

# Calling out Cheaters: Covert Security With Public Verifiability\*

Gilad Asharov

Department of Computer Science,  
Bar-Ilan University, ISRAEL.  
asharog@cs.biu.ac.il

Claudio Orlandi<sup>†</sup>

Department of Computer Science,  
Aarhus University, DENMARK.  
orlandi@cs.au.dk.

December 17, 2012

## Abstract

We introduce the notion of *covert security with public verifiability*, building on the covert security model introduced by Aumann and Lindell (TCC 2007). Protocols that satisfy covert security guarantee that the honest parties involved in the protocol will notice any cheating attempt with some constant probability  $\epsilon$ . The idea behind the model is that the fear of being caught cheating will be enough of a deterrent to prevent any cheating attempt. However, in the basic covert security model, the honest parties are not able to persuade any third party (say, a judge) that a cheating occurred.

We propose (and formally define) an extension of the model where, when an honest party detects cheating, it also receives a *certificate* that can be published and used to persuade other parties, without revealing any information about the honest party's input. In addition, malicious parties cannot create fake certificates in the attempt of framing innocents.

Finally, we construct a secure two-party computation protocol for any functionality  $f$  that satisfies our definition, and our protocol is almost as efficient as the one of Aumann and Lindell. We believe that the fear of a public humiliation or even legal consequences vastly exceeds the deterrent given by standard covert security. Therefore, even a small value of the deterrent factor  $\epsilon$  will suffice in discouraging any cheating attempt. As the overall complexity of covert security and the parameter  $\epsilon$  are inversely proportional to each other, we believe that the small price to pay to get the public verifiability property on top of covert security will be dominated by the efficiency gain obtained by using a smaller value  $\epsilon$ .

**Keywords:** Secure computation; covert security;

---

\*The work was funded by the European Research Council under the European Unions Seventh Framework Programme (FP/2007-2013) / ERC Grant Agreement n. 239868.

<sup>†</sup>This work was carried out while at Bar-Ilan University.

# Contents

<b>1</b>	<b>Introduction</b>	<b>1</b>
<b>2</b>	<b>Definitions</b>	<b>5</b>
2.1	Covert Security . . . . .	5
2.2	Covert Security with Public Verifiability . . . . .	7
2.3	The Formal Definition . . . . .	8
<b>3</b>	<b>Signed Oblivious Transfer</b>	<b>10</b>
3.1	Definitions . . . . .	10
3.2	Constructing an EU-CMRA Signature Scheme . . . . .	12
3.3	PVW-based Signed OT . . . . .	15
3.4	One-out-of- $\ell$ Signed OT . . . . .	17
<b>4</b>	<b>Two-Party Computation with Publicly Verifiable Covert Security</b>	<b>18</b>
4.1	Lemma: Adding Signatures Does Not Break Security . . . . .	22
4.2	Proof of Theorem 4.1. . . . .	24
	<b>References</b>	<b>26</b>

# 1 Introduction

One of the main goals of the theory of cryptographic protocols is to find security definitions that provide the participants with meaningful guarantees and that can, at the same time, be achieved by reasonably efficient protocols. Both standard security notions lack one of these two properties: the level of security offered by *semi-honest secure* protocols is unsatisfactory (as the only guarantee is that security is achieved if all parties follow the protocol specification) while *malicious secure* protocols (that offer security against arbitrarily behaving adversaries) are orders of magnitude slower than semi-honest ones (see e.g., the comparison in [NNOB12]).

In *covert security*, introduced by Aumann and Lindell in 2007 [AL07], the honest parties have the guarantee that if the adversary tries to cheat in order to break some of the security properties of the protocol (correctness, confidentiality, input independence, etc.) then the honest parties will notice the cheating attempt with some constant probability  $\epsilon$ . Here, unlike the malicious model where the adversary cannot cheat at all, the adversary can effectively cheat while taking the risk of being caught. This relaxation of the security model allows protocol designers to construct highly efficient protocols, essentially only a small factor away from the efficiency of semi-honest protocols.

The main justification for covert security is that, in many practical applications, the relationship between the participants of the protocol is such that the fear of being caught cheating is enough of a deterrent to avoid any cheating attempt. For example, two companies that decide to engage in a secure computation protocol might value their reputation and the possibility of future trading with the other company more than the possibility of learning a few bits of information about the other company's input, and therefore have no incentive in trying to cheat in the protocol at all.

However, a closer look at the covert model reveals that the repercussions of a cheating attempt are somewhat limited: Indeed, if Alice tries to cheat, the protocol guarantees that she will be caught by Bob with some predetermined probability, and so Bob will know that Alice is dishonest. Nevertheless, Bob will not be able to bring Alice in front of a judge or to persuade a third party Charlie that Alice cheated, and therefore Alice's reputation will only be hurt in Bob's eyes and no one else. This is due to the fact that Charlie has no way of telling apart the situation where Alice cheated from the situation where Bob is trying to frame Alice to hurt her reputation: Bob can always generate fake transcripts that will be indistinguishable from a real interaction between a cheating Alice and Bob.

This becomes a problem, as the fact that only Bob knows that Alice has tried to cheat may not be enough of a deterrent for Alice. In particular, consider the scenario where there is some social asymmetry between the parties, for instance if a very powerful company engages in a protocol with a smaller entity (i.e., a citizen). If the citizen does not have any clear evidence of the cheating she will not be able to get any compensation for the cheating attempt, as she will not be able to sue the company or persuade any other party of the misbehavior – who would believe her without any proof? This means that if we run a covert protocol between these parties, the fact that a party can detect the cheating may not be enough to prevent the more powerful one from attempting to cheat.

The scenario described above can be dramatically changed if, once a party is caught cheating, the other party receives some undeniable evidence of this fact, and this evidence can be independently verified by any third party. We therefore introduce the notion of *covert security with public verifiability* where if a party is caught cheating, then the honest parties receives a certificate – a small piece of evidence – that can be published and used to prove to all those who are interested that indeed there was a dishonest behavior during the interaction. Clearly, this provides a stronger

deterrent than the one given by covert security.

Intuitively, we want cheating parties to be *accountable* for their actions i.e., if a party cheats then everyone can be persuaded of this fact. At the same time, we need also the system to be *defamation-free* in the sense that no honest parties can be framed i.e., no party can produce a fake cheating certificate.

**Towards better efficiency: choosing the right  $\epsilon$ .** In order to fully understand the benefit of covert-security with public verifiability, consider the utilities of a rational Alice, running a cryptographic protocol with Bob for some task. Let  $(U_h, U_c, U_f, U_f^{pub})$  be real numbers modeling Alice utilities: Alice’s utility is  $U_h$  when she runs the protocol honestly, and so both parties learn the output and nothing else. If Alice attempts to cheat, she will receive utility  $U_c$  if the cheating attempt succeeds. If the cheating attempt fails (i.e., Alice gets caught), the utility received by Alice will be  $U_f$  in the standard covert security setting and  $U_f^{pub}$  in the setting with public verifiability. We assume that  $U_c > U_h > U_f > U_f^{pub}$ , namely, Alice prefers to succeed cheating over the outcome of an honest execution, prefers the latter over being caught cheating, and prefers losing her reputation in the eye of one parties over losing it publicly.

Remember that, since the protocol is  $\epsilon$ -deterrent, whenever Alice attempts to cheat she will be caught with probability  $\epsilon$  and succeed with probability  $1 - \epsilon$ . Therefore, assuming that Bob is honest, Alice’s expected payoff is  $U_h$  when she plays the honest strategy and  $\epsilon \cdot U_f' + (1 - \epsilon) \cdot U_c$  when she plays cheating, with  $U_f' \in \{U_f, U_f^{pub}\}$  depending on whether the protocol satisfies public verifiability or not. Therefore if we set

$$\epsilon > \frac{U_c - U_h}{U_c - U_f'}$$

then Alice will maximize her expected utility by playing honest. This implies that the value of  $\epsilon$  needed to discourage Alice from cheating is much higher in the standard covert security setting than in our framework.

As the value of the deterrent factor  $\epsilon$  determines the replication factor and thus the efficiency of covert secure protocols, we believe that in practice using covert security with public verifiability will lead to an increase in efficiency, as the benefits obtained by the reduced replication factor will exceed the limited price to pay for achieving the public verifiability property on top of the covert secure protocol.

**Main Ideas.** It is clear that no solution to our problem exists in the plain model and that we need to be able to publicly identify parties. We therefore set our study in the public-key infrastructure (PKI) model, where the keys of all parties are registered in some public database. Note that in practice this is not really an additional assumption, as most cryptographic protocols already assume the existence of authenticated point-to-point channels, that can be essentially only implemented by having some kind of PKI and letting the parties sign all the messages they exchange to each other.

At this point it might seem that the problem we are trying to solve is trivial, and that the solution is simply to let all parties sign all the exchanged messages in a covert secure protocol. Here is why this naïve solution does not reach our goal: As a first problem, we need to make sure that the adversary cannot abort as a consequence of being caught cheating; think of a zero-knowledge (ZK) protocol with one bit challenge, where the prover only knows how to answer to a challenge  $c = 0$ . If the verifier asks for  $c = 1$ , the malicious prover has no reason to reply with an invalid proof and will abort instead. Surely, the honest party will suspect the prover of cheating but

will have no certificate to show to a judge. The problem of an adversary aborting as an escape from being caught cheating was already raised in [AL07, Section 3.5], and the solution is to run all the *cut-and-choose* via an oblivious transfer (OT): here the prover (acting as a sender) inputs openings to all possible challenges and the verifier (acting as the receiver) inputs his random challenge. Due to the security of the OT, the prover now cannot choose whether to continue or abort the protocol as a function of the verifier’s challenge. The prover needs to decide in advance whether to take the risk of being caught, or abort before the execution of the OT protocol.

Secondly, we need to ensure that the published certificate does not leak information about the honest party’s input: when the honest party detects cheating, it computes a certificate as a function of its view i.e., the (signed) transcript of the protocol, his input and his random tape. Therefore, this certificate may (even indirectly) leak information about the input of the honest party. This is clearly unsatisfactory and leads us to the following unfortunate situation: a party knows that the other party has cheated, however, in order to prove this fact to the public he is required to reveal to the adversary his private information.

For the sake of concreteness, consider a protocol where Alice chooses a key pair  $(pk, sk)$  for a homomorphic encryption scheme  $E$ , and sends Bob  $(pk, E_{pk}(x))$  where  $x$  is Alice’s input. Later in the protocol, Alice and Bob use the homomorphic properties of  $E$  for a cut-and-choose; i.e., Bob sends the first message of a ZK proof, Alice sends an encrypted challenge  $E_{pk}(c)$  and Bob obliviously computes the last message of the ZK proof for the challenge  $c$ , and signs all the transcripts of the protocol. Alice finally decrypts and checks the validity of the proof. Note that Bob cannot abort as a function of  $c$  (due to the semantic security of the encryption scheme). If Bob cheats and Alice detects it, she receives a proof, a signature on the (encrypted) incriminating messages. Alice can now publish the transcript and her secret key  $sk$  in order to enable the judge to verify that Bob cheated. However, once the certificate is made public, Bob will learn the secret, decrypt the first ciphertext and learn  $x$ .

Moreover, a malicious Alice might have a strategy to compute a different secret key  $sk'$  that makes the signed ciphertext decrypt to some “illegal” message that can be used to frame an innocent Bob. These examples show that things can easily go wrong, and motivates the need for a formal study of covert security with public verifiability.

**Signed oblivious transfer.** As a building block for our construction we introduce a new cryptographic primitive, that we shall call *signed oblivious-transfer*. In this primitive, the sender inputs two message  $(m_0, m_1)$  and a signature key  $sk$ , and the receiver inputs a bit  $b$ . At the end of the protocol, the receiver will learn the message  $m_b$  together with a signature on it, while the sender learns nothing. That is, the receiver learns:  $(m_b, \text{Sign}_{sk}(b, m_b))$ .

To see the importance of this tool in constructing protocols that satisfy covert security with public verifiability it is useful to see how it can be used to fix the problems with the zero-knowledge protocols described before. A very high level description of the signed-OT based zero-knowledge protocol is: (1) First the prover prepares the first message of the zero-knowledge protocol and sends it to the verifier together with a valid signature on it; (2) Now the prover prepares the answers to both challenges  $c = 0$  and  $c = 1$  and inputs them, together with his secret key, to the signed OT; (3) The verifier inputs a random choice bit  $c$  to the signed OT and receives the last message of the zero-knowledge protocol together with a valid signature on it. The verifier checks this message and, if the proof passes the verification, it outputs **accept**. On the other hand, if the proof is invalid, the verifier can take the transcript of the protocol and send them to any third party as an undeniable proof that the prover attempted to cheat.

Note that this works only because  $b$  is included in the signature. Had  $b$  not be signed, the prover could input the simulated opening to both branches of the OT. This makes the (signed) transcript always look legit (in particular, it does not depend on the challenge bit  $b$ ), and the verifier cannot persuade a third party that the prover did not properly answer to his challenge. Also, note that it is not enough to run a standard OT, where the prover inputs  $(m_0, \text{Sign}(0, m_0)), (m_1, \text{Sign}(1, m_1))$ , as in this case the prover could cheat by sending a valid signature on the valid opening, and no signature on the wrong opening – it is crucial for the security of the protocol that the verifier is persuaded that *both* signatures are valid, even if only one is received.

Finally we claim that the example protocol (zero-knowledge using signed-OT) satisfies all the security requirements stated in this introduction: the protocol has *accountability* – the prover never gets to see the verifier’s challenge and therefore cannot choose to abort if he simulated on the wrong branch; it also trivially protects the privacy of the verifier has no input (this will be non-trivial to achieve for generic protocols). Finally, the protocol is *defamation free*: this is due to the unforgeability of the signature produced by the signed-OT.

**Our model.** Our security definition guarantees that when an honest party publishes the certificate, the adversary cannot gain any additional information from this certificate even when it is combined with the adversary’s view, in a strong simulation sense. This, together with the fact that in the *strong explicit cheat formulation* of covert security a cheating party does not learn any information about the honest party’s input and output, guarantees that the certificate does not leak any unintentional information to anyone seeing the certificate (i.e., the certificate can be simulated without the input/output of the honest party).

A covert secure protocol with public verifiability is composed of an “honest” protocol and two extra algorithms to deal with cheating situations: the first is used to produce a certificate when a cheating is detected, and the other to decide whether a certificate is authentic or not. The requirements for the two latter algorithms are the following: any time that an honest party outputs that the other party is corrupted, the evaluation of the verification algorithm on the produced certificate should output the identity of the corrupted party. In addition, no one should be able to produce incriminating certificates against honest parties.

**On the definitional choice and fairness.** In our model, we still have the problem of “fairness” where one of the parties may get the output without the other party (and the honest party does not have a certificate for this case). We remark that the problem of fairness in two party computation is unavoidable in the standard model [Cle86] for general functionalities (even if some, partial, cases where complete fairness is possible have recently been found [GHKL11, GK12, GK09]). In the model of covert security with public verifiability, a judge cannot distinguish between the case where one of the parties aborts prematurely from the case where the protocol was completed correctly and the corrupted party removes the last message before showing the transcript to the judge.

Assuming that the judge is a semi-trusted third party, several solutions to this problem exist (see, e.g. [Mic03]). However in this work we aim to achieve *public* verifiability: the judge can be *anyone* and a certificate of cheating can be published online and verified by anyone.

**Organization and Results.** In Section 2, we define and justify the model of covert security with public verifiability. In Section 3 we show how to construct a signed-OT protocol: our starting point is the very efficient OT protocol due to Peikert, Vaikuntanathan and Waters [PVW08]. The resulting protocol is only slightly less efficient than the protocol of PVW.

Signed-OT will also be the main ingredient in our protocol for two-party secure computation

using Yao’s garbled circuit, described in Section 4. Here we show that for any two party functionality  $f$ , there exists an efficient covert secure protocol with  $\epsilon$ -deterrent and public verifiability. Our protocol is roughly  $1/\epsilon$  slower than a semi-honest secure protocol, and has essentially the same complexity as an  $\epsilon$ -deterrent secure protocol without public verifiability.

Technically, our starting point is the protocol presented in [AL07, Section 6.3] (the variant where aborting is not considered cheating) the only differences with the original protocol are that every call to an OT is replaced by a call to a signed-OT, and that the circuit constructor will also send a few signatures in the right places. We believe that this is a very positive fact as the resulting protocol is only slightly less efficient than the original covert secure protocol, showing how covert security with public verifiability offers a much greater deterrent to cheating than standard covert security (as a cheater can face huge loss in reputation or even legal consequences), while only slightly decreasing the efficiency of the protocol.

**Related Work.** The idea of allowing malicious parties to cheat as long as this is detected with significant probability can be found in several works, e.g. [FY92, IKNP03, MNPS04], and it was first formally introduced under the name of covert security by Aumann and Lindell [AL07]. Since then, several protocols satisfying this definition have been constructed, for instance [HL08, GMS08, DGN10]. It is possible to add the public verifiability property to any of these protocols.<sup>1</sup> Doing so in the most efficient way is left as a future work.

## 2 Definitions

**Preliminaries.** A function  $\mu(\cdot)$  is negligible, if for every positive polynomial  $p(\cdot)$  and all sufficiently large  $n$ ’s it holds that  $\mu(n) < 1/p(n)$ . A probability ensemble  $X = \{X(a, n)\}_{a \in \{0,1\}^*; n \in \mathbb{N}}$  is an infinite sequence of random variables indexed by  $a$  and  $n \in \mathbb{N}$ . Usually, the value  $a$  represents the parties’ inputs and  $n$  the security parameter. Two distributions ensembles  $X = \{X(a, n)\}_{a \in \{0,1\}^*; n \in \mathbb{N}}$  and  $Y = \{Y(a, n)\}_{a \in \{0,1\}^*; n \in \mathbb{N}}$  are said to be computationally indistinguishable, denoted  $X \stackrel{c}{\equiv} Y$ , if for every non-uniform polynomial-time algorithm  $D$  there exists a negligible function  $\mu(\cdot)$  such that for every  $a \in \{0, 1\}^*$  and every  $n \in \mathbb{N}$ ,

$$|\Pr [D(X(a, n)) = 1] - \Pr [D(Y(a, n)) = 1]| \leq \mu(n)$$

We assume the reader to be familiar with the standard definition for secure multiparty computation [Can00, Gol04].

**The PKI model.** As was discussed in the introduction, we work in the PKI model. We describe the PKI functionality here. Intuitively, each party can register its public verification key in the functionality, and anyone can query the functionality to receive any verification key of any other identity.

### 2.1 Covert Security

Aumann and Lindell [AL07] present three possible definitions for this covert security, where the three definitions constitute a strict hierarchy. We adopt the strongest definition that is presented,

---

<sup>1</sup>Note that malicious security implies covert security with public verifiability (see discussion after Definition 3), and that each of these protocols can be turned into a maliciously secure protocol, using the *commit-and-prove* approach. Therefore we see little value in providing a general compiler to add public verifiability to any protocol with covert security. What is interesting, is to add the public verifiability protocol with the smallest possible effort.

**FUNCTIONALITY 1 (The  $\mathcal{F}^{\text{PKI}}$  functionality)**

1. Upon receiving the first message (**Register**,  $id, v$ ) from party  $P$ , check that  $id = P$  and this is the first request from  $P$ . If so, record the pair  $(P, v)$ .
2. Upon receiving a message (**Retrieve**,  $id$ ) from party  $P'$ , check that there is a recorded pair  $(id, v)$ . If so, output (**Retrieve**,  $sid, v$ ) to  $P'$ . Else output (**Retrieve**,  $id, \perp$ ) to  $P'$ .

which is called “strong explicit cheat formulation” (Section 3.4 in [AL07]). A protocol that is secure with respect to this definition is also secure with respect to the two other suggested definitions. The formal definition is reported for completeness, almost verbatim from [AL07].

**Strong Explicit Cheat Formulation.** As in the standard secure computation definition, we compare a real execution of the protocol to an execution in the ideal model. The real execution is the same as in the standard definition, however, the ideal model is changed in order to allow the adversary to successfully cheat or to be caught cheating. Let  $\epsilon : \mathbb{N} \rightarrow [0, 1]$  be a function. This is the “deterrent” value, as a function of the security parameter  $n$ , and denotes the probability that the adversary succeeds to cheat once it attempts to cheat. Informally, in this stronger formulation, the adversary may choose to input a special input **cheat** to the ideal functionality. The ideal functionality will then flip a coin and with probability  $(1 - \epsilon)$  will give to the adversary full control: the adversary will learn the honest party’s input and instruct the functionality to deliver any output of its choice. However, with probability  $\epsilon$ , the ideal functionality will inform the honest party of the cheating attempt by sending him a special symbol **corrupted**, and crucially, the adversary will not learn any information about the honest party’s input. The modified ideal model is as follows:

**Inputs:** Each party obtains an input; the  $i$ th party’s input is denoted by  $x_i$ . We assume that all inputs are of the same length, denoted  $n$ . The adversary receives an auxiliary-input  $z$ .

**Send inputs to trusted party:** Any honest party  $P_j$  sends its received input  $x_j$  to the trusted party. The corrupted parties, controlled by  $\mathcal{A}$ , may either send their received input, or send some other input of the same length to the trusted party. This decision is made by  $\mathcal{A}$  and may depend on the values  $x_i$  for  $i \in I$  and auxiliary input  $z$ . Denote the vector of inputs sent to the trusted party by  $\vec{w}$ .

**Abort options:** If a corrupted party sends  $\vec{w} = \text{abort}_i$  to the trusted party as its input, then the trusted party sends  $\text{abort}_i$  to all the honest parties and halts. If a corrupted party sends  $\vec{w} = \text{corrupted}_i$ <sup>2</sup> to the trusted party as its input, then the trusted party sends  $\text{corrupted}_i$  to all of the honest parties and halts. If multiple parties sends  $\text{abort}_i$  (resp.  $\text{corrupted}_i$ ), then the trusted party disregards all but one of them (say, the one with the smallest index,  $i$ ). If both  $\text{corrupted}_i$  and  $\text{abort}_i$  messages are sent, then the trusted party ignores the  $\text{corrupted}_i$  message.

**Attempted cheat option:** If a corrupted party sends  $w_i = \text{cheat}_i$  to the trusted party as its input, then the trusted party works as follows:

1. With probability  $\epsilon$ , the trusted party sends  $\text{corrupted}_i$  to the adversary and all of the honest parties.

---

<sup>2</sup>This corresponds to “blatant cheating” and it is needed for the simulation.

2. With probability  $1 - \epsilon$ , the trusted party sends **undetected** to the adversary along with the honest parties inputs  $\{x_j\}_{j \notin I}$ . Following this, the adversary sends the trusted party output values  $\{y_j\}_{j \notin I}$  of its choice for the honest parties. Then, for every  $j \notin I$ , the trusted party sends  $y_j$  to  $P_j$ .

The ideal execution then ends at this point.

If no  $w_i$  equals **abort** <sub>$i$</sub> , **corrupted** <sub>$i$</sub>  or **cheat** <sub>$i$</sub> , the ideal execution continues below.

**Trusted party answers adversary:** The trusted party computes  $(y_1, \dots, y_m) = f(\vec{w})$  and sends  $y_i$  to  $\mathcal{A}$  for all  $i \in I$ .

**Trusted party answers honest parties:** After receiving its outputs, the adversary sends either **abort** <sub>$i$</sub>  for some  $i \in I$ , or **continue** to the trusted party. If the trusted party receives **continue** then it sends  $y_j$  for all honest parties  $P_j$  (where  $j \notin I$ ). Otherwise, if it receives **abort** <sub>$i$</sub>  for some  $i \in I$ , it sends **abort** <sub>$i$</sub>  to all honest parties.

**Outputs:** An honest party always outputs the message it obtained from the trusted party. The corrupted parties outputs nothing. The adversary  $\mathcal{A}$  outputs any arbitrary (probabilistic) polynomial-time computation function of the initial inputs  $\{x_i\}_{i \in I}$ , the auxiliary inputs  $z$ , and the messages obtained from the the trusted party.

Denote by  $\text{IDEALC}_{f,S(z),I}^\epsilon(\vec{x}, n)$  the outputs of the honest parties and the adversary in an execution in the ideal world as described above, and let  $\text{REAL}_{\pi,\mathcal{A}(z),I}(\vec{x}, n)$  denotes the outputs of the honest parties and the adversary in a real execution of the protocol. We are now ready to define security of protocols in the presence of covert adversary with  $\epsilon$ -deterrent.

**Definition 1 (Security in the presence of covert adversary)** *Let  $f$ ,  $\pi$  and  $\epsilon$  be as above. Protocol  $\pi$  is said to securely compute  $f$  in the presence of covert adversaries with  $\epsilon$ -deterrent if for every non-uniform probabilistic polynomial-time adversary  $\mathcal{A}$  for the real model, there exists a non-uniform probabilistic polynomial-time adversary  $\mathcal{S}$  for the ideal model such that for every  $I \subseteq [m]$ :*

$$\left\{ \text{IDEALC}_{f,S(z),I}^\epsilon(\vec{x}, n) \right\}_{\vec{x}, z \in (\{0,1\}^*)^{m+1}, n \in \mathbb{N}} \stackrel{c}{\equiv} \left\{ \text{REAL}_{\pi,\mathcal{A}(z),I}(\vec{x}, n) \right\}_{\vec{x}, z \in (\{0,1\}^*)^{m+1}, n \in \mathbb{N}}$$

In our definition, we require that the protocol will be **non-halting detection accurate**, namely that the adversary cannot abort as a consequence of being caught cheating. This is formalized by using **fail-stop** adversaries, who act semi-honestly, except that it may halt prematurely.

**Definition 2** *A protocol  $\pi$  is non-halting detection accurate if for every fail-stop adversary  $\mathcal{A}$  controlling parties  $I \subseteq [m]$ , for every honest party  $j \notin I$ , the probability that  $P_j$  outputs **corrupted** <sub>$i$</sub>  where  $i \in I$  is negligible.*

## 2.2 Covert Security with Public Verifiability

We start with an informal discussion motivating our choices and then proceed to the actual definition. For the sake of simplicity, we will present the definition and the motivation for the two-party case. The definition can be easily extended to the multi-party case.

**Motivation:** As discussed in the introduction, we work in the  $\mathcal{F}^{\text{PKI}}$ -hybrid model where each party  $P_i$  registers a verification key  $vk_i$  for a signature scheme. This key will be used to uniquely identify a party. Note that we do not require parties to prove knowledge of their secret keys (i.e., the simulator will not know these secret keys), so this is the weakest  $\mathcal{F}^{\text{PKI}}$  formulation possible [BCNP04].

We extend the covert security model of Aumann and Lindell [AL07] and enhance it with the public verifiability property: As in covert security, if the adversary chooses to cheat it will be caught with probability  $\epsilon$ , and the honest party outputs **corrupted**. However, in this latter case, the protocol in addition provides this party an algorithm **Blame** to distil a *certificate* from its view in the protocol. A third party who wants to verify the cheating (“the judge”) should take the certificate and decide whether the certificate is authentic (i.e., some cheater has been caught) or it is a fake (i.e., someone is trying to frame an innocent). The verification is performed using an additional algorithm, which is called **Judgement**. We require the verification procedure to be *non-interactive*, which will enable the honest party to send the certificate to a judge or to publish it on a public “wall of shame”.

In addition, as our interest is mainly to protect the interest of the honest party, we want to make sure that the certificate of cheating does not reveal any unnecessary information to the verifier. Therefore, we cannot simply publish the view (transcript and random tape) of the honest party, as those might reveal some information about the input or output of the honest party. Moreover, we need to remember that the adversary sees the certificate once it is published and therefore we should take care that no one will be able to learn any meaningful information from this certificate, even when combining it with the adversary’s view. To capture this fact, we use the convention that when a party detects a cheating, it creates the certificate and sends it to the adversary.

The fact that the certificate is part of the view of the adversary means that the simulator needs to include this certificate as a part of the view when it receives **corrupted** from the ideal functionality. Remember that in this case the simulator does not learn anything from the trusted party rather than the adversary got caught, and therefore this implies that our definition ensures that the certificate cannot reveal the private information of the honest party.

Until now, we talked about the certificate that the honest party publishes, and its privacy. However, the main goal of this certificate is to enable others to verify that indeed there was a cheating attempt. Therefore, the designer of the protocol should construct, in addition to the protocol specification, the **Judgement** algorithm. This algorithm takes the certificate and decides whether it is an authentic one or not. We require two properties from the **Judgement** algorithm: whenever an honest party outputs **corrupted**, running the algorithm on the certificate will output the identity of the corrupted party. Moreover, no adversary (even interacting with polynomially many honest parties) can produce a certificate for which the verification algorithm outputs the identity of an honest party.

### 2.3 The Formal Definition

Let  $f$  be a two party functionality. We consider the triple  $(\pi, \text{Blame}, \text{Judgement})$ . The algorithm **Blame** gets as input the view of the honest party (in case of cheat detection) and outputs a certificate  $Cert$ . The verification algorithm, **Judgement**, takes as input a certificate  $Cert$  and outputs the identity  $id$  (for instance, the verification key) of the party to blame or *none* in the case of an invalid certificate.

**The protocol:** Let  $\pi$  be a two party protocol. If an honest party detects a cheating in  $\pi$  then the honest party is instructed to compute  $Cert = \text{Blame}(\text{view})$  and send it to the adversary.

Let  $\text{REAL}_{\pi, \mathcal{A}(z), i^*}(x_1, x_2; 1^n)$  denote the output of the honest party and the adversary on a real execution of the protocol  $\pi$  where  $P_1, P_2$  are invoked with inputs  $x_1, x_2$ , the adversary is invoked with an auxiliary input  $z$  and corrupts party  $P_{i^*}$  for some  $i^* \in \{1, 2\}$ .

**The ideal world.** The ideal world is exactly as Definition 1. Let  $\text{IDEAL}_{\pi, \mathcal{A}(z), i^*}(x_1, x_2)$  denote the output of the honest party, together with the output of the simulator, on an ideal execution with the functionality  $f$ , where  $P_1, P_2$  are invoked with inputs  $x_1, x_2$ , respectively, the simulator  $\mathcal{S}$  is invoked with an auxiliary input  $z$  and the corrupted party is  $P_{i^*}$ , for some  $i^* \in \{1, 2\}$ .

**Notations.** Let  $\text{EXEC}_{\pi, \mathcal{A}(z)}(x_1, x_2; r_1, r_2; 1^n)$  denote the messages and the outputs of the parties in an execution of the protocol  $\pi$  with adversary  $\mathcal{A}$  on auxiliary input  $z$ , where the inputs of  $P_1, P_2$  are  $x_1, x_2$ , respectively, and the random tapes are  $(r_1, r_2)$ . Let  $\text{EXEC}_{\pi, \mathcal{A}(z)}(x_1, x_2; 1^n)$  denote the probability distribution of  $\text{EXEC}_{\pi, \mathcal{A}(z)}(x_1, x_2; r_1, r_2)$  where  $(r_1, r_2)$  are chosen uniformly at random. Let  $\text{OUTPUT}(\text{EXEC}_{\pi, \mathcal{A}(z)}(x_1, x_2))$  denote the output of the honest party in the execution described above. We are now ready to define the security properties.

**Definition 3 (covert security with  $\epsilon$ -deterrent and public verifiability)** *Let  $f, \pi, \text{Blame}$  and  $\text{Judgement}$  be as above. We say that  $(\pi, \text{Blame}, \text{Judgement})$  securely computes  $f$  in the presence of a covert adversary with  $\epsilon$ -deterrent and public verifiability if the following conditions hold:*

1. **(Simulatability with  $\epsilon$ -deterrent:)** *The protocol  $\pi$  (where the honest party broadcasts  $Cert = \text{Blame}(\text{view})$  if it detects cheating) is secure against a covert adversary according to the strong explicit cheat formulation with  $\epsilon$ -deterrent (see Definition 1) and non-halting detection accurate (Definition 2).*
2. **(Accountability:)** *For every PPT adversary  $\mathcal{A}$  corrupting party  $P_{i^*}$  for  $i^* \in \{1, 2\}$ , there exists a negligible function  $\mu(\cdot)$  such that for all sufficiently large  $x_1, x_2, z \in (\{0, 1\}^*)^3$  the following holds:*

*If  $\text{OUTPUT}(\text{EXEC}_{\pi, \mathcal{A}(z), i^*}(x_1, x_2; 1^n)) = \text{corrupted}_{i^*}$  then:*

$$\Pr[\text{Judgement}(Cert) = id_{i^*}] > 1 - \mu(n)$$

*where  $Cert$  is the output certificate of the honest party in the execution.*

3. **(Defamation-Free:)** *For every PPT adversary  $\mathcal{A}$  controlling  $i^* \in \{1, 2\}$  and interacting with the honest party, there exists a negligible function  $\mu(\cdot)$  such that for all sufficiently large  $x_1, x_2, z \in (\{0, 1\}^*)^3$ :*

$$\Pr[Cert^* \leftarrow \mathcal{A}; \text{Judgement}(Cert^*) = id_{3-i^*}] < \mu(n)$$

**Every Malicious Secure Protocol is also Covert Secure with Public Verifiability.** As a sanity check, we note that any protocol that is secure against malicious adversaries satisfies all of the above requirements, with deterrence factor  $\epsilon = 1 - \text{negl}(n)$ : aborting is the only possible malicious behavior. Therefore the function  $\text{Blame}$  will never be invoked and the function  $\text{Judgement}$  outputs *none* on every input. In other words, given that no cheating strategy can succeed except with negligible probability, we have that by definition no one ever “cheats” and no one can be “framed”.

### 3 Signed Oblivious Transfer<sup>3</sup>

As discussed in the introduction, signed oblivious transfer (signed OT) is one of the main ingredient in our construction. For the sake of presentation, one can think of signed OT as a protocol implementing the following functionality:

$$(\lambda; (m_b, \text{Sign}_{sk}(b, m_b))) \leftarrow \mathcal{F}((m_0, m_1, sk), (b, vk)) \quad (1)$$

where  $\lambda$  denotes the empty string. However it turns out that while this formulation certainly suffices for our goal, it is not necessary for our secure two-party computation protocol in Section 4. In particular, we will define a relaxed version of the signed OT functionality, that allows corrupted senders and receivers to influence up to some extent the randomness used in the generation of the signatures: a malicious sender will be allowed to choose any two strings  $(\sigma_0^*, \sigma_1^*)$  and input them to the functionality. If  $(\sigma_0^*, \sigma_1^*)$  are valid signatures on the messages  $(0, m_0)$  and  $(1, m_1)$  respectively, the functionality delivers  $(m_b, \sigma_b^*)$  to the receiver or abort otherwise. A malicious receiver will be able to specify the randomness used by the signature algorithms, and this is going to be fine when using a signature scheme with the extra property that unforgeability holds also against chosen message and partially chosen randomness attacks. We call such signature schemes *existentially unforgeable under adaptive chosen message and randomness attack* (EU-CMRA). The functionality is given in Functionality 2.

The idea behind our signed-OT protocol is very simple: we take the OT protocol proposed in [PVW08, HL10] and let the sender sign the only message he sends to the receiver, and therefore our signed-OT protocol is essentially as efficient as the original OT protocol.<sup>4</sup>

Essentially this can be seen as a signature of two encryptions, where the receiver knows only one of the two secret keys. The receiver has an influence in generating the “public keys” while the sender has an influence in generating the “ciphertexts”. The combined signature consists of a signed ciphertext plus the secret key. As long as the encryption scheme is a binding commitment, a corrupted receiver will not be able to forge the signature by creating a different secret key that makes the ciphertext decrypt to a different value. The rest of the section is devoted to formally develop this intuition.

#### 3.1 Definitions

We start by defining a signature scheme that is existentially unforgeable under adaptive chosen message and randomness attack (EU-CMRA for short). The definition is parametrized by a function  $\rho : \{0, 1\}^* \times \{0, 1\}^* \rightarrow \{0, 1\}^*$  where the first argument is a truly random tape and the second argument is the randomness chosen by adversary. The output is the randomness used by the signature scheme. The function describes how much control the adversary has over the randomness

---

<sup>3</sup>*Editorial Note:* In the proceeding version of this article there is a technical flaw, namely Lemma 1 is wrong and Functionality 1 does not fully capture the “insecurities” of our protocol. Here we fix this by allowing a malicious sender control the randomness in the generation of the signature in the signed-OT functionality. Therefore we need a stronger definition of security for the signature scheme, that resists chosen message and randomness attack. Finally, we modify the signed-OT protocol by letting the receiver choose the “CRS” and prove that it is not a DDH tuple (while in the proceeding version the sender chooses the “CRS” and proves that it is a DDH tuple). These changes have essentially no impact on the way the signed-OT functionality is used later in Section 4.

<sup>4</sup>Given that the focus of this paper is on efficiency, we chose to keep the protocol simple and efficient at the price of a more involved functionality (and proof of security). It is an interesting open problem to see how efficiently the functionality described in (1) can be implemented.

**FUNCTIONALITY 2 (The Signed OT Functionality –  $\mathcal{F}_{\Pi}^{\text{SignedOT}}$ )**

The functionality is parameterized by a  $\rho$ -EU-CMRA signature scheme  $\Pi = (\text{Gen}, \text{Sign}, \text{Vrfy})$ .

**Inputs:** The receiver inputs  $(vk, b, r^*)$  – a verification key together with a bit  $b \in \{0, 1\}$  and a random tape  $r^*$ . The input of the sender is  $(m_0, m_1, sk, \sigma_0^*, \sigma_1^*)$ . An honest receiver is restricted to input  $r^* = \perp$  and an honest sender is restricted to input  $(\sigma_0^*, \sigma_1^*) = (\perp, \perp)$ .

**Output:** If no parties are corrupted the functionality computes  $\sigma = \text{Sign}_{sk}(b, m_b; r)$  (with truly random coins  $r$ ) and verifies that  $\text{Vrfy}_{vk}((b, m_b), \sigma) = 1$ . It then outputs  $(m_b, \sigma)$  to the receiver or **abort** in case where the verification fails.

*(Corrupted Receiver)* If  $r^* \neq \perp$  the functionality computes  $\sigma = \text{Sign}_{sk}(b, m_b; \rho(r, r^*))$ , and verifies that  $\text{Vrfy}_{vk}((b, m_b), \sigma) = 1$ . It then outputs  $(m_b, \sigma)$  to the receiver or **abort** in case where the verification fails.

*(Corrupted Sender)* If  $(\sigma_0^*, \sigma_1^*) \neq (\perp, \perp)$  the functionality outputs  $(m_b, \sigma_b^*)$  to the receiver if  $\text{Vrfy}_{vk}((0, m_0), \sigma_0^*) = 1$  and  $\text{Vrfy}_{vk}((1, m_1), \sigma_1^*) = 1$  or **abort** otherwise.

of the signature scheme. When  $\rho(r, r^*) = r^*$  the adversary has full control over the random tape. When  $\rho(r, r^*) = r$  the definition is equivalent to standard EU-CMA. Jumping forward to our construction we will use  $\rho(r, r^*) = r || r^*$ , that is the final signature scheme will use some maliciously chosen coins and some truly random ones.

**$\rho$ -EU-CMRA Signature Scheme.** Let  $\Pi = (\text{Gen}, \text{Sign}, \text{Vrfy})$  be a tuple of three PPT algorithms with the standard syntax for signature schemes:

- The key-generation algorithm  $\text{Gen}$  takes as input a security parameter  $1^n$  and outputs a pair of keys  $(pk, sk)$ .
- The signing algorithm  $\text{Sign}$  takes as input a private key  $sk$ , a random tape  $r$ , a message  $m \in \{0, 1\}^*$  and outputs a signature  $\sigma$ .
- The deterministic verification algorithm  $\text{Vrfy}$  takes as input a public key  $pk$ , a message  $m$  and a signature  $\sigma$ . It outputs 1 when  $\sigma$  is a valid signature for  $m$ , and 0 otherwise.

Our security requirements are stronger than the standard definition for signature schemes. In particular, we require unforgeability even when the adversary has influence on the randomness that is used to produce the signature. We define the oracle  $\text{Sign}'_{sk}(m, \rho(r, r^*))$  as follows: On input  $(m, r^*)$  from the adversary, the  $\text{Sign}'$  oracle chooses uniformly random string  $r$  and returns  $\sigma = \text{Sign}_{sk}(m; \rho(r, r^*))$ . That is, the adversary has the ability to choose part of the the random tape of the signature algorithm, where the function  $\rho$  determines the exact influence of the adversary. We proceed to the formal definition:

**Definition 4** *We say that a signature scheme  $\Pi = (\text{Gen}, \text{Sign}, \text{Vrfy})$  is  $\rho$  existentially unforgeable under an adaptive chosen-message and randomness attack (or  $\rho$ -EU-CMRA for short) if any PPT  $\mathcal{A}$  adversary succeeds in the following experiment, denoted as  $\text{Sig-forge}_{\mathcal{A}, \Pi}^{\text{CMRA}}(n)$ , with at most negligible probability:*

1.  $\text{Gen}(1^n)$  is run to obtain keys  $(pk, sk)$ .

2. Adversary  $\mathcal{A}$  is given  $pk$  and an oracle access to  $\text{Sign}'_{sk}(\cdot, \cdot)$  as defined above. The adversary then outputs a pair  $(m, \sigma)$ . Let  $\mathcal{Q}$  denote the set of messages whose signatures were requested by  $\mathcal{A}$  during its execution.
3. The output of the experiment is defined to be 1 iff (1)  $\text{Vrfy}_{pk}(m, \sigma) = 1$ , and (2)  $m \notin \mathcal{Q}$ .

When  $\rho(r, r^*) = r$  then the definition collapses to EU-CMA. We will call all-EU-CMRA a signature scheme that is  $\rho$ -EU-CMRA secure when  $\rho(r, r^*) = r^*$ .

In the following we will show a non-trivial example of a signature scheme that is unforgeable even against an adversary that can influence some of the randomness.

**Binding Commitments.** We use the standard non-interactive commitment scheme definition. A commitment scheme is a tuple of three PPT algorithms  $\text{Com} = (\text{Setup}, \text{Com}, \text{Open})$  with the following syntax. The setup algorithm takes as input a security parameter  $1^n$  and outputs a public commitment key  $ck$ . The commitment algorithm  $(c, d) \leftarrow \text{Com}_{ck}(m)$  takes the commitment key  $ck$ , the message  $m$ , and outputs the commitment  $c$  together with the opening value  $d$ . The  $\text{Open}_{ck}(c, d)$  algorithm takes the commitment key  $ck$ , the commitment  $c$  and the opening value  $d$  (note that  $d$  does not need to be random tape used by  $\text{Com}$ ). It then outputs either a message  $\tilde{m}$  or a special symbol  $\perp$  if  $d$  is not a valid commitment to any message.

Regarding security, we consider the binding property only. Roughly speaking, it is infeasible, even to an all-powerful adversary  $\mathcal{A}$ , to come up with a triple  $(r, c, d, d')$  such that  $ck = \text{Setup}(1^n, r)$  and  $(c, d)$  and  $(c, d')$  are valid commitments for some valid  $m, m'$  under commitment key  $ck$ , where  $m \neq m'$ . Formally, the binding experiment, denoted as  $\text{Binding}_{\text{Com}, \mathcal{A}}(1^n)$  is defined as follows:

1. The adversary  $\mathcal{A}$  is given  $ck$ , and outputs  $r, c, d, d'$ .
2.  $\text{Setup}(1^n; r)$  is run to obtain  $ck$ .
3. The output of the experiment is 1 iff  $\text{Open}_{ck}(c, d) = m$ ,  $\text{Open}_{ck}(c, d') = m'$ ,  $m \neq m'$  and both are not  $\perp$ .

A commitment scheme is unconditionally binding if for any (even unbounded) adversary  $\mathcal{A}$ , the probability that the adversary wins in  $\text{Binding}_{\text{Com}, \mathcal{A}}(1^n)$  is negligible.

An important observation is that in the  $\text{Binding}$  game, the adversary has a complete control on the randomness of the commitment. This means that even if the adversary computes  $\text{Com}_{ck}$  on some message  $m$  where it chooses the randomness of the commitment and the commitment key (as long as the commitment key is generated correctly), still it is hard to find a collusion. This observation will help us to build an EU-CMRA signature scheme.

### 3.2 Constructing an EU-CMRA Signature Scheme

Here we construct a (somewhat contrived) signature scheme, designed to combine efficiently with the OT protocol, where the adversary does have the ability to influence the randomness of the signature scheme.

Our signature scheme combines any EU-CMA signature scheme  $\Pi' = (\text{Gen}', \text{Sign}', \text{Vrfy}')$  with any statistically binding commitment  $\text{Com} = (\text{Setup}, \text{Com}, \text{Open})$  (we do not need the commitment to be hiding) into a new signature scheme that will be  $\rho$ -EU-CMRA with  $\rho(r, r^*) = (r || r^*)$  (in particular, the signature scheme will use the truly random coins while the commitment scheme will

be fed the adversarially chosen coins). We further present an explicit commitment scheme that is used in our protocol, while the original signature scheme  $\Pi'$  can be an arbitrary EU-CMA signature

**Construction 1 (Combined signature scheme)** *Given an unforgeable signature scheme  $\Pi' = (\text{Gen}', \text{Sign}', \text{Vrfy}')$  and a commitment scheme  $\text{Com} = (\text{Setup}, \text{Com}, \text{Open})$ , we define the signature scheme  $\Pi = (\text{Gen}, \text{Sign}, \text{Vrfy})$  as follows:*

- **The Gen algorithm:** *On input  $1^n$ , invoke  $\text{Gen}'(1^n)$  and receive  $(vk, sk)$ . Output  $(vk, sk)$ .*
- **The Sign algorithm:** *On input message  $m$  a key  $sk$  and random tape  $r, r^*$ , where we parse  $r^* = (r_1^*, r_2^*)$ , the algorithm chooses a commitment key  $ck$  by invoking  $ck \leftarrow \text{Setup}(1^n, r_1^*)$ . Then, it computes the commitment  $(c, d) = \text{Com}_{ck}(m; r_2^*)$ . Finally, it outputs:*

$$\sigma = (ck, d, c, \text{Sign}'_{sk}(ck, c; r))$$

- **The Vrfy algorithm:** *On input  $(m, \sigma)$  and a verification key  $vk$ , parse  $\sigma$  as  $(ck, d, c, \sigma')$ . Output 1 iff  $\text{Open}(c, d) = m$  and  $\text{Vrfy}'_{vk}((ck, c), \sigma') = 1$ .*

Unforgeability of the combined scheme follows from the unforgeability of the original scheme together with the binding property of the commitment scheme. We now prove the following Claim:

**Claim 1** *If the signature scheme  $\Pi' = (\text{Gen}', \text{Sign}', \text{Vrfy}')$  is existentially unforgeable under an adaptive chosen-message attack and  $\text{Com}$  is an unconditionally binding commitment scheme (EU-CMA), then  $\Pi = (\text{Gen}, \text{Sign}, \text{Vrfy})$  is  $\rho$ -EU-CMRA as in Definition 4, with  $\rho(r, r^*) = (r || r^*)$ .*

**Proof:** Assume that there exists an adversary  $\mathcal{A}$  and a non-negligible function  $\epsilon(\cdot)$  such that for infinitely many  $n$ 's it holds that:

$$\Pr [\text{Sig-forge}_{\mathcal{A}, \Pi}^{\text{CMRA}}(n) = 1] \geq \epsilon(n)$$

We construct an adversary  $\mathcal{A}'$  for  $\Pi'$  that succeeds with the Sig-forge experiment (for EU-CMA) with some non-negligible probability.

The adversary  $\mathcal{A}'$  is invoked with the verification key  $vk$ . It then invokes the adversary  $\mathcal{A}$  and gives it  $vk$  (note that both  $\Pi$  and  $\Pi'$  have the same public key and verification key, however, the signing and verification algorithms are different). Whenever  $\mathcal{A}$  queries its signing oracle for a message  $m$  with randomness  $r^*$ ,  $\mathcal{A}'$  invokes  $\text{Setup}(1^n; r_1^*)$  receives  $ck$ , computes  $(c, d) \leftarrow \text{Com}_{ck}(m; r_2^*)$  and queries its oracle on the message  $(ck, c)$ . It then receives the signature  $\sigma$  and replies to  $\mathcal{A}'$  with  $(ck, d, c, \sigma)$ . Moreover, it stores  $(ck, c)$  in the list  $\mathcal{Q}_{\mathcal{A}'}$  and the message  $m$  in the list  $\mathcal{Q}_{\mathcal{A}}$ . When  $\mathcal{A}$  outputs  $(m, \sigma)$ ,  $\mathcal{A}'$  verifies that  $\text{Vrfy}_{vk}(m, \sigma) = 1$  and that  $m \notin \mathcal{Q}_{\mathcal{A}}$ . If so, it then parses  $\sigma$  as  $(ck, d, c, \sigma')$  and checks that  $(ck, c) \notin \mathcal{Q}_{\mathcal{A}'}$ . If the check passes – it outputs  $((ck, c), \sigma')$ . Otherwise, it outputs  $\perp$ .

The only case where  $\text{Sig-forge}_{\mathcal{A}, \Pi}^{\text{CMRA}}(n)$  equals 1 while  $\text{Sig-forge}_{\mathcal{A}', \Pi'}$  equals 0, is when  $\text{Vrfy}_{vk}(m, \sigma) = 1$ ,  $m \notin \mathcal{Q}_{\mathcal{A}}$  but  $(ck, c) \in \mathcal{Q}_{\mathcal{A}'}$ . We now show that this happens with at most some negligible probability. Let  $\sigma = (ck, d, c, \sigma')$  and assume that  $m \notin \mathcal{Q}_{\mathcal{A}}$  but  $(ck, c) \in \mathcal{Q}_{\mathcal{A}'}$ . This means that  $\mathcal{A}$  has queried its oracle before on some message  $m' \neq m$  (with some randomness  $r', r^*$ , respectively), and received back the signature  $(ck, d', c, \sigma')$ , where  $(c, d') = \text{Com}_{ck}(m')$ . This implies that  $\text{Open}_{ck}(c, d) = m$ , and  $\text{Open}_{ck}(c, d') = m'$ . However, from the binding property of the commitment scheme, this can happen only with some negligible probability. We conclude that

the probability that  $\mathcal{A}$  succeeds in the experiment is negligibly-close to the probability that  $\mathcal{A}'$  succeeds in its experiment, and thus  $\mathcal{A}'$  succeeds with some non-negligible probability. This is in contradiction to the assumption that  $\Pi'$  is unforgeable signature scheme.  $\blacksquare$

**The commitment scheme.** We present the commitment scheme that we use in the above template. Let  $(\mathbb{G}, q)$  be a prime order group.

The commitment scheme is based on some variant of ElGamal encryption (see [PVW08]) and, like in ElGamal encryption, there are two ways of encrypting/committing and opening: one using the secret key and the other using the randomness for the encryption. (In the signed OT protocol, the standard mode will be used in the real world while the procedure using the secret key will be used in the ideal world). The two modes are perfectly equivalent (they give rise to an identical distribution). The commitment scheme is as follows:

**Construction 2** *The commitment scheme  $\text{Com} = (\text{Setup}, \text{Com}, \text{Open})$  is defined as follows:*

- **The setup algorithm  $\text{Setup}$ :** *On input security parameter  $1^n$ , the setup chooses a non-DDH tuple  $(g_0, h_0, g_1, h_1)$  in  $\mathbb{G}$  and defines  $ck = (g_0, h_0, g_1, h_1)$ . Concretely, the algorithm chooses a random  $\alpha \in \mathbb{Z}_q$ , random  $g_0, g_1 \in \mathbb{G}$  and compute  $h_0 = g_0^\alpha$  and  $h_1 = g_1^{\alpha^{-1}}$*
- **The commitment algorithm  $\text{Com}_{ck}$ :** *On input message  $(b, m) \in \{0, 1\} \times \mathbb{G}$  and randomness  $(r_1, \dots, r_3, s_1, \dots, s_7) \in \mathbb{Z}_q^3 \times \mathbb{G}^7$  it checks if  $r_1 = \alpha$  (i.e., check if  $h_0 = g_0^\alpha$  and  $h_1 = g_1^{\alpha^{-1}}$ ).*
  1. *If yes (compute the commitment using the secret key): let  $(g, h) = (s_1, s_2)$ . Check that  $b \in \{0, 1\}$  satisfies  $h = g^{\alpha-b}$  and abort if it does not. If it does, let  $(u_{1-b}, w_{1-b}) = (s_3, s_4)$ ,  $u_b = g_b^{r_2}$  and  $w_b = m \cdot g^{r_2}$ . Then, it defines  $c = (g, h, u_0, w_0, u_1, w_1)$  and the decommitting value  $d = ((r_1, r_2); (b, m))$ .*
  2. *If no (compute the commitment “normally”): compute  $(g, h) = (g_b, h_b)^{r_3}$ ,  $u_b = s_5$ ,  $w_b = m \cdot u_b^{r_3}$ , and  $(u_{1-b}, w_{1-b}) = (s_6, s_7)$ . Then, it defines  $c = (g, h, u_0, w_0, u_1, w_1)$  and the decommitting value  $d = (r_3; (b, m))$ .*
- **The opening algorithm  $\text{Open}_{ck}(c, d)$ :** *On input key  $ck = (g_0, h_0, g_1, h_1)$ , commitment  $c = (g, h, u_0, w_0, u_1, w_1)$  and parse the decommitting value  $d = (r; (b, m))$ :*
  1. *If  $r \in \mathbb{Z}_q^2$  parse  $r = (r_1, r_2)$  and check that  $g_i = h_i^{r_1^{-i}}$  for both  $i \in \{0, 1\}$  and that  $h = g^{r_1-b}$ . Then check that  $u_b = g_b^{r_2}$  and  $w_b = m \cdot g^{r_2}$ . If so, outputs  $(b, m)$ .*
  2. *If  $r \in \mathbb{Z}_q$  check that  $(g, h) = (g_b, h_b)^r$  and  $w_b = m \cdot u_b^r$ . If so it outputs  $(b, m)$ .*
  3. *If none of the two checks passed, output  $\perp$ .*

**Remark 1:** Note that if the randomness for  $\text{Com}$  is chosen uniformly at random, branch 1 will be used only with negligible probability.

**Remark 2:** Note that branches 1 and 2 of  $\text{Com}$  use disjoint parts of the random tape.

**Remark 3:** Note that in the opening algorithm, the values  $(u_{1-b}, w_{1-b})$  are never used. However, they are signed during the signed-OT protocol and therefore they are included in the definition of the commitment scheme, although they are not necessary for the binding property.

**Claim 2** *The scheme  $(\text{Setup}, \text{Com}, \text{Open})$  is unconditionally binding.*

**Proof:** First remember that  $ck$  is not a DDH tuple (in our definition of the binding game the adversary can choose the random coins used by **Setup** and not directly the commitment key  $ck$ , this ensures that  $ck$  is wellformed). In fact:

$$ck = (g_0, h_0, g_1, h_1) = (g_0, g_0^\alpha, g_1, g_1^{\alpha-1})$$

and therefore there cannot exist  $r'_{3,0}, r'_{3,1}$  s.t.  $(g, h) = (g_i, h_i)^{r'_{3,i}}$  for both  $i \in \{0, 1\}$ . Therefore  $b$  is uniquely defined to be the value such that there exists a  $r'_3$  for which  $(g, h) = (g_b, h_b)^{r'_3}$ . This implies that  $m$  is uniquely defined as the value  $m = w_b \cdot u_b^{-r'_3}$ . This shows that when the commitment key is well formed there is a unique value that makes **Open** accepts on the second branch.

When the **Open** algorithm takes the first branch, there is at most one value  $r'_1$  such that  $h_0 = g_0^{r'_1}$  and  $h_1 = g_1^{r'_1-1}$ . Let  $\delta \in \mathbb{Z}_q$  be s.t.  $h = g^\delta$ . As  $\delta$  can either be  $\delta = r'_1$  or  $\delta = r'_1 - 1$ , but not both at the same time, this defines uniquely  $b$ . In turns, this defines the unique value  $r'_2$  s.t.,  $u_b = g_b^{r'_2}$  and therefore the only value  $m$  s.t.,  $m = w_b \cdot g^{-r'_2}$ .

We showed that both branches accept at most one message  $(b, m_b)$ . But the two messages are actually the same due to the following bijection between  $r_2$  and  $r_3$ : let  $z \in \mathbb{Z}_q$  be the unique value s.t.  $u_b = g_b^z$ , then  $z = r_2 \cdot r_3$ . ■

Let  $\Pi' = (\text{Gen}', \text{Sign}', \text{Vrfy}')$  be any signature scheme. Our PVW compatible signature scheme  $\Pi = (\text{Gen}, \text{Sign}, \text{Vrfy})$  is the a combination of the signature scheme  $\Pi'$  and the commitment scheme **Com** as defined in Construction 2. From the combination of Claim 1 and Claim 2 we conclude:

**Corollary 1** *If  $\Pi' = (\text{Gen}', \text{Sign}', \text{Vrfy}')$  is an existentially unforgeable under an adaptive chosen-message attack signature scheme then  $\Pi = (\text{Gen}, \text{Sign}, \text{Vrfy})$  is an EU-CMRA signature scheme.*

### 3.3 PVW-based Signed OT

We present the protocol for signed OT in Protocol 1, combining the PVW OT protocol with the signature scheme described above. Note that the protocol is just the DDH-based instantiation of the PVW OT framework (as described in [HL10, Protocol 7.5.1]) with the only difference that the sender signs the transfer messages it sends to the receiver.

We present the protocol in the  $(\mathcal{F}_{\text{ZK}}^{\text{DDH}})$ -hybrid model. The  $\mathcal{F}_{\text{ZK}}^{\text{DDH}}$  is the zero-knowledge proof of membership for DH-tuple, which formally, the functionality receives from both parties the common input  $x = (\mathbb{G}, q, g_0, h_0, g_1, h_1)$ , and in addition, it receives from the prover the witness  $\alpha$ . The functionality sends 1 to the verifier iff  $g_1 = g_0^\alpha$  and  $h_1 = g_1^\alpha$  (i.e, that  $x$  is a DDH-tuple). (Remember that we want to prove that  $g_0, h_0, g_1, h_1$  is *not* a DDH tuple, but given the way that the tuple is constructed this is equivalent to show that  $(g_0, h_0, g_1, h_1 \cdot g_1)$  is a DDH tuple. The functionality  $\mathcal{F}_{\text{ZK}}^{\text{DDH}}$  can be implemented efficiently using sigma protocols, see [HL10].

Note that the **Com** algorithm is *distributed*, in the sense that both parties contribute to the input and randomness: in particular the receiver chooses  $b$  while the sender specifies  $(m_0, m_1)$  without knowing which message is going to be chosen.

Define the randomized function:

$$\text{RAND}(g_0, h_0, g_1, h_1) = (u, v), \quad \text{where } u = (g_0)^s \cdot (h_0)^t, \quad v = (g_1)^s \cdot (h_1)^t \text{ and } s, t \in_R \mathbb{Z}_q.$$

Observe that if  $(g_0, h_0, g_1, h_1)$  is a DDH tuple for some  $x$  (i.e, there exists an  $x$  such that  $g_1 = g_0^x$  and  $h_1 = h_0^x$ ) then  $u$  is distributed at random in  $\mathbb{G}$  and  $v = u^x$ . In case where  $(g_0, h_0, g_1, h_1)$  is not

a DDH tuple (i.e,  $\log_{g_0} g_1 \neq \log_{h_0} h_1$ ) then the pair  $(u, v)$  is distributed uniformly at random in  $\mathbb{G}^2$ . See [PVW08] for more details.

**Lemma 1** *Let  $\Pi = (\text{Gen}, \text{Sign}, \text{Vrfy})$  be the PVW-compatible signature scheme defined above. Then, Protocol 1 securely implements the  $\mathcal{F}_{\Pi}^{\text{SignedOT}}$ -functionality in the  $\mathcal{F}_{\text{ZK}}^{\text{DDH}}$ -hybrid model in the presence of a malicious, static adversary.*

**PROTOCOL 1 (Signed  $\binom{2}{1}$ -OT Protocol in the  $\mathcal{F}_{\text{ZK}}^{\text{DDH}}$ -hybrid model)**

**Setup:** This step can be done once and reused for multiple runs of the OT: The receiver  $R$  chooses  $(g_0, g_1) \in_R \mathbb{G}^2$ , a random  $\alpha \in_R \mathbb{Z}_q$  and compute  $h_0 = g_0^\alpha$  and  $h_1 = g_1^{\alpha^{-1}}$ . The receiver sends  $(\mathbb{G}, q, g_0, h_0, g_1, h_1)$  to the receiver  $R$  and using the  $\mathcal{F}_{\text{ZK}}^{\text{DDH}}$ -functionality proves that  $(g_0, h_0, g_1, h_1 g_1)$  is a DDH tuple.

**Choose:**  $R$  chooses random  $r \in_R \mathbb{Z}_q$ , computes  $g = (g_b)^r$ ,  $h = (h_b)^r$  and sends  $(g, h)$  to  $S$ ;

**Transfer:** The sender operates in the following way:

1.  $S$  computes  $(u_0, v_0) = \text{RAND}(g_0, g, h_0, h)$  and  $(u_1, v_1) = \text{RAND}(g_1, g, h_1, h)$ ;
2.  $S$  sends  $R$  the values  $(u_0, w_0)$  where  $w_0 = v_0 \cdot m_0$ , and  $(u_1, w_1)$  where  $w_1 = v_1 \cdot m_1$ ;
3. **(diff)**  $S$  sends to the receiver

$$\sigma' = \text{Sign}'_{sk'}((g_0, h_0, g_1, h_1), (g, h, u_0, w_0, u_1, w_1));$$

**Retrieve:** **(diff)** Let  $vk = vk'$ .  $R$  checks that  $\sigma'$  is a valid signature on the transcript of the protocol. If so,  $R$  outputs:  $m_b = w_b \cdot (u_b)^{-r}$  and **(diff)**

$$\sigma = ((g_0, h_0, g_1, h_1), (r; (b, m_b)), (g, h, u_0, w_0, u_1, w_1), \sigma') .$$

Otherwise, it outputs **abort**.

**Proof Sketch:** The proof of security of the underlying OT protocol is by now standard and can be found in [PVW08, HL10].

As discussed in Corollary 1,  $\sigma$  is a proper signature on the message  $(b, m_b)$ , and therefore the correct functionality is implemented when both parties are honest.

When the receiver is corrupted, the simulator  $\mathcal{S}$  plays as an honest sender and interacts with the adversary  $\mathcal{A}$ . At the step “Setup”, the simulator extracts the witness  $\alpha$  from the  $\mathcal{F}_{\text{ZK}}^{\text{DDH}}$  functionality.

The corrupted receiver  $\mathcal{A}$  replies with the tuple  $(g, h)$ . The simulator finds  $1 - b$  s.t.  $h \neq g^{\alpha^{-1-b}}$  and inputs  $b, r^*$  and  $vk$  to the ideal functionality on behalf of the corrupted receiver. In particular, the simulator constructs

$$r^* = (r_1, r_2, r_3, s_1, s_2, s_3, s_4, s_5, s_6, s_7)$$

with  $r_1 = \alpha$ ,  $r_2 \in_R \mathbb{Z}_q$ ,  $r_3 = \perp$ ,  $(s_1, s_2) = (g, h)$ ,  $(s_3, s_4) \in_R \mathbb{G}^2$ ,  $(s_5, s_6, s_7) = (\perp, \perp, \perp)$ . That is, the simulator makes sure that the **Com** algorithm takes the first branch. It receives back  $m_b$  and a signature  $\sigma = (ck, d, c, \sigma')$ .  $\mathcal{S}$  parses  $c$  as  $(g, h, u_0, w_0, u_1, w_1)$ , and sends back to the adversary  $\mathcal{A}$  the values  $(u_0, w_0, u_1, w_1)$  and the signature  $\sigma'$ .

The fact that the view of a corrupted receiver is indistinguishable in the real world and in the simulation follows from the security of the underlying OT protocol (see [HL10]) and that the signature generated by the ideal functionality is distributed identically in the real and ideal world.

In the case of a corrupted sender, the simulator chooses  $(g_0, h_0, g_1, h_1)$  to be a DDH tuple (let  $\beta$  be the values such that  $g_1 = g_0^\beta$  and  $h_1 = h_0^\beta$ ), and “cheat” the sender by controlling the zero-knowledge functionality  $\mathcal{F}_{\text{ZK}}^{\text{DDH}}$ . Then it plays like an honest receiver (with  $b = 1$ ). It chooses random  $r$  as instructed in the protocol, and computes  $(g, h) = (g_1, h_1)^r = (g_0, h_0)^{\beta \cdot r}$  and receives from the adversary  $\mathcal{A}$  the message  $(u_0, w_0, u_1, w_1)$  together with the signature  $\sigma'$ .

Observe that  $(u_0, v_0) = \text{RAND}(g_0, g, h_0, h) = \text{RAND}(g_0, g_0^{\beta \cdot r}, h_0, h_0^{\beta \cdot r})$  and thus  $v_0 = u_0^{\beta \cdot r}$ . This implies that the simulator can decrypt both  $m_0$  and  $m_1$ .

Thus, when the sender sends  $(u_0, w_0, u_1, w_1)$  the simulator can extract both messages  $m_0, m_1$  as follows:  $m_0 = w_0/u_0^{\beta \cdot r}$  and  $m_1 = w_1/u_1^r$ . Then, it computes the two signatures  $\sigma_0^*, \sigma_1^*$  as follows:

$$\begin{aligned}\sigma_0^* &= ((g_0, h_0, g_1, h_1), (\beta \cdot r, (0, m_0)), (g, h, u_0, w_0, u_1, w_1), \sigma') \\ \sigma_1^* &= ((g_0, h_0, g_1, h_1), (r, (1, m_1)), (g, h, u_0, w_0, u_1, w_1), \sigma')\end{aligned}$$

In order to see that these are valid signatures on  $(0, m_0), (1, m_1)$  respectively, recall that  $(g, h) = (g_1, h_1)^r = (g_0, h_0)^{\beta \cdot r}$ . This implies that  $\beta \cdot r$  is a valid opening of  $c$  for  $(0, m_0)$  whereas  $r$  is the opening of  $c$  for  $(1, m_1)$ .

Finally, the distribution of the constructed signatures are the same as in the real execution except that in the real world  $(g_0, h_0, g_1, h_1)$  is not a DDH tuple while in the simulation it is a DDH tuple. It is straightforward to show that an adversary that if the environment distinguished between the real and the simulated world, it can be used to break the DDH assumption.  $\blacksquare$

### 3.4 One-out-of- $\ell$ Signed OT

Like the original OT protocol [PVW08], our signed OT protocol can be extended in the straightforward way to an 1-out-of- $\ell$  signed OT. The definition of the functionality and the protocol are changed accordingly, see Functionality 3 and Protocol 2.

**FUNCTIONALITY 3 (The Signed 1-out-of- $\ell$  OT Functionality –  $\mathcal{F}_{\Pi}^{\text{SignedOT}}$ )**  
The functionality is parameterized by a  $\rho$ -EU-CMRA signature scheme  $\Pi = (\text{Gen}, \text{Sign}, \text{Vrfy})$ .

**Inputs:** The receiver inputs  $(vk, b, r^*)$  – a verification key together with an index  $b \in [\ell]$  and a random tape  $r^*$ . The input of the sender is  $(\{m_i\}_{i \in [\ell]}, sk, \{\sigma_i^*\}_{i \in [\ell]})$ . An honest receiver is restricted to input  $r^* = \perp$  and an honest sender is restricted to input  $\{\sigma_i^*\}_{i \in [\ell]} = \perp$ .

**Output:** If no parties are corrupted the functionality computes  $\sigma = \text{Sign}_{sk}(b, m_b; r)$  (with truly random coins  $r$ ) and verifies that  $\text{Vrfy}_{vk}((b, m_b), \sigma) = 1$ . It then outputs  $(m_b, \sigma)$  to the receiver or **abort** in case where the verification fails.

*(Corrupted Receiver)* If  $r^* \neq \perp$  the functionality computes  $\sigma = \text{Sign}_{sk}(b, m_b; \rho(r, r^*))$ , and verifies that  $\text{Vrfy}_{vk}((b, m_b), \sigma) = 1$ . It then outputs  $(m_b, \sigma)$  to the receiver or **abort** in case where the verification fails.

*(Corrupted Sender)* If  $\{\sigma_i^*\}_{i \in [\ell]} \neq \perp$  the functionality outputs  $(m_b, \sigma_b^*)$  to the receiver if  $\text{Vrfy}_{vk}((i, m_i), \sigma_i^*) = 1$  for all  $i \in [\ell]$  or **abort** otherwise.

**PROTOCOL 2 (Signed  $\binom{\ell}{1}$ -OT Protocol in the  $\mathcal{F}_{\text{ZK}}^{\text{DDH}}$ -hybrid model)**

**Setup:** This step can be done once and reused for multiple runs of the OT: The receiver  $R$  chooses  $\{g_i\}_{i \in [\ell]} \in_R \mathbb{G}^2$ , a random  $\alpha \in_R \mathbb{Z}_q$  and compute  $\{h_i = g_i^{\alpha-i+1}\}_{i \in [\ell]}$ . The receiver sends  $(\mathbb{G}, g, \{g_i, h_i\}_{i \in [\ell]})$  to the receiver  $R$  and using the  $\mathcal{F}_{\text{ZK}}^{\text{DDH}}$ -functionality proves, for all  $i = 2, \dots, \ell$  that  $(g_1, h_1, g_i, h_i(g_i)^{i-1})$  is a DDH tuple.

**Choose:**  $R$  chooses random  $r \in_R \mathbb{Z}_q$ , computes  $g = (g_b)^r$ ,  $h = (h_b)^r$  and sends  $(g, h)$  to  $S$ ;

**Transfer:** The sender operates in the following way:

1.  $S$  computes for all  $i \in [\ell]$ ,  $(u_i, v_i) = \text{RAND}(g_i, g, h_i, h)$ ;
2.  $S$  sends  $R$  the values  $\{(u_i, w_i)\}_{i \in [\ell]}$  where  $w_i = v_i \cdot m_i$ ;
3. **(diff)**  $S$  sends to the receiver

$$\sigma' = \text{Sign}'_{sk'}(\{(g_i, h_i)\}_{i \in [\ell]}, (g, h), \{(u_i, w_i)\}_{i \in [\ell]});$$

**Retrieve:** **(diff)** Let  $vk = vk'$ .  $R$  checks that  $\sigma'$  is a valid signature on the transcript of the protocol. If so,  $R$  outputs:  $m_b = w_b \cdot (u_b)^{-r}$  and **(diff)**

$$\sigma = (\{(g_i, h_i)\}_{i \in [\ell]}, (r; (b, m_b)), (g, h, \{(u_i, w_i)\}_{i \in [\ell]}), \sigma').$$

Otherwise, it outputs **abort**.

## 4 Two-Party Computation with Publicly Verifiable Covert Security

The protocol is an extension of the two party protocol of [AL07], which is based on Yao's garbled circuit protocol for secure two-party computation. We will start with an informal discussion of the ways that a malicious adversary can cheat in Yao's protocol<sup>5</sup> and we will present the (existing) countermeasures to make sure that such attacks will be detected with significant probability, thus leading to covert security. Finally we will describe how to add the public verifiability property on top of this. The ways that a malicious adversary can cheat in Yao's protocol are as follows:

1. **Constructing bad circuits:** To prevent  $P_1$  from constructing a circuit that computes a function different than  $f$ ,  $P_1$  constructs  $\ell$  independent garbled circuits and  $P_2$  checks  $\ell - 1$  of them. Therefore if  $P_1$  cheats in the construction of the circuits,  $P_2$  will notice this with probability  $> 1 - 1/\ell$ . To make sure  $P_1$  cannot abort if it is challenged on an incorrect circuit, we run the cut-and-choose through a 1-out-of- $\ell$  signed OT, so that  $P_2$  will always receive some (signed) opening of the circuits that can be used to prove a cheating attempt to a third party.
2. **Selective failure attack on  $P_2$ 's input values:** When  $P_2$  retrieves its keys (using the OT protocol),  $P_1$  may take a guess  $g$  at one of the inputs bits of  $P_2$ . Then, it may use some string  $r$  instead of the valid key  $k_{1-g}$ , as input to the OT protocol. Now, in case where that  $P_1$  guesses correctly and indeed the input bit equals  $g$ ,  $P_2$  receives  $k_g$  and does not notice that

<sup>5</sup>We assume the reader to be familiar with Yao's garbled circuit protocol. See [LP09] for more details and full proof of security.

there was anything wrong. However, in case the guess is incorrect,  $P_1$  receives  $r$  instead of  $k_{1-g}$  which is an invalid key and thus it aborts. In both cases, the way  $P_2$  reacts completely reveals this input bit. This problem can be fixed by computing a different circuit, where  $P_2$ 's input is an  $m$ -out-of- $m$  linear secret sharing of each one of the input bits of  $P_2$ . Now every  $m - 1$  input bits of  $P_2$  to the protocol are uniformly random and therefore  $P_2$  will get caught with probability  $1 - 2^{-m+1}$  if it attempts to guess (the encoding of) an input bit. By using a signed OT we will ensure that  $P_2$  receives a certificate on the wrong keys if  $P_1$  cheats.

Let  $\text{Com}$  denote a perfectly-binding commitment scheme, where  $\text{Com}(x; r)$  denotes a commitment to  $x$  using randomness  $r$ .  $(\text{Gen}_{\text{ENC}}, \text{Enc}, \text{Dec})$  is a semantically secure symmetric encryption scheme.  $(\text{Gen}, \text{Sign}, \text{Vrfy})$  is an existentially unforgeable signature scheme under an adaptive chosen-message attack. Note that it is crucial that every message is signed together with some extra-information about the role of this message (i.e., with unique identifiers for the parties executing the protocols, the instance of the protocol, which type of message in the protocol, which gate/wire label is the message associated too etc.) but we will neglect these extra information in the description of our protocol for the sake of readability.

### PROTOCOL 3 [Two-Party Secure Computation in the $\mathcal{F}^{\text{PKI}}$ -hybrid model]

**Inputs:** Party  $P_1$  has input  $x_1$  and Party  $P_2$  has input  $x_2$ , where  $|x_1| = |x_2|$ . In addition, both parties have parameters  $\ell$  and  $m$ , and a security parameter  $n$ . For simplicity, we will assume that the length of the inputs are  $n$ . (**diff**) Party  $P_1$  knows a secret key  $sk$  for a signature scheme and  $P_2$  received the corresponding verification key  $vk$  from the  $\mathcal{F}^{\text{PKI}}$ .

**Auxiliary input:** Both parties have the description of a circuit  $C$  for inputs of length  $n$  that computes the function  $f$ . The input wires associated with  $x_1$  are  $w_1, \dots, w_n$  and the input wires associated with  $x_2$  are  $w_{n+1}, \dots, w_{2n}$ .

#### The Protocol<sup>6</sup>:

1. Parties  $P_1$  and  $P_2$  define a new circuit  $C'$  that receives  $m + 1$  inputs  $x_1, x_2^1, \dots, x_2^m$  each of length  $n$ , and computes the function  $f(x_1, \oplus_{i=1}^m x_2^i)$ . Note that  $C'$  has  $n + mn$  input wires. Denote the input wires associated with  $x_1$  by  $w_1, \dots, w_n$ , and the input wires associated with  $x_2^i$  by  $w_{n+(i-1)m}, \dots, w_{n+im}$  for  $i = 1, \dots, m$ .
2.  $P_2$  chooses  $m - 1$  strings  $x_2^1, \dots, x_2^{m-1}$  uniformly and independently at random from  $\{0, 1\}^n$ , and defines  $x_2^m = (\oplus_{i=1}^{m-1} x_2^i) \oplus x_2$ , where  $x_2$  is  $P_2$ 's original input. Observe that  $\oplus_{i=1}^m x_2^i = x_2$ .
3. For each  $i = 1, \dots, mn$  and  $\beta = 0, 1$ , party  $P_1$  chooses  $\ell$  encryption keys by running  $\text{Gen}_{\text{ENC}}(1^n)$  for  $\ell$  times. Denote the  $j$ th key associated with a given  $i$  and  $\beta$  by  $k_{w_{n+i}, \beta}^j$ .
4.  $P_1$  and  $P_2$  invoke  $mn$  times the (**diff**)  $\mathcal{F}_{\Pi}^{\text{SignedOT}}$  functionality with the following inputs: In the  $i$ th execution, party  $P_1$  inputs the pair:

$$\left( \left[ k_{w_{n+i}, 0}^1, \dots, k_{w_{n+i}, 0}^\ell \right], \left[ k_{w_{n+i}, 1}^1, \dots, k_{w_{n+i}, 1}^\ell \right] \right)$$

and party  $P_2$  inputs the bit  $x_2^i$  ( $P_2$  receives the keys  $\left[ k_{w_{n+i}, x_2^i}^1, \dots, k_{w_{n+i}, x_2^i}^\ell \right]$  and a signature on this as output). If  $P_2$  output in the OT is  $\text{abort}_i$ , then it outputs  $\text{abort}_i$  and halts.

<sup>6</sup>The description of the protocol is almost verbatim from [AL07] to help the reader identify the few (clearly marked) differences between our protocol and the original protocol.

5. Party  $P_1$  constructs  $\ell$  garbled circuits  $GC_1, \dots, GC_\ell$  using independent randomness for the circuit  $C'$  described above. The keys for the input wires  $w_{n+1}, \dots, w_{n+m}$  in the garbled circuits are taken from above (i.e., the keys associated with  $w_{n+i}$  are  $k_{w_{n+i},0}^j$  and  $k_{w_{n+i},1}^j$ ). The keys for the inputs wires  $w_1, \dots, w_n$  are chosen randomly, and are denoted in the same way.  $P_1$  sends the  $\ell$  garbled circuits to  $P_2$  (**diff**) together with a signature on those.
6.  $P_1$  commits to the keys associated with its inputs. That is, for every  $i = 1, \dots, n$ ,  $\beta = 0, 1$  and  $j = 1, \dots, \ell$ , party  $P_1$  computes (**diff**):

$$c_{w_i,\beta}^j = \text{Com} \left( k_{w_i,\beta}^j; r_{i,\beta}^j \right), \sigma_{w_i,\beta}^j = \text{Sign}_{sk}(c_{w_i,\beta}^j)$$

The commitments and the signatures are sent as  $\ell$  vectors of pairs (one vector for each circuit); in the  $j$ th vector the  $i$ th pair is  $\{(c_{w_i,0}^j, \sigma_{w_i,0}^j), (c_{w_i,1}^j, \sigma_{w_i,1}^j)\}$  in a random order (the order is randomly chosen independently for each pair). (**diff**) Party  $P_2$  verifies that all the signatures are correct. If not, it halts and outputs  $\text{abort}_1$ .

7.  $P_2$  chooses a random index  $\gamma \in_R \{1, \dots, \ell\}$ .
8. (**diff**)  $P_1$  and  $P_2$  engage in a  $\binom{\ell}{1}$ -signed OT where  $P_1$  is the sender and  $P_2$  is the receiver. The input of  $P_2$  is  $\gamma$  and, the  $i$ th input of  $P_1$  (for  $i = 1, \dots, \ell$ ) is all of the keys for the inputs wires in all garbled circuits except for  $GC_i$ , together with the associated mappings and the decommitment values.  $P_1$  sends also decommitments to the input keys associated with its input for the circuit  $GC_i$ .

$P_2$  receives the openings for  $\ell - 1$  circuits (all but  $GC_\gamma$ ) together with a signature on them.  $P_2$  receives also the decommitments and the keys associated with  $P_1$ 's input for circuit  $GC_\gamma$  together with signatures on them. If any of the signatures are incorrect, it halts and outputs  $\text{abort}_1$ .

9.  $P_2$  checks that:
  - That the keys it received for all  $GC_j$ ,  $j \neq \gamma$ , indeed decrypt the circuits and the decrypted circuits are all  $C'$ . (**diff**) If not, add  $\text{key} = \text{wrongCircuit}$  to its view.
  - That the decommitment values correctly open all the commitments  $c_{w_i,\beta}^j$  that were received, and these decommitments reveal the keys  $k_{w_i,\beta}^j$  that were sent for  $P_1$ 's wires. (**diff**) If not, add  $\text{key} = \text{wrongDecommitment}$  to its view.
  - That the keys received in the signed OT in Step 4 match the appropriate keys that it received in the opening. (**diff**) If not, add  $\text{key} = \text{selectiveOTattack}$  to its view.

If all check pass, proceed to the next step, else (**diff**),  $P_2$  computes  $\text{Cert} = \text{Blame}(\text{view}_2)$  (see the description of **Blame** for its output on different key values), it publishes  $\text{Cert}$  and output  $\text{corrupted}_1$ .

10.  $P_2$  checks that the values received are valid decommitments to the commitments received above. If not, it outputs  $\text{abort}_1$ . If yes, it uses the keys to compute  $C'(x_1, z_2) = C'(x_1, x_2^1, \dots, x_2^m) = C(x_1, x_2)$ , and outputs the result.

**ALGORITHM 1 (The Blame Algorithm – Blame)****Input:** The view of a honest party, containing an error tag key.**Output:** A certificate  $Cert = (id, key, message, \sigma)$ .**The Algorithm:**

- **Case 1: key = wrongCircuit:** Let  $j$  be the smallest index s.t. the garbled circuit  $GC_j$  is not a garbling of  $C'$ . Let **message** be the commitment to  $GC_j$  concatenated with the opening obtained via the  $\binom{\ell}{1}$ -signed OT in Step 8, and  $\sigma$  the signature on these messages.
- **Case 2: key = wrongDecommitment:** Let **message** be  $(c, x, r)$  be a commitment where  $c \neq \text{Com}(x; r)$  and  $\sigma$  the signatures on  $c$  and  $(x, r)$ .
- **Case 3: key = selectiveOTattack:** let **message** be a garbled circuit  $GC_i$  and two keys to one of its input gates. Let  $\sigma$  be the signature on the circuit and the signatures on the keys obtained in Step 8.

On any other case, output  $\perp$ .**ALGORITHM 2 (The Public Verification Algorithm – Judgement)****Input:** A certificate  $Cert = (id, key, message, \sigma)$ .**Output:** The identity  $id$  or  $none$ .**The Algorithm:** If  $\sigma$  is not a valid signature on the message **message** according to verification key  $vk_{id}$  halt and output  $none$ . Else:

- **Case 1: key = wrongCircuit:** Parse **message** as a garbled circuit  $GC$  and the randomness  $r$  used to generate it. If  $GC$  is not an encryption of the circuit computing  $C'$  using randomness  $r$  output  $id$  or  $none$  otherwise.
- **Case 2: key = wrongDecommitment:** Parse **message** as  $(c, x, r)$ . If  $c \neq \text{Com}(x; r)$  output  $id$  or  $none$  otherwise.
- **Case 3: key = selectiveOTattack:** Parse **message** as a circuit  $GC$  and two keys  $k^i, k^j$  for an input gate  $g$  of the circuit  $GC$ . If  $k^i, k^j$  do not decrypt the gate  $g$  output  $id$  or  $none$  otherwise.

**Theorem 4.1** *Let  $\ell$  and  $m$  be parameters in the protocol that are both upper-bound by  $\text{poly}(n)$ , and set  $\epsilon = (1 - 1/\ell)(1 - 2^{-m+1})$ , and let  $f$  be a probabilistic polynomial-time function and let  $\pi$  denote Protocol 3. Then, assuming the DDH assumption, security of the commitment scheme, signature scheme and symmetric encryption scheme as described above,  $(\pi, \text{Blame}, \text{Judgement})$  securely computes  $f$  in the presence of covert adversaries with  $\epsilon$ -deterrent and public verifiability (i.e, satisfies Definition 3).*

Note that even for very small replication factors this construction gives reasonable level of deterrence factor e.g.,  $\ell = 3$  and  $m = 3$  lead to  $\epsilon = 50\%$ .

Before proceeding to the proof, we first state an important lemma: we show a general transformation, where we take some protocol  $\pi$  in the  $OT$ -hybrid model and convert it to a protocol  $\pi'$  in the  $\mathcal{F}_{\Pi}^{\text{SignedOT}}$ -hybrid model where each party simply sign each message it sends and verifies each message it receives. We show that the if  $\pi$  is secure in the presence of covert adversary with

$\epsilon$ -deterrent, then  $\pi'$  is also secure in the presence of covert adversary with  $\epsilon$ -deterrent, as well. We remark that Protocol 3 is essentially the same as the one of [AL07], with the only exception that each party signs his message and any OT is replaced with Signed-OT. Therefore, this lemma alone proves almost all we need, where the missing part for the simulation is only the simulation of the certificate in case of cheat detection.

#### 4.1 Lemma: Adding Signatures Does Not Break Security

In this section we show a general transformation, where we take some protocol  $\pi$  in the  $OT$ -hybrid model and covert it to a protocol  $\pi'$  in the  $\mathcal{F}_{\Pi}^{\text{SignedOT}}$ -hybrid model where each party simply sign each message it sends and verifies each message it receives. We show that the if  $\pi$  is secure in the presence of covert adversary with  $\epsilon$ -deterrent, then  $\pi'$  is also secure in the presence of covert adversary with  $\epsilon$ -deterrent, as well.

More formally, let  $f$  be a two party functionality, and let  $\pi$  be a protocol that implements  $f$  in the  $OT$ -hybrid model. We build a protocol  $\pi'$  in the  $(\mathcal{F}_{\Pi}^{\text{SignedOT}}, \mathcal{F}^{\text{PKI}})$  hybrid model as follows:

- A party  $P$  has identity  $id$ , holds an input  $x$  and the identity of the other party  $id^*$ .
- At the first step,  $P$  runs  $\text{Gen}_{\text{Sign}}(1^n)$  and receives back  $(sk_{id}, vk_{id})$ .  $P$  invokes the  $\mathcal{F}^{\text{PKI}}$  functionality with command  $(\text{Register}, id, vk_{id})$ .
- The party  $P_i$  queries the  $\mathcal{F}^{\text{PKI}}$  functionality for the verification key of the other party. That is, it sends  $(\text{Retrieve}, id^*, vk_{id^*})$ , and receives back  $vk_{id^*}$ .
- The party runs the protocol  $\pi$  with input  $x$ , controlling party  $P$  in  $\pi$ .
  - Whenever  $P$  should send a message  $m$  in the protocol  $\pi$ , it sends  $(m, \text{Sign}_{sk_{id}}(m))$  instead.
  - Whenever a party  $P$  receives a pair  $(m, \sigma)$ , it verifies that  $\text{Vrfy}_{vk_{id^*}}(m, \sigma) = 1$ . If the verification passes, it relates to  $m$  as the incoming message in  $\pi$  and computes the next message accordingly. If the verification fails, it aborts and outputs  $\text{abort}_{id^*}$ .
  - Whenever the parties in  $\pi$  invoke the  $OT$ -functionality, the parties invoke the  $\mathcal{F}_{\Pi}^{\text{SignedOT}}$  functionality instead where:
    - \* If  $P$  is the sender, it sends to the  $\mathcal{F}_{\Pi}^{\text{SignedOT}}$  in addition to the inputs  $(m_0, m_1)$  the secret key  $sk_{id}$  and pair of empty signatures  $(\perp, \perp)$ .
    - \* If  $P$  is the receiver, it sends to  $\mathcal{F}_{\Pi}^{\text{SignedOT}}$  in addition to the input  $b$  the verification key  $vk_{id^*}$  and randomness  $\perp$ .
- $P$  outputs the output of  $P$  in  $\pi$ .

**Lemma 2** *Let  $f, \pi, \pi'$  be as above. If  $\pi$  securely computes  $f$  in the presence of a covert adversary with  $\epsilon$ -deterrent in the  $OT$ -hybrid model, then  $\pi'$  also securely computes  $f$  with in the presence of a covert adversary with  $\epsilon$ -deterrent in the  $(\mathcal{F}^{\text{PKI}}, \mathcal{F}_{\Pi}^{\text{SignedOT}})$ -hybrid model.*

**Proof:** We start with an adversary  $\mathcal{A}'$  for  $\pi'$  and then convert it to an adversary  $\mathcal{A}$  for the protocol  $\pi$ . Since  $\pi$  is secure, there exists some simulator  $\mathcal{S}$  for which the ideal and real executions are indistinguishable. Using  $\mathcal{S}$  we build a simulator  $\mathcal{S}'$  for  $\mathcal{A}'$  in the protocol  $\pi'$ , and show that  $\mathcal{S}'$  indeed simulates the adversary  $\mathcal{A}'$  in the ideal execution.

Let  $\mathcal{A}'$  be an adversary for  $\pi'$ . We build an adversary  $\mathcal{A}$  as follows:

**The adversary  $\mathcal{A}$ .**

- **Input:** The security parameter  $1^n$ , the input for the protocol  $x_{i^*}$  and an auxiliary input  $z$ .
- **The adversary:**
  1.  $\mathcal{A}$  invokes  $\mathcal{A}'$  with input  $(1^n, x_{i^*})$  and the auxiliary input  $z$ .
  2. **Setup:**
    - (a) When  $\mathcal{A}'$  queries the  $\mathcal{F}^{\text{PKI}}$ -functionality with the command  $(\text{Retrieve}, id^*)$  to receive the verification key of identity  $id^*$ , run  $\text{Gen}_{\text{Sign}}(1^n)$ , get back  $(sk_{id^*}, vk_{id^*})$  and replies with  $(\text{Retrieve}, id^*, vk_{id^*})$ .
    - (b) When  $\mathcal{A}'$  queries the  $\mathcal{F}^{\text{PKI}}$ -functionality with the command  $(\text{Register}, id, vk_{id})$ , store the pair  $(id, vk_{id})$ .
  3. Whenever  $\mathcal{A}$  receives a message  $m$  during the interaction, it computes  $\sigma = \text{Sign}_{sk}(m)$  and hands  $\mathcal{A}'$  the pair  $(m, \sigma)$ .
  4. Whenever  $\mathcal{A}'$  outputs a message  $(m, \sigma)$ , it checks that  $\text{Vrfy}_{vk_{id^*}}(m, \sigma) = 1$  and output it as an output message. In case that the verification fail, it aborts.
  5. Whenever  $\mathcal{A}'$  queries the  $\mathcal{F}_{\Pi}^{\text{SignedOT}}$ -functionality as Sender with input  $(m_0, m_1, sk_{id}, \sigma_0^*, \sigma_1^*)$  act as follows. If  $(\sigma_0^*, \sigma_1^*) = (\perp, \perp)$ , then check that  $sk_{id}$  and the verification key  $vk_{id}$  are consistent<sup>7</sup>. If  $(\sigma_0^*, \sigma_1^*) \neq (\perp, \perp)$ , then check that  $\text{Vrfy}((0, m_0), \sigma_0^*) = 1$  and  $\text{Vrfy}((1, m_1), \sigma_1^*) = 1$ . If the checks hold, then use  $(m_0, m_1)$  as the inputs for the OT functionality. Otherwise, send  $\perp$  to the functionality, and abort the execution.
  6. Whenever  $\mathcal{A}'$  queries the  $\mathcal{F}_{\Pi}^{\text{SignedOT}}$ -functionality as Receiver with input  $(b, vk'_{id}, r^*)$ , then query the OT functionality with input  $b$ . Upon receiving  $m_b$  from the OT, check if  $r^* = \perp$ . If so, compute  $\sigma = \text{Sign}_{sk_{id}}(b, m_b)$  with fresh uniformly distributed randomness. If  $r^* \neq \perp$ , compute  $\sigma = \text{Sign}_{sk_{id}}(b, m_b; r^*)$ . Then, check that the verification key that the adversary has sent is valid, by checking that  $\text{Vrfy}_{vk'_{id}}((b, m_b), \sigma) = 1$ . If so, send  $(m_b, \sigma)$  to  $\mathcal{A}'$ . Otherwise, send  $\perp$  and abort the execution.
  7. At the end of the executions,  $\mathcal{A}$  whatever  $\mathcal{A}'$  outputs and halts. (without loss of generality, this is the view of  $\mathcal{A}'$ ).

From the definition of the adversary  $\mathcal{A}$  and the construction of  $\pi'$ , it is clear that:

$$\left\{ \text{HYBRID}_{\mathcal{A}(z), \pi, I}^{\text{OT}}(x, y, n) \right\}_{x, y, z \in (\{0,1\}^*)^3, n \in \mathbb{N}} \stackrel{c}{=} \left\{ \text{HYBRID}_{\mathcal{A}'(z), \pi', I}^{\mathcal{F}^{\text{PKI}}, \mathcal{F}_{\Pi}^{\text{SignedOT}}}(x, y, n) \right\}_{x, y, z \in (\{0,1\}^*)^3, n \in \mathbb{N}}$$

Since  $\pi$  is secure, for every PPT adversary  $\mathcal{A}$  (and in particular the adversary that we have just constructed) there exists a simulator  $\mathcal{S}$  such that:

$$\left\{ \text{IDEALC}_{f, \mathcal{S}(z), I}^{\epsilon}(x, y, n) \right\}_{x, y, z \in (\{0,1\}^*)^3, n \in \mathbb{N}} \stackrel{c}{=} \left\{ \text{HYBRID}_{\pi, \mathcal{A}(z), I}^{\text{OT}}(x, y, n) \right\}_{x, y, z \in (\{0,1\}^*)^3, n \in \mathbb{N}}$$

<sup>7</sup>This property may not exist in a general signature scheme. However, we can always define the secret key of a signature scheme as the randomness used in the key generation algorithm. Then, the adversary just checks that the verification key  $vk_{id}$  is obtained when invoking  $\text{Gen}_{\text{Sign}}$  with the given randomness.

and therefore we conclude that:

$$\left\{ \text{IDEALC}_{f,S(z),I}^{\epsilon}(x,y,n) \right\}_{x,y,z \in (\{0,1\}^*)^3, n \in \mathbb{N}} \stackrel{c}{=} \left\{ \text{HYBRID}_{\mathcal{A}', \pi', I}^{\mathcal{F}^{\text{PKI}}, \mathcal{F}_{\Pi}^{\text{SignedOT}}}(x,y,n) \right\}_{x,y,z \in (\{0,1\}^*)^3, n \in \mathbb{N}}$$

■

## 4.2 Proof of Theorem 4.1.

We show that our protocol satisfies each one of the properties as in Definition 3. We will use the similarity between our protocol and the one of [AL07] to argue for covert security with  $\epsilon$ -deterrent.

**Corrupted  $P_2$ .** Our protocol achieves security in the presence of a *malicious*  $P_2$ . The security follows from the  $\mathcal{F}_{\Pi}^{\text{SignedOT}}$ -functionality (that as we have seen, can be implemented efficiently with malicious security) and the same reasoning as in [AL07], with the exception that here we use a fully secure malicious OT instead of a covert. We are therefore left with the case where  $P_1$  is corrupted.

**Simulatability with  $\epsilon$ -deterrent.** Our protocol is in fact the same protocol as in [AL07], with the following differences:

1. In Steps 5 and 6,  $P_1$  sends its messages together with a signature on those.
2. In Steps 4 and 8, signed OT is used instead of standard OT.
3. In Step 9, if  $P_2$  outputs `corruptedi`, then it sends  $Cert = \text{Blame}(\text{view}_1)$  to the adversary.

Let  $\pi_0$  be the protocol of [AL07, Section 6.3] and  $\pi_1, \pi_2, \pi_3$  the protocols after the changes explained in bullets 1, 2, 3 respectively.

Protocols  $\pi_1$  and  $\pi_2$  differ from  $\pi_0$  only because  $P_1$  signs the messages it sends to  $P_2$ . In Lemma 2, we show that if  $\pi$  is a covert secure protocol with  $\epsilon$ -deterrent and  $\pi'$  is the same protocol as  $\pi$  with the only change that parties sign on all the message they send, and use signed OT instead of standard OT, then  $\pi'$  is also a covert secure protocol with  $\epsilon$ -deterrent. We therefore conclude that  $\pi_2$  is also a covert secure protocol with  $\epsilon$ -deterrent.

The only difference between  $\pi_3$  and  $\pi_2$  is that if  $P_2$  outputs `corrupted1`, then the adversary learns the certificate  $Cert$ . By inspection of the algorithm `Blame`, it is clear that the output certificate only contains messages that  $P_1^*$  sends to  $P_2$  (so  $P_1^*$  does not learn anything new) with one exception:  $P_1^*$  may learn the exact reason for why it has been detected. That is, by publishing the certificate, the honest party  $P_2$  may reveal, for example, the challenge  $\gamma$ .

However, [AL07, Theorem 6.2] showed that the protocol is secure in the presence of covert adversary with  $\epsilon$ -deterrent, even when  $P_2$  sends the challenges on the clear and not via an oblivious transfer (we consider now the simulation according to Definition 1 and *ignore* for now the issue of non-halting detection accuracy). Note that the input of the adversary for the  $\mathcal{F}_{\Pi}^{\text{SignedOT}}$  functionality is the list of messages  $(m_1, \dots, m_{\ell})$  together with a list of signatures  $(\sigma_0^*, \dots, \sigma_{\ell}^*)$ . Thus, since the simulator is able to simulate the challenge as well, the certificate can be computed deterministically from the view of the adversary.

**Non-halting detection accuracy.** The output of the honest party is `corrupted1`, only if it has received wrong circuit, wrong decommitment or the opening of the keys are not consistent with the keys it has received (selective OT attack). However,  $P_1$  must deviate from the protocol

specifications and actively cheat in order that the above will occurred. Due to the security of the Signed OT functionality, the adversary cannot know the challenges of  $P_2$  and therefore cannot abort as a consequence of being detected cheating. We therefore conclude that an honest party  $P_2$  never outputs  $\text{corrupted}_1$  when  $P_1^*$  is a fail-stop adversary.

**Accountability.** We need to show that for every PPT adversary  $\mathcal{A}$  corrupting party  $P_{i^*}$  for  $i^* \in \{1, 2\}$ , there exists a negligible function  $\mu(\cdot)$  such that for all sufficiently large  $x_1, x_2, z \in (\{0, 1\}^*)^3$  the following holds: If  $\text{OUTPUT}(\text{EXEC}_{\pi, \mathcal{A}(z), 1}(x_1, x_2; 1^n)) = \text{corrupted}_1$  then  $\Pr[\text{Judgement}(Cert) = id_1] > 1 - \mu(n)$  where  $Cert$  is the output certificate of the honest party in the execution. This follows from the description of the protocol  $\pi$  and the **Blame, Judgement** algorithms: an adversarial  $P_1$  who constructs one faulty circuit must decide before the oblivious transfer in Step 9 if it wishes to abort (in which case there is no successful cheating) or if it wishes to proceed (in which case  $P_2$  will receive an explicitly invalid opening and a signature on it). Note that due to the security of the oblivious transfer,  $P_1$  cannot know what value  $\gamma$  party  $P_2$  inputs, and so cannot avoid being detected.

Once the honest party outputs the certificate, it contains all the necessary information that caused the party to decide on the corruption. The verification algorithm **Judgement** performs exactly the same check as the honest party, and so accountability holds.

**Defamation-Free.** We need to show that for every PPT adversary  $\mathcal{A}$  controlling  $i^* \in \{1, 2\}$  and interacting with the honest party, there exists a negligible function  $\mu(\cdot)$  such that for all sufficiently large  $x_1, x_2, z \in (\{0, 1\}^*)^3$ :

$$\Pr[Cert^* \leftarrow \mathcal{A}; \text{Judgement}(Cert^*) = id_{3-i^*}] < \mu(n)$$

The above holds from the security of the signature scheme. Since **Judgement** never outputs the identity of  $P_2$  and may just output the identity of  $P_1$ , the only interesting case is when the adversary controls  $P_2$  and succeeds in creating a forged certificate  $Cert^*$  for which  $\text{Judgement}(Cert^*) = id_1$ . Since  $P_1$  is honest, it follows the protocol specifications and creates all the circuits correctly, consistent and open the commitments correctly. Remember also that every signature the honest  $P_1$  produces contains meta-information about the message (such as identity of the participating parties, protocol unique identifier, message identifier etc.) to ensure that a corrupted  $P_2^*$  cannot mix and match signatures obtained during different protocols to create a forged certificate. Therefore, if the adversary produces a certificate that passes the verification, it must have forged one of the messages.

More formally, assume the existence of an adversary  $\mathcal{A}$  that succeeds to forge a certificate  $Cert^*$  on an execution with some inputs  $x_1, x_2, z$ . We now build an adversary  $\mathcal{A}'$  to the signature scheme. The adversary  $\mathcal{A}'$  is given a verification key  $vk$  and a signing-oracle, and wins the game when it forges a signature on a message that it did not query its oracle for. The adversary  $\mathcal{A}'$  invokes the adversary  $\mathcal{A}$  with auxiliary input  $z$  and input  $x_2$ , and plays the role of the honest party  $P_1$  in the execution of the protocol  $\pi$ , with input  $x_1$ . When  $\mathcal{A}'$  queries the  $\mathcal{F}^{\text{PKI}}$  to the verification key of  $P_1$ , it sends back to it the verification key  $vk$ . Whenever it should send a message to  $\mathcal{A}$ , it computes the next message function according to the protocol  $\pi$  and query the signing oracle for a signature on it. Then, in case where  $\mathcal{A}$  outputs a certificate  $Cert^*$  for which  $\text{Judgement}(Cert^*) = id_{3-i^*}$ , it must be that  $\mathcal{A}$  was able to replace a message of the honest party with some valid signature on it.  $\mathcal{A}'$  outputs this message and its forged signature, and succeeds with the same probability as the adversary  $\mathcal{A}$ . ■

**Acknowledgements.** The authors would like to thank Yehuda Lindell for many helpful discussions and his invaluable guidance.

## References

- [AL07] Yonatan Aumann and Yehuda Lindell. Security against covert adversaries: Efficient protocols for realistic adversaries. In Salil P. Vadhan, editor, *TCC*, volume 4392 of *Lecture Notes in Computer Science*, pages 137–156. Springer, 2007.
- [BCNP04] Boaz Barak, Ran Canetti, Jesper Buus Nielsen, and Rafael Pass. Universally composable protocols with relaxed set-up assumptions. In *FOCS*, pages 186–195. IEEE Computer Society, 2004.
- [Can00] Ran Canetti. Security and composition of multiparty cryptographic protocols. *J. Cryptology*, 13(1):143–202, 2000.
- [Cle86] Richard Cleve. Limits on the security of coin flips when half the processors are faulty (extended abstract). In *STOC*, pages 364–369, 1986.
- [DGN10] Ivan Damgård, Martin Geisler, and Jesper Buus Nielsen. From passive to covert security at low cost. In Daniele Micciancio, editor, *TCC*, volume 5978 of *Lecture Notes in Computer Science*, pages 128–145. Springer, 2010.
- [FY92] Matthew K. Franklin and Moti Yung. Communication complexity of secure computation (extended abstract). In S. Rao Kosaraju, Mike Fellows, Avi Wigderson, and John A. Ellis, editors, *STOC*, pages 699–710. ACM, 1992.
- [GHKL11] S. Dov Gordon, Carmit Hazay, Jonathan Katz, and Yehuda Lindell. Complete fairness in secure two-party computation. *J. ACM*, 58(6):24, 2011.
- [GK09] S. Dov Gordon and Jonathan Katz. Complete fairness in multi-party computation without an honest majority. In *TCC*, pages 19–35, 2009.
- [GK12] S. Dov Gordon and Jonathan Katz. Partial fairness in secure two-party computation. *J. Cryptology*, 25(1):14–40, 2012.
- [GMS08] Vipul Goyal, Payman Mohassel, and Adam Smith. Efficient two party and multi party computation against covert adversaries. In Nigel P. Smart, editor, *EUROCRYPT*, volume 4965 of *Lecture Notes in Computer Science*, pages 289–306. Springer, 2008.
- [Gol04] Oded Goldreich. *Foundations of Cryptography Volume 2, Basic Applications*. Cambridge University Press, 2004.
- [HL08] Carmit Hazay and Yehuda Lindell. Efficient protocols for set intersection and pattern matching with security against malicious and covert adversaries. In Ran Canetti, editor, *TCC*, volume 4948 of *Lecture Notes in Computer Science*, pages 155–175. Springer, 2008.
- [HL10] Carmit Hazay and Yehuda Lindell. Efficient secure two-party protocols: Techniques and constructions. Springer-Verlag, 2010.

- [IKNP03] Yuval Ishai, Joe Kilian, Kobbi Nissim, and Erez Petrank. Extending oblivious transfers efficiently. In Dan Boneh, editor, *CRYPTO*, volume 2729 of *Lecture Notes in Computer Science*, pages 145–161. Springer, 2003.
- [LP09] Yehuda Lindell and Benny Pinkas. A proof of security of yao’s protocol for two-party computation. *J. Cryptology*, 22(2):161–188, 2009.
- [Mic03] Silvio Micali. Simple and fast optimistic protocols for fair electronic exchange. In *PODC*, pages 12–19, 2003.
- [MNPS04] Dahlia Malkhi, Noam Nisan, Benny Pinkas, and Yaron Sella. Fairplay - secure two-party computation system. In *USENIX Security Symposium*, pages 287–302. USENIX, 2004.
- [NNOB12] Jesper Buus Nielsen, Peter Sebastian Nordholt, Claudio Orlandi, and Sai Sheshank Burra. A new approach to practical active-secure two-party computation. In Reihaneh Safavi-Naini and Ran Canetti, editors, *CRYPTO*, volume 7417 of *Lecture Notes in Computer Science*, pages 681–700. Springer, 2012.
- [PVW08] Chris Peikert, Vinod Vaikuntanathan, and Brent Waters. A framework for efficient and composable oblivious transfer. In David Wagner, editor, *CRYPTO*, volume 5157 of *Lecture Notes in Computer Science*, pages 554–571. Springer, 2008.