Secure *Committed* Computation

Amir Herzberg* and Haya Shulman†

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

August 18, 2010

Abstract

We introduce *secure committed computation*, where n parties commit in advance to compute a function over their private inputs; we focus on two party computations ($n = 2$). In committed computation, parties initially commit to the computation 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 commitments requires a *trusted enforcement authority (*TEA*)*; however, the protocol protects confidentiality even from the TEA. Secure committed 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 commitment can be either symmetric (both parties commit) or asymmetric (e.g., only a server commits to a client). Symmetric commitment should also be *fair*, i.e., one party cannot obtain commitment by the other party without committing as well. Our secure committed computation protocols are *optimistic*, i.e., the TEA is involved only if and when a party fails to participate (correctly).

The protocols we present use two new building blocks, which may be of independent interest. The first is a protocol for *optimistic fair secure computation*, which is simpler and more efficient than previously known. The second is a protocol for *two party computation secure against malicious participants*, which is simple and efficient, and relies on a weakly-trusted third party. This protocol can be useful where a trusted third party is unavoidable, e.g., in secure committed or fair computation protocols.

Keywords: Two-party computation, trusted third party, 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 *committed 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 [32] and Goldreich et al. [17], investigates how to securely compute functionalities over inputs from multiple players, under different circumstances and in the presence of different adversaries. Such computation can trivially be done securely by a trusted third party; the goal of secure computation is to achieve the same security impact without a trusted party, running only a protocol between the parties. However, as shown in [10], two-party protocols cannot achieve fairness for general computation without an honest majority. Our focus is on the problem of delivery failures, aka *abort attacks*, and fairness. Namely, what should the result of a secure computation be, when a party fails to deliver ('valid') input?

Committed network services focus on the problem of delivery failures. As network services become more and more important, failure to provide services can become a serious concern. Many works (and systems) address the basic concerns of unintentional failures and congestion, as well as (intentional) denial-of-service attacks. A more difficult issue is the *intentional delivery failures* by one of the parties, e.g., 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

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

price; and suppose the customer sends an order to execute the option, close to its expiration time. If the broker fails to process the order, this could cause significant loss to the customer (and illegitimate gain to the broker).

Secure *committed* computation, introduced in Section 5, provides an interesting variant, where parties have an *incentive* to complete the protocol. This incentive is achieved, by running the computation in two phases. In the first, commitment, phase, a party commits to participate in the second, execution, phase, by inputing some secret value whose exposure would penalise that party (and compensate the other party); in the second phase, if a party does not participate correctly, then the protocol exposes its commitment from the first phase. To provide compensation we involve an additional, weakly trusted, participant, which we call the TEA (Trusted Enforcement Authority), who does not provide inputs, but helps to identify and penalise faulty participants. Specifically, in the commitment phase, the parties agree on the terms and send to the TEA a pre-agreed *compensation*, e.g., as a signed payment order. Later, in the execution phase, either the service is performed correctly, or if the other participant fails to deliver the agreed upon service or content, e.g., digitally signed payment, (either due to early abort, i.e., if it is malicious, or due to communication failures), the TEA compensates the honest party. The compensation is based on the inputs sent to the TEA in advance by both parties. This provides an incentive for the parties to complete protocol execution correctly, i.e., to provide the services that they committed to provide. The solution can extend secure computation protocol to be used as a building block in design of more complex incentive-based protocols, ensuring security goals which involve rational adversaries.

In a naive implementation, the TEA is aware of the terms of the service, as well as of the inputs provided by the parties and of the compensation. This limits the use of sensitive transactions to situations where a fully trusted intermediary is available, or where the service provider or peer is sufficiently trusted. As a result, the potential of the Internet, to allow arbitrary parties to perform commerce, with automated, trustworthy dispute-resolution and compensation mechanisms, is only partially used. We believe that in this paper, we make a significant, if yet initial, step towards this goal.

Our focus is on minimising the exposure of the private inputs and outputs. Specifically, we use secure computation techniques to ensure that the TEA is *oblivious* to the terms, inputs and compensation. Namely, the TEA engages in secure computation with the parties, and as a result either the service is provided correctly, or compensation is given, while the TEA is unaware of the inputs and outputs of the process, which contain sensitive data of the parties.

The definition of correct service is trivial, e.g., when the service is exchange of well defined signed documents, such as contracts, or payment orders (e.g., in different currencies). This case is related to existing works on certified mail, non-repudiation (evidences) and on fair exchange (esp. of signed documents), see [1, 2]. Note, however, that these works do not ensure *confidentiality* against the TEA (TTP). Yet confidentiality can be very important, e.g., the exposure of (future) trading positions can allow entities to react, possibly harming the customer.

Often, the correct service is more complex, and may involve computation based on inputs from both parties. For example, a customer sends a complex order involving multiple stocks, and the broker has to provide updated, valid (signed) quotas, and if there is a match then the result will be specific buy and/or sell transactions.

The discussion above focused on the case of asymmetric commitment: only Bob commits to Alice. We also present a protocol, in Section 5.2, for symmetric commitment, 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 commitment if and only if Alice receives Bob's commitment.

Our secure committed computation protocols make 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 4.2, which we use as a module in our committed fair secure computation protocol to ensure that the commitment process is *fair*, i.e., one party cannot obtain commitment by other party without committing 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 also 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 weakly-trusted 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 [7], however, our protocol is significantly more efficient (also, the protocol in [7] was not proven secure yet).

1.1 Related Work

There are many works, beginning with Yao [32], 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 a 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 [7, 25, 22, 9].

In malicious model, an adversary can always abort after receiving its output and before the honest party receives output. Cleve [10] 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 [6, 3, 11, 5, 19, 12, 30, 14, 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. In order to release the output gradually many rounds of interaction are required, which may render this approach impractical for realistic applications.

A second approach is to provide only a relaxed notion of fairness. In particular, in [23] Lindell presented *legally enforceable fair* secure two-party computation, where 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. Our results support 'real' fair computation.

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. Optimistic protocols were mostly proposed for specific tasks, esp. fair exchange [1, 2, 27]. Cachin and Camenisch, [7], presented optimistic fair secure computation protocol, with constant number of interactions. Our protocols essentially improve over this earlier work, in efficiency, see comparison in Section 4.2.1, provable security, and most notably, by allowing commitment to the computation.

Organisation and Contributions

In Section 2 we present preliminaries (model, notations, building blocks) and an outline of our results. In Section 3, we present a basic building block used by our protocols: an efficient, practical protocol to securely compute any two-party functionality, using a trusted third party, but limiting the involvement of the third party to preprocessing prior to receiving inputs, i.e., off-line. We also mention how the same goal can be achieved using a group of 'third parties', if their majority is honest, or using a secure two-party computation protocol; these solutions would be less efficient, of course. In Section 4 we present an optimistic fair secure computation protocol. The resulting protocol is practical - simple, efficient and *optimistic*, i.e., it makes use of a Trusted Third Party only when faults occur. It improves on the known optimistic secure computation protocol of [7] in efficiency and security[1].In Section 5 we define ideal functionality for committed fair secure computation, and present a protocol realising it.

2 Preliminaries and Overview

This section provides a high level overview of the constructions, presents the model along with cryptographic assumptions and notations.

Model

We prove security of our protocols in the universal composability framework, which ensures that security of the protocols is maintained under a composition with arbitrary other protocols in the system, see [8] for more details. 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 corrupt 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 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. Let Δ_C be a bound on the channel communication delay, then $\zeta(\Delta_C)$ (for some function ζ) is the maximal waiting time. For instance, after sending a

[1]Security is not proven in [7], in fact, their protocol appears amenable to the 'corrupt encoding of 0 value' attack, where a party holding the encodings of bits, w.l.o.g., corrupts just the encoding of the 0 value of an input bit, to detect the bit its peer has provided in an input to the oblivious transfer protocol.

message to Alice, Bob has to wait $\zeta(\Delta_C) = 2\Delta_C$, for his message to reach Alice and for Alice's response to arrive to him. We assume faulty channels between Alice and Bob and that messages that the parties exchange may be lost or delayed by at most a factor of $\zeta(\Delta_C)$. We 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.

Notations and Building Blocks

We use the following cryptographic schemes as building blocks for our protocols:
In all our constructions we use an authenticated encryption scheme $(\mathcal{K}, \mathcal{E}, \mathcal{D})$ to ensure confidentiality and integrity of the inputs and outputs of the participants. For ease of exposition, we consider the message authentication code (MAC) key and the secret encryption key as one key K comprised of K_1 for authentication and K_2 for encryption, e.g., see authenticated encryption in [4]; an alternative implementation can be based on a one-time pad encryption with information theoretic MAC, see [25]. When applying $\mathcal{E}_{K_P}(x)$ we perform an authenticated encryption of input x using the key K_P of party P. In the implementation of the resolver we use a non-malleable encryption (see [13] for details) $(\mathcal{NG}, \mathcal{NE}, \mathcal{ND})$ to ensure fairness. We also use a signature scheme $(\mathcal{G}, \mathcal{E}, \mathcal{D})$, to ensure integrity: in Section 3.1, Algorithm 1, Section 4.2, Algorithm 4, Section 5.2, Algorithm 8. When validating authenticated inputs, we use $\perp$ to denote authentication failure. In subsequent sections we use ideal functionality $\mathcal{F}^2_{\mathsf{ot}}$ (a functionality implementing a two-party (1-2) oblivious transfer protocol), and $\mathcal{F}_{\mathsf{ca}}$ (representing certification authority). We use parameters n and m (in Section 5) to define the inputs' length to functions throughout the work. The length parameters may differ depending on the definition of the function at hand.

Outline of the Results and Techniques

In this section we provide a high level overview of our protocols for two-party computation, and the techniques that underly their construction. In Section 3 we present a protocol with output at one party only, secure against malicious adversaries. The protocol relies on a weakly trusted third party $\mathcal{F}^e_{\mathsf{offline}}$, that generates the garbled circuit during the preprocessing phase; the construction ensures integrity and confidentiality. The garbled circuit is then used for the evaluation of the function during the execution phase. Next, in Section 4 define a Δ-delayed fairness where a malicious party can delay the output of the honest party by at most a factor of Δ. We then construct a protocol with output at both parties, using any two-party protocol secure against malicious adversaries with output at one party. The resulting fair protocol involves a resolver $\mathcal{F}_{\mathsf{Resolve}}$ only in case one of the parties misbehaves, or in case of faults. The resolver is an oblivious and optimistic, and performs the resolution without learning the private inputs or outputs of the participants. Note that we assume that the resolver is trusted to perform its functionality correctly, and to restore fairness in case of malicious behaviour. However, even if the resolver is malicious, and deviates from the protocol or colludes with one of the parties, it can only breach fairness, but confidentiality and integrity of the inputs and the corresponding outputs of the parties are ensured, and the resolver cannot make the honest party accept an incorrect input or output; this is a direct implication of the fact that the resolver is oblivious, and its view is comprised of the private inputs and outputs of the parties encrypted and authenticated with their respective secret keys. Eventually, in Section 5, we present the notion of guaranteed output delivery, that ensures that a malicious party will compensate honest party in case of malicious behaviour or faults using a $\mathcal{F}_{\mathsf{tea}}$ (trusted enforcement authority). This is accomplished by having the parties commit to participate in protocol execution, and the commitment is executed in case of failures. If the $\mathcal{F}_{\mathsf{tea}}$ is malicious, it will not be able to learn the inputs or the outputs of the parties and will not be able to generate an incorrect result without the parties detecting this.

3 Secure Two-Party Computation in Malicious Setting

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), see e.g., [24, 25], for standard definitions of two-party computation. 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.

Secure function evaluation based on garbled circuits, see [32], allows to perform such a computation in a secure manner, i.e., ensuring privacy, correctness and inputs independence (see proof in [24]). 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 secret inputs 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, e.g., one that computes a different function which may expose the input of Bob. Although, any two-party protocol can be securely computed in the malicious setting, e.g., see [17, 15], they are inefficient for practical purposes, and a series of works [28, 31, 25, 29] 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. We take an alternative approach, and attempt to improve efficiency by using an additional offline third party, with reduced trust, i.e., it does not learn anything about the inputs of the participants or of the result of the computation.

3.1 Offline Functionality $\mathcal{F}^e_{\mathsf{offline}}$

An Offline Party functionality $\mathcal{F}^e_{\mathsf{offline}}$, in Algorithm 1, represents an offline third party. The Offline Party is used during the preprocessing phase to ensure privacy and correctness against malicious Alice. The functionality $\mathcal{F}^e_{\mathsf{offline}}$ runs with two security parameters n and s (presented below), and is parametrised by a function $e : \{0,1\}^n \times \{0,1\}^n \rightarrow \{0,1\}^n$. Upon request, $\mathcal{F}^e_{\mathsf{offline}}$ generates a garbled circuit that computes e. Specifically, $\mathcal{F}^e_{\mathsf{offline}}$ receives the $\mathsf{ID_A}, \mathsf{ID_B}$ from Alice (the originator) and Bob (the circuit evaluator) respectively, and generates a 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 [25] for details of this threat and defense mechanism. 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 fact that a trusted party generates the circuit ensures that the garbled circuit computes the correct function. We use an ideal functionality computing the function in Algorithm 2.

Input: $\mathsf{ID_A}$ from Alice, or $\mathsf{ID_B}$ from Bob, security parameters n, s
Registration Phase
- generate signature key-pair: $(vk_T, sk_T) \leftarrow \mathcal{G}(1^n)$
- register the verification key: (register, offline, vk_T) to $\mathcal{F}_{\mathsf{ca}}$

end
Computation Phase
1. Construct a circuit C 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: $\bar{\mathcal{K}_A} = ((\mathcal{K}^0_A[0], \mathcal{K}^1_A[0]), ..., (\mathcal{K}^0_A[n], \mathcal{K}^1_A[n]))$
   b. Random strings corresponding to all possible input bits of Bob: $\bar{\mathcal{K}_B} = ((\mathcal{K}^0_B[0], \mathcal{K}^1_B[0]), ..., (\mathcal{K}^0_B[sn], \mathcal{K}^1_B[sn]))$
   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: $\bar{\sigma}_A = ((\sigma^0_A[0], \sigma^1_A[0]), ..., (\sigma^0_A[n], \sigma^1_A[n]))$ where $\forall i, j,\ \sigma^j_A[i] = \mathcal{S}_{sk_T}(\mathcal{K}^j_A[i], i)$
5. Sign the random input strings $\bar{\mathcal{K}_B}$ of Bob: $\bar{\sigma}_B = ((\sigma^0_B[0], \sigma^1_B[0]), ..., (\sigma^0_B[sn], \sigma^1_B[sn]))$ where $\forall i, j,\ \sigma^j_B[i] = \mathcal{S}_{sk_T}(\mathcal{K}^j_B[i], i, j)$

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 1: 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.

3.2 Secure Two-Party Protocol Against Malicious Adversaries

We next present an implementation, in Algorithm 2, of Yao's protocol using an offline third party 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 [23, 20, 21] for standard definition of security with abort). During the preprocessing phase $\mathcal{F}^e_{\mathsf{offline}}$ is used to generate the garbled circuit, and sends the signed random strings to Alice and garbled tables along with output decryption tables to Bob. This phase ensures that the circuit was correctly constructed and prevents cheating by either party, essentially replacing the computation and communication overhead, which are required against malicious adversaries, with a weakly trusted third party. 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.

Input: security parameters n, s
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 implementation in Algorithm 8)
  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: $((\mathcal{K}_A^{a[0]}[0], \sigma_A^{a[0]}[0]), ..., (\mathcal{K}_A^{a[n]}[n], \sigma_A^{a[n]}[n]))$, $(\forall i), \mathcal{K}_A^{a[i]}[i] \in \bar{\mathcal{K}}_A, \sigma_A^{a[i]}[i] \in \bar{\sigma}_A$
  Bob:
    send (retrieve, offline) to $\mathcal{F}_{\mathsf{ca}}$ and obtain vk_T
    if $\exists (\mathcal{K}_A^{a[i]}[i], \sigma_A^{a[i]}[i])$, $s.t.$, $\mathcal{V}_{vk_T}(\mathcal{K}_A^{a[i]}[i], i, \sigma_A^{a[i]}[i]) =$ 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 $\mathcal{V}_{vk_T}(\mathcal{K}_B^{b'[i]}[i], \sigma_B^{b'[i]}[i]) ==$ false **then**
        output $\perp$ and halt
    $(y_B = (y_B[0], ..., y_B[n])) \leftarrow \mathcal{C}(\bar{\mathcal{K}_\mathcal{A}}, \bar{\mathcal{K}_\mathcal{B}})$
    (see implementation in Algorithm 8)
  end
end

Algorithm 2: 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 $e : \{0,1\}^n \times \{0,1\}^n \rightarrow \{0,1\}^n$.

Claim 1 *Let $e : \{0,1\}^n \times \{0,1\}^n \rightarrow \{0,1\}^n$ be a polynomial time two-party functionality. Assume that the signature scheme $(\mathcal{G}, \mathcal{S}, \mathcal{V})$ is existentially unforgeable under 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 $(\mathcal{F}^e_{\mathsf{offline}}, \mathcal{F}^2_{\mathsf{ot}}, \mathcal{F}_{\mathsf{ca}})$-hybrid model with* abort.

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

3.2.1 Efficiency Analysis and Comparison

Classical way, see [16], of making two-party protocols secure against malicious adversaries, is based on running a zero-knowledge protocol, see [17, 18], which renders them inefficient for practical purposes. In [28] 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 $\mathcal{O}(s|C|s + sn^2)$ (where n is the number of input bits to the circuit C and s is the statistical security parameter). Then [31] improved the communication complexity of [28] to $\mathcal{O}(s|C|)$ using expanders. However as [25] observed the protocol in [28] is susceptible to 'input corruption' attack (see [25]) for details); [25] also present a protocol with roughly the same communication complexity as [28], of $\mathcal{O}(s|C| + s^2n)$ (this protocol was also implemented in [26]). Another improvement to two-party computation in malicious setting was made by [] 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 [25]) and computational complexity of $\mathcal{O}(|C|)$ (as opposed to $\mathcal{O}(n)$ in [25]). Subsequently, the work of [29], also followed the cut-and choose approach and improved the complexity to $\mathcal{O}(\frac{s|C|}{log(|C|)})$.

Our protocol, in Algorithm 2, 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). Specifically during the (offline) preprocessing the $\mathcal{F}^{e}_{\mathsf{offline}}$ sends the corresponding random strings and tables of the circuit to Alice and Bob, 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 by Bob. Thus the resulting protocol is of similar computational and communication complexity as the construction of Yao's garbled circuit, [32, 24]. 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 [25], which due to the cut-and-choose incur a multiplicative increase by a factor of s (the second security parameter) and results in communication complexity of $\mathcal{O}(s|C| + s^2 n)$.

4 Fair Two-Party Protocol Against Malicious Adversaries

In Section 3, we considered scenario where only one party receives the output. Yet in many applications it is desirable to allow for output at both participants. In this case, an additional property of fairness is required. Specifically, Alice receives her result if and only if Bob receives his, or no party receives the output. Fairness is trivial to achieve in honest or semi-honest setting. However, this is not so when considering malicious adversaries that may arbitrarily deviate from the protocol.

In this section, in Algorithm 5, we present an optimistic weakly trusted (oblivious) third party, involved only for resolution in case one of the parties misbehaves. We believe that the model based on the separation between the functionality 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. Specifically, the (offline) third party 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. Neither party of the third parties learns anything about the inputs or the result of the computation. In Algorithm 3 we present the notion of Δ-delayed fairness, where a corrupt party may delay the output of the honest party by at most a factor of Δ. We then construct a protocol Π^F_f, that computes functionality $f : \{0,1\}^n \times \{0,1\}^n \to \{0,1\}^n \times \{0,1\}^n$, 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. To construct Π^F_f we use the protocol Π^E_e, in Section 3, that allows to compute securely any functionality $e : \{0,1\}^n \times \{0,1\}^n \to \{0,1\}^n$ with output only at Bob. Let $e(a,b) = y$, then construct f as follows: $f_{ek_R}(a||K_A, b||K_B) = \{\mathcal{E}_{K_A}(e(a,b), \mathcal{E}_{K_B}(e(a,b))), \mathcal{NE}_{ek_R}(\mathcal{E}_{K_A}(e(a,b)), \mathcal{E}_{K_B}(e(a,b)))\}$.

4.1 Δ-delayed fairness

In the Δ-delayed fairness model in Algorithm 3, either both parties receive the output or no one does. Alice receives her output first, and should send to Bob his output (encrypted with his secret key). The delay Δ is the maximal time till Bob obtains his part of the output. If Bob does not receive the output from Alice after $2\Delta_C$ (the maximal delay on the channel from himself to Alice, and then from Alice back to him), he contacts the resolver $\mathcal{F}_{\mathsf{Resolve}}$ and obtains the result (this takes another $2\Delta_C$). In the worst case, after at most $4\Delta_C$ Bob obtains his part of the output. Since Alice receives her output first, Bob cannot breach fairness, thus fairness should be ensured w.r.t. malicious Alice. Malicious Bob can either abort without obtaining the result (in which case neither does Alice), or may contact the $\mathcal{F}_{\mathsf{Resolve}}$ (in which case Alice also receives her output). Thus fairness is preserved.

4.2 Fair Two-Party ($\mathcal{F}_{\mathsf{Resolve}}$,$\mathcal{F}_{\mathsf{ca}}$)-Hybrid Protocol Π^F_f

We use the protocol in Algorithm 2 to evaluate a family of functions $\mathbb{E} = \{e_{pk}\}_{pk \in \mathcal{G}(1^n)}$, i.e., functions defined by a public encryption key. Let $(dk_R, ek_R) \leftarrow \mathcal{G}(1^n)$ be the key pair of the resolver $\mathcal{F}_{\mathsf{Resolve}}$ (see Algorithm 5). We take the function e for Π^E_e (that provides input at Bob only) to be the function computing the following: $e_{ek_R}((a, K_A), (b, K_B)) = \mathcal{E}_{ek_R}(c_A||c_B)||(\mathcal{E}_{K_A}(f_A(a,b), c_B))$, where $c_A = \mathcal{E}_{K_A}(f_A(a,b))$ and $c_B = \mathcal{E}_{K_B}(f_B(a,b))$.

In Algorithm 4, we construct the protocol Π^F_f using protocol Π^E_e. Alice and Bob retrieve the public encryption key ek_R of the resolver $\mathcal{F}_{\mathsf{Resolve}}$, and use a symmetric authenticated encryption scheme $(\mathcal{K}, \mathcal{E}, \mathcal{D})$ with secret keys K_A and K_B respectively. Alice and Bob run protocol Π^E_e and provide their inputs $(a||K_A)$ and $(b||K_B)$ respectively.

Input: n, Δ_C, Δ
a from Alice, b from Bob
Computation Phase
  if $a == \perp \lor b == \perp$ **then**
    send $\perp$ to Alice and to Bob, and halt
  else
    send $y_A = f_A(a,b)$ to Alice
    sleep('wait for response', $2\Delta_C$)
    onReceive(fair)
      stopTimer('wait for response')
      send($y_B = f_B(a,b)$) to Bob
    onWakeup('wait for response')
      sleep($\Delta - 2\Delta_C$)
      send($y_B = f_B(a,b)$) to Bob
end

Algorithm 3: The ideal functionality $\mathcal{F}_{\Delta\text{-delayed-fairness}}$ 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.

The inputs consist of their private inputs, and their respective secret encryption keys. The protocol evaluates the function on 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 $\mathcal{F}_{\text{Resolve}}$ (containing both the output of Bob and of Alice). Since Bob performed the computation, he is assured that the output is constructed correctly. Bob sends the output (without the part of the resolver) to Alice. If Alice misbehaves, Bob runs contacts the resolver with the part of the output encrypted with the key of the resolver. The $\mathcal{F}_{\text{Resolve}}$ validates, decrypts and sends to Alice her output, and to Bob his (restoring fairness). The resolver uses a non-malleable encryption scheme ($\mathcal{NG},\mathcal{NE},\mathcal{ND}$) (see [13] for details), which encrypts the part of the output of the resolver (with the encryption key ek_R), which is essential to ensure that its part of the output cannot be maliciously changed in a meaningful way. This part of the output is sent to resolver by Bob in case Alice misbehaves.

Upon receipt of an output from Bob, Alice decrypts and obtains her part of the output and Bob's output encrypted with his secret key, and send this part to Bob. Bob obtains and decrypts his part of the output, which concludes the protocol. Thus fairness is ensured.

Claim 2 *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^F securely realises $\mathcal{F}_{\Delta\text{-delayed-fairness}}$ in the presence of malicious static adversaries in the ($\mathcal{F}_{\text{Resolve}}$,$\mathcal{F}_{\text{ca}}$,$\mathcal{F}_e$)-hybrid model with Δ-delayed fairness.*

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

4.2.1 Efficiency Analysis and Comparison

There are two central approaches to fairness, the gradual release of secrets and the optimistic model. The number of rounds in a two-party protocol in [30] (that ensures fairness by gradually releasing the secrets) is high and proportional to the security parameter. The high communication complexity is required even in case the parties are honest. In [7], the authors designed an efficient optimistic fair protocol using proofs of knowledge. In contrast to [30] 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. However, the protocol of [7] 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, their paper lacks a full proof of security. To date we are not aware of other works on optimistic fair secure computation, that provide proofs of security and reasonable efficiency.

In our protocol, when the parties are honest and follow the steps of the protocol, 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 input to the resolver and top receive a decrypted response back. The analysis and comparison of the initial steps are the same as in Section 3.2.1.

Input: security parameters n, s, maximal communication delay Δ_C, maximal fairness delay Δ
$a = [a_i]_{i=1}^n$ from Alice
$b = [b_i]_{i=1}^n$ from Bob
Output: $y = (y_A, y_B)$
Computation Phase
Alice and Bob do:
send (retrieve, resolve) to $\mathcal{F}_{\mathsf{ca}}$ and obtain ek_R (each)
generate secret keys K_A and K_B respectively
run a protocol $\Pi^E_{e_{ek_R}}(a||K_A, b||K_B)$ (in Algorithm 2)
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 4: Secure Two Party Protocol Π^F_f 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 $f : \{0,1\}^n \times \{0,1\}^n \to \{0,1\}^n \times \{0,1\}^n$.

Input: n
generate encryption key-pair: $(ek_R, dk_R) \leftarrow \mathcal{K}(1^n)$
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 5: The ideal functionality $\mathcal{F}_{\mathsf{Resolve}}$

5 Committed Two-Party Computation

In Section 4 we constructed a protocol that achieves fairness in two party computation, i.e., either both receive the result of the computation or no one does. However, fairness alone may not suffice for some applications. Specifically, a participant may decide to abort the protocol, not provide an input to the protocol or provide an invalid input. Such an outcome may not be plausible in many applications, e.g., online market. In addition, parties often agree to participate in some computation in advance, possibly before they have inputs to that computation, by exchanging each others commitments, e.g., by signing a contract together. The commitment phase should ensure fairness and prevent a malicious party from aborting after it receives its commitment, if the honest party has not received a commitment. In addition, the commitments should be validated to prevent the malicious party from providing an invalid commitment, e.g., one that expired. To encompass these requirements we introduce the $\mathcal{F}_{\mathsf{tea}}$ (trusted enforcement authority), that is used to compensate the honest party in case of failure to participate by the other.

For applications based on the client-server architecture, it suffices to ensure one sided, asymmetric, commitment, since most Internet transactions are asymmetric. In this section we focus on symmetric commitments, where both parties commit to participate in the protocol. We present the symmetric commitment functionality, ensuring guaranteed output delivery, defined in Algorithm 6, and then construct a protocol, in Algorithm 8.

During the second, computation phase, the protocol relies on the TEA, in Algorithm 7, to restore guaranteed output property of misbehaviour.

5.1 Committed Two-Party Computation Functionality $\mathcal{F}^{v,g}_{\textsf{commited-computation}}$

The committed two-party computation functionality $\mathcal{F}^{v,g}_{\textsf{commited-computation}}$, in Algorithm 6, consists of two-phases: during the first phase the parties commit to participate in protocol execution, and during the second phase, they evaluate a function over their inputs. Both the commitments and the inputs are validated by the functionality. This functionality is reactive, i.e., parties can adaptively choose their inputs to the second phase, based on the output from the commitment phase. During the commitment phase, both Alice and Bob provide their inputs a_1 and b_1, respectively, to validation function v_1 that validates the inputs. If inputs are valid both parties receive each others' commitments, i.e., $v_1(a_1)$ and $v_1(b_1)$ respectively, and can participate in the second phase, i.e., the evaluation of agreed upon function g. During the computation 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; and 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 commitment of the other party) and if valid, recovers the commitment and grants it to the party which sent the input.

When functionality receives both inputs, the input of each party is validated against the commitment of the other party, and the time that both inputs were received. In case one of the inputs is invalid, the party with the valid input is compensated. Otherwise, when both inputs are valid, the functionality evaluates the function g on the inputs, 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 $2\Delta_C$.

5.2 Two-Party ($\mathcal{F}^e_{\textsf{offline}}$,$\mathcal{F}_{\textsf{ca}}$,$\mathcal{F}_{\textsf{tea}}$)-Hybrid Protocol $\Pi^G_{(v,g)}$

Committed two-party computation, in Algorithm 8, is a two-phase protocol, s.t., during the first phase the parties commit to participate in protocol execution, and in second phase, they evaluate a function over their inputs. Both the commitments and the inputs are validated using validations functions v_1 and v_2 for first and second phases respectively. If the commitment of one of the parties is not valid, the execution is terminated. Once the commitment phase completed, the parties can engage in computation of the second phase. At this stage each party holds the commitment by the other, and can contact the trusted enforcement authority functionality $\mathcal{F}_{\textsf{tea}}$ (in Algorithm 7) in case a malicious party fails to participate, and provide an input, or provides an incorrect input to the computation. The $\mathcal{F}_{\textsf{tea}}$ attempts to complete the protocol with the other party on behalf of the party originating resolution. In case of failure, the $\mathcal{F}_{\textsf{tea}}$ opens the commitment and sends it to the originating party. Otherwise, it concludes the protocol, and returns the result of the computation to the originating party. Let Π^F_f (Algorithm 4) be a protocol that computes $f : \{0,1\}^n \times \{0,1\}^n \to \{0,1\}^n \times \{0,1\}^n$, and allows for outputs at both parties, while ensuring fairness. We use it to construct a protocol $\Pi^G_{(g,v)}$ that computes function $g, v = (v_1, v_2)$ with output at both parties and ensures Guaranteed Output Delivery. The $\mathcal{F}_{\textsf{tea}}$ uses a non-malleable encryption scheme $(\mathcal{NG}, \mathcal{NE}, \mathcal{ND})$, and Alice and Bob use an authenticated encryption scheme $(\mathcal{K}, \mathcal{E}, \mathcal{D})$.

Claim 3 *Let $G : \{0,1\}^* \times \{0,1\}^* \to \{0,1\}^* \times \{0,1\}^*$ be a polynomial two-party functionality, and let $(\mathcal{G}, \mathcal{E}, \mathcal{D})$ be a secure shared key encryption scheme, and $(\mathcal{NG}, \mathcal{NE}, \mathcal{DE})$ be a non-malleable encryption scheme. Then protocol $\Pi^G_{(v,g)}$ securely realises $\mathcal{F}^{v,g}_{\textsf{commited-computation}}$ in the presence of malicious static adversaries in the ($\mathcal{F}^e_{\textsf{offline}}$,$\mathcal{F}_{\textsf{ca}}$,$\mathcal{F}_{\textsf{tea}}$)-hybrid model with* Guaranteed Output Delivery.

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

Input: n, maximal channel delay Δ_C, fairness delay Δ
Commitment Phase
  Input: a_1 from Alice, b_1 from Bob
  $V_1(a_1, b_1) == (y_A^1, y_B^1)$
  if $y_A^1 == \perp \vee y_B^1 == \perp$ **then**
    send $\perp$ to Alice and Bob and halt
  else
    send y_A^1, y_B^1 to Alice and Bob respectively
end
Computation Phase

  onReceive(a_2) from Alice
    $t_A \leftarrow getTime()$
    sleep('wait for input from Bob')
    onWakeup('wait for input from Bob')
      if $(V_2(a_2, y_A^1, t_A, getTime()) \neq \perp)$ **then**
      send (b_1) to Alice

  onReceive(b_2) from Bob
    $t_B \leftarrow getTime()$
    sleep('wait for input from Alice')
    onWakeup('wait for input from Alice')
      if $(V_2(b_2, y_B^1, t_A, getTime()) \neq \perp)$ **then**
      send (a_1) to Bob

  if $V_2(a_2, y_A^1, t_A, t_B) == \perp$ **then**
    send a_1 to Bob and halt
  if $V_2(b_2, y_B^1, t_A, t_B) == \perp$ **then**
    send b_1 to Alice and halt
  $(y_A^2, y_B^2) \leftarrow g(a_2, b_2)$
  send y_B^2 to Bob
  sleep('wait for response', $2\Delta_C$)
  onReceive(fair)
    stopTimer('wait for response')
    send(y_A^2) to Alice
  onWakeup('wait for response')
    sleep($\Delta - 2\Delta_C$)
    send(y_A^2) to Alice
end

Algorithm 6: The ideal functionality $\mathcal{F}^{v,g}_{\text{commited-computation}}$ for computing (v, g) with guaranteed output delivery, runs with Alice and Bob, and an adversary S, where $v = (v_1, v_2)$ is inputs validation function used at each phase.

References

[1] 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.

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

[3] Donald Beaver and Shafi Goldwasser. Multiparty computation with faulty majority. In G. Brassard, editor, *Proc. CRYPTO 89*, pages 589–590. Springer-Verlag, 1990. Lecture Notes in Computer Science No. 435.

[4] M. Bellare and C. Namprempre. Authenticated Encryption: Relations among Notions and Analysis of the Generic Composition Paradigm. *LECTURE NOTES IN COMPUTER SCIENCE*, pages 531–545, 2000.

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

[6] Ernest F. Brickell, David Chaum, Ivan 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.

[7] Christian Cachin and Jan 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.

[8] 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.

Input: n, s, Δ_C, Δ

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_T
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($((\bar{\mathcal{K}}^a_A, \bar{\sigma}^a_A), (\bar{\mathcal{K}_B}, \bar{\sigma}), \mathcal{NE}_{ek_T}(\mathcal{E}_{K_A}(b_1)))$)
  if $\forall i : (\mathcal{V}_{vk_T}(\mathcal{K}^{a[i]}_A[i], \sigma^{a[i]}_A[i]) \neq \perp) \vee (\mathcal{V}_{vk_T}(\mathcal{K}^{b[i]}_B[i], \sigma^{b[i]}_B[i]) \neq \perp)$ **then**
    send $(\bar{\mathcal{K}}^a_A, \bar{\sigma}^a_A)$ to Bob
    for $i \leftarrow 1$ ***to*** $n \cdot s$ **do**
      run $\mathcal{F}^2_{\mathsf{ot}}((\mathcal{K}^0_B[i], \sigma^0_B[i]), (\mathcal{K}^1_B[i], \sigma^1_B[i]), b_i)$ with Bob, Bob receives $(\mathcal{K}^{b[i]}_B[i], \sigma^{b[i]}_B[i])$
    sleep('time to Bob',$2\Delta_C$)
    onReceive(c_A)
      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 7: The ideal functionality $\mathcal{F}_{\mathsf{tea}}$

[9] 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.

[10] 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.

[11] 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.

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

[13] 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.

[14] J. Garay, P. MacKenzie, M. Prabhakaran, and K. Yang. Resource Fairness and Composability of Cryptographic Protocols. *LECTURE NOTES IN COMPUTER SCIENCE*, 3876:404, 2006.

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

[16] O. Goldreich. Foundations of Cryptography: Volume 2, Basic Applications. *Basic Applications*, 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.

Input: security params n, s, maximal communication delay Δ_C, maximal fairness delay Δ

Commitment 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

generate secret keys K_A and K_B respectively

run $\Pi^F_{v_1}((a_1, K_A, ek_T), (b_1, K_B, ek_T))$ (in Algorithm 4), to generate and validate commitments

Bob receives $(\mathcal{NE}_{ek_T}(\mathcal{E}_{K_B}(a_1))||\mathcal{E}_{K_B}(v_1(a_1)))$, Alice receives $(\mathcal{NE}_{ek_T}(\mathcal{E}_{K_A}(b_1))||\mathcal{E}_{K_A}(v_1(b_1)))$

if $((\mathcal{E}_{K_A}(v_1(b_1)) == \perp \wedge \mathcal{E}_{K_B}(v_1(a_1)) == \perp) \vee ((v_1(b_1) == \perp) \wedge (v_1(a_1) == \perp))$ **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 $(b^1_1, ..., b^1_s, ..., b^n_1, ..., b^n_s)$: $[b'_i]^{n\cdot s}_{i=1} \leftarrow encodeInput(b_2)$

Alice and Bob run functionality $\mathcal{F}^e_{\mathsf{offline}}$ (in Algorithm 1) to generate circuit G computing function g

Bob sends (retrieve, offline) to $\mathcal{F}_{\mathsf{ca}}$ and obtains vk_T

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_2, K_A, sk_A: $((\mathcal{K}^{a[0]}_A[0], \sigma^{a[0]}_A[0]), ..., (\mathcal{K}^{a[n]}_A[n], \sigma^{a[n]}_A[n]))$

if $\exists(\mathcal{K}^{a[i]}_A[i], \sigma^{a[i]}_A[i])$, *s.t.*, $\mathcal{V}_{vk_T}(\mathcal{K}^{a[i]}_A[i], i, \sigma^{a[i]}_A[i]) == \perp$ **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 $\mathcal{F}^2_{\mathsf{ot}}((\mathcal{K}^0_B[i], \sigma^0_B[i]), (\mathcal{K}^1_B[i], \sigma^1_B[i]), b_i)$, Bob receives $(\mathcal{K}^{b[i]}_B[i], \sigma^{b[i]}_B[i])$

if $\mathcal{V}_{vk_T}(\mathcal{K}^{b[i]}_B[i], \sigma^{b[i]}_B[i]) == \perp$ **then** Bob sends $\mathcal{NE}_{ek_T}(\mathcal{E}_{K_B}(a_1))$ to $\mathcal{F}_{\mathsf{tea}}$

<u>Bob:</u>

$((\mathcal{E}_{K_A}(y_A), \sigma_A)||y_B) \leftarrow \mathcal{C}(\bar{\mathcal{K}_\mathcal{A}}, \bar{\mathcal{K}_\mathcal{B}})$

(see implementation in **Circuit Evaluation** below)

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 $\forall i$, $(\mathcal{K}^{b[i]}_B[i], \sigma^{b[i]}_B[i])$

$((\mathcal{E}_{K_A}(y_A), \sigma_A)||y_B) \leftarrow \mathcal{C}(\bar{\mathcal{K}_\mathcal{A}}, \bar{\mathcal{K}_\mathcal{B}})$

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

<u>Alice:</u>

sleep('response from Bob',$2\Delta_C$)

onReceive($\mathcal{E}_{K_A}(y_A), \sigma_A$)

if $((\mathcal{E}_{K_A}(y_A), \sigma) \neq \perp)$ **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 $\mathcal{F}_{\mathsf{tea}}$

Circuit Evaluation

$\mathcal{C}(\bar{\mathcal{K}_\mathcal{A}}, \bar{\mathcal{K}_\mathcal{B}})$ {

$(\mathcal{K}^{a[0]}_A[0], ..., \mathcal{K}^{a[n]}_A[n]) \leftarrow \bar{\mathcal{K}_\mathcal{A}}$, $(\mathcal{K}^{b[0]}_B[0], ..., \mathcal{K}^{b[sn]}_B[sn]) \leftarrow \bar{\mathcal{K}_\mathcal{B}}$

$(\mathcal{K}^{y[0]}_Y[0], ..., \mathcal{K}^{y[n]}_Y[n]) \leftarrow \bar{\mathcal{T}_G}((\mathcal{K}^{a[0]}_A[0], ..., \mathcal{K}^{a[n]}_A[n]), (\mathcal{K}^{b[0]}_B[0], ..., \mathcal{K}^{b[sn]}_B[sn]))$

return $\omega \leftarrow \bar{\mathcal{T}_D}(\mathcal{K}^{y[0]}_Y[0], ..., \mathcal{K}^{y[n]}_Y[n])$ }

end

Input Encoding

encodeInput$([b_i]^n_{i=1})$ { $b' = \emptyset$

for $i \leftarrow 1$ ***to*** n **do**

Let $b^i_1, ..., b^i_s \in_R \{0,1\}$ s.t. $b_i = b^i_1 \oplus ... \oplus b^i_s$

$b' = b'||b^i_1, ..., b^i_s$

return b' }//`after` n `iterations` $b' = [b'_i]^{n\cdot s}_{i=1} = (b^1_1, ..., b^1_s, ..., b^n_1, ..., b^n_s)$

end

end

Algorithm 8: Committed fair secure two-party protocol $\Pi^G_{(v,g)}$ in the $(\mathcal{F}^e_{\mathsf{offline}}, \mathcal{F}_{\mathsf{ca}}, \mathcal{F}_{\mathsf{tea}}, \mathcal{F}_{\Delta\text{-delayed-fairness}})$-hybrid model for computing $v : \{0,1\}^m \times \{0,1\}^m \rightarrow \{0,1\}^m \times \{0,1\}^m$ and $g : \{0,1\}^n \times \{0,1\}^n \rightarrow \{0,1\}^n \times \{0,1\}^n$.

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

[23] A.Y. Lindell. Legally-enforceable fairness in secure two-party computation. *Lecture Notes in Computer Science*, 4964: 121, 2008.

[24] 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.

[25] Y. Lindell and B. Pinkas. An Efficient Protocol for Secure Two-Party Computation in the Presence of Malicious Adversaries. *LECTURE NOTES IN COMPUTER SCIENCE*, 4515:52, 2007.

[26] Dahlia Malkhi, Noam Nisan, Benny Pinkas, and Yaron 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`.

[27] 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.

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

[29] 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.

[30] B. Pinkas. Fair Secure Two-Party Computation. *LECTURE NOTES IN COMPUTER SCIENCE*, pages 87–105, 2003.

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

[32] 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 $\mathcal{F}^e_{\mathsf{offline}}$, $\mathcal{F}^2_{\mathsf{ot}}$ 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 $\mathcal{F}^e_{\mathsf{offline}}$, $\mathcal{F}^2_{\mathsf{ot}}$ 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 $\mathcal{F}^e_{\mathsf{offline}}$, $\mathcal{F}^2_{\mathsf{ot}}$ and $\mathcal{F}_{\mathsf{ca}}$, 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 2. 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.

$S(a, \mathsf{ID_A}, 1^n)$

$\mathsf{ID_{A'}} \overset{\mathsf{OfflineParty}}{\longleftarrow} A(a, \mathsf{ID_A}, 1^n)$

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 2. 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 4.2)

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 2 in Propositions 6 and 7 respectively.

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 $\mathcal{F}_{\mathsf{Resolve}}$, $\mathcal{F}_{\mathsf{ca}}$ 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 $\mathcal{F}_{\Delta\text{-}\mathsf{delayed\text{-}fairness}}$, 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.

$S(b, \mathsf{ID_B}, 1^n)$

$\mathsf{ID_{B'}} \overset{\mathsf{OfflineParty}}{\longleftarrow} A(b, \mathsf{ID_B}, 1^n)$

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.

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 $\mathcal{F}_{\Delta\text{-delayed-fairness}}$

if $a' \neq a \vee ek' \neq ek$ **then**

send $\perp$ to $\mathcal{F}_{\Delta\text{-delayed-fairness}}$

send $\perp$ to A

output whatever A outputs and halt

else

S sends a to the trusted party computing $\mathcal{F}_{\Delta\text{-delayed-fairness}}$, 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 $\mathcal{F}_{\Delta\text{-delayed-fairness}}$, 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 $\mathcal{F}_{\mathsf{ca}}$, there exists a non-uniform polynomial time simulator S corrupting Alice and running in the ideal model with access to an ideal functionality $\mathcal{F}_{\Delta\text{-delayed-fairness}}$, such that for every $a, b, z \in \{0,1\}^*$ holds:*

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

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 $\mathcal{F}_{\Delta\text{-delayed-fairness}}$
if $b' \neq b \vee ek' \neq ek$ **then**
  send $\perp$ to $\mathcal{F}_{\Delta\text{-delayed-fairness}}$
  send $\perp$ to A
  output whatever A outputs and halt
else
  S sends b to the trusted party computing $\mathcal{F}_{\Delta\text{-delayed-fairness}}$, 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 $\mathcal{F}_{\Delta\text{-delayed-fairness}}$, 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 5.2)

Proof We analyse $\Pi^G_{(v,g)}$ in a ($\mathcal{F}^e_{\mathsf{offline}}$,$\mathcal{F}_{\mathsf{ca}}$,$\mathcal{F}_{\mathsf{tea}}$)-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 3 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 $\mathcal{F}^e_{\mathsf{offline}}$, $\mathcal{F}_{\mathsf{ca}}$ and $\mathcal{F}_{\mathsf{tea}}$, there exists a non-uniform polynomial time simulator S corrupting Alice and running in the ideal model with access to an ideal functionality $\mathcal{F}^{v,g}_{\mathsf{commited\text{-}computation}}$, such that for every $a, b, z \in \{0,1\}^*$ holds:*

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

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.

The view of A in a simulation with S is identical to its view in an ($\mathcal{F}^e_{\mathsf{offline}}$,$\mathcal{F}_{\mathsf{ca}}$,$\mathcal{F}_{\mathsf{tea}}$)-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 $\mathcal{F}^e_{\mathsf{offline}}$, $\mathcal{F}_{\mathsf{ca}}$, and $\mathcal{F}_{\mathsf{tea}}$ there exists a non-uniform polynomial time simulator S corrupting Bob and running in the ideal model with access to an ideal functionality $\mathcal{F}^{v,g}_{\mathsf{commited\text{-}computation}}$, such that for every $a, b, z \in \{0,1\}^*$ holds:*

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

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 $\mathcal{F}^{v,g}_{\mathsf{commited\text{-}computation}}$
if $a_1' \neq a_1 \vee ek' \neq ek$ **then**
  send $\perp$ to $\mathcal{F}^{v,g}_{\mathsf{commited\text{-}computation}}$
  send $\perp$ to A
  output whatever A outputs and halt
else
  S sends a_1 to the trusted party computing $\mathcal{F}^{v,g}_{\mathsf{commited\text{-}computation}}$, 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 $\mathcal{F}^{v,g}_{\mathsf{commited\text{-}computation}}$
    send $\perp$ to A
    output whatever A outputs and halt
  else
    simulate $\mathcal{F}^{e}_{\mathsf{offline}}$ for A according to steps in Algorithm 9
    Send a_2 to $\mathcal{F}^{v,g}_{\mathsf{commited\text{-}computation}}$
    Upon input y_A^2 from $\mathcal{F}^{v,g}_{\mathsf{commited\text{-}computation}}$, 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 $\mathcal{F}^{v,g}_{\mathsf{commited\text{-}computation}}$, and simulating the view of 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 $\mathcal{F}^{v,g}_{\mathsf{commited\text{-}computation}}$
if $b_1' \neq b_1 \vee ek' \neq ek$ **then**
  send $\perp$ to $\mathcal{F}^{v,g}_{\mathsf{commited\text{-}computation}}$
  send $\perp$ to A
  output whatever A outputs and halt
else
  S sends b_1 to the trusted party computing $\mathcal{F}^{v,g}_{\mathsf{commited\text{-}computation}}$, 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 $\mathcal{F}^{v,g}_{\mathsf{commited\text{-}computation}}$
    send $\perp$ to A
    output whatever A outputs and halt
  else
    simulate $\mathcal{F}^{e}_{\mathsf{offline}}$ for A according to steps in Algorithm 10
    Send b_2 to $\mathcal{F}^{v,g}_{\mathsf{commited\text{-}computation}}$
    Upon input y_B^2 from $\mathcal{F}^{v,g}_{\mathsf{commited\text{-}computation}}$, 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 $\mathcal{F}^{v,g}_{\mathsf{commited\text{-}computation}}$
S outputs whatever A outputs.

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

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 $(\mathcal{F}^{e}_{\mathsf{offline}}, \mathcal{F}_{\mathsf{ca}}, \mathcal{F}_{\mathsf{tea}})$-hybrid execution of $\Pi^G_{(v,g)}$ 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.