Calling out Cheaters: Covert Security With Public Verifiability*

Gilad Asharov[1] and Claudio Orlandi[2,**]

[1] Department of Computer Science, Bar-Ilan University, Israel
[2] Department of Computer Science, Aarhus University, Denmark
asharog@cs.biu.ac.il, orlandi@cs.au.dk

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 ϵ. 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 ϵ will suffice in discouraging any cheating attempt.

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.

* The research was supported by the European Research Council as part of the ERC project LAST.

** Research conducted while at Bar-Ilan University, Israel.

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 ϵ. 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 repercussion of a cheating attempt is 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 receive 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 ϵ. 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 ϵ-deterrent, whenever Alice attempts to cheat she will be caught with probability ϵ 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 ϵ 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 ϵ 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: $(m_b, \mathsf{Sig}_{sk}(b, m_b))$.

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 look always 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 $(m_0, \mathsf{Sig}(0, m_0)), (m_1, \mathsf{Sig}(1, m_1))$, 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.

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.

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 ϵ-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 ϵ-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. 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].

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

2.1 Covert Security with Public Verifiability

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 ϵ, 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. In addition, 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.

Regarding the Judgement algorithm, we require two security properties: 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.

2.2 The Formal Definition

Let f be a two party functionality. We consider the triple $(\pi, \mathsf{Blame}, \mathsf{Judgement})$. 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 π be a two party protocol. If an honest party detects a cheating in π then the honest party is instructed to compute $Cert = \mathsf{Blame}(\text{view})$ 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 π 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 [AL07, Definition 3.4]. 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 π 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 1 (covert security with ϵ-deterrent and public verifiability) *Let f, π,* Blame *and* Judgement *be as above. We say that* $(\pi, \mathsf{Blame}, \mathsf{Judgement})$ securely computes f in the presence of a covert adversary with ϵ-deterrent and public verifiability *if the following conditions hold:*

1. **(Simulatability with ϵ-deterrent:)** *The protocol π (where the honest party broadcasts $Cert = \mathsf{Blame}(\text{view})$ if it detects cheating) is secure against a covert adversary according to the* strong explicit cheat formulation *with ϵ-deterrent (see [AL07, Definition 3.4]).*

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

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

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* $x_1, x_2, z \in (\{0,1\}^*)^3$*:*

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

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 $\epsilon = 1 - \mathsf{negl}(n)$: aborting is the only possible malicious behavior. Therefore the function Blame will never be invoked and the function $\mathsf{Judgement}$ 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

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:

$$(\perp; (m_b, \mathsf{Sig}_{sk}(b, m_b))) \leftarrow \mathcal{F}\left((m_0, m_1, sk), (b, vk)\right)$$

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 don't need the signature to be computed by the ideal functionality. We therefore use a relaxed version of the signed OT functionality, that allows a malicious sender to choose any two strings (σ_0^*, σ_1^*) and input them to the functionality. If (σ_0^*, σ_1^*) are valid signatures on the messages $(0, m_0)$ and $(1, m_1)$ respectively, the functionality delivers (m_b, σ_b^*) to the receiver or abort otherwise. In other words, we allow a corrupted sender to influence the randomness involved in the generation of the signature, as long as it provides correct signatures for both messages. See Functionality 1 for the formal description.

3.1 A PVW Compatible Signature Scheme

As a first step, we will construct a (somewhat contrived) signature scheme, designed to combine efficiently with the OT protocol. Essentially, we are combining

FUNCTIONALITY 1 (The Signed OT Functionality – $\mathcal{F}_{\Pi}^{\mathbf{SignedOT}}$)

The functionality is parameterized by a signature Scheme $\Pi = (\mathsf{Gen}, \mathsf{Sig}, \mathsf{Ver})$.

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

Output: If $(\sigma_0^*, \sigma_1^*) = (\perp, \perp)$ the functionality computes $\sigma = \mathsf{Sig}_{sk}(b, m_b)$ and verifies that $\mathsf{Ver}_{vk}((b, m_b), \sigma) = 1$. It then outputs (m_b, σ) to the receiver or abort in case where the verification fails.
If $(\sigma_0^*, \sigma_1^*) \neq (\perp, \perp)$ the functionality outputs (m_b, σ_b^*) to the receiver if $\mathsf{Ver}_{vk}((0, m_0), \sigma_0^*) = 1$ and $\mathsf{Ver}_{vk}((0, m_0), \sigma_0^*) = 1$ or abort otherwise.

a signature scheme $\Pi' = (\mathsf{Gen}', \mathsf{Sig}', \mathsf{Ver}')$ with a computationally binding commitment $\mathsf{Com} = (\mathsf{Setup}, \mathsf{Com}, \mathsf{Open})$ (we do not need the commitment to be hiding). The verification key vk of the combined scheme is the same as the verification key of the original scheme vk'. On input a message m, the combined signature algorithm Sig chooses a random commitment key $ck = \mathsf{Setup}(1^n)$ and a string r, compute the commitment $(c, d) = \mathsf{Com}_{ck}(m; r)$ and outputs:

$$(ck, d, c, \mathsf{Sig}'_{sk}(ck, c)) \leftarrow \mathsf{Sig}_{sk}(m) \ . \tag{1}$$

On input $(m, (ck, d, c, \sigma))$, the verification algorithm Ver outputs 1 if and only if $\mathsf{Open}_{ck}(c, d) = m$ and $\mathsf{Ver}_{vk}((ck, c), \sigma) = 1$. Unforgeability of the combined scheme follows from the unforgeability of the original scheme together with the binding property of the commitment scheme. (Note that here is the signer creates both the commitment key and the commitment itself – differently from the standard game for computationally binding commitments, where the receiver needs to generate the key.) See the full version for details.

We present the commitment scheme that we use in the above template. Let $(\mathbb{G}, q)$ be a prime order group where the DDH assumption is believed to hold. Define the randomized function $RAND(g_0, h_0, g_1, h_1) = (u, v)$, where $u = (g_0)^s \cdot (h_0)^t$ and $v = (g_1)^s \cdot (h_1)^t$ 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. The commitment scheme is as follows:

- **The setup algorithm** Setup**:** On input security parameter 1^n, the setup chooses a DDH tuple (g_0, h_0, g_1, h_1) in $\mathbb{G}$ and defines $ck = (g_0, h_0, g_1, h_1)$.
- **The commitment algorithm** Com_{ck}**:** On input message $(b, m) \in \{0,1\} \times \mathbb{G}$, the Com algorithm chooses a random $r \in_R \mathbb{Z}_q$ and computes $(g, h) = (g_b, h_b)^r$ and $(u_b, v_b) = RAND(g_b, g, h_b, h)$, $w_b = m \cdot v_b$, $(u_{1-b}, w_{1-b}) \in_R \mathbb{G}^2$. Then, it defines $c = (g, h, u_0, w_0, u_1, w_1)$ and the decommitment value $d = (r; (b, m))$.

– **The opening algorithm** $\mathsf{Open}_{ck}(c, d)$: 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 decommitment $d = (r; (b, m))$, the opening algorithm checks that $(g, h) = (g_b, h_b)^r$ and $w_b = m \cdot u_b^r$. If so it outputs (b, m), otherwise $\perp$.

Claim 1 *Assuming computing discrete logarithms is hard in $\mathbb{G}$, the scheme $(\mathsf{Setup}, \mathsf{Com}, \mathsf{Open})$ is computationally binding.*

Proof Sketch: To see that the scheme is binding, observe that there is a unique mapping between r and (b, m) in the following way: given a commitment $c = (g, h, u_0, w_0, u_1, w_1)$ and the decommitment r, we search for b for which $(g, h) = (g_b, h_b)^r$. Given (r, b), the message m is defined as: $w_b \cdot (u_b)^{-r}$. Therefore, the only way that an adversary can break the binding property of a given commitment c is by finding r' for which $(g, h) = (g_{1-b}^{r'}, h_{1-b}^{r'})$. But, to find such an r' the adversary needs to break the discrete logarithm assumption. ∎

Our PVW compatible signature scheme $\Pi = (\mathsf{Gen}, \mathsf{Sig}, \mathsf{Ver})$ is the a combination of the signature scheme Π' and the commitment scheme Com as defined in Eq. (1). We conclude:

Corollary 1 *If $\Pi' = (\mathsf{Gen}', \mathsf{Sig}', \mathsf{Ver}')$ is an* existentially unforgeable under an adaptive chosen-message attack *signature scheme and the discrete logarithm problem is hard in $(\mathbb{G}, g_0, q)$, then $\Pi = (\mathsf{Gen}, \mathsf{Sig}, \mathsf{Ver})$ is also existentially unforgeable under an adaptive chosen-message attack.*

3.2 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. Like the original OT protocol [PVW08], our signed OT protocol can be extended in the straightforward way to an 1-out-of-ℓ signed OT (see the full version). Note that the overall protocol is just the DDH-based instantiation of the PVW OT framework with the following differences (clearly marked in the protocol description): (1) The sender chooses the "CRS" (g_0, h_0, g_1, h_1) and proves that it is a DDH tuple. (Remember that in this case the receiver's message hides his choice bit *statistically*). (2) The sender signs all the messages it sends to the receiver.

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.

Lemma 1 *Let $\Pi = (\mathsf{Gen}, \mathsf{Sig}, \mathsf{Ver})$ be the PVW-compatible signature scheme defined above. Then, Protocol 1 securely implements the $\mathcal{F}_{\Pi}^{\mathrm{SignedOT}}$-functionality in the presence of a malicious adversary.*

PROTOCOL 1 (Signed One-out-of-Two OT Protocol)

Setup: This step can be done once and reused for multiple runs of the OT: The sender S chooses $(g_0, h_0) \in_R \mathbb{G}^2$ and a random $\alpha \in_R \mathbb{Z}_q$ and compute $g_1 = g_0^\alpha$ and $h_1 = h_0^\alpha$. The sender sends (g_0, h_0, g_1, h_1) to the receiver R and gives a zero-knowledge proof-of-knowledge that (g_0, h_0, g_1, h_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 $(u_0, v_0) = RAND(g_0, g, h_0, h)$ and $(u_1, v_1) = RAND(g_1, g, h_1, h)$;
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*
$$\sigma' = \mathsf{Sig}'_{sk'}((g_0, h_0, g_1, h_1), (g, h, u_0, w_0, u_1, w_1));$$

Retrieve: *(**diff**)* Let $vk = vk'$. R checks that σ' 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: As discussed in Corollary 1, σ is a proper signature on the message (b, m_b), and therefore the correct functionality is implemented when both parties are honest.

The proof of security of the underlying OT protocol is by now standard and can be found in [PVW08, HL10]. When the receiver is corrupted, the simulator plays as an honest sender except that it chooses instead a non-DDH tuple in step "Setup" (i.e., some (g_0, g_0^x, g_1, g_1^y)) and then, given the pair (g, h) and using the trapdoor (x, y), it can extract the receiver input's bit b by finding whether h equals g^x or g^y. It then sends b to the functionality $\mathcal{F}_\Pi^{\text{SignedOT}}$. Clearly, adding the signature σ' does not break any security property of the original OT protocol (it is easy to see that any attack to this protocol can be reduced to an attack to the original protocol, where the reduction will simply produce this extra signature).

For the case of a corrupted sender, the simulator plays as an honest receiver (with $b = 1$) except that it extracts α from the zero-knowledge proof in step "Setup". Using this trapdoor, it can compute both messages m_0, m_1 (as in the proof of the original protocol). Then, it computes the two signatures σ_0^*, σ_1^* as follows:

$$\sigma_0^* = ((g_0, h_0, g_1, h_1), (\alpha \cdot r, (0, m_0)), (g, h, u_0, w_0, u_1, w_1), \sigma')$$
$$\sigma_1^* = ((g_0, h_0, g_1, h_1), (r, (1, m_1)), (g, h, u_0, w_0, u_1, w_1), \sigma')$$

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

opening of c for $(0, m_0)$ whereas r is the opening of c for $(1, m_1)$. Finally, it is easy to see that the distribution of the constructed signatures are the same as in the real execution. ∎

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[3] 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 ℓ 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-ℓ 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 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 Com denote a perfectly-binding commitment scheme, where $\mathsf{Com}(x; r)$ denotes a commitment to x using randomness r. $(\mathsf{Gen}_{\mathsf{ENC}}, \mathsf{Enc}, \mathsf{Dec})$ is a semantically secure symmetric encryption scheme. $(\mathsf{Gen}, \mathsf{Sig}, \mathsf{Ver})$ is an existentially unforgeable signature scheme under an adaptive chosen-message attack. Note that

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

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 simplicity.

PROTOCOL 2 [Two-Party Secure Computation]

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 ℓ 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 $w_1, \ldots, w_n$ and the input wires associated with x_2 are $w_{n+1}, \ldots, w_{2n}$.*

The Protocol[4]:

1. *Parties P_1 and P_2 define a new circuit C' that receives $m+1$ inputs $x_1, x_2^1, \ldots, x_2^m$ 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 $w_1, \ldots, w_n$, and the input wires associated with x_2^i by $w_{n+(i-1)m}, \ldots, w_{n+im}$ for $i = 1, \ldots, n$.*
2. *P_2 chooses $m-1$ strings $x_2^1, \ldots, x_2^{m-1}$ uniformly and independently at random form $\{0,1\}^n$, and defines $x_2^m = \left(\oplus_{i=1}^{m-1} x_2^i\right) \oplus x_2$, where x_2 is P_2's original input. Observe that $\oplus_{i=1}^m x_2^i = x_2$.*
3. *For each $i = 1, \ldots, mn$ and $\beta = 0, 1$, party P_1 chooses ℓ encryption keys by running $\mathsf{Gen}_{\mathsf{ENC}}(1^n)$ for ℓ times. Denote the jth key associated with a given i and β by $k^j_{w_{n+i},\beta}$.*
4. *P_1 and P_2 invoke the mn times the (***diff***) $\mathcal{F}_{\Pi}^{\text{SignedOT}}$ functionality with the following inputs: In the ith execution, party P_1 inputs the pair:*

$$\left(\left[k^1_{w_{n+i},0}, \ldots, k^\ell_{w_{n+i},0}\right], \left[k^1_{w_{n+i},1}, \ldots, k^\ell_{w_{n+i},1}\right]\right)$$

   and party P_2 inputs the bit x_2^i (P_2 receives the keys $\left[k^1_{w_{n+i},x_2^i}, \ldots, k^\ell_{w_{n+i},x_2^i}\right]$ and a signature on this as output). If P_2 output in the OT is abort_i, then it outputs abort_i and halts.
5. *Party P_1 constructs ℓ garbled circuits $GC_1, \ldots, GC_\ell$ using independent randomness for the circuit C' described above. The keys for the input wires $w_{n+1}, \ldots, w_{n+mn}$ in the garbled circuits are taken from above (i.e., the keys*

[4] 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.

associated with w_{n+i} *are* $k^j_{w_{n+i},0}$ *and* $k^j_{w_{n+i},1}$*). The keys for the inputs wires* $w_1, \ldots, w_n$ *are chosen randomly, and are denoted in the same way.* P_1 *sends the* ℓ *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* $i = 1, \ldots, n$*,* $\beta = 0, 1$ *and* $j = 1, \ldots, \ell$*, party* P_1 *computes (***diff***):*

$$c^j_{w_i,\beta} = \mathsf{Com}\left(k^j_{w_i,\beta}; r^j_{i,\beta}\right), \sigma^j_{w_i,\beta} = \mathsf{Sig}_{sk}(c^j_{w_i,\beta})$$

The commitments and the signatures are sent as ℓ *vectors of pairs (one vector for each circuit); in the* j*th vector the* i*th pair is* $\{(c^j_{w_i,0}, \sigma^j_{w_i,0}), (c^j_{w_i,1}, \sigma^j_{w_i,0})\}$ *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* abort_1*.*
7. P_2 *chooses a random index* $\gamma \in_R \{1, \ldots, \ell\}$*.*
8. *(***diff***)* P_1 *and* P_2 *engage in a* $\binom{\ell}{1}$*-signed OT, where* P_2 *inputs* γ *and, for* $i = 1, \ldots, \ell$*,* P_1 *inputs as the* i*th message of the signed OT 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_γ*) together with a signature on them.* P_2 *receives also the decommitments and the keys associated with* P_1*'s input for circuit* GC_γ *together with signatures on them. If any of the signatures are incorrect, it halts and outputs* 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* $\mathsf{key} = \mathsf{wrongCircuit}$ *to its view.*
   - *That the decommitment values correctly open all the commitments* $c^j_{w_i,\beta}$ *that were received, and these decommitments reveal the keys* $k^j_{w_i,\beta}$ *that were sent for* P_1*'s wires. (***diff***) If not, add* $\mathsf{key} = \mathsf{wrongDecommitment}$ *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* $\mathsf{key} = \mathsf{selectiveOTattack}$ *to its view.*

   *If all check pass, proceed to the next step, else (***diff***),* P_2 *computes Cert =* $\mathsf{Blame}(\mathrm{view}_2)$ *(see the description of* Blame *for its output on different* key *values), it publishes Cert and output* $\mathsf{corrupted}_1$*.*
10. P_2 *checks that the values received are valid decommitments to the commitments received above. If not, it outputs* abort_1*. If yes, it uses the keys to compute* $C'(x_1, z_2) = C'(x_1, x_2^1, \ldots, x_2^m) = C(x_1, x_2)$*, and outputs the result.*

Theorem 1 *Let* ℓ *and* m *be parameters in the protocol that are both upper-bound by* $\mathrm{poly}(n)$*, and set* $\epsilon = (1 - 1/\ell)(1 - 2^{-m+1})$*, and let* f *be a probabilistic polynomial-time function and let* π *denote Protocol 2. Then, assuming the*

ALGORITHM 1 (The Blame Algorithm – Blame)

Input: The view of a honest party view, containing an error tag key.

Output: A certificate $Cert = (id, \mathsf{key}, \mathsf{message}, \sigma)$.

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 σ the signature on these messages.
- **Case 2:** key = wrongDecommitment**:** Let message be (c, x, r) be a commitment where $c \neq \mathsf{Com}(x; r)$ and σ 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 σ 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 $Cert = (id, \mathsf{key}, \mathsf{message}, \sigma)$.

Output: The identity id or *none*.

The Algorithm: If σ 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 $c \neq \mathsf{Com}(x; r)$ 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.

DDH assumption, security of the commitment scheme, signature scheme and symmetric encryption scheme as described above, $(\pi, \mathsf{Blame}, \mathsf{Judgement})$ *securely computes* f *in the presence of covert adversaries with* ϵ*-deterrent and public verifiability (i.e, satisfies Definition 1).*

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 $\epsilon = 50\%$. We can now proceed to the proof.

Proof Sketch: We show that our protocol satisfies each one of the properties as in Definition 1. We will use the similarity between our protocol and the one of [AL07] to argue for covert security with ϵ-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 a covert. We are therefore left with the case where P_1 is corrupted.

Simulatability with ϵ-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 $\mathsf{corrupted}_i$, then it sends $\mathit{Cert} = \mathsf{Blame}(\text{view}_2)$ to the adversary. Let π_0 be the protocol of [AL07, Section 6.3] and π_1, π_2, π_3 the protocols after the changes explained in bullets $1, 2, 3$ respectively.

Protocols π_1 and π_2 differ from π_0 only because P_1 signs the messages it sends to P_2. In the full version, we show that if π is a covert secure protocol with ϵ-deterrent and π' is the same protocol as π with the only change that parties sign on all the message they send, then π' is also a covert secure protocol with ϵ-deterrent. We therefore conclude that π_2 is also a covert secure protocol with ϵ-deterrent.

The only difference between π_3 and π_2 is that if P_2 outputs $\mathsf{corrupted}_1$, then the adversary learns the certificate Cert. In the full version, we show that this extra information can be simulated as well and so the overall protocol is covert protocol with ϵ-deterrent.

Accountability. Accountability follows from the description of the protocol π and the $\mathsf{Blame}, \mathsf{Judgement}$ 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 γ 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 $\mathsf{Judgement}$ 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 $x_1, x_2, z \in (\{0,1\}^*)^3$:

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

The above holds from the security of the signature scheme. Since $\mathsf{Judgement}$ 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 $\mathsf{Judgement}(\mathit{Cert}^*) = id_1$. 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. A more formal argument appears in the full version ∎

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 *TCC*, pages 137–156, 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.

[DGN10] Ivan Damgård, Martin Geisler, and Jesper Buus Nielsen. From passive to covert security at low cost. In *TCC*, pages 128–145, 2010.

[FY92] Matthew K. Franklin and Moti Yung. Communication complexity of secure computation (extended abstract). In *STOC*, pages 699–710, 1992.

[GMS08] Vipul Goyal, Payman Mohassel, and Adam Smith. Efficient two party and multi party computation against covert adversaries. In *EUROCRYPT*, pages 289–306, 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 *TCC*, pages 155–175, 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 *CRYPTO*, pages 145–161, 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.

[MNPS04] Dahlia Malkhi, Noam Nisan, Benny Pinkas, and Yaron Sella. Fairplay - secure two-party computation system. In *USENIX Security Symposium*, pages 287–302, 2004.

[PVW08] Chris Peikert, Vinod Vaikuntanathan, and Brent Waters. A framework for efficient and composable oblivious transfer. In *CRYPTO*, pages 554–571, 2008.