Oblivious and Fair Server-Aided Two-Party Computation

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

***Abstract*—We show efficient, practical (server-aided) secure two-party computation protocols ensuring privacy, correctness and fairness in the presence of malicious (Byzantine) faults. Our requirements from the server are modest: to ensure privacy and correctness, we only assume *offline set-up* prior to protocol execution; and to also ensure fairness, we further assume a *trusted-decryption* service, providing decryption service using known public key. The fairness-ensuring protocol is *optimistic*, i.e., the decryption service is invoked only in case of faults. Both assumptions are feasible in practice and formally presented in the hybrid model. The resulting protocols may be sufficiently efficient, to allow deployment, in particular for financial applications.**

***Index Terms*—Two-party computation, fair optimistic protocols, server-aided computation.**

I. Introduction

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

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

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

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

Such practical implementation requires modest computational resources and very limited trust; security is assured as long as the service does not collude with one of the parties. The use of these ideal functionalities (or, in practice, weakly-trusted services), allows us to address the two main challenges: *efficiency* and *fairness*. We next briefly discuss these challenges.

Efficiency. Secure computation protocols suffer from a significant overhead, especially when considering arbitrary (byzantine) faults, as required in practice. The basic secure function evaluation technique for two-party computation, [1], ensures security, privacy and integrity only against passive, semi-honest, adversaries that follow the steps of the protocol. This protection however does not suffice for real life systems, where malicious participants may arbitrarily deviate from the prescribed steps of the protocol, e.g., in an attempt to gain an unfair advantage. The basic protocol of [1] was extended by Goldreich et al. [2] to ensure security against malicious adversaries, e.g., those that may try to alter the agreed computation. Several later works improved efficiency, e.g., [8], [9], [10]. However, these protocols are still so computationally intensive, that they are impractical for many real life applications, and specifically for most financial applications. Indeed, there is little hope of sufficient improvements in the efficiency of secure computation protocols, to allow their use for such tasks in the forseenable future.

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

In this work, we present two simple models for secure two party computation: the *preprocessing oracle model*, which allows efficient constructions of protocols secure against malicious faults, and the *decryption oracle model*, which produces fair and efficient protocols. Both oracles are simple, generic and oblivious to the computation, including the inputs of the parties to the protocol and outputs from the protocol. This is similar to the use of other set-up assumptions such as common reference string (CRS) model[1], random oracle model or registration authority, e.g., see [11], [12].

Namely, these simple models capture weak assumptions which appear necessary for fair and efficient secure two-party computation protocols; we present and analyse such protocols. The ideal functionalities capturing the models can be realised, e.g., by weakly-trusted additional parties. Further research is required to determine whether these models are indeed the minimal necessary, or whether the same goals (efficiency, fairness) be achieved in even weaker models; furthermore, the models we present may be useful for other tasks.

Following many previous fair protocols, our fair protocol is *optimistic*; specifically, the parties involve the ideal functionality $\mathcal{F}_{\mathsf{Decryption}}$ only in case of a misbehaviour by the peer or in case of faults. In this sense, our work is related to many existing works on optimistic fair exchange and similar tasks, e.g., see [13], [14], except that these works support only specific, relatively simple interactions, e.g., certified mail, contract signing, and do not hide the values exchanged from the trusted third party. In contrast, our protocol supports arbitrary computations, and does not expose the inputs (or respective outputs) of the parties to the ideal functionality that ensures fairness.

Cachin and Camenisch, [15], presented optimistic fair secure computation protocol, with constant number of interactions. However, their protocol is not practical since it requires running a zero knowledge protocol for every gate of the circuit; furthermore, they assume a highly specialised and powerful third party. In contrast, our protocols assume only a very simple, minimal ideal functionality $\mathcal{F}_{\mathsf{Decryption}}$, and furthermore are much more efficient; see Section V-A for efficiency comparison.

CONTRIBUTIONS. This work has both practical and theoretical contributions:

- Efficient, practical protocols for (fair) two party computation, with set-up assumptions formulating the new models which we introduce in this work. The models are implementable using a weakly-trusted 'third party'. Our protocols are secure against malicious (byzantine) faults, but with efficiency comparable to that of the existing protocols that are secure only against honest-but-curious participants.
- Introduction of two new models, the *preprocessing model* allowing trusted preprocessing phase for greater efficiency, and the *decryption model* which provides basic generic facility allowing resolution of conflicts (e.g., for fairness).

CRYPTOGRAPHIC TOOLS. Our constructions use several tools, i.e., standard cryptographic mechanisms: (1) *authenticated symmetric-key encryption scheme* $(\mathcal{K}, \mathcal{E}, \mathcal{D})$ to ensure confidentiality and integrity of the inputs and outputs of the participants, [16]. When applying $\mathcal{E}_{K_P}(x)$ we perform an authenticated encryption of input x using the key K_P of party P. (2) *Non-malleable public-key encryption scheme* $(\mathcal{NG}, \mathcal{NE}, \mathcal{ND})$ to ensure confidentiality and non-malleability; (3) *Signature scheme* $(\mathcal{G}, \mathcal{S}, \mathcal{V})$, [17], where we use $\perp$ to denote authentication failure; (4) *Two-party (1-2) oblivious transfer* which we denote by the ideal functionality $\mathcal{F}^2_{\mathsf{ot}}$, [12].

ORGANISATION. In Section II we introduce the *preprocessing model* allowing offline set-up assumption for greater efficiency; we subsequently use this model in Section III. Then in Section IV we introduce the *assisted-decryption functionality* that upon requests validates and decrypts the input ciphertext. We later use this functionality in Section V in the construction of a fair two-party protocol, where we use the assisted-decryption functionality to restore fairness in case of failures or misbehaviour by one of the participants. In Section VI we present sample financial applications based on our protocols. Finally, in Section VII, we conclude and present future research directions.

II. PREPROCESSING MODEL

In this section we introduce the preprocessing model allowing efficient protocols for secure computation in the malicious setting. The model assumes that two (or multiple[2]) parties in the protocol have an access to an ideal functionality (oracle), during the preprocessing phase, which upon invocation provides them with output strings (one for each party) which they later use to carry out the protocol between them. The oracle is only used during the preprocessing phase, to generate the respective output strings to the parties, and the oracle is not involved in the computation itself. The preprocessing oracle functionality is one of our basic set-up assumptions and it is captured with the ideal functionality $\mathcal{F}_{\mathsf{pre}}$, in Algorithm 1. In the next section we show how the strings, generated by the oracle, ensure privacy and correctness to the computation during the protocol execution, while allowing efficiency equivalent to that of the semi-honest setting. Since the preprocessing oracle does not obtain the secret inputs of the parties (which may not even be available during the preprocessing phase) it does not learn anything about the inputs of the parties from the computation that it performs.

[1]In fact preprocessing model is a generalisation of CRS.

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

```
Functionality $\mathcal{F}_{\mathsf{pre}}$

Input: $\mathsf{Prg}_1$ from party $P_1$ and $\mathsf{Prg}_2$ from party $P_2$, input length $n$
Computation Phase
    $r \xleftarrow{R} \{0,1\}^n$
    if $\mathsf{Prg}_1 \neq \mathsf{Prg}_2$ then
        output $\perp$
    Evaluate $\mathsf{Prg}_1$ on $r$ and obtain $(p_1||p_2) \leftarrow \mathsf{Prg}_1(r)$
end
Output: send $p_1$ to $P_1$ and $p_2$ to $P_2$
```

Algorithm 1: The functionality $\mathcal{F}_{\mathsf{pre}}$ for executing a 'preprocessing' circuit, which description it receives as input from the parties. The functionality checks if the circuits supplied by both parties are equivalent, in which case it evaluates the circuit on a random string r, and obtains $p_1||p_2$ where p_1 is the output of P_1 and p_2 is the output of P_2, and sends the outputs to the respective parties. Otherwise, if the circuits are not the same it returns $\perp$.

The preprocessing oracle receives a function that it should compute. This allows for a general definition, one that allows the parties to define the computation that they wish the oracle to perform. Our preprocessing model is a generalisation of the Common Reference String (CRS) model, [11], [12]. In the CRS model the parties are given a common public reference string that was ideally chosen from a given distribution. Note that, similarly to CSR, the computation that the preprocessing oracle performs is not a function of the private inputs of the parties, and neither does it observe the output from the protocol. Furthermore, the parties are not required to identify themselves before participating in the protocol.

For generality, we will assume that the preprocessing oracle computes a universal function which receives in an input description of a function (or rather a description of a circuit Prg implementing a function that the parties agreed upon) from the parties and evaluates that function on a random string. We capture the preprocessing model with the ideal functionality $\mathcal{F}_{\mathsf{pre}}$: $\mathcal{F}_{\mathsf{pre}}$ receives a description of a circuit, P_1 and P_2, (computing an agreed upon function) from both participants, executes that circuit on a random string r, obtains two outputs p_1, p_2 and returns each output to the corresponding party. The $\mathcal{F}_{\mathsf{pre}}$ ideal functionality is not involved during the computation performed by the parties.

III. Two-Party Protocol with Preprocessing

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. We construct a two-party protocol, in Algorithm 4, for evaluation of inputs of Alice and Bob on a known function e using the preprocessing oracle presented in Algorithm 1.

During the preprocessing phase, Alice and Bob send a description of a circuit, which they agreed on, to the third party, and receive an output. The output is an encoding of a garbled circuit, see description in Section III-B, which is then used by Alice and Bob to evaluate the function $e : \{0,1\}^n \times \{0,1\}^n \to \{0,1\}^n$ on their respective inputs.

When Alice and Bob have the inputs they run the two-party computation protocol, whereby Alice transfers the strings representing her input to Bob and runs an $\mathcal{F}^2_{\mathsf{ot}}$ functionality (capturing the oblivious transfer protocol, [12]) for strings representing Bob's input bits; Bob evaluates the circuit on both inputs, concluding the protocol. After evaluating the functionality e on a and b, Bob obtains $e(a,b)$, while Alice learns nothing at all.

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

In Section III-B present a description of the circuit that Alice and Bob send to the preprocessing functionality, which allows for garbled circuit pre-generation, and in Section III-C we construct the two-party protocol in the preprocessing model.

A. Definition: Two-Party Computation with Output at Bob

In our definitions below (and constructions of the protocols throughout this work) we assume that Alice is the originator of the computation and Bob is the evaluator.

Execution in the Ideal Model: In the ideal model, there is a universally trusted third party, that is parametrised by $e : \{0,1\}^n \times \{0,1\}^n \to \{0,1\}^n$, the function which Alice and Bob wish to evaluate on their inputs. The parties are Alice and Bob and the adversary A that has corrupted one of them. The parties Alice and Bob send their inputs to the trusted party, who performs the computation and returns the output to Bob. We assume that one of the two parties, either Alice or Bob, can be corrupted by the adversary A. The pair of outputs of the honest party and an adversary S in an ideal execution where the trusted party computes e is denoted $\text{IDEAL}_{e,S(z)}(a,b,n)$, where n is the security parameter, a and b are inputs of Alice and Bob respectively, and z is an auxiliary input of the adversary S.

```
Private Inputs : Alice and Bob send $a$ and $b$ (respectively) to
                 the trusted party
Public Inputs  : $n$
Computation Phase
    if $a = \perp \vee b = \perp$ then
        send $\perp$ to Bob
    send $e(a,b)$ to Bob
end
Output: The honest party outputs whatever it received from
        the trusted party, the corrupted party does not output
        anything, and the adversary $S$ outputs an arbitrary
        function of its view.
```

Algorithm 2: The ideal functionality $\mathcal{F}_e$ for computing e, runs with Alice and Bob, and an adversary S, and provides an output at only one party (Bob).

Execution in the Real Model: In the real model a two party protocol Π_e is executed between Alice and Bob without a trusted party. The adversary A controls one of the parties, obtains the inputs of the corrupted party, and sends messages on behalf of that party. The honest party follows the protocol Π_e and returns the output specified by Π_e. The adversary outputs an arbitrary function of its view. The pair of outputs of the honest party and an adversary A in the real protocol execution is denoted $\text{REAL}_{\Pi_e, A(z)(a,b,n)}$.

Definition 3.1 (Two-Party Protocol with Output at One party): Let Π_e be a protocol and let e be a polynomial two-party functionality. Protocol Π_e is said to securely compute e if for every probabilistic polynomial-time adversarial algorithm A in the real model running with Π_e, there exists a probabilistic polynomial-time simulator S in the ideal model, such that for every $a, b, z \in \{0,1\}^n$, holds

$$\left\{\text{REAL}_{\Pi_e, A(z)}(a,b,n)\right\}_{n \in \mathbb{N}} = \left\{\text{IDEAL}_{e, S(z)}(a,b,n)\right\}_{n \in \mathbb{N}}$$

B. Garbled Circuit Pre-Generation

To allow for evaluation of arbitrary functions we define Prg_e to be a circuit that generates a garbled circuit computing a common function which the parties agreed on. The circuit Prg_e is parametrised by a function $e : \{0,1\}^n \times \{0,1\}^n \to \{0,1\}^n$ (which is encoded as a circuit C). Thus the output that the preprocessing functionality produces is a garbled circuit, which will be used by the parties during the protocol execution. In Algorithm 3 we present the circuit Prg_e; the circuit generates a signature key pair (sk_{pre}, vk_{pre}), then constructs and gables a circuit, and signs the circuit with the freshly generated signature key sk_{pre}. We use part of the string r (which is selected at random by $\mathcal{F}_{\mathsf{pre}}$) for garbled circuit generation, and another part for keys generation and signatures. The public verification key is registered with the $\mathcal{F}_{\mathsf{reg}}$[3] (the registration authority) and thus Alice and Bob can later retrieve it to verify signatures.

The circuit Prg_e, in Algorithm 3, is encoded as a circuit C that computes function e. The circuit C consists of gates; in the first level each gate has two input wires, one for input bit of Alice and another for input bit of Bob. $\mathcal{F}_{\mathsf{pre}}$ generates a garbled circuit $\mathbf{C}$ from C as follows: it first 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 his input values (during the oblivious transfer protocol); see [8] for details. Next, random strings, corresponding to each input bit of Alice and Bob, are generated, and the gates of the circuit are replaced with garbled gates, i.e., boolean tables incorporating the outputs from a gate for all possible combinations of inputs of Alice and Bob.

Eventually, $\mathcal{F}_{\mathsf{pre}}$ sends the random input strings (corresponding to all possible inputs) to Alice, and the garbled gates and output decryption tables to Bob. Note that according to the model in Algorithm 1, part of the output should be sent to Alice and another part to Bob. However, in this case the output of Bob is $\perp$ (i.e., no private output) and the entire output (the signed garbled circuit) from $\mathcal{F}_{\mathsf{pre}}$ can be sent to Alice, and Alice in turn will forward to Bob the signed garbled tables and output decryption tables. For efficiency (and simplicity) we let $\mathcal{F}_{\mathsf{pre}}$ send the gabled tables and output decryption tables directly to Bob (this is only a simplifying assumption since these tables constitute an output that is not secret, i.e., known to both Alice and Bob).

C. Two-Party Protocol Construction

In Algorithm 4 we construct a two-party protocol by applying the $\mathcal{F}_{\mathsf{pre}}$ functionality during the preprocessing phase. The resulting two-party protocol illustrates one of the applications of the preprocessing model. We also use this mechanism in the subsequent section as a building block in our fair two-party computation protocol. The procedure that generates the garbled circuit using a third party[4] during the preprocessing phase can be of independent interest to enhance efficiency (see Section III-C2 for efficiency comparison of techniques employed in malicious model) of two-party protocols in malicious setting. Assuming preprocessing phase allows a simpler and much more efficient protocol (cf. to [26], [27], [8], [20]).

```
Registration Phase
    generate signature key-pair: $(vk_{pre}, sk_{pre}) \leftarrow \mathcal{G}(1^s)$
    register the verification key: (register, preprocessing, $vk_{pre}$) to $\mathcal{F}_{\mathsf{reg}}$
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 $\mathbf{C}$. The garbled circuit consists of:
        a. Random strings corresponding to all possible input bits of Alice:
    $\overline{\mathcal{K}_A} = ((\mathcal{K}_A^0[1], \mathcal{K}_A^1[1]), ..., (\mathcal{K}_A^0[n], \mathcal{K}_A^1[n]))$
        b. Random strings corresponding to all possible input bits of Bob:
    $\overline{\mathcal{K}_B} = ((\mathcal{K}_B^0[1], \mathcal{K}_B^1[1]), ..., (\mathcal{K}_B^0[sn], \mathcal{K}_B^1[sn]))$
        c. Garbled boolean tables $\overline{\mathcal{T}_G}$ for each garbled gate $G$ of the circuit
        d. Output decryption tables $\overline{\mathcal{T}_D}$ mapping output strings to bits
    4. Sign the random input strings $\overline{\mathcal{K}_A}$ of Alice:
    $\overline{\sigma}_A = ((\sigma_A^0[1], \sigma_A^1[1]), ..., (\sigma_A^0[n], \sigma_A^1[n]))$ where
    $\forall i, j : \sigma_A^j[i] = \mathcal{S}_{sk_{pre}}(\mathcal{K}_A^j[i], i)$
    5. Sign the random input strings $\overline{\mathcal{K}_B}$ of Bob:
    $\overline{\sigma}_B = ((\sigma_B^0[1], \sigma_B^1[1]), ..., (\sigma_B^0[sn], \sigma_B^1[sn]))$ where
    $\forall i, j : \sigma_B^j[i] = \mathcal{S}_{sk_{pre}}(\mathcal{K}_B^j[i], i, j)$
end
Output: $(\overline{\mathcal{K}_A}, \overline{\sigma}_A), (\overline{\mathcal{K}_B}, \overline{\sigma}_B)$ to Alice and $(\overline{\mathcal{T}_G}, \overline{\mathcal{T}_D})$ to Bob.
```

Algorithm 3: The circuit Prg_e generates a garbled circuit $\mathbf{C}$, based on a circuit $\mathcal{C}$ computing e that both parties agreed upon.

1) Security Analysis: We analyse Π_e in a hybrid model where there is a trusted party computing $\mathcal{F}_{\mathsf{pre}}$, $\mathcal{F}_{\mathsf{ot}}^2$ and $\mathcal{F}_{\mathsf{reg}}$. 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 $\mathcal{F}_{\mathsf{pre}}$, $\mathcal{F}_{\mathsf{ot}}^2$ and $\mathcal{F}_{\mathsf{reg}}$.

Claim 3.2: 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 a chosen-message attack. Then protocol Π_e^E securely realises a two-party functionality with output at Bob only (according to Definition 3.1), in the presence of malicious adversaries in the ($\mathcal{F}_{\mathsf{pre}}$,$\mathcal{F}_{\mathsf{ot}}^2$,$\mathcal{F}_{\mathsf{reg}}$)-hybrid model.

[3]The registration authority maintains a database of signed public keys, and allows anyone to retrieve a key by providing the identity of the owner of the key.

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

```
**Input**: security parameter $s$, number of bits $n$
**Output**: $y_B = e(a, b)$
**Offline Generation Phase**
    Alice and Bob send $\mathsf{ID_A},\ \mathsf{ID_B}$ (respectively) to $\mathcal{F}_{\mathsf{pre}}$
    Alice receives $(\overline{\mathcal{K}_A}, \overline{\sigma}_A), (\overline{\mathcal{K}_B}, \overline{\sigma}_B)$ (see Algorithm 3)
    Bob receives $\overline{\mathcal{T}_G}, \overline{\mathcal{T}_D}$
**end**
**Computation Phase**
    Alice receives $\overline{a} = [a_i]_{i=1}^n$
    Bob receives $\overline{b} = [b_i]_{i=1}^n$
    **Input Encoding**
        $encodeInput([b_i]_{i=1}^n)$ {
        $b' = \emptyset$
        **for** $i \leftarrow 1$ ***to*** $n$ **do**
            Let $b_1^i, ..., b_s^i \in_R \{0, 1\}$ s.t. $b_i = b_1^i \oplus ... \oplus b_s^i$
            $b' \leftarrow (b'||b_1^i, ..., b_s^i)$
        return $b'$ }
        //in $n$ iterations $b' = [b'_i]_{i=1}^{n \cdot s} = (b_1^1, ..., b_s^1, ..., b_1^n, ..., b_s^n)$
    **end**
    Alice: sends to Bob: $((\mathcal{K}_A^{a[1]}[1], \sigma_A^{a[1]}[1]), ..., (\mathcal{K}_A^{a[n]}[n], \sigma_A^{a[n]}[n]))$
    Bob:
        send (retrieve, preprocessing) to $\mathcal{F}_{\mathsf{reg}}$ and obtain $vk_{pre}$
        **if**
        $\exists(\mathcal{K}_A^{a[i]}[i], \sigma_A^{a[i]}[i]),\ s.t.,\ \mathcal{V}_{vk_{pre}}(\mathcal{K}_A^{a[i]}[i], i, \sigma_A^{a[i]}[i]) = \mathsf{false}$
        **then**
            output $\perp$ and halt
        **for** $i \leftarrow 1$ ***to*** $n \cdot s$ **do**
            run with Alice $\mathcal{F}_{\mathsf{ot}}^2((\mathcal{K}_B^0[i], \sigma_B^0[i]), (\mathcal{K}_B^1[i], \sigma_B^1[i]), b'_i)$
            //run oblivious transfer, Alice provides
             $(\mathcal{K}_B^0[i], \sigma_B^0[i]), (\mathcal{K}_B^1[i], \sigma_B^1[i])$ and Bob $b'_i$
            receive $(\mathcal{K}_B^{b'[i]}[i], \sigma_B^{b'[i]}[i])$
            **if** $\mathcal{V}_{vk_{pre}}(\mathcal{K}_B^{b'[i]}[i], \sigma_B^{b'[i]}[i]) == \mathsf{false}$ **then**
                output $\perp$ and halt
        $(y_B = (y_B[1], ..., y_B[n])) \leftarrow$
        $\mathcal{C}((\mathcal{K}_A^{a[1]}[1], ..., \mathcal{K}_A^{a[n]}[n]), (\mathcal{K}_B^{b'[1]}[1], ..., \mathcal{K}_B^{b'[sn]}[sn]))$ (see below)
    **end**
**end**
**Circuit Evaluation**
    $\mathcal{C}((\mathcal{K}_A^{a[1]}[1], ..., \mathcal{K}_A^{a[n]}[n]), (\mathcal{K}_B^{b'[1]}[1], ..., \mathcal{K}_B^{b'[sn]}[sn]))$ {
      $(\mathcal{K}_Y^{y[1]}[1], ..., \mathcal{K}_Y^{y[n]}[n]) \leftarrow$
    $\overline{\mathcal{T}_G}((\mathcal{K}_A^{a[1]}[1], ..., \mathcal{K}_A^{a[n]}[n]), (\mathcal{K}_B^{b'[1]}[1], ..., \mathcal{K}_B^{b'[sn]}[sn]))$
    return $\omega \leftarrow \overline{\mathcal{T}_D}(\mathcal{K}_Y^{y[1]}[1], ..., \mathcal{K}_Y^{y[n]}[n])$ }
**end**
//$C(a, b) = \overline{\mathcal{T}_G}(\overline{\mathcal{T}_D}(\overline{\mathcal{K}}_A^a, \overline{\mathcal{K}}_B^b))$
```

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

Proof: We analyse Π_e^E in a ($\mathcal{F}_{\mathsf{pre}}$,$\mathcal{F}_{\mathsf{ot}}^2$,$\mathcal{F}_{\mathsf{reg}}$)-hybrid model, and show that the execution of Π_e^E is computationally indistinguishable from computation of e in the ideal model. We prove the Claim 3.2 in Propositions 3.3 and 3.4 for cases where the adversary controlls Alice or Bob, respectively. ■

Proposition 3.3 (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 $\mathcal{F}_{\mathsf{pre}}$, $\mathcal{F}_{\mathsf{ot}}^2$ and $\mathcal{F}_{\mathsf{reg}}$, there exists a probabilistic polynomial-time simulator S corrupting Alice and running in the ideal model with access to an ideal functionality $\mathcal{F}_e$, such that for every $a, b, z \in \{0, 1\}^*$ holds:

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

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 5 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 ($\mathcal{F}_{\mathsf{pre}}$,$\mathcal{F}_{\mathsf{reg}}$,$\mathcal{F}_{\mathsf{ot}}^2$)-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 $\mathcal{F}_e$ as Alice's input
        send $\perp$ to $A$ as its input from $\mathcal{F}_{\mathsf{pre}}$
        output whatever $A$ outputs and halt
    **else**
          simulate functionality $\mathcal{F}_{\mathsf{pre}}$ for $A$:
        1. choose a key pair $(vk, sk) \leftarrow \mathcal{G}(1^n)$
        2. construct a circuit $C$ computing $f'_A$
        3. construct **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: $\bar{\mathcal{K}_A} = ((\mathcal{K}_A^0[0], \mathcal{K}_A^1[0]), ..., (\mathcal{K}_A^0[n], \mathcal{K}_A^1[n]))$
            b. Random strings corresponding to all possible input bits of
        Bob: $\bar{\mathcal{K}_B} = ((\mathcal{K}_B^0[0], \mathcal{K}_B^1[0]), ..., (\mathcal{K}_B^0[n], \mathcal{K}_B^1[n]))$
            c. Garbled boolean tables $\bar{\mathcal{T}_G}$ for each garbled gate $G$ of the
        circuit $\mathcal{C}$
            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 $\bar{\sigma} = ((\sigma_0^0, \sigma_0^1), ..., (\sigma_n^0, \sigma_n^1))$
        6. send $\bar{\mathcal{K}_A}$, $(\bar{\mathcal{K}_B}, \bar{\sigma})$ to $A$ as its output from $\mathcal{F}_{\mathsf{pre}}$
        $A$ sends $\bar{\mathcal{K}'_A}$, intended for Bob and $(\bar{\mathcal{K}'}_B, \bar{\sigma}')$ for ideal
        functionality $\mathcal{F}_{\mathsf{ot}}^2$
        **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 $\mathcal{F}_e$ as Alice's input
            send $\perp$ to $A$ as its input from $\mathcal{F}_{\mathsf{ot}}^2$
            output whatever $A$ outputs and halt
    $A$ outputs its view and halts, $S$ outputs the same and halts
**end**
```

Algorithm 5: Simulator S, simulating the view of Alice.

■

Proposition 3.4 (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 $\mathcal{F}_{\mathsf{pre}}$,$\mathcal{F}_{\mathsf{ot}}^2$ and $\mathcal{F}_{\mathsf{reg}}$, there exists a probabilistic polynomial-time simulator S corrupting Bob and running in the ideal model with access to an ideal functionality computing $\mathcal{F}_e$, such that for every $a, b, z \in \{0, 1\}^*$ holds:

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

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 $\mathcal{F}_e$, and can simulate the view of the execution of the protocol. Assume that Bob is corrupted by a hybrid model adversary A. In Algorithm 6 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 $\mathcal{F}_{\mathsf{ot}}^2$ 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 ($\mathcal{F}_{\mathsf{pre}}$,$\mathcal{F}_{\mathsf{ot}}^2$,$\mathcal{F}_{\mathsf{reg}}$)-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. ■

```
$S(b, \mathsf{ID_B}, 1^n)$
    $\mathsf{ID_{B'}} \overset{\mathsf{OfflineParty}}{\longleftarrow} A(b, \mathsf{ID_B}, 1^n)$
    **if** $\mathsf{ID_{B'}} = \perp \vee \mathsf{ID_{B'}} \neq \mathsf{ID_B}$ **then**
        send $\perp$ to the trusted party computing $\mathcal{F}_e$ as Bob's input
        send $\perp$ to $A$ as its input from $\mathcal{F}_{\mathsf{pre}}$
        output whatever $A$ outputs and halt
    **else**
          simulate functionality $\mathcal{F}_{\mathsf{pre}}$ for $A$:
        1. choose a key pair $(vk, sk) \leftarrow \mathcal{G}(1^n)$
        3. when $A$ sends (retrieve, $\mathcal{F}_{\mathsf{pre}}$) to $\mathcal{F}_{\mathsf{reg}}$, respond with (retrieve
        $\mathcal{F}_{\mathsf{pre}}$, $vk$):
        4. construct a circuit $C$ computing $f'_A$
        5. construct **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: $\bar{\mathcal{K}_A} = ((\mathcal{K}_A^0[0], \mathcal{K}_A^1[0]), ..., (\mathcal{K}_A^0[n], \mathcal{K}_A^1[n]))$
            b. Random strings corresponding to all possible input bits of
        Bob: $\bar{\mathcal{K}_B} = ((\mathcal{K}_B^0[0], \mathcal{K}_B^1[0]), ..., (\mathcal{K}_B^0[n], \mathcal{K}_B^1[n]))$
            c. Garbled boolean tables $\bar{\mathcal{T}_G}$ for each garbled gate $G$ of the
        circuit $\mathcal{C}$
            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 $\bar{\sigma} = ((\sigma_0^0, \sigma_0^1), ..., (\sigma_n^0, \sigma_n^1))$
        8. send $\bar{\mathcal{T}_G}, \bar{\mathcal{T}_D}$ to $A$ as its output from $\mathcal{F}_{\mathsf{pre}}$
        **for** $i \leftarrow 1$ ***to*** $|b|$ **do**
            run $\mathcal{F}_{\mathsf{ot}}^2((\mathcal{K}_B^0[i], \sigma_i^0), (\mathcal{K}_B^1[i], \sigma_i^1), b_i)$, providing
            $(\mathcal{K}_B^0[i], \sigma_i^0), (\mathcal{K}_B^1[i], \sigma_i^1)$ and $A$ provides $b_i$
            $A$ receives $(\mathcal{K}_B^{b_i}[i], \sigma_i^{b_i})$
            output whatever $A$ outputs and halt
**end**
```

Algorithm 6: Simulator S, simulating the view of Bob.

2) Efficiency Analysis: Secure function evaluation based on garbled circuits, [1], allows to perform a two-party computation in a secure manner, i.e., ensuring privacy, correctness and inputs independence (see [21]). The computation is constant-round but ensures security only against semi-honest adversaries. When considering malicious adversaries, which is the typical model in practice, additional security concerns arise, that are not addressed by the basic secure function evaluation protocol, [1], [21].

Any two-party protocol can be transformed into a secure protocol in the malicious setting, e.g.,[2], but the resulting protocol is not constant round. Subsequently, constant round protocols were presented, e.g., see [17], [23], [15], [25], [22], [2]. However, these protocols are based on zero knowledge proofs, which renders them inefficient for practical purposes. Also protocols that do not employ zero knowledge were constructed, e.g., [24], however their round complexity is linear in the depth of the circuit.

In [26] 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 [27] improved the communication complexity of [26] to $\mathcal{O}(s|C|)$ using expanders. However as [8] observed, the protocol in [26] is susceptible to 'input corruption' attack; [8] also present a protocol with roughly the same communication complexity as [26], of $\mathcal{O}(s|C|+s^2n)$ (this protocol was implemented in [28]). Another improvement to two-party computation in malicious setting was made by [9] using homomorphic encryption; they present a protocol in the common reference string (CRS) model, that has a constant number of rounds, and has an $\mathcal{O}(|C|)$ public-key operations (cf. $\mathcal{O}(s|C| + s^2n)$ in [8]), and computational complexity of $\mathcal{O}(|C|)$ (as opposed to $\mathcal{O}(n)$ in [8]). Subsequently, the work of [20], also followed the cut-and choose approach in a different manner and improved the complexity to $\mathcal{O}(\frac{s|C|}{log(|C|)})$. Efficiency improvements were also designed for multi-party computation, [29], by optimising AES encryption; their ideas can be applied when implementing the encrypion in our protocols. A new multi-party protocol to securely evaluate reactive arithmetic circuits, offering security against an active adversary in the universally composable security framework, was proposed by [30]; the protocol is based on a design of an efficient 'cut-and-choose' technique. Techniques reducing the size of garbled tables, thus improving computation and communication complexity, were proposed in [31]; the design of the gates rely on a 'free-XOR' technique. [32] present a framework for secure function evaluation using 'privately programmable blocks'.

Our protocol, in Algorithm 4, is computationally efficient as it uses public key operations only for signing (by Prg_e) 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, see [1], [21]). Our protocol is efficient in that it has only a constant number of rounds and uses only one oblivious transfer operation per each input bit. This is in contrast to the complexity of [8], which due to the cut-and-choose incur a multiplicative increase by a factor of s (the statistical security parameter) and results in communication complexity of $\mathcal{O}(s|C| + s^2n)$.

IV. Assisted-Decryption Model

The decryption model provides the parties in the two-party protocol with an access to the decryption oracle that upon invocation with a ciphertext, validates and decrypts the ciphertext, and returns the result to Alice and Bob. This model requires minimal trust (and computation) and allows for modular constructions that guarantee fairness.

The decryption model is captured with the $\mathcal{F}_{\mathsf{Decryption}}$ functionality, Algorithm 8. In contrast to $\mathcal{F}_{\mathsf{pre}}$ (Algorithm 1), even if $\mathcal{F}_{\mathsf{Decryption}}$ is corrupt, the implication is on fairness only, but not on privacy or correctness. We use $\mathcal{F}_{\mathsf{Decryption}}$ ideal functionality in the fair protocol which we present in Section V. However, we believe that the $\mathcal{F}_{\mathsf{Decryption}}$ could be useful for other tasks too, and not only to ensure fairness. For instance, this functionality can be implemented using simple, secure (stateless) hardware, or via a set of servers (using distributed/proactive decryption), [33].

V. Fair Two-Party Protocol with Assisted-Decryption

The standard definition of two-party computation [17] allows Alice and Bob to securely evaluate a function over their private inputs; however, a corrupted party can abort the protocol execution prematurely after it receives its output, while preventing the honest party from receiving output. In many scenarios both parties should receive output, which requires an additional property of *fairness*. Specifically, Alice receives her output if and only if Bob receives his, or no party receives the output. Fairness is especially important for financially oriented tasks, e.g., exchange of signed checks, or currency exchange.

In Algorithm 7 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. We assume synchronous communication model with bounded delay Δ_C, i.e., the messages are never lost and are delivered within the assumed delay bound. In Δ-delayed fairness (which generalises the standard definition with abort, [17]), either both parties receive the output or neither 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 of security with abort. Specifically, any protocol that securely computes a functionality with Δ-delayed fairness, also securely computes the functionality of fairness with abort. 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. In Algorithm 10 we construct a protocol Π^{F}_{f} that realises the $\mathcal{F}^{f}_{\Delta\text{-delayed-fairness}}$ functionality presented in Algorithm 7; the protocol Π^{F}_{f} uses as a building block a secure (with abort) two-party protocol in the malicious setting, e.g., the one we presented in Section III. 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 neither party 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 10 uses a weakly trusted (oblivious) third party, captured by the ideal functionality $\mathcal{F}_{\mathsf{Decryption}}$ in Algorithm 8, involved only for resolution in case one of the parties misbehaves. The fair protocol, in Algorithm 10, uses as a building block the $\mathcal{F}_{\mathsf{pre}}$

```
Input: $a$ from Alice, $b$ from Bob, $n$
Computation Phase
    if $a == \perp \vee b == \perp$ then
        send $\perp$ to Alice and to Bob, and halt
    else
        send $y_A = f_A(a, b)$ to Alice
        sleep('wait for response', $\Delta$)
        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 7: 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.

ideal functionality. Specifically, the preprocessing functionality, $\mathcal{F}_{\mathsf{pre}}$, ensures correctness and privacy, and the optimistic ideal functionality $\mathcal{F}_{\mathsf{Decryption}}$, involved during the evaluation phase in case of malicious behaviour, ensures fairness of the computation. The third parties do not learn anything about the inputs or the result of the computation.

To construct Π^{F}_{f} we use, as a module, the ideal two-party computation functionality with output at Bob, as implemented by the protocol Π^{E}_{e} in the ($\mathcal{F}_{\mathsf{pre}}$,$\mathcal{F}^{2}_{\mathsf{ot}}$,$\mathcal{F}_{\mathsf{reg}}$)-hybrid model, in Section III (Algorithm 4). The decryption functionality $\mathcal{F}_{\mathsf{Decryption}}$, generates a key pair $(dk_R, ek_R) \leftarrow \mathcal{NG}(1^n)$ (see Algorithm 8), and this key ek_R is part of the function e. The functionality $\mathcal{F}_{\mathsf{pre}}$ generates a garbled circuit that computes 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:

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

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

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

generate encryption key-pair: $(dk_R, ek_R) \leftarrow \mathcal{NG}(1^n)$
register the encryption key: (register, decryption, $ek_R$) to $\mathcal{F}_{\text{reg}}$
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 decryption functionality $\mathcal{F}_{\text{Decryption}}$

Claim 5.1: 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 ($\mathcal{F}_{\text{Decryption}}$,$\mathcal{F}_{\text{reg}}$,$\mathcal{F}_e$)-hybrid model, with $\Delta = 4\Delta_C$, where Δ_C is the typical channel delay.

Proof: We analyse Π_f^F in a ($\mathcal{F}_{\text{Decryption}}$,$\mathcal{F}_{\text{reg}}$,$\mathcal{F}_e$)-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 5.1 in Propositions 5.2 and 5.3 respectively. ■

Proposition 5.2 (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 $\mathcal{F}_{\text{Decryption}}$, $\mathcal{F}_{\text{reg}}$ 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:

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

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 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 ($\mathcal{F}_{\text{Decryption}}$,$\mathcal{F}_{\text{reg}}$,$\mathcal{F}_e$)-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 5.3 (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 $\mathcal{F}_{\text{pre}}$ and $\mathcal{F}_{\text{reg}}$, there exists a non-uniform polynomial time simulator S corrupting Alice and running in the ideal

```
$S$ generates $(dk, ek) \leftarrow \mathcal{K}(1^n)$ and selects a random key $K_S \in \{0,1\}^n$
$S$ invokes $A$ with input $a, \text{ID}_\text{A}, n$
When $A$ sends (retrieve,resolve) for $\mathcal{F}_{\text{reg}}$, $S$ 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 9: The simulator S running in ideal model with trusted party computing $\mathcal{F}^f_{\Delta\text{-delayed-fairness}}$, and simulating the view of Alice.

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:

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

The proof is omitted due to space restrictions, and appears in the full version of this paper.

A. Efficiency Analysis

In [15], 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 [15] seems to be susceptible to 'inputs corruption' attack, whereby Alice corrupts one of the inputs to oblivious transfer protocol, and based on the behaviour of Bob learns the corresponding value of his input bit. In our protocol, when the parties are honest and follow the steps of the protocol (which is the typical case), the computation complexity is roughly as that of the Yao's original protocol (see Section III-C2 for discussion). When one of the parties misbehaves, the protocol requires an additional round, to send the encrypted result to the $\mathcal{F}_{\text{Decryption}}$ and to receive a decrypted response back.

VI. Applications

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

We believe that these protocols could be particularly useful for financial applications, which typically involve third parties (that often cannot be avoided, and are often trusted much

```
Input: security parameters $n, s$, maximal communication delay $\Delta_C$, $a = [a_i]_{i=1}^n$ from Alice, $b = [b_i]_{i=1}^n$ from Bob
Output: $y = (y_A, y_B)$
Alice and Bob send (retrieve, decryption) to $\mathcal{F}_{\text{reg}}$ and obtain $ek_R$ (each)
Computation Phase
    Alice and Bob do:
        generate secret keys $K_A$ and $K_B$ respectively
        run a protocol $\Pi^E$ (in Algorithm 4 realising $\mathcal{F}_e$) computing $e$ ($e$ is constructed from $f$ as in Equation 1) with key $ek_R$ of $\mathcal{F}_{\text{Decryption}}$, 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 $\mathcal{F}_{\text{Decryption}}$
            onReceive($c_B$)
                stopTimer('time to $\mathcal{F}_{\text{Decryption}}$')
                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 10: Secure Two Party Protocol Π_f^F that realises $\mathcal{F}^f_{\Delta\text{-delayed-fairness}}$ with $\Delta = 4\Delta_C$, in the ($\mathcal{F}_{\text{Decryption}}$,$\mathcal{F}_{\text{reg}}$,$\mathcal{F}_{\text{e}}$)-hybrid model for computing $f(a,b) = (f_A(a), f_B(b))$, where $f : \{0,1\}^m \times \{0,1\}^m \rightarrow \{0,1\}^m \times \{0,1\}^m$; and $e_{ek_R}((a,K_A),(b,K_B)) = \mathcal{NE}_{ek_R}(c_A||c_B)||(\mathcal{E}_{K_A}(f_A(a,b), c_B))$, $c_A = \mathcal{E}_{K_A}(f_A(a,b))$ and $c_B = \mathcal{E}_{K_B}(f_B(a,b))$.

more than needed by our protocols). We suggest to use these 'already present' third parties, to produce efficient and practical systems. We next illustrate such application:

A. Stocks Trade

Consider the following sample application for stocks trade, utilising the protocols presented in this work. Alice wishes to sell IBM stocks and Bob wants to buy them. Alice contacts a third party (which they both trust to some extent) e.g., a broker, to obtain a program for this specific transaction. Both Alice and Bob provide some secret policy, as their respective inputs, and possibly their secret signature keys (which are essential since the algorithm produces signed orders); then the algorithm, constructed by the third party, will produce corresponding signed orders if there is a match between the policies, or $\perp$ if there is no match. Concretely, the protocol satisfies the following requirements: (1) it does not expose the policies (nor the secret inputs, e.g., secret keys); (2) it ensures correctness of computation (since the program, implementing the algorithm, was supplied and signed by a third party); (3) ensures fairness (if, say, Alice aborts after receiving her signed order, Bob can contact the resolver, e.g., a broker, or a clearing house, to recover his order).

B. Currency Exchange

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

VII. Conclusion

Two-party computation received a lot of attention during the last two decades, with numerous works, and although it was shown to be practical (see [3]), there are no 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] A. C. Yao, "How to generate and exchange secrets," in *Proc. 27th IEEE Symp. on Foundations of Comp. Science*. Toronto: IEEE, 1986, pp. 162–167.

[2] 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*, 1987, pp. 218–229.

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

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

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

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

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

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

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

[10] S. 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*. Asiacrypt, Springer-Verlag, 2007, pp. 298–314.

[11] M. Blum, P. Feldman, and S. Micali, "Non-interactive zero-knowledge and its applications," in *Proceedings of the twentieth annual ACM symposium on Theory of computing*. ACM, 1988, pp. 103–112.

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

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

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

[15] C. Cachin and J. Camenisch, "Optimistic fair secure computation," in *Advances in Cryptology – CRYPTO ' 2000*, ser. Lecture Notes in Computer Science, M. Bellare, Ed., vol. 1880, International Association for Cryptologic Research. Springer-Verlag, Berlin Germany, 2000, pp. 93–111.

[16] 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*. London, UK: Springer-Verlag, 2000, pp. 531–545.

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

[18] A. Lindell, "Legally-enforceable fairness in secure two-party computation," in *Proceedings of the 2008 The Cryptopgraphers' Track at the RSA conference on Topics in cryptology*. Springer-Verlag, 2008, pp. 121–137.

[19] D. Gordon and J. Katz, "Partial fairness in secure two-party computation," Cryptology ePrint Archive, Report 2008/206, 2008, Tech. Rep., 2008.

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

[21] Y. Lindell and B. Pinkas, "A Proof of Yao Protocol for Secure Two-Party Computation," in *Electronic Colloquium on Computational Complexity*, vol. 11, 2004, p. 063.

[22] J. Kilian, "Founding crytpography on oblivious transfer," in *Proceedings of the twentieth annual ACM symposium on Theory of computing*. ACM, 1988, pp. 20–31.

[23] J. Katz and R. Ostrovsky, "Round-optimal secure two-party computation," in *Advances in Cryptology–CRYPTO 2004*. Springer, 2004, pp. 3–34.

[24] J. Garay, "Efficient and universally composable committed oblivious transfer and applications," *Theory of Cryptography*, pp. 297–316, 2004.

[25] 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)*, vol. 38, no. 3, pp. 690–728, 1991.

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

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

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

[29] I. Damgard and M. Keller, "Secure multiparty aes," *Financial Cryptography and Data Security*, pp. 367–374, 2010.

[30] I. Damgard and C. Orlandi, "Multiparty computation for dishonest majority: From passive to active security at low cost," *Advances in Cryptology-CRYPTO 2010*, pp. 558–576, 2010.

[31] V. Kolesnikov, A. Sadeghi, and T. Schneider, "Improved garbled circuit building blocks and applications to auctions and computing minima," *Cryptology and Network Security*, pp. 1–20, 2009.

[32] A. Paus, A. Sadeghi, and T. Schneider, "Practical secure evaluation of semi-private functions," in *Applied Cryptography and Network Security*. Springer, 2009, pp. 89–106.

[33] A. Herzberg, M. Jakobsson, S. Jarecki, H. Krawczyk, and M. Yung, "Proactive public key and signature systems," in *Proceedings of the 4th ACM conference on Computer and communications security*. ACM, 1997, pp. 100–110.