $\mathcal{F}^e_{\mathsf{offline}}$, $\mathcal{F}_{ca}$, $\mathcal{F}^f_{\Delta\text{-delayed-fairness}}$ and $\mathcal{F}_{tea}$, there exists a non-uniform polynomial time simulator S corrupting* Alice *and running in the ideal model with access to an ideal functionality $\mathcal{F}^{v,g}_{\Delta-guaranteed\text{-}computation}$, such that for every $a,b,z\in\{0,1\}^*$ holds:*

$$\left\{\text{IDEAL}_{f,S(z)}(a,b,n)\right\}_{n\in\mathbb{N}}=\left\{\text{HYBRID}^{\mathcal{F}_{offline},\mathcal{F}_{\mathsf{ca}},\mathcal{F}_{\mathsf{tea}},\mathcal{F}_{\Delta-delayed\text{-}fairness}}_{\Pi_f,A(z)}(a,b,n)\right\}_{n\in\mathbb{N}}$$

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 13 we construct a simulator S given a black-box access to A.

S generates $(dk, ek) \leftarrow \mathcal{K}(1^n)$ and selects a random key $K_S \in \{0,1\}^n$
S invokes A with input $a_1, a_2, \mathsf{ID_A}, n$
When A sends (retrieve, TEA) for $\mathcal{F}_{\mathsf{ca}}$, S responds with (retrieve, TEA, ek)
S obtains A's inputs (a_1', K_A, ek') for the trusted party $\mathcal{F}^{v,g}_{\Delta-\text{guaranteed-computation}}$
if $a_1' \neq a_1 \vee ek' \neq ek$ **then**
  send $\perp$ to $\mathcal{F}^{v,g}_{\Delta-\text{guaranteed-computation}}$
  send $\perp$ to A
  output whatever A outputs and halt
else
  S sends a_1 to the trusted party computing $\mathcal{F}^{v,g}_{\Delta-\text{guaranteed-computation}}$, and receives back y_A^1
  if $y_A^1 == \perp$ send $\perp$ to A and halt
  Otherwise S chooses a random string $s_B \in \{0,1\}^n$, computes $(\mathcal{NE}_{ek}(\mathcal{E}_{K_A}(s_B))||\mathcal{E}_{K_A}(y_A^1))$, and hands the result to A.
  Upon input a_2' from A: **if** $a_2' \neq a_2$ **then**
    send $\perp$ to $\mathcal{F}^{v,g}_{\Delta-\text{guaranteed-computation}}$
    send $\perp$ to A
    output whatever A outputs and halt
  else
    simulate $\mathcal{F}^{e}_{\text{offline}}$ for A according to steps in Algorithm 9
    Send a_2 to $\mathcal{F}^{v,g}_{\Delta-\text{guaranteed-computation}}$
    Upon input y_A^2 from $\mathcal{F}^{v,g}_{\Delta-\text{guaranteed-computation}}$, send $\mathcal{E}_{K_A}(y_A^2)$ to A
S outputs whatever A outputs.

Algorithm 13: The simulator S running in ideal model with trusted party computing $\mathcal{F}^{v,g}_{\Delta-\text{guaranteed-computation}}$, and simulating the view of Alice.

The view of A in a simulation with S is identical to its view in an $(\mathcal{F}^{e}_{\text{offline}}, \mathcal{F}_{\mathsf{ca}}, \mathcal{F}_{\mathsf{tea}}, \mathcal{F}^{f}_{\Delta\text{-delayed-fairness}})$-hybrid execution of $\Pi^{G}_{(v,g)}$ 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.

Proposition 9 (Security Against Malicious Bob) *For every non-uniform polynomial time adversary A corrupting* Bob *and running $\Pi^{G}_{(v,g)}$ in a hybrid model with access to $\mathcal{F}^{e}_{\text{offline}}$, $\mathcal{F}_{ca}$, $\mathcal{F}^{f}_{\Delta\text{-delayed-fairness}}$ and $\mathcal{F}_{tea}$ there exists a non-uniform polynomial time simulator S corrupting* Bob *and running in the ideal model with access to an ideal functionality $\mathcal{F}^{v,g}_{\Delta-guaranteed\text{-}computation}$, such that for every $a, b, z \in \{0,1\}^*$ holds:*

$$\left\{\text{IDEAL}_{f,S(z)}(a,b,n)\right\}_{n\in\mathbb{N}} = \left\{\text{HYBRID}^{\mathcal{F}_{offline},\mathcal{F}_{\mathsf{ca}},\mathcal{F}_{\mathsf{tea}},\mathcal{F}_{\Delta-delayed\text{-}fairness}}_{\Pi_f,A(z)}(a,b,n)\right\}_{n\in\mathbb{N}}$$

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 14 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 $(\mathcal{F}^{e}_{\text{offline}}, \mathcal{F}_{\mathsf{ca}}, \mathcal{F}_{\mathsf{tea}}, \mathcal{F}^{f}_{\Delta\text{-delayed-fairness}})$-hybrid execution of $\Pi^{G}_{(v,g)}$ with a honest Alice.

S generates $(dk, ek) \leftarrow \mathcal{K}(1^n)$ and selects a random key $K_S \in \{0,1\}^n$
S invokes A with input $b_1, b_2, \mathsf{ID_B}, n$
When A sends (retrieve, TEA) for $\mathcal{F}_{\mathsf{ca}}$, S responds with (retrieve, TEA, ek)
S obtains A's inputs (b_1', K_B, ek') for the trusted party $\mathcal{F}^{v,g}_{\Delta-\text{guaranteed-computation}}$
if $b_1' \neq b_1 \vee ek' \neq ek$ **then**
  send $\perp$ to $\mathcal{F}^{v,g}_{\Delta-\text{guaranteed-computation}}$
  send $\perp$ to A
  output whatever A outputs and halt
else
  S sends b_1 to the trusted party computing $\mathcal{F}^{v,g}_{\Delta-\text{guaranteed-computation}}$, and receives back y_B^1
  if $y_B^1 == \perp$ send $\perp$ to A and halt
  Otherwise S chooses a random string $s_A^1, s_A^2 \in \{0,1\}^n$, computes $(\mathcal{NE}_{ek}(\mathcal{E}_{K_B}(s_A^1))||\mathcal{E}_{K_B}(y_B^1))$, and hands the result to A.
  Upon input b_2' from A: **if** $b_2' \neq b_2$ **then**
    send $\perp$ to $\mathcal{F}^{v,g}_{\Delta-\text{guaranteed-computation}}$
    send $\perp$ to A
    output whatever A outputs and halt
  else
    simulate $\mathcal{F}^{e}_{\text{offline}}$ for A according to steps in Algorithm 10
    Send b_2 to $\mathcal{F}^{v,g}_{\Delta-\text{guaranteed-computation}}$
    Upon input y_B^2 from $\mathcal{F}^{v,g}_{\Delta-\text{guaranteed-computation}}$, send $\mathcal{E}_{K_A}(s_A^2)||\sigma_A||y_B^2$ to A
    When A sends $\mathcal{E}_{K_A}(s_A^2)||\sigma_A$, check that the authentication is valid and that s_A^2 is correct
    if not, send $\perp$ to $\mathcal{F}^{v,g}_{\Delta-\text{guaranteed-computation}}$
S outputs whatever A outputs.

Algorithm 14: The simulator S running in ideal model with trusted party computing $\mathcal{F}^{v,g}_{\Delta-\text{guaranteed-computation}}$, and simulating the view of Bob.