Public Verifiability in the Covert Model (Almost) for Free

Vladimir Kolesnikov[1(✉)] and Alex J. Malozemoff[2]

[1] Bell Labs, Murray Hill, USA
kolesnikov@research.bell-labs.com
[2] University of Maryland, College Park, USA
amaloz@cs.umd.edu

Abstract. The covert security model (Aumann and Lindell, TCC 2007) offers an important security/efficiency trade-off: a covert player may arbitrarily cheat, but is caught with a certain fixed probability. This permits more efficient protocols than the malicious setting while still giving meaningful security guarantees. However, one drawback is that cheating cannot be proven to a third party, which prevents the use of covert protocols in many practical settings. Recently, Asharov and Orlandi (ASIACRYPT 2012) enhanced the covert model by allowing the honest player to generate a *proof of cheating*, checkable by any third party. Their model, which we call the PVC (*publicly verifiable covert*) model, offers a very compelling trade-off.

Asharov and Orlandi (AO) propose a practical protocol in the PVC model, which, however, relies on a specific expensive oblivious transfer (OT) protocol incompatible with OT extension. In this work, we improve the performance of the PVC model by constructing a PVC-compatible OT extension as well as making several practical improvements to the AO protocol. As compared to the state-of-the-art OT extension-based two-party covert protocol, our PVC protocol adds relatively little: four signatures and an $\approx 67\,\%$ wider OT extension matrix. This is a significant improvement over the AO protocol, which requires public-key-based OTs per input bit. We present detailed estimates showing (up to orders of magnitude) concrete performance improvements over the AO protocol and a recent malicious protocol.

Keywords: Secure computation · Publicly verifiable covert security

1 Introduction

Two-party secure computation addresses the problem where two parties need to evaluate a common function f on their inputs while keeping the inputs private. Several security models for secure computation have been proposed. The most

A.J. Malozemoff—Work partially done while the author was at Bell Labs.

T. Iwata and J.H. Cheon (Eds.): ASIACRYPT 2015, Part II, LNCS 9453, pp. 210–235, 2015.
DOI: 10.1007/978-3-662-48800-3_9

basic is the *semi-honest* model, where the parties are expected to follow the protocol description but must not be able to learn anything about the other party's input from the protocol transcript. A much stronger guarantee is provided by the *malicious* model, where parties may deviate arbitrarily from the protocol description. This additional security comes at a cost. Recent garbled circuit-based protocols [3,17] have an overhead of at least 40× that of their semi-honest counterparts, and are considerably more complex.

Aumann and Lindell [8] introduced a very practical compromise between these two models, that of *covert* security. In the covert security model, a party can deviate arbitrarily from the protocol description but is caught with a fixed probability ϵ, called the *deterrence factor*. In many practical scenarios, this guaranteed risk of being caught (likely resulting in loss of business and/or embarrassment) is sufficient to deter would-be cheaters. Importantly, covert protocols are much more efficient and simpler than their malicious counterparts.

Motivating the Publicly Verifiable Covert (PVC) Model. At the same time, the cheating deterrent introduced by the covert model is relatively weak. Indeed, a party catching a cheater certainly knows what happened and can respond accordingly, e.g., by taking their business elsewhere. However, the impact is largely limited to this, since the honest player cannot credibly *accuse* the cheater publicly. If, however, credible public accusation were possible, the deterrent for the cheater would be immeasurably greater: suddenly, *all* the cheater's customers would be aware of the cheating and thus any cheating may affect the cheater's global customer base.

The addition of credible accusation greatly improves the covert model even in scenarios with a small number of players, such as those involving the government. Consider, for example, the setting where two agencies are engaged in secure computation on their respective classified data. The covert model may often be insufficient here. Indeed, consider the case where one of the two players deviates from the protocol, perhaps due to an insider attack. The honest player detects this, but we are now faced with the problem of identifying the culprit across two domains, where the communication is greatly restricted due to trust, policy, data privacy legislation, or all of the above. On the other hand, credible accusation immediately provides the ability to exclude the honest player from the suspect list, and focus on tracking the problem *within one organization/trust domain*, which is dramatically simpler.

PVC Definition and Protocol. Asharov and Orlandi [7] proposed a security model, *covert with public verifiability*, and an associated protocol, addressing these concerns. At a high level, they proposed that when cheating is detected, the honest player is able to publish a "certificate of cheating" which can be checked by any third party. In this work, we abbreviate their model as *PVC: publicly verifiable covert*. Their proposed protocol (which we call the "AO protocol") has performance similar to the original covert protocol of Aumann and Lindell [8], with the exception of requiring signed-OT, a special form of oblivious transfer (OT). Their signed-OT construction is based on the OT of Peikert et al. [18], and thus requires several expensive public-key operations.

In this work, we propose several critical performance improvements to the AO protocol. Our most technically involved contribution is a novel signed-OT *extension* protocol which eliminates per-instance public-key operations. Before discussing our contributions and technical approach in Sect. 1.1, we review the AO protocol.

The Asharov-Orlandi (AO) PVC Protocol [7]**.** The AO protocol is based on the covert construction of Aumann and Lindell [8]. Let P_1 be the circuit generator, P_2 be the evaluator, and $f(\cdot,\cdot)$ be the function to be computed. Recall the standard garbled circuit (GC) construction in the semi-honest model: P_1 constructs a garbling of f and sends it to P_2 along with the wire labels associated with its input. The parties then run OT, with P_1 acting as the sender and inputting the wire labels associated with P_2's input, and P_2 acting as the receiver and inputting as its choice bits the associated bits of its input.

We now adapt this protocol to the PVC setting. Recall the "selective failure" attack on P_2's input wires, where P_1 can send P_2 via OT an invalid wire label for one P_2's two inputs and learn one of P_2's input bits based on whether P_2 aborts. To protect against this attack, the parties construct $f'(\mathbf{x}_1, \mathbf{x}_2^1, \ldots, \mathbf{x}_2^\nu) = f(\mathbf{x}_1, \bigoplus_{i\in[\nu]} \mathbf{x}_2^i)$, where ν is the *XOR-tree replication factor*, and compute f' instead of f. Party P_1 then constructs λ (the *GC replication factor*) garblings of f' and P_2 checks that $\lambda - 1$ of the GCs are correctly constructed, evaluating the remaining GC to derive the output. The main difficulty of satisfying the PVC model is ensuring that neither party can improve its odds by aborting (e.g., based on the other party's challenge). For example, if P_1 could abort whenever P_2's challenge would reveal P_1's cheating, this would enable P_1 to cheat without the risk of generating a proof of cheating. Thus, P_1 sends the GCs to P_2 through a 1-out-of-λ OT; namely, in the ith input to the OT P_1 provides openings for all the GCs but the ith, as well as the input wire labels needed to evaluate GC_i. Party P_2 inputs a random γ, checks that all GCs besides GC_γ are constructed correctly, and if so, evaluates GC_γ.

Finally, it is necessary for P_1 to operate in a *verifiable* manner, so that an honest P_2 has proof if P_1 tries to cheat and gets caught. (Note that GCs guarantee that P_2 cannot cheat in the GC evaluation at all, so we only worry about catching P_1.) The AO protocol addresses this by having P_1 sign all its messages and the parties using *signed*-OT in place of all standard OTs (including wire label transfers and GC openings). Informally, the signed-OT functionality proceeds as follows: rather than the receiver R getting message $\mathbf{m}_b$ from the sender S for choice bit b, R receives $((b, \mathbf{m}_b), \sigma)$, where σ is S's signature of $(b, \mathbf{m}_b)$. This guarantees that if R detects any cheating by S, it has S's signature on an inconsistent set of messages, which can be used as proof of this cheating. Asharov and Orlandi show that this construction is ϵ-PVC-secure for $\epsilon = (1 - 1/\lambda)(1 - 2^{-\nu+1})$.

1.1 Our Contribution

Our main contribution is a signed-OT extension protocol built on the recent malicious OT extension of Asharov et al. [6]. Informally, signed-OT extension

ensures that (1) a cheating sender S is held accountable in the form of a "certificate of cheating" that the honest receiver R can generate, and (2) a malicious R cannot *defame* an honest S by presenting a false "certificate of cheating". Achieving the first goal is fairly straightforward by having S simply sign all its messages. The challenge is in simultaneously protecting against a malicious R. In particular, we need to commit R to its particular choices throughout the OT extension protocol to prevent it from defaming an honest S, while maintaining that those commitments do not leak any information about R's choices.

Recall that in the standard OT extension protocol of Ishai et al. [12] (cf. Fig. 3), R constructs a random matrix M, and S obtains a matrix M' derived from M, S's random string $\mathbf{s}$ and R's vector of OT inputs $\mathbf{r}$. The key challenge of adapting this protocol to the signed variant is to efficiently prevent R from submitting a malleated M as part of the proof without it ever explicitly revealing M to S (as this would leak R's choice bits). We achieve this by observing that S does in fact learn some of M, as in the OT extension construction some of the columns of M and M' are the same (i.e., those corresponding to zero bits of S's string $\mathbf{s}$). We prevent R from cheating by having S include in its signature carefully selected information from the columns in M which S sees. Finally, we require that R generates each row of M from a seed, and that R's proof of cheating includes this seed such that the row rebuilt from the seed is consistent with the columns included in S's signature. We show that this makes it infeasible for R to successfully present an invalid row in the proof of cheating. We describe this approach in greater detail in Sect. 3[1].

As another contribution, we present a new more communication efficient PVC protocol, building off of the AO protocol; see Sect. 4. Our main (simple) trick there is a careful amendment allowing us to send GC hashes instead of GCs; this is based on an idea from Goyal et al. [11].

We work in the random oracle model, a slight strengthening of the assumptions needed for standard OT extension and free-XOR, two standard secure computation tools.

Comparison with Existing Approaches. The cost of our protocol is almost the same as that of the covert protocol of Goyal et al. [11]; the only extra cost is essentially a $\approx 67\,\%$ wider OT extension matrix and four signatures. This often negligible additional overhead (versus covert protocols) provides us with dramatically stronger (than covert) deterrent. We believe that our PVC protocol could be used in many applications where covert security is insufficient at the order-of-magnitude cost advantage over previously-needed malicious protocols or the PVC protocol of Asharov and Orlandi [7]. See Sect. 5 for more details.

Related Work. The only directly related work is that of Asharov and Orlandi [7], already discussed at length. We also note a recent line of work on secure

[1] Our construction is also interesting from a theoretical perspective in that we construct signed-OT from *any* maliciously secure OT protocol, whereas Asharov and Orlandi [7] build a specific construction based on the Decisional Diffie-Hellman problem.

computation with cheaters (including fairness violators) punished by an external entity, such as the Bitcoin network [4,10,16]. Similarly to the PVC model and our protocols, this line of work relies on generating proofs of misbehavior which could be accepted by a third-party authority. However, these works address a different setting and use different techniques; in particular, they build on maliciously-secure computation and require the Bitcoin framework.

2 Preliminaries

Let κ denote the (computational) security parameter, let ρ denote the statistical security parameter, and let τ denote the field size. When considering concrete costs, we utilize the security parameter and field size settings for key lengths recommended by NIST [9]; see Fig. 1. We use PPT to denote "probabilistic polynomial time" and let $\mathsf{negl}(\cdot)$ denote a negligible function in its input. We consider two-party protocols between parties P_1 and P_2, and when we use subscript $i \in \{1,2\}$ to denote a party we let subscript $\text{-}i = 3 - i$ denote the other party. We use $i^* \in \{1,2\}$ to denote a malicious party and $\text{-}i^* = 3 - i^*$ to denote the associated honest party.

Security	κ	**FCC**	**ECC**
Short	80	1024	160
Long	128	3072	256

Fig. 1. Settings for (computational) security parameter κ and field size τ for various security settings as recommended by NIST [9]. FCC denotes the setting of τ when using finite field cryptography and ECC denotes the setting of τ when using elliptic curve cryptography.

We use bold lowercase letters (e.g., $\mathbf{x}$) to denote bitstrings and use the notation $\mathbf{x}[i]$ to denote the ith bit in bitstring $\mathbf{x}$. Likewise, we use bold uppercase letters (e.g., $\mathbf{T}$) to denote matrices over bits. We use $[n]$ to denote $\{1,\ldots,n\}$. Let "$a \leftarrow f(x_1, x_2, \ldots)$" denote setting a to be the deterministic output of f on inputs $x_1, x_2, \ldots$; the notation "$a \leftarrow\!\!\$\, f(x_1, x_2, \ldots)$" is the same except that f here is randomized. We abuse notation and let $a \leftarrow\!\!\$\, S$ denote selecting a uniformly at random from set S.

Our constructions are in the $\mathcal{F}_{\mathsf{PKI}}$ model, where each party P_i can register a verification key, and other parties can retrieve P_i's verification key by querying $\mathcal{F}_{\mathsf{PKI}}$ on id_i. We use the notation $\mathsf{Sign}_{P_i}(\cdot)$ to denote a signature signed by P_i's secret key, and we assume that this signature can be verified by any third party. We often leave off the subscript if the identity of the signing party is clear.

2.1 Publicly Verifiable Covert Security

We assume the reader is familiar with the covert security model; however, we review the less familiar *publicly verifiable covert* (PVC) security model of

Asharov and Orlandi [7] below. When we say a protocol is "secure in the covert model," we assume it is secure under the strong explicit cheat formulation with ϵ-deterrent [8, §3.4], for some value of ϵ.

Let π be a two-party protocol between parties P_1 and P_2 implementing function f. Following Aumann and Lindell [8], we call π *non-halting* if for honest P_i and fail-stop adversary[2] $P_{\text{-}i}$, the probability that P_i outputs $\mathsf{corrupted}_{\text{-}i}$ is negligible. Consider the triple of algorithms $(\pi', \mathsf{Blame}, \mathsf{Judgment})$ defined as follows:

- Protocol π' is the same as π except that if an honest party $P_{\text{-}i^*}$ outputs $\mathsf{corrupted}_{i^*}$ when executing π, it computes $\mathsf{Cert} \leftarrow \mathsf{Blame}(\mathsf{id}_{i^*}, \mathsf{key}, \mathsf{View}_{\text{-}i^*})$, where key denotes the type of cheating detected, and sends Cert to P_{i^*}.
- Algorithm Blame is a deterministic algorithm which takes as input a cheating identity id, a cheating type key, and a view View of a protocol execution, and outputs a certificate Cert.
- Algorithm $\mathsf{Judgment}$ is a deterministic algorithm which takes as input a certificate Cert and outputs either an identity id or $\perp$.

Before proceeding to the definition, we first introduce some notation. Let $\mathsf{Exec}_{\pi,\mathcal{A}(z)}(x_1, x_2; 1^\kappa)$ denote the transcript (i.e., messages and output) produced by P_1 with input x_1 and P_2 with input x_2 running protocol π, where adversary $\mathcal{A}$ with auxiliary input z can corrupt parties before execution begins. Let $\mathsf{Output}_{P_i}(\mathsf{Exec}_{\pi,\mathcal{A}(z)}(x_1, x_2; 1^\kappa))$ denote the output of P_i on the input transcript.

Definition 1. *We say that* $(\pi', \mathsf{Blame}, \mathsf{Judgment})$ *securely computes* f *in the presence of a publicly verifiable covert adversary with* ϵ*-deterrent (or, is* ϵ*-PVC-secure) if the following conditions hold:*

1. *The protocol* π' *is a non-halting and secure realization of* f *in the covert model with* ϵ*-deterrent.*
2. *(Accountability) For every* PPT *adversary* $\mathcal{A}$ *corrupting party* P_{i^*}*, there exists a negligible function* $\mathsf{negl}(\cdot)$ *such that if* $\mathsf{Output}_{P_{\text{-}i^*}}(\mathsf{Exec}_{\pi,\mathcal{A}(z)}(x_1, x_2; 1^\kappa)) = \mathsf{corrupted}_{i^*}$ *then* $\Pr[\mathsf{Judgment}(\mathsf{Cert}) = \mathsf{id}_{i^*}] > 1 - \mathsf{negl}(\kappa)$*, where* $\mathsf{Cert} \leftarrow \mathsf{Blame}(\mathsf{id}_{i^*}, \mathsf{key}, \mathsf{View}_{\text{-}i^*})$ *and the probability is over the randomness used in the protocol execution.*
3. *(Defamation-free) For every* PPT *adversary* $\mathcal{A}$ *corrupting party* P_{i^*} *and outputting a certificate* Cert*, there exists a negligible function* $\mathsf{negl}(\cdot)$ *such that* $\Pr[\mathsf{Judgment}(\mathsf{Cert}) = \mathsf{id}_{\text{-}i^*}] < \mathsf{negl}(\kappa)$*, where the probability is over the randomness used by* $\mathcal{A}$*.*

Note that, in particular, the PVC definition implicitly disallows Blame to reveal $P_{\text{-}i^*}$'s input. This is because π' specifies that Cert is sent to P_{i^*}.

2.2 Signed Oblivious Transfer

A central functionality for constructing PVC protocols is *signed oblivious transfer* (signed-OT). Introduced by Asharov and Orlandi [7], we can define the basic

[2] A *fail-stop adversary* is one which acts semi-honestly but may halt at any time.

signed-OT functionality $\mathcal{F}$ as

$$(\perp, (m_b, \mathsf{Sign}_{\mathsf{sk}}(b, m_b))) \leftarrow\!\!\$\ \mathcal{F}((m_0, m_1, \mathsf{sk}), (b, \mathsf{vk})),$$

where the signature scheme is assumed to be existentially unforgeable under adaptive chosen message attack (EU-CMA). Namely, the sender S inputs two messages m_0 and m_1 along with a signing key sk; the receiver R inputs a choice bit b and a verification key vk; S receives no output whereas R receives m_b alongside a signature on (b, m_b).

However, as in prior work [7], this definition is too strong for our signed-OT extension construction to satisfy. We introduce a relaxed signed-OT variant (slightly different from Asharov and Orlandi's variant [7]) which is tailored for OT extension and is sufficient for obtaining PVC-security. Essentially, we need a signature scheme that satisfies a weaker notion than EU-CMA in which the signing algorithm takes randomness, a portion of which can be controlled by the adversary[3]. This is because in our signed-OT extension construction, a malicious party can influence the randomness used in the signing algorithm. In addition, we introduce an *associated data* parameter to the signing algorithm which allows the signer to specify some additional information unrelated to the message being signed but used in the signature. In our construction, we use the associated data to tie the signature to a specific counter (such as a session ID or message ID), preventing a malicious receiver from "mixing" properly signed values to defame an honest sender.

Let $\Pi = (\mathsf{Gen}, \mathsf{Sign}, \mathsf{Verify})$ be a tuple of PPT algorithms over message space $\mathcal{M}$, associated data space $\mathcal{D}$, and randomness spaces $\mathcal{R}_1$ and $\mathcal{R}_2$, defined as follows:

1. $\mathsf{Gen}(1^\kappa)$: On input security parameter 1^κ, output key pair $(\mathsf{vk}, \mathsf{sk})$.
2. $\mathsf{Sign}_{\mathsf{sk}}(m, a; (r_1, r_2))$: On input secret key sk, message $m \in \mathcal{M}$, associated data $a \in \mathcal{D}$, and randomness $r_1 \in \mathcal{R}_1$ and $r_2 \in \mathcal{R}_2$, output signature $\sigma = (a, \sigma')$.
3. $\mathsf{Verify}_{\mathsf{vk}}(m, \sigma)$: On input verification key vk, message $m \in \mathcal{M}$, and signature σ, output 1 if σ is a valid signature for m and 0 otherwise.

For security, we need the condition that unforgeability remains even if the adversary inputs some arbitrary r_1 or r_2. However, the adversary is prevented from inputting values for *both* r_1 and r_2. This reflects the fact that in our signed-OT extension construction, a malicious sender can control only r_1 and a malicious receiver can control only r_2. We place a further restriction that the choice of r_1 must be *consistent*; namely, all queries to Sign must use the same value for r_1. Looking ahead, this property exactly captures the condition we need (r_1 corresponds to the zero bits in the sender's column selection string in the OT

[3] Our notion is similar to the ρ-EU-CMRA notion introduced by Asharov and Orlandi [7]. It differs in that we allow different portions of the randomness to be corrupted, but not both portions at once. Looking forward, this is needed because the sender in our signed-OT functionality is only allowed to control some of the randomness.

extension), where the choice of r_1 is made once and then fixed throughout the protocol execution.

Towards our definition, we define an oracle $\mathcal{O}_{\mathsf{sk}}(\cdot,\cdot,\cdot,\cdot)$ as follows. Let $\perp$ be a special symbol. On input (m, a, r_1, r_2), proceed as follows. If neither r_1 nor r_2 equal $\perp$, output $\perp$. Otherwise, proceed as follows. If $r_1 = \perp$ and r'_1 has not been set, set r'_1 uniformly at random; if $r_1 \neq \perp$ and r'_1 has not been set, set $r'_1 = r_1$; if $r_2 = \perp$, set r'_2 uniformly at random; otherwise, set $r'_2 = r_2$. Finally, output $\mathsf{Sign}_{\mathsf{sk}}(m, a; (r'_1, r'_2))$.

Now, consider the following game $\mathsf{Sig\text{-}forge}^{\mathsf{CMPRA}}_{\mathcal{A},\Pi}(\kappa)$ for signature scheme Π between PPT adversary $\mathcal{A}$ and PPT challenger $\mathcal{C}$.

1. $\mathcal{C}$ runs $(\mathsf{vk}, \mathsf{sk}) \leftarrow\!\!\$\ \mathsf{Gen}(1^\kappa)$ and sends vk to $\mathcal{A}$.
2. $\mathcal{A}$, who has oracle access to $\mathcal{O}_{\mathsf{sk}}(\cdot,\cdot,\cdot,\cdot)$, outputs a tuple $(m, (a, \sigma'))$. Let $\mathcal{Q}$ be the set of messages and associated data pairs input to $\mathcal{O}_{\mathsf{sk}}(\cdot,\cdot,\cdot,\cdot)$.
3. $\mathcal{A}$ succeeds if and only if (1) $\mathsf{Verify}_{\mathsf{vk}}(m, (a, \sigma')) = 1$ and (2) $(m, a) \notin \mathcal{Q}$.

Definition 2. *Signature scheme* $\Pi = (\mathsf{Gen}, \mathsf{Sign}, \mathsf{Verify})$ *is existentially unforgeable under adaptive chosen message and partial randomness attack (EU-CMPRA) if for all* PPT *adversaries* $\mathcal{A}$ *there exists a negligible function* $\mathsf{negl}(\cdot)$ *such that* $\Pr[\mathsf{Sig\text{-}forge}^{\mathsf{CMPRA}}_{\mathcal{A},\Pi}(\kappa)] < \mathsf{negl}(\kappa)$.

Functionality $\mathcal{F}^{\Pi}_{\mathbf{signedOT}}$

The functionality is parameterized by an EU-CMPRA signature scheme $\Pi = (\mathsf{Gen}, \mathsf{Sign}, \mathsf{Verify})$.

Input: The sender inputs messages m_0 and m_1 such that $|m_0| = |m_1|$, secret key sk, associated data a, randomness r_1^*, and signatures σ_0^* and σ_1^*. The receiver inputs choice bit b, verification key vk, and randomness r_2^*. If the sender (resp., the receiver) is honest, then $r_1^* = \sigma_0^* = \sigma_1^* = \perp$ (resp., $r_2^* = \perp$).

Output: The functionality computes $\sigma_b = \mathsf{Sign}_{\mathsf{sk}}((b, m_b), a; (r_1^*, r_2^*))$ for $b \in \{0, 1\}$. The sender receives no output. The receiver receives the following output based on if the sender is corrupt or not:

- If either $\sigma_0^* \neq \perp$ or $\sigma_1^* \neq \perp$, the functionality outputs $((b, m_b), \sigma_b^*)$ if and only if $\mathsf{Verify}_{\mathsf{vk}}((0, m_0), \sigma_0^*) = \mathsf{Verify}_{\mathsf{vk}}((1, m_1), \sigma_1^*) = 1$, where $\sigma_b^* \leftarrow \sigma_b$ if $\sigma_b^* = \perp$; otherwise it outputs abort.
- If $\sigma_0^* = \sigma_1^* = \perp$, the functionality outputs $((b, m_b), \sigma_b)$.

Fig. 2. Signed oblivious transfer functionality.

Signed-OT Functionality. We are now ready to introduce our relaxed signed-OT functionality. As is our EU-CMPRA signature, it is tailored for OT extension, and is sufficient for building PVC protocols. This functionality, denoted

by $\mathcal{F}_{\mathbf{signedOT}}^{\Pi}$, is parameterized by an EU-CMPRA signature scheme Π and is defined in Fig. 2. As in standard OT, the sender inputs two messages (of equal length) and the receiver inputs a choice bit. However, in this formulation we allow a malicious sender to specify some random value r_1^* as well as signatures σ_0^* and σ_1^*. Likewise, a malicious receiver can specify some random value r_2^*. (Honest players input $\perp$ for these values.) If both players are honest, the functionality computes $\sigma \leftarrow \mathsf{Sign}((b, m_b); (r_1, r_2))$ with uniformly random values r_1 and r_2 and outputs $((b, m_b), \sigma)$ to the receiver. However, if either party is malicious and specifies some random value, this is fed into the Sign algorithm. Likewise, if the sender is malicious and specifies some signature $\sigma_b^* \neq \perp$, this value is used as the signature sent to the receiver.

Note that $\mathcal{F}_{\mathbf{signedOT}}^{\Pi}$ is nearly identical to the signed-OT functionality presented by Asharov and Orlandi [7, Functionality 2]; it differs in the use of EU-CMPRA signature schemes instead of ρ-EU-CMRA schemes. We also note that it is straightforward to adapt $\mathcal{F}_{\mathbf{signedOT}}^{\Pi}$ to realize OTs with more than two inputs from the sender. We let $\binom{\lambda}{1}$-$\mathcal{F}_{\mathbf{signedOT}}^{\Pi}$ denote a 1-out-of-λ variant of $\mathcal{F}_{\mathbf{signedOT}}^{\Pi}$.

A Compatible Commitment Scheme. Our construction of an EU-CMPRA signature scheme (cf. Sect. 3.3) uses a non-interactive commitment scheme, which we define here. Our definition follows the standard commitment definition, except we tweak the Com algorithm to take an additional associated data value.

Let $\Pi_{\mathsf{Com}} = (\mathsf{ComGen}, \mathsf{Com})$ be a tuple of PPT algorithms over message space $\mathcal{M}$ and associated data space $\mathcal{D}$, defined as follows:

1. $\mathsf{ComGen}(1^\kappa)$: On input security parameter 1^κ, compute parameters params.
2. $\mathsf{Com}(m, a; r)$: On input message $m \in \mathcal{M}$, associated data $a \in \mathcal{D}$, and randomness r, output commitment com.

A commitment can be opened by revealing the randomness r used to construct that commitment.

We now define security for our commitment scheme. We only consider the *binding* property; namely, the inability for a PPT adversary to open a commitment to some other value than that committed to. Security is the same as for standard commitment schemes, except we allow the adversary to control the randomness used in ComGen.

Consider the game $\mathsf{Com\text{-}bind}_{\mathcal{A},\Pi_{\mathsf{Com}}}(\kappa)$ for commitment scheme Π_{Com} between a PPT adversary $\mathcal{A}$ and a PPT challenger $\mathcal{C}$, defined as follows.

1. $\mathcal{A}$ sends randomness r to $\mathcal{C}$.
2. $\mathcal{C}$ computes $\mathsf{params} \leftarrow \mathsf{ComGen}(1^\kappa; r)$ and sends params to $\mathcal{A}$.
3. $\mathcal{A}$ outputs $(\mathsf{com}, m_1, a_1, r_1, m_2, a_2, r_2)$ and wins if and only if (1) $m_1 \neq m_2$, and (2) $\mathsf{com} = \mathsf{Com}(\mathsf{params}, m_1, a_1; r_1) = \mathsf{Com}(\mathsf{params}, m_2, a_2; r_2)$.

Definition 3. *A commitment scheme* $\Pi_{\mathsf{Com}} = (\mathsf{ComGen}, \mathsf{Com})$ *is (computationally) binding if for all* PPT *adversaries* $\mathcal{A}$*, there exists a negligible function* $\mathsf{negl}(\cdot)$ *such that* $\Pr[\mathsf{Com\text{-}bind}_{\mathcal{A},\Pi_{\mathsf{Com}}}(\kappa)] < \mathsf{negl}(\kappa)$.

3 Signed Oblivious Transfer Extension

We now present our main contribution: an efficient instantiation of signed oblivious transfer (signed-OT) extension. We begin in Sect. 3.1 by describing in detail the logic of the construction, iteratively building it up from the passively secure protocol of Ishai et al. [12]. We motivate the need for EU-CMPRA signature schemes in Sect. 3.2 and present a compatible such scheme in Sect. 3.3. In Sect. 3.4 we present the proof of security.

3.1 Intuition for the Construction

Consider the OT extension protocol of Ishai et al. [12] in Fig. 3, run between sender S and receiver R. This protocol is secure against a semi-honest R and malicious S. We show how to convert this protocol into one which satisfies the $\mathcal{F}^{\Pi}_{\mathbf{signedOT}}$ functionality defined in Fig. 2. For clarity of presentation, we build on the protocol of Fig. 3 and later discuss how to support a malicious R as well, based on the malicious OT extension protocol of Asharov et al. [6].

S**'s inputs:** Message pairs $\{(\mathbf{x}_j^0, \mathbf{x}_j^1)\}_{j\in[m]}$, where each $\mathbf{x}_j^0, \mathbf{x}_j^1 \in \{0,1\}^n$.
R**'s inputs:** Selection bits $\mathbf{r} = (r_1, \ldots, r_m)$.
Common inputs: Security parameter κ; number of base OTs ℓ $(= \kappa)$; hash function $H : \mathbb{N} \times \{0,1\}^\ell \to \{0,1\}^n$; ideal functionality $\mathcal{F}_{\mathsf{OT}}$.

1. Initial OT Phase:
   - S computes $\mathbf{s} \leftarrow\!\!\$\ \{0,1\}^\ell$.
   - R generates a random $m \times \ell$ matrix $\mathbf{T}$, where the jth row is $\mathbf{t}_j$ and the ith column is $\mathbf{t}^i$. Likewise, R generates a random $m \times \ell$ matrix $\mathbf{V}$, where the jth row is $\mathbf{v}_j$ and the ith column is $\mathbf{v}^i$.
   - S and R run $\mathcal{F}_{\mathsf{OT}}$ ℓ times in parallel, where S acts as the *receiver* with input s_i in the ith OT and R acts as the *sender* with input $(\mathbf{t}^i, \mathbf{v}^i)$ in the ith OT.
2. OT Extension Phase (Part I):
   - For $i \in [m]$, R sets $\mathbf{u}^i \leftarrow \mathbf{t}^i \oplus \mathbf{v}^i \oplus \mathbf{r}$, and sends $\mathbf{u}^i$ to S.
3. OT Extension Phase (Part II):
   - Let $\mathbf{Q}$ be the $m \times \ell$ matrix where each column $\mathbf{q}^i = (s_i \cdot (\mathbf{u}^i \oplus \mathbf{v}^i)) \oplus ((1 - s_i) \cdot \mathbf{t}^i)$. Note that $\mathbf{q}^i = (s_i \cdot \mathbf{r}) \oplus \mathbf{t}^i$ and $\mathbf{q}_j = (r_j \cdot \mathbf{s}) \oplus \mathbf{t}_j$.
   - For $j \in [m]$, S computes $\mathbf{y}_j^0 \leftarrow \mathbf{x}_j^0 \oplus H(j, \mathbf{q}_j)$ and $\mathbf{y}_j^i \leftarrow \mathbf{x}_j^i \oplus H(j, \mathbf{q}_j \oplus \mathbf{s})$, and sends $\mathbf{y}_j^0$ and $\mathbf{y}_j^1$ to R.
   - For $j \in [m]$, R computes $\mathbf{x}_j \leftarrow \mathbf{y}_j^{r_j} \oplus H(j, \mathbf{t}_j)$.
4. Output:
   - S outputs $\perp$ and R outputs $\{\mathbf{x}_j\}_{j\in[m]}$.

Fig. 3. Protocol implementing passively secure OT extension [5,12].

As a first attempt, suppose S simply signs all its messages in Step 3. Recall that we will use this construction to have P_1 send the appropriate input wire labels to P_2; namely, P_1 acts as S in the OT extension and inputs the wire labels for P_2's input wires whereas P_2 acts as R and inputs its input bits. Thus, our first step is to enhance the protocol in Fig. 3 to have S send $\sigma' \leftarrow_{\$} \mathsf{Sign}((j, \mathbf{y}_j^0))$ and $\sigma'' \leftarrow_{\$} \mathsf{Sign}((j, \mathbf{y}_j^1))$ in Step 3.

Now, if P_2 gets an invalid (with respect to a signed GC sent in the PVC protocol of Sect. 4) wire label $\mathbf{x}_j$, it can easily construct a certificate Cert which demonstrates P_1's cheating. Namely, it outputs as its certificate the tuple $\left(b, j, \mathbf{y}_j^0, \mathbf{y}_j^1, \sigma', \sigma'', \mathbf{t}_j\right)$ along with the (signed by P_1 and opened) GC containing the invalid wire label. A third party can (1) check that σ' and σ'' are valid signatures and (2) compute $\mathbf{x}_j^b \leftarrow H(j, \mathbf{t}_j) \oplus \mathbf{y}_j^b$ and check that $\mathbf{x}_j^b$ is indeed an invalid wire label for the given garbled circuit.

This works for protecting against a malicious P_1; however, note that P_2 can easily *defame* an honest P_1 by outputting $\mathbf{t}_j^* \neq \mathbf{t}_j$ as part of its certificate (in which case $\mathbf{x}_j^b \leftarrow H(j, \mathbf{t}_j^*) \oplus \mathbf{y}_j^b$ will very likely be an invalid wire label). Thus, the main difficulty in constructing signed-OT extension is tying P_2 to its choice of the matrix $\mathbf{T}$ generated in Step 1 of the protocol so it cannot blame an honest P_1 by using invalid rows $\mathbf{t}_j^*$ in its certificate.

Towards this end, consider the following modification. In Step 1, R now additionally sends commitments to each $\mathbf{t}_j$ to S, and S signs these and sends them as part of its messages in Step 3. This prevents R from later changing $\mathbf{t}_j$ to blame S. This does not quite work, however, as R could simply commit to an incorrect $\mathbf{t}_j^*$ in the first place! Clearly, R cannot send $\mathbf{T}$ to S, as this would leak R's selection bits, yet we still need R to somehow be committed to its choice of the matrix $\mathbf{T}$. The key insight is noting that S does in fact know *some* of the bits of $\mathbf{T}$; namely, it knows those columns at which $s_i = 0$ (as it learns $\mathbf{t}^i$ in the base OT). We can use this information to tie R to its choice of $\mathbf{T}$ such that it cannot later construct some matrix $\mathbf{T}^* \neq \mathbf{T}$ to defame S.

We do this by enhancing Step 3 as follows. Let I^0 be the set of indices i such that $s_i = 0$ (recall that $\mathbf{s}$ is the random selection bits of S input to the base OTs in Step 1). Let $t_{j,i}$ denote the ith bit in row $\mathbf{t}_j$. Note that S knows the values of $t_{j,i}$ for $i \in I^0$, and could thus compute $\{(i, t_{j,i})\}_{i \in I^0}$ as a "binding" of R's choice of $\mathbf{t}_j$. By including this information in its signature, S enforces that any $\mathbf{t}_j^*$ that R tries to use to blame S must match in the given positions. This brings us closer to our goal; however, there are still two issues that we need to resolve:

1. Sending $\{(i, t_{j,i})\}_{i \in I}$ to R leaks $\mathbf{s}$, which allows R to learn both of S's inputs. We address this by increasing the number of base OTs in Step 1 and having S only send some subset $I \subseteq I^0$ such that $|I| = \kappa$. Thus, while R learns that $s_i = 0$ for $i \in I$, by increasing the number of base OTs enough, R does not have enough information to recover $\mathbf{s}$.
2. R can still flip one bit in $\mathbf{t}_j$ and pass the check with high probability. We fix this by having each $\mathbf{t}_j$ be generated by a seed $\mathbf{k}_j$. Namely, R computes

$\mathbf{t}_j \leftarrow G(\mathbf{k}_j)$ in Step 1, where G is a random oracle[4]. Then, when blaming S, R must reveal $\mathbf{k}_j$ instead of $\mathbf{t}_j$. Thus, with high probability a malicious polytime R cannot find some $\mathbf{k}_j^* \neq \mathbf{k}_j$ such that the Hamming distance between $G(\mathbf{k}_j^*)$ and $G(\mathbf{k}_j)$ is small enough that the above check succeeds.

Finally, note that we have thus far considered the passively secure OT extension protocol, which is insecure against a malicious R. We thus utilize the maliciously secure OT extension protocol of Asharov et al. [6]. The only way R can cheat in passively secure OT extension is by using different $\mathbf{r}$ values in Step 2. Asharov et al. add a "consistency check" phase between Steps 1 and 2 to enforce that $\mathbf{r}$ is consistent. This does not affect our construction, and thus we can include this step to complete the protocol[5]. We refer the reader to Asharov et al. [6] for the justification and intuition of this step; as far as this work is concerned we can treat this consistency check as a "black box".

Observation 1 (OT Extension Matrix Size). *We set ℓ, the number of base OTs, so that leaking κ bits to R does not allow it to recover $\mathbf{s}$ and thus both messages. We do this as follows. Let ℓ' be the number of base OTs required in malicious OT extension [6]. We set $\ell = \ell' + \kappa$ and require that when S chooses $\mathbf{s}$, it first fixes κ randomly selected bits to zero before randomly setting the rest of the bits. Now, when S reveals I to R, the number of unknown bits in $\mathbf{s}$ is equal to ℓ' and thus the security of the Asharov et al. scheme carries over to our setting. Asharov et al. set $\ell' \approx 1.6\kappa$, and thus us using κ extra columns results in an $\approx 67\,\%$ matrix size increase.*

Observation 2 (Batching Signatures). *The main computational cost of our protocol is the signatures sent by S in Step 4. This cost can easily be brought to negligible, as follows. Recall that when using our protocol for transferring the input wire labels of a GC using free-XOR we can optimize the communication slightly by setting $\mathbf{x}_j^0 \leftarrow H(j, \mathbf{q}_j)$ and $\mathbf{y}_j^1 \leftarrow \mathbf{x}_j^0 \oplus \Delta \oplus H(j, \mathbf{q}_j \oplus \mathbf{s})$, where Δ is the free-XOR global offset. Thus, S only needs to send (and sign) $\mathbf{y}_j^1$.*

The most important idea, however, is to batch messages across OT executions and have S sign (and send) only one *signature which includes all the necessary information across many OTs. Namely, using the free-XOR optimization above, S signs and sends the tuple $(I, \{\mathbf{y}_j^1, \{t_{j,i}\}_{i \in I}\}_{j \in [m]})$ to R. We note that the j values need not be sent as they are implied by the protocol execution.*

Figure 4 gives the full protocol for signed-OT extension. For clarity of presentation, this description, and the following proof of security, does not take into account the optimizations described in Observation 2.

[4] Note that G *cannot* be a pseudorandom generator because the input to G is not necessarily uniform as the inputs may be adversarially chosen by R.

[5] The reason this does not affect our construction is because the consistency check phase only involves R sending messages to S. A malicious R cannot defame S because we are only enforcing that R's value $\mathbf{r}$ is consistent.

S's inputs: Messages $\{(\mathbf{x}_j^0, \mathbf{x}_j^1)\}_{j\in[m]}$ where $\mathbf{x}_j^0, \mathbf{x}_j^1 \in \{0,1\}^n$; signing key sk.
R's inputs: Selection bits $\mathbf{r} = (r_1, \ldots, r_m)$; verification key vk.
Common inputs: Security parameter κ; statistical security parameter ρ; number of base OTs ℓ; number of check functions μ; random oracle $G : \{0,1\}^\kappa \to \{0,1\}^\ell$; random oracle $H : \mathbb{N} \times \{0,1\}^\ell \to \{0,1\}^n$; random oracle $H' : \{0,1\}^m \to \{0,1\}^\kappa$; $\mathsf{EU\text{-}CMA}$ signature scheme $\Pi = (\mathsf{KeyGen}', \mathsf{Sign}', \mathsf{Verify}')$; ideal functionality $\mathcal{F}_{\mathsf{OT}}$.

1. Initial OT Phase:
   - S computes $\mathbf{s} \in \{0,1\}^\ell$ as follows. Let I be a set of indices, where $|I| = \kappa$. For $i \in I$, S sets $s_i = 0$. Then, S fills the remaining bits at random.
   - For $j \in [m]$, R computes $\mathbf{k}_j \leftarrow\!\!\$\ \{0,1\}^\kappa$ and sets $\mathbf{t}_j \leftarrow G(\mathbf{k}_j)$.
   - Let $\mathbf{T}$ be an $m \times \ell$ matrix, where the jth row is $\mathbf{t}_j$ and the ith column is $\mathbf{t}^i$. Let $\mathbf{V}$ be an $m \times \ell$ matrix, where the jth row is $\mathbf{v}_j$ and the ith column is $\mathbf{v}^i$. S and R run $\mathcal{F}_{\mathsf{OT}}$ ℓ times in parallel, where S acts as the *receiver* with input s_i and R acts as the *sender* with input $(\mathbf{t}^i, \mathbf{v}^i)$.
2. OT Extension Phase (Part I):
   - For $i \in [\ell]$, R sets $\mathbf{u}^i \leftarrow \mathbf{t}^i \oplus \mathbf{v}^i \oplus \mathbf{r}$, and sends $\mathbf{u}^i$ to S.
3. Consistency check of $\mathbf{r}$:
   - Same as in maliciously-secure OT extension protocol of Asharov et al. [6].
4. OT Extension Phase (Part II):
   - Let $\mathbf{Q}$ be the $m \times \ell$ matrix where each column $\mathbf{q}^i = (s_i \cdot (\mathbf{u}^i \oplus \mathbf{v}^i)) \oplus ((1 - s_i) \cdot \mathbf{t}^i)$. Note that $\mathbf{q}^i = (s_i \cdot \mathbf{r}) \oplus \mathbf{t}^i$ and $\mathbf{q}_j = (r_j \cdot \mathbf{s}) \oplus \mathbf{t}_j$.
   - Let I be the set defined in Step 1, and let $t_{j,i}$ denote the ith bit in row $\mathbf{t}_j$. S sends I to R, who checks that $|I| = \kappa$ and otherwise aborts.
   - For $j \in [m]$, S computes $\mathbf{y}_j^0 \leftarrow \mathbf{x}_j^0 \oplus H(j, \mathbf{q}_j)$ and $\mathbf{y}_j^1 \leftarrow \mathbf{x}_j^1 \oplus H(j, \mathbf{q}_j \oplus \mathbf{s})$ and signatures $\sigma_j' \leftarrow \mathsf{Sign}'_{\mathsf{sk}}\left((I, j, \mathbf{y}_j^0, \{t_{j,i}\}_{i\in I})\right)$, and $\sigma_j'' \leftarrow \mathsf{Sign}'_{\mathsf{sk}}\left((I, j, \mathbf{y}_j^1, \{t_{j,i}\}_{i\in I})\right)$, and sends $\left(j, \mathbf{y}_j^0, \mathbf{y}_j^1, \{t_{j,i}\}_{i\in I}, \sigma_j', \sigma_j''\right)$ to R.
   - For $j \in [m]$, R computes $\mathbf{x}_j \leftarrow \mathbf{y}_j^{r_j} \oplus H(j, \mathbf{t}_j)$.
5. Output:
   - S outputs $\perp$; R outputs $\left\{\mathbf{x}_j, \left(j, r_j, \mathbf{k}_j, I, \mathbf{y}_j^0, \mathbf{y}_j^1, \{t_{j,i}\}_{i\in I}, \sigma_j', \sigma_j''\right)\right\}_{j\in[m]}$.

Fig. 4. Signed-OT extension, based on the OT extension protocol of Asharov et al. [6].

3.2 Towards a Proof of Security

Before presenting the security proof, we first motivate the need for $\mathsf{EU\text{-}CMPRA}$ signature schemes. As mentioned in Sect. 3.1, ideally we could just have S sign everything using an $\mathsf{EU\text{-}CMA}$ signature scheme; however, this presents opportunities for R to defame S. Thus, we need to enforce that R cannot output an $\mathbf{x}_j^b$ value different from the one sent by S. We do so by using a binding commitment scheme $\Pi_{\mathsf{Com}} = (\mathsf{ComGen}, \mathsf{Com})$, and show that the messages sent by S in Step 4 are essentially binding commitments to the underlying $\mathbf{x}_j^b$ values.

We define Π_{Com} as follows, where $G : \{0,1\}^\kappa \to \{0,1\}^\ell$ and $H : \mathbb{N} \times \{0,1\}^\ell \to \{0,1\}^\kappa$ are random oracles, and $\ell \geq \kappa$.

1. $\mathsf{ComGen}(1^\kappa)$: choose set $I \subseteq [\ell]$ uniformly at random subject to $|I| = \kappa$; output $\mathsf{params} \leftarrow I$.
2. $\mathsf{Com}(\mathsf{params}, \mathbf{m}, j; \mathbf{r})$: On input parameters $I \leftarrow \mathsf{params}$, message $\mathbf{m}$, counter j, and randomness $\mathbf{r} \in \{0,1\}^\kappa$, proceed as follows. Compute $\mathbf{t} \leftarrow G(\mathbf{r})$, set $\mathsf{com} \leftarrow (j, \mathbf{m} \oplus H(j, \mathbf{t}), I, \{t_i\}_{i \in I})$, and output com.

We make the assumption that given I, one can derive the randomness input to ComGen. (We use this when defining our EU-CMPRA signature scheme below, which uses a generic binding commitment scheme). We can satisfy this by simply letting the randomness input to ComGen be the set I.

In our signed-OT extension protocol, the set I chosen by S is used as params and the $\mathbf{k}_j$ values chosen by R are used as the randomness to Com. The commitment value com is exactly the message signed and sent by S in Step 4. Thus, ignoring the signatures for now, we have an OT extension protocol that binds S to its $\mathbf{x}_j^b$ values, and thus prevents a malicious R from defaming an honest S. Adding in the signatures (cf. Sect. 3.3) gives us an EU-CMPRA signature scheme. Namely, S is tied to its messages due to the signatures and R is prevented from "changing" the messages to defame S due to the binding property of the commitment scheme.

We now prove that the commitment scheme described above is binding. We actually prove something stronger than what is required in our protocol. Namely, we prove that an adversary who can control *both* random values still cannot win, whereas when we use this commitment scheme in our signed-OT extension protocol, only one of the two random values can be controlled by any one party.

Theorem 1. *Protocol Π_{Com} is binding according to Definition 3.*

Proof. Adversary $\mathcal{A}$ needs to come up with choices of I, $\mathbf{m}$, $\mathbf{m}'$, j, j', $\mathbf{r}$, and $\mathbf{r}'$ such that $(j, \mathbf{m} \oplus H(j, \mathbf{t}), I, \{t_i\}_{i \in I}) = (j', \mathbf{m}' \oplus H(j', \mathbf{t}'), I, \{t'_i\}_{i \in I'})$, where $\mathbf{t} \leftarrow G(\mathbf{r})$ and $\mathbf{t}' \leftarrow G(\mathbf{r}')$. Clearly, $j = j'$. Thus, $\mathcal{A}$ must find $\mathbf{t}$ and $\mathbf{t}'$ such that $t_i = t'_i$ for all $i \in I$. However, by the property that G is a random oracle, the values $\mathbf{t}$ and $\mathbf{t}'$ are distributed uniformly at random in $\{0,1\}^\ell$. Thus, the probability that $\mathcal{A}$ finds two bitstrings $\mathbf{t}$ and $\mathbf{t}'$ that match in κ bits is negligible, regardless of the choice of I. ■

3.3 An EU-CMPRA Signature Scheme

We now show that the messages sent by S in Step 4 form an EU-CMPRA signature scheme. Let $\Pi' = (\mathsf{Gen}', \mathsf{Sign}', \mathsf{Verify}')$ be an EU-CMA signature scheme and $\Pi_{\mathsf{Com}} = (\mathsf{ComGen}, \mathsf{Com})$ be a commitment scheme satisfying Definition 3 (e.g., the scheme presented in Sect. 3.2). Consider the scheme $\Pi = (\mathsf{Gen}, \mathsf{Sign}, \mathsf{Verify})$ defined as follows.

1. $\mathsf{Gen}(1^\kappa)$: On input 1^κ, run $(\mathsf{vk}, \mathsf{sk}) \leftarrow\!\!\$\ \mathsf{Gen}'(1^\kappa)$ and output $(\mathsf{vk}, \mathsf{sk})$.
2. $\mathsf{Sign}_{\mathsf{sk}}(\mathbf{m}, j; (\mathbf{r}_1^*, \mathbf{r}_2^*))$: On input message $\mathbf{m} \in \{0,1\}^\kappa$, counter $j \in \mathbb{N}$, and randomness $\mathbf{r}_1^*$ and $\mathbf{r}_2^*$, proceed as follows. Compute $\mathsf{params} \leftarrow \mathsf{ComGen}(1^\kappa; \mathbf{r}_1^*)$ and $\mathsf{com} \leftarrow \mathsf{Com}(\mathsf{params}, \mathbf{m}, j; \mathbf{r}_2^*)$. Next, choose $\mathbf{m}' \leftarrow\!\!\$\ \{0,1\}^\kappa$ and compute

$\mathsf{com}' \leftarrow \mathsf{Com}(\mathsf{params}, \mathbf{m}', j; \mathbf{r}_2^*)$[6]. Output $\sigma \leftarrow (j, \mathsf{params}, \mathbf{r}_2^*, \mathsf{com}, \mathsf{com}', \mathsf{Sign}'_{\mathsf{sk}}((\mathsf{params}, \mathsf{com})), \mathsf{Sign}'_{\mathsf{sk}}((\mathsf{params}, \mathsf{com}')))$.

3. $\mathsf{Verify}_{\mathsf{pk}}(\mathbf{m}, \sigma)$: On input message $\mathbf{m}$ and signature σ, parse σ as $(j, \mathsf{params}, \mathbf{r}, \mathsf{com}', \mathsf{com}'', \sigma', \sigma'')$, and output 1 if and only if (1) $\mathsf{Com}(\mathsf{params}, \mathbf{m}; \mathbf{r}) = \mathsf{com}'$, (2) $\mathsf{Verify}'_{\mathsf{vk}}((\mathsf{params}, \mathsf{com}'), \sigma') = 1$, and (3) $\mathsf{Verify}'_{\mathsf{vk}}((\mathsf{params}, \mathsf{com}''), \sigma'') = 1$; otherwise output 0.

As explained in Sect. 3.2, this signature scheme exactly captures the behavior of S in our signed-OT extension protocol. We now prove that this is indeed an EU-CMPRA signature scheme.

Theorem 2. *Given an EU-CMA signature scheme* $\Pi' = (\mathsf{Gen}', \mathsf{Sign}', \mathsf{Verify}')$ *and a commitment scheme* $\Pi_{\mathsf{Com}} = (\mathsf{ComGen}, \mathsf{Com})$ *secure according to Definition 3, then* $\Pi = (\mathsf{Gen}, \mathsf{Sign}, \mathsf{Verify})$ *described above is an EU-CMPRA signature scheme.*

Proof. Let $\mathcal{A}$ be a PPT adversary attacking Π. We construct an adversary $\mathcal{B}$ attacking Π'. Adversary $\mathcal{B}$ receives vk from the challenger and initializes $\mathcal{A}$ with vk as input. Let $(\mathbf{m}, j, \mathbf{r}_1^*, \mathbf{r}_2^*)$ be the input of $\mathcal{A}$ to its signing oracle. Adversary $\mathcal{B}$ emulates the execution of $\mathcal{A}$'s signing oracle as follows: it computes $\mathsf{params} \leftarrow \mathsf{ComGen}(1^\kappa; \mathbf{r}_1^*)$ and $\mathsf{com} \leftarrow \mathsf{Com}(\mathsf{params}, \mathbf{m}, j; \mathbf{r}_2^*)$, chooses $\mathbf{m}'$ uniformly at random and computes $\mathsf{com}' \leftarrow \mathsf{Com}(\mathsf{params}, \mathbf{m}', j; \mathbf{r}_2^*)$, constructs $\sigma \leftarrow (j, \mathsf{params}, \mathbf{r}_2^*, \mathsf{com}, \mathsf{com}', \mathsf{Sign}'_{\mathsf{sk}}((\mathsf{params}, \mathsf{com})), \mathsf{Sign}'_{\mathsf{sk}}((\mathsf{params}, \mathsf{com}')))$, and sends σ to $\mathcal{A}$. After each of $\mathcal{A}$'s queries, $\mathcal{B}$ stores $(\mathbf{m}, j)$ in set $\mathcal{Q}_{\mathcal{A}}$ and stores all the messages it sent to its signing oracle in set $\mathcal{Q}_{\mathcal{B}}$.

Eventually, $\mathcal{A}$ outputs $(\mathbf{m}, (j, \sigma'))$ as its forgery. Adversary $\mathcal{B}$ checks that $\mathsf{Verify}_{\mathsf{vk}}(\mathbf{m}, (j, \sigma')) = 1$ and that $(\mathbf{m}, j) \notin \mathcal{Q}_{\mathcal{A}}$. If not, $\mathcal{B}$ outputs 0. Otherwise, $\mathcal{B}$ parses σ' as $(\mathsf{params}, \mathbf{r}, \mathsf{com}', \mathsf{com}'', \sigma', \sigma'')$ and checks that $\mathsf{com}' \notin \mathcal{Q}_{\mathcal{B}}$. If so, it outputs (com', σ'); otherwise it outputs 0.

Note that $\mathsf{Sig\text{-}forge}^{\mathsf{CMPRA}}_{\mathcal{A},\Pi}(\kappa) = 1$ and $\mathsf{Sig\text{-}forge}^{\mathsf{CMA}}_{\mathcal{B},\Pi'}(\kappa) = 0$ if and only if $\mathsf{Verify}_{\mathsf{vk}}(\mathbf{m}, (j, \mathsf{params}, \mathbf{r}, \mathsf{com}', \mathsf{com}'', \sigma', \sigma'')) = 1$ and $(\mathbf{m}, j) \notin \mathcal{Q}_{\mathcal{A}}$ but $\mathsf{com}' \in \mathcal{Q}_{\mathcal{B}}$. Fix some $(\mathbf{m}, (j, \mathsf{params}, \mathbf{r}, \mathsf{com}_1, \mathsf{com}_{1'}, \sigma_1, \sigma_{1'}))$ such that this is the case. Thus it holds that $\mathsf{com}_1 \in \mathcal{Q}_{\mathcal{B}}$. This implies that $\mathcal{B}$ queried Sign' on com_1, which means that $\mathcal{A}$ queried *its* signing oracle on some $(\mathbf{m}', j', \mathbf{r}_1^*, \mathbf{r}_2^*)$, where $\mathbf{m}' \neq \mathbf{m}$, and received back $(j', \mathsf{params}, \mathbf{r}', \mathsf{com}_1, \mathsf{com}_{2'}, \sigma_{1''}, \sigma_{2'})$. However, this implies that $\mathsf{Com}(\mathsf{params}, \mathsf{com}_1; \mathbf{r}) = \mathbf{m}$ and $\mathsf{Com}(\mathsf{params}, \mathsf{com}_1; \mathbf{r}') = \mathbf{m}'$. Thus, $\Pr[\mathsf{Sig\text{-}forge}^{\mathsf{CMPRA}}_{\mathcal{A},\Pi}(\kappa)] = \Pr[\mathsf{Sig\text{-}forge}^{\mathsf{CMA}}_{\mathcal{B},\Pi}(\kappa)] + \Pr[\mathsf{Com\text{-}bind}_{\mathcal{B}',\Pi_{\mathsf{Com}}}(\kappa)]$ for some PPT adversary $\mathcal{B}'$. We now bound $\Pr[\mathsf{Com\text{-}bind}_{\mathcal{B}',\Pi_{\mathsf{Com}}}(\kappa)]$.

Adversary $\mathcal{B}'$ runs almost exactly like $\mathcal{B}$. On the first query $(\mathbf{m}, j, \mathbf{r}_1^*, \mathbf{r}_2)$ by $\mathcal{A}$, it sets $\mathbf{r} = \mathbf{r}_1^*$ if $\mathbf{r}_1^* \neq \perp$ and otherwise it sets $\mathbf{r}$ uniformly at random; $\mathcal{B}'$ then sends $\mathbf{r}$ to $\mathcal{C}$, receiving back params.

Let $(\mathbf{m}_1, j_1, \mathbf{r}_1^*, \mathbf{r}_2^*)$ and $(\mathbf{m}_2, j_2, \mathbf{r}_1^*, \mathbf{r}_2^{*'})$ be the two queries made by $\mathcal{A}$ resulting in a common commitment value. Let $(j_1, \mathsf{params}, \mathbf{r}_1, \mathsf{com}_1, \mathsf{com}'_1, \sigma_1, \sigma_{1'})$ and $(j_2, \mathsf{params}, \mathbf{r}_2, \mathsf{com}_1, \mathsf{com}'_2, \sigma_{1''}, \sigma_{2'})$ be the corresponding signatures resulting

[6] This extra commitment on a random message is needed for our signed-OT extension proof.

from $\mathcal{A}$'s queries. Adversary $\mathcal{B}'$ sends $(\mathsf{com}_1, \mathbf{m}_1, j_1, \mathbf{r}_2^*, \mathbf{m}_2, j_2, \mathbf{r}_2^{*'})$ to its challenger and wins with probability one, contradicting the security of the commitment scheme. Thus, we have that $\Pr[\mathsf{Com\text{-}bind}_{\mathcal{B}', \Pi_{\mathsf{Com}}}(\kappa)] < \mathsf{negl}(\kappa)$, completing the proof. ■

3.4 Proof of Security

We are now ready to prove the security of our signed-OT extension protocol. Most of the proof complexity is hidden in the proofs of the associated EU-CMPRA signature scheme and commitment scheme. Thus, the signed-OT extension simulator is relatively straightforward, and mostly involves parsing the output of $\mathcal{F}_{\mathbf{signedOT}}^{\Pi}$ and passing the correct values to the adversary. The analysis follows almost exactly that of Asharov et al. [6] and thus we elide most of the details.

Theorem 3. *Let $\Pi = (\mathsf{Gen}, \mathsf{Sign}, \mathsf{Verify})$ be the EU-CMPRA signature scheme in Sect. 3.3. Then the protocol in Fig. 4 is a secure realization of $\mathcal{F}_{\mathbf{signedOT}}^{\Pi}$ in the $\mathcal{F}_{\mathsf{OT}}$-hybrid model.*

Proof. We separately consider the case where S is malicious and R is malicious. The case where the parties are either both honest or both malicious is straightforward.

Malicious S . Let $\mathcal{A}$ be a PPT adversary corrupting S. We construct a simulator $\mathcal{S}$ as follows.

1. The simulator $\mathcal{S}$ acts as an honest R would in Step 1, extracting $\mathbf{s}$ from $\mathcal{A}$'s input to $\mathcal{F}_{\mathsf{OT}}$.
2. The simulator $\mathcal{S}$ acts as an honest R would in Steps 2 and 3.
3. Let I and $\left(j, \mathbf{y}_j^0, \mathbf{y}_j^1, \{t_{j,i}\}_{i \in I}, \sigma'_{j,0}, \sigma'_{j,1}\right)$, for $j \in [m]$, be the messages sent by $\mathcal{A}$ in Step 4. If any of these are invalid, $\mathcal{S}$ sends abort to $\mathcal{F}_{\mathbf{signedOT}}^{\Pi}$ and simulates R aborting, outputting whatever $\mathcal{A}$ outputs.
4. For $j \in [m]$, proceed as follows. The simulator $\mathcal{S}$ extracts $\mathbf{x}_j^0 \leftarrow \mathbf{y}_j^0 \oplus H(j, \mathbf{q}_j)$ and $\mathbf{x}_j^1 \leftarrow \mathbf{y}_j^1 \oplus H(j, \mathbf{q}_j \oplus \mathbf{s})$, constructs $\sigma_{j,b}^* \leftarrow (j, I, \mathbf{k}_j, (I, (j, \mathbf{y}_j^b, I, \{t_{j,i}\}_{i \in I})), (I, (j, \mathbf{y}_j^{1-b}, I, \{t_{j,i}\}_{i \in I})), \sigma'_{j,b}, \sigma'_{j,1-b})$ for $b \in \{0, 1\}$, and sends $\mathbf{x}_j^0$, $\mathbf{x}_j^1$, $\sigma_{j,0}^*$, and $\sigma_{j,1}^*$ to $\mathcal{F}_{\mathbf{signedOT}}^{\Pi}$, receiving back either $((b, m_b), \sigma_{j,b})$ or abort.
5. If $\mathcal{S}$ received abort in any of the above iterations, it simulates R aborting, outputting whatever $\mathcal{A}$ outputs. Otherwise, for $j \in [m]$, $\mathcal{S}$ parses $\sigma_{j,b}$ as $(j, I, \mathbf{k}_j, (I, (j, \mathbf{y}_j^b, I, \{t_{j,i}\}_{i \in I})), (I, (j, \mathbf{y}_j^{1-b}, I, \{t_{j,i}\}_{i \in I})), \sigma'_{j,b}, \sigma'_{j,1-b})$, constructs message $\sigma_j \leftarrow (j, \mathbf{y}_j^0, \mathbf{y}_j^1, \{t_{j,i}\}_{i \in I}, \sigma'_{j,0}, \sigma'_{j,1})$, and acts as an honest R would when receiving messages I and $\{\sigma_j\}_{j \in [m]}$.
6. The simulator $\mathcal{S}$ outputs whatever $\mathcal{A}$ outputs.

It is easy to see that this protocol perfectly simulates a malicious sender since $\mathcal{S}$ acts exactly as an honest R would (beyond feeding the appropriate messages to $\mathcal{F}_{\mathbf{signedOT}}^{\Pi}$).

Malicious R . Let $\mathcal{A}$ be a PPT adversary corrupting R. We construct a simulator $\mathcal{S}$ as follows.

1. The simulator $\mathcal{S}$ acts as an honest S would in Step 1, extracting matrices $\mathbf{T}$ and $\mathbf{V}$ through S's $\mathcal{F}_{\mathsf{OT}}$ inputs, and thus the values $\{\mathbf{k}_j\}_{j\in[m]}$.
2. The simulator $\mathcal{S}$ uses the values extracted above to extract selection bits $\mathbf{r}$ after receiving the $\mathbf{u}^i$ values from $\mathcal{A}$ in Step 2.
3. The simulator $\mathcal{S}$ acts as an honest S would in Step 3.
4. Let I^0 be the indices at which $\mathbf{s}$ (generated in Step 1) is zero, and let $I \subseteq I^0$ be a set of size κ. For $j \in [m]$, $\mathcal{S}$ sends r_j, vk, and I to $\mathcal{F}^{\Pi}_{\mathbf{signedOT}}$, receiving back $((r_j, \mathbf{x}_j^{r_j}), \sigma_{j,r_j})$; $\mathcal{S}$ parses σ_{j,r_j} as $(j, I, \mathbf{r}, (I, (j, \mathbf{c}_{r_j}, I, \{t_{j,i}\}_{i\in I})), (I, (j, \mathbf{c}_{1-r_j}, I, \{t_{j,i}\}_{i\in I})), \sigma'_{j,r_j}, \sigma'_{j,1-r_j})$.
5. In Step 4, $\mathcal{S}$ sends I and $(j, \mathbf{c}_0, \mathbf{c}_1, \{t_{j,i'}\}_{i'\in I'}, \sigma'_{j,0}, \sigma'_{j,1})$, for $j \in [m]$, to $\mathcal{A}$.
6. The simulator $\mathcal{S}$ outputs whatever $\mathcal{A}$ outputs.

The analysis is almost exactly that of the malicious receiver proof in the construction of Asharov et al. [6]; we thus give an informal security argument here and refer the reader to the aforementioned work for the full details.

A malicious R has two main attacks: using inconsistent choices of its selection bits $\mathbf{r}$ and trying to cheat in the signature creation in Step 4. This latter attack is prevented by the security of our EU-CMPRA signature scheme. The former is prevented by the consistency check in Step 3. Namely, Asharov et al. show that the consistency check guarantees that: (1) most inputs are consistent with some string $\mathbf{r}$, and (2) the number of inconsistent inputs is small and thus allow R to only learn a small number of bits of $\mathbf{s}$. Thus, for specific choices of ℓ and μ, the probability of a malicious R cheating is negligible. Asharov et al. provide concrete parameters for various settings of the security parameter [6, §3.2]; let ℓ' denote the number of base OTs used in their protocol. Now, in our protocol we set $\ell = \ell' + \kappa$; S leaks κ bits of $\mathbf{s}$ when revealing the set I in Step 4, and so is left with ℓ' unknown bits of $\mathbf{s}$. Thus, the security argument presented by Asharov et al. carries over into our setting. ■

4 Our Complete PVC Protocol

As noted above, the main technical challenge of the PVC model is in the signed-OT construction and model definitions. The AO protocol in the $\mathcal{F}^{\Pi}_{\mathbf{signedOT}}$-hybrid model is relatively straightforward: the natural (but careful) combination of taking a non-halting covert protocol, having the GC generator P_1 sign appropriate messages, and replacing OTs with signed-OTs works. In particular, our signed-OT extension can be naturally modified and used in place of the signed-OT primitive in the AO protocol.

In this section we present a new PVC protocol based on signed-OT extension. Our protocol is similar to the AO protocol in the $\mathcal{F}^{\Pi}_{\mathbf{signedOT}}$-hybrid model, but with applying several simple yet very effective optimizations, resulting in a much lower communication cost.

We present our protocol by starting off with the AO protocol and pointing out the differences. We presented the AO protocol intuition in the Introduction; see Fig. 5 for its formal description; due to lack of space, we omit the (straightforward) Blame and Judgment algorithms. In presenting our changes, we sketch

Private inputs: P_1 has input $\mathbf{x}_1 \in \{0,1\}^n$ and P_2 has input $\mathbf{x}_2 \in \{0,1\}^n$.
Common inputs: Security parameter κ; XOR-tree replication factor ν; garbled circuit replication factor λ; circuit $C(\cdot,\cdot)$; commitment scheme $\Pi_{\mathsf{Com}} = (\mathsf{Com}, \mathsf{Open})$; ideal functionalities $\mathcal{F}^{\Pi}_{\mathbf{signedOT}}$ and $\binom{\lambda}{1}$-$\mathcal{F}^{\Pi}_{\mathbf{signedOT}}$ for EU-CMPRA signature scheme Π.

1. P_1 and P_2 define a new circuit $C'(\mathbf{x}_1, \mathbf{x}_2^1, \ldots, \mathbf{x}_2^\nu) = C(\mathbf{x}_1, \bigoplus_{i\in[\nu]} \mathbf{x}_2^i)$. Let $w_1, \ldots, w_n$ denote the input wires of $\mathbf{x}_1$ and let $w_{n+(i-1)\nu}, \ldots, w_{n+i\nu}$ denote the input wires of $\mathbf{x}_2^i$.
2. For $i \in [\nu - 1]$, P_2 chooses $\mathbf{x}_2^i \leftarrow\!\!\$\, \{0,1\}^n$. P_2 sets $\mathbf{x}_2^\nu \leftarrow (\bigoplus_{i\in[\nu-1]} \mathbf{x}_2^i) \oplus \mathbf{x}_2$.
3. For $j \in [\lambda]$, $i \in [n + \nu n]$, and $b \in \{0,1\}$, P_1 chooses $\mathbf{k}^j_{w_{n+i},b} \leftarrow\!\!\$\, \{0,1\}^\kappa$.
4. P_1 and P_2 run νn instantiations of $\mathcal{F}^{\Pi}_{\mathbf{signedOT}}$, where in the ith execution P_1 acts as the sender with input $(\mathbf{k}^1_{w_{n+i},0} \| \ldots \| \mathbf{k}^\lambda_{w_{n+i},0}, \mathbf{k}^1_{w_{n+i},1} \| \ldots \| \mathbf{k}^\lambda_{w_{n+i},1})$ and P_2 acts as the receiver with input $\mathbf{x}_2^{\lceil i/n \rceil}[i \bmod \nu]$. If P_2's output is abort_1, it outputs abort_1.
5. For $j \in [\lambda]$, P_1 constructs garbled circuit GC_j of circuit C', where for $i \in [n + \nu n]$ the keys for input wire w_i are $\mathbf{k}^j_{w_i,0}$ and $\mathbf{k}^j_{w_i,1}$. P_1 sends $(\mathrm{GC}_j, \mathsf{Sign}(\mathrm{GC}_j))$ to P_2, who checks that the signature is valid; if not, P_2 outputs abort_1.
6. For $i \in [n]$ and $j \in [\lambda]$, P_1 chooses $b \leftarrow\!\!\$\, \{0,1\}$, computes commitments $(\mathbf{c}^j_{w_i,0}, \mathbf{o}^j_{w_i,0}) \leftarrow\!\!\$\, \mathsf{Com}(\mathbf{k}^j_{w_i,0})$ and $(\mathbf{c}^j_{w_i,1}, \mathbf{o}^j_{w_i,0}) \leftarrow\!\!\$\, \mathsf{Com}(\mathbf{k}^j_{w_i,1})$, and sends $(\mathbf{c}_{w_i,b}, \mathsf{Sign}(\mathbf{c}_{w_i,b}))$ and $(\mathbf{c}_{w_i,\bar{b}}, \mathsf{Sign}(\mathbf{c}_{w_i,\bar{b}}))$ to P_2, who checks that the signatures are valid; if not, P_2 outputs abort_1.
7. P_1 and P_2 run $\binom{\lambda}{1}$-$\mathcal{F}^{\Pi}_{\mathbf{signedOT}}$ with P_1 as the sender inputting $(\{\mathbf{k}^i_{w_p,b}\}_{i\in[\lambda]\setminus\{j\}, p\in[n+\nu n], b\in\{0,1\}}, \{\mathbf{o}^i_{w_p,b}\}_{i\in[\lambda]\setminus\{j\}, p\in[n], b\in\{0,1\}}, \{\mathbf{k}^j_{w_i,\mathbf{x}_1[i]}\}_{i\in[n]})$ as its jth input and P_2 as the receiver inputting $\gamma \leftarrow\!\!\$\, [\lambda]$ as its input; if P_2's output is abort_1, it outputs abort_1.
8. P_2 does the following:
   - For $j \in [\lambda]\setminus\{\gamma\}$, $i \in [n]$, and $b \in \{0,1\}$, P_2 checks that $\mathsf{Open}(\mathbf{c}^j_{w_i,b}, \mathbf{o}^j_{w_i,b}) = \mathbf{k}^j_{w_i,b}$. If not, P_2 sets $\mathsf{key} \leftarrow \mathsf{InvalidDecommitment}$ and moves to Step 9.
   - For $j \in [\lambda]\setminus\{\gamma\}$, P_2 uses the input wire keys received from the signed-OT in Step 7 to check that GC_j is a correctly garbled circuit. If not, P_2 sets $\mathsf{key} \leftarrow \mathsf{InvalidCircuit}$ and moves to Step 9.
   - For $j \in [\lambda]\setminus\{\gamma\}$, P_2 checks that the keys received in the signed-OT in Step 4 match the keys sent by P_1 in Step 7. If not, P_2 sets $\mathsf{key} \leftarrow \mathsf{SelectiveOTAttack}$ and moves to Step 9.
9. If any of the above checks fail, P_2 computes $\mathsf{Cert} \leftarrow \mathsf{Blame}(\mathsf{id}_1, \mathsf{key}, \mathsf{View}_2)$, publishes Cert, and outputs $\mathsf{corrupted}_1$. Otherwise, P_2 uses the keys to compute $C'(\mathbf{x}_1, \mathbf{x}_2^1, \ldots, \mathbf{x}_2^\nu)$ and outputs the result.

Fig. 5. The AO PVC protocol [7, Protocol 3].

the improvement each of them brings. Thus, we start by reviewing the communication cost of the AO protocol.

Communication Cost of the AO Protocol. Using state-of-the-art optimizations [13,19,20], the size of each GC sent in Step 5 is $2\kappa|G_C|$, where $|G_C|$ is

the number of non-XOR gates in circuit C (note that $|G_C| = |G_{C'}|$ for circuit C' generated in Step 1 since the XOR-tree only adds XOR gates to the circuit, which are "free" [13]). Let τ be the field size (in bits), ν the XOR-tree replication factor, λ the GC replication factor, and n the length of the inputs, and assume that each signature is of length τ and the commitment and decommitment values are of length κ. Using the signed-OT instantiations of Asharov and Orlandi [7, Protocols 1 and 2], we get a total communication cost of $\tau(7\nu n + 11) + 2\lambda\kappa\nu n + \ell(2\kappa|G_C| + \tau) + 2n\lambda(\kappa + \tau) + \tau(3 + 2\lambda + 11(\lambda - 1)) + \lambda\kappa(2(n + \nu n)(\lambda - 1) + 2n(\lambda - 1) + n)$.

As an example, consider the secure computation of AES$(\mathbf{m}, \mathbf{k})$, where P_1 inputs message $\mathbf{m} \in \{0,1\}^{128}$ and P_2 inputs key $\mathbf{k} \in \{0,1\}^{128}$, and suppose we set both the GC replication factor λ and the XOR-tree replication factor ν to 3, giving a cheating probability of $\epsilon = 1/2$. Letting $\kappa = 128$ and $\tau = 256$, we have a total communication cost of 9.3 Mbit (where we assume that the AES circuit has 9,100 non-XOR gates [15]).

Our Modifications. We make the following modifications to the AO protocol:

- In Step 6, instead of using a commitment scheme we can use a hash function. This saves on communication in Step 7 as P_1 no longer needs to send the openings $\{\mathbf{o}^i_{w_p,b}\}$ to the commitments in the signed-OT, and is secure when treating H as a random oracle since the keys are generated uniformly at random and thus it is infeasible for P_2 to guess the committed values. The total savings are $2n(\lambda - 1)\kappa\lambda$ bits; in our example, this saves us 196 kbit.
- In Step 3, we use a random seed to generate the input wire keys. Namely, for all $j \in [\lambda]$ we compute $\mathbf{s}_j \leftarrow\!\!\$\, \{0,1\}^\kappa$, and compute the input wire keys for circuit j as $\mathbf{k}^j_{w_1,0} \| \mathbf{k}^j_{w_1,1} \| \cdots \| \mathbf{k}^j_{w_{n+\nu n},0} \| \mathbf{k}^j_{w_{n+\nu n},1} \leftarrow G(\mathbf{s}_j)$, where G is a pseudorandom generator. Now, in the 1-out-of-λ signed-OT in Step 7 we can just send the seeds to the input wire keys rather than the input wire keys themselves. The total savings are $2(n + \nu n)(\lambda - 1)\lambda\kappa - n(\lambda - 1)\lambda\kappa$ bits; in our example, this saves us 688 kbit.
- In Step 5, P_1 generates each GC_j from a seed $\mathbf{s}^j_{\mathrm{GC}}$. (This idea was first put forward by Goyal et al. [11].) That is, $\mathbf{s}^j_{\mathrm{GC}}$ specifies the randomness used to construct all wire keys *except* for the input wire keys which were set in Step 3. Instead of P_1 sending each GC to P_2 in Step 5, P_1 instead sends a commitment $\mathbf{c}^j_{\mathrm{GC}} \leftarrow H(\mathrm{GC}_j)$. Now, in Step 7, P_1 can send the appropriate seeds $\{\mathbf{s}^j_{\mathrm{GC}}\}_{j \in [\lambda] \setminus \{j\}}$ in the jth input of the 1-out-of-λ signed-OT to allow P_2 to check the correctness of the check GCs. We then add an additional step where, if the checks pass, P_1 sends GC_γ (along with a signature on GC_γ) to P_2, who can check whether $H(\mathrm{GC}_\gamma) = \mathbf{c}^\gamma_{\mathrm{GC}}$. Note that this does not violate the security conditions required by the PVC model because P_2 catches any cheating of P_1 before the evaluation circuit is sent. If P_1 tries to cheat here, P_2 already has a commitment to the circuit so can detect any cheating. The total savings are $(\lambda - 1)2\kappa|G_C| - \lambda\tau - \lambda\kappa(\lambda - 1)$ bits; in our example, this saves us 4.6 Mbit.

Private inputs: P_1 has input $\mathbf{x}_1 \in \{0,1\}^n$; P_2 has input $\mathbf{x}_2 \in \{0,1\}^n$.
Common inputs: Security parameter κ; XOR-tree replication factor ν; garbled circuit replication factor λ; circuit $C(\cdot,\cdot)$; hash function $H : \{0,1\}^* \to \{0,1\}^\kappa$; pseudorandom generator $G : \{0,1\}^\kappa \to \{0,1\}^{2(n+\nu n)\kappa}$; ideal functionalities $\mathcal{F}_{\mathbf{signedOT}}^{\Pi}$ and $\binom{\lambda}{1}$-$\mathcal{F}_{\mathbf{signedOT}}^{\Pi}$ for EU-CMPRA signature scheme Π.

1. P_1 and P_2 define a new circuit $C'(\mathbf{x}_1, \mathbf{x}_2^1, \ldots, \mathbf{x}_2^\nu) = C(\mathbf{x}_1, \bigoplus_{i\in[\nu]} \mathbf{x}_2^i)$. Let $w_1, \ldots, w_n$ denote the input wires of $\mathbf{x}_1$ and let $w_{n+(i-1)\nu}, \ldots, w_{n+i\nu}$ denote the input wires of $\mathbf{x}_2^i$.
2. For $i \in [\nu - 1]$, P_2 chooses $\mathbf{x}_2^i \leftarrow\!\!\$\ \{0,1\}^n$ and sets $\mathbf{x}_2^\nu \leftarrow (\bigoplus_{i\in[\nu-1]} \mathbf{x}_2^i) \oplus \mathbf{x}_2$.
3. For $j \in [\lambda]$, P_1 chooses $\mathbf{s}_j \leftarrow\!\!\$\ \{0,1\}^\kappa$ and computes $\mathbf{k}^j_{w_1,0} \| \mathbf{k}^j_{w_1,1} \| \cdots \| \mathbf{k}^j_{w_{n+\nu n},0} \| \mathbf{k}^j_{w_{n+\nu n},1} \leftarrow G(\mathbf{s}_j)$.
4. P_1 and P_2 run νn instantiations of $\mathcal{F}_{\mathbf{signedOT}}^{\Pi}$, where in the ith execution P_1 acts as the sender with input $(\mathbf{k}^1_{w_{n+i},0} \| \cdots \| \mathbf{k}^\lambda_{w_{n+i},0}, \mathbf{k}^1_{w_{n+i},1} \| \cdots \| \mathbf{k}^\lambda_{w_{n+i},1})$ and P_2 acts as the receiver with input $\mathbf{x}_2^{\lceil i/n \rceil}[i \bmod \nu]$. If P_i's output is abort_i, it outputs abort_i.
5. For $j \in [\lambda]$, P_1 computes $\mathbf{s}^j_{\mathrm{GC}} \leftarrow\!\!\$\ \{0,1\}^\kappa$ and uses $\mathbf{s}^j_{\mathrm{GC}}$ as the randomness used to generate garbled circuit GC_j, where for $i \in [n + \nu n]$ the keys for input wire w_i are $\mathbf{k}^j_{w_i,0}$ and $\mathbf{k}^j_{w_i,1}$. P_1 computes $\mathbf{c}^j_{\mathrm{GC}} \leftarrow H(GC_j)$ and sends $(\mathbf{c}^j_{\mathrm{GC}}, \mathsf{Sign}(\mathbf{c}^j_{\mathrm{GC}}))$ to P_2, who checks that the signature is valid; if not, P_2 outputs abort_1.
6. For $i \in [n]$ and $j \in [\lambda]$, P_1 computes $\mathbf{c}^j_{w_i,0} \leftarrow H(\mathbf{k}^j_{w_i,0})$ and $\mathbf{c}^j_{w_i,1} \leftarrow H(\mathbf{k}^j_{w_i,1})$, and sends $(\mathbf{c}_{w_i,b}, \mathsf{Sign}(\mathbf{c}_{w_i,b})), (\mathbf{c}_{w_i,\bar{b}}, \mathsf{Sign}(\mathbf{c}_{w_i,\bar{b}}))$ to P_2, where $b \leftarrow\!\!\$\ \{0,1\}$. P_2 checks that the signatures are valid; if not, P_2 outputs abort_1.
7. P_1 and P_2 run $\binom{\lambda}{1}$-$\mathcal{F}_{\mathbf{signedOT}}^{\Pi}$ with P_1 as the sender and P_2 as the receiver. P_2 uses $\gamma \leftarrow\!\!\$\ [\lambda]$ as its input and P_1 uses $(\{\mathbf{s}_i, \mathbf{s}^i_{\mathrm{GC}}\}_{i\in[\lambda]\setminus\{j\}}, \{\mathbf{k}^j_{w_i,\mathbf{x}_1[i]}\}_{i\in[n]})$ as its jth input. If P_i's output is abort_i, it outputs abort_i.
8. P_2 does the following:
   - For $j \in [\lambda]\setminus\{\gamma\}$, $i \in [n]$, and $b \in \{0,1\}$, P_2 checks that $H(\mathbf{k}^j_{w_i,b}) = \mathbf{c}^j_{w_i,b}$. If not, P_2 sets $\mathsf{key} \leftarrow \mathsf{InvalidDecommitment}$ and moves to Step 12.
   - For $j \in [\lambda]\setminus\{\gamma\}$, P_2 uses $\mathbf{s}_j$ and $\mathbf{s}^j_{\mathrm{GC}}$ received from $\binom{\lambda}{1}$-$\mathcal{F}_{\mathbf{signedOT}}^{\Pi}$ to check that GC_j is a correctly garbled circuit and that $H(\mathrm{GC}_j) = \mathbf{c}^j_{\mathrm{GC}}$. If not, P_2 sets $\mathsf{key} \leftarrow \mathsf{InvalidCircuit}$ and moves to Step 12.
   - For $j \in [\lambda]\setminus\{\gamma\}$, P_2 checks that the keys received in $\mathcal{F}_{\mathbf{signedOT}}^{\Pi}$ match the keys generated by $\mathbf{s}_j$ received in Step 7. If not, P_2 sets $\mathsf{key} \leftarrow \mathsf{SelectiveOTAttack}$ and moves to Step 12.
9. Let $((\gamma, m_\gamma), \sigma)$ be P_2's output of $\binom{\lambda}{1}$-$\mathcal{F}_{\mathbf{signedOT}}^{\Pi}$. P_2 sends (γ, σ) to P_1, who checks that the signature is valid and otherwise outputs abort_2.
10. P_1 sends $(\mathrm{GC}_\gamma, \mathsf{Sign}(\mathrm{GC}_\gamma))$ to P_2, who checks that the signature is valid; if not, P_2 outputs abort_1.
11. P_2 checks that $H(\mathrm{GC}_\gamma) = \mathbf{c}^\gamma_{\mathrm{GC}}$. If not, P_2 sets $\mathsf{key} \leftarrow \mathsf{InvalidCircuitHash}$ and moves to Step 12.
12. If any of the above checks fail, P_2 computes $\mathsf{Cert} \leftarrow \mathsf{Blame}(\mathsf{id}_1, \mathsf{key}, \mathsf{View}_2)$, publishes Cert, and outputs $\mathsf{corrupted}_1$. Otherwise, P_2 uses the keys to compute $C'(\mathbf{x}_1, \mathbf{x}_2^1, \ldots, \mathbf{x}_2^\nu)$ and outputs the result.

Fig. 6. Our PVC protocol.

Our PVC Protocol and Its Cost. Fig. 6 presents our optimized protocol. For simplicity, we sign each message in Steps 5 and 6 separately; however, we note that we can group all the messages in a given step into a single signature (cf. Observation 2). The Blame and Judgment algorithms are straightforward and similar to the AO protocol (Blame outputs the relevant parts of the view, including the cheater's signatures, and Judgment checks the signatures). We prove the following theorem in the full version.

Theorem 4. *Let $\lambda < p(\kappa)$ and $\nu < p(\kappa)$, for some polynomial $p(\cdot)$, be parameters to the protocol, and set $\epsilon = (1-1/\lambda)(1-2^{-\nu+1})$. Let f be a* PPT *function, let H be a random oracle, let $\mathcal{F}_{\mathbf{signedOT}}^{\Pi}$ and $\binom{\lambda}{1}$-$\mathcal{F}_{\mathbf{signedOT}}^{\Pi}$ be the $\binom{2}{1}$-signed-OT and $\binom{\lambda}{1}$-signed-OT ideal functionalities, respectively, where Π is an EU-CMPRA signature scheme. Then the protocol in Fig. 6 securely computes f in the presence of (1) an ϵ-PVC adversary corrupting P_1 and (2) a malicious adversary corrupting P_2.*

Using our AES circuit example, we find that the total communication cost is now 2.5 Mbit, plus the cost of signed-OT/signed-OT extension. In this particular example, signed-OT requires around 1 Mbit and signed-OT extension requires around 1.4 Mbit. However, as we show below, as the number of OTs required grows, signed-OT extension quickly outperforms signed-OT, both in communication and computation.

5 Comparison with Prior Work

We now compare our signed-OT extension construction (including optimizations, and in particular, the signature batching of Observation 2) with the signed-OT protocol of Asharov and Orlandi [7], along with a comparison of existing covert and malicious protocols and our PVC protocol using both signed-OT and signed-OT extension. All comparisons are done through calculating the number of bits transferred and estimated running times based on the relative cost of public-key versus symmetric-key operations. We use a very conservative (low-end) estimate on the public/symmetric speed ratio. We note that this ratio does vary greatly across platforms, being much higher on low power mobile devices, which often employ a weak CPU but have hardware AES support. For such platforms our numbers would be even better.

Recall that τ is the field size (in bits), ν is the XOR-tree replication factor, λ is the GC replication factor, n is the input length, and we assume that each signature is of length τ.

Communication Cost. We first focus on the *communication cost* of the two protocols. The signed-OT protocol of Asharov and Orlandi [7] is based on the maliciously secure OT protocol of Peikert et al. [18], and inherits similar costs. Namely, the communication cost of executing ℓ OTs each of length n is $(6\ell+11)\tau$ if $n \leq \tau$, and $(6\ell + 11)\tau + 2n\ell$ if $n > \tau$. Signed-OT requires the additional communication of a signature per OT, adding an additional $\tau\ell$ bits. In the

underlying secure computation protocol we have that $n = \lambda\kappa$, where λ is the garbled circuit replication factor. For simplicity, we set $\lambda = 3$ (which along with an XOR-tree replication factor of three equates to a deterrence factor of $\epsilon = 1/2$) and thus $n = 3\kappa$. Thus, the total communication cost of executing t signed-OTs is $\boldsymbol{\tau(7t + 11)}$ bits if $3\kappa \leq \tau$ and $\boldsymbol{\tau(7t + 11) + 6\kappa t}$ bits otherwise.

On the other hand, the cost of signed-OT extension for t OTs is $(6\ell + 11)\tau + 2\ell t + \ell t + \mu\ell \log \ell + 4\mu\ell\kappa + \kappa \log \ell + (n + \kappa)t + \tau$. Asharov et al. [6, §3.2] present concrete choices of μ and ℓ for various security parameters. However, in our setting we need to increase ℓ by κ bits. Thus, let ℓ' be the particular choice of ℓ specified by Asharov et al. We then set $\ell = \ell' + \kappa$. Thus, for short security parameter we set $\ell = 133 + 80 = 213$ and $\mu = 3$, and for long security parameter we set $\ell = 190 + 128 = 318$ and $\mu = 2$. Thus, the total communication cost of executing t signed-OTs when using signed-OT extension is $\boldsymbol{(6\ell + 12)\tau + (3\ell + n + \kappa)t + \mu\ell \log \ell + 4\mu\ell\kappa + \kappa \log \ell}$ bits.

	1,000 OTs			10,000 OTs		
Security	sOT	sOT-ext	Improvement	sOT	sOT-ext	Improvement
Short (FFC)	7,179	2,539	2.8×	71,691	11,305	6.3×
Short (ECC)	1,602	1,398	1.1×	16,002	10,164	1.6×
Long (FFC)	21,538	7,694	2.8×	215,074	20,888	10.3×
Long (ECC)	2,563	2,288	1.1×	25,603	15,482	1.7×

Fig. 7. Communication cost (in kbits) of transferring the input wire labels for P_2 when using signed-OT (sOT) versus signed-OT extension (sOT-ext) for 1,000 and 10,000 OTs.

Figure 7 presents a comparison of the communication cost of both approaches when executing 1,000 and 10,000 OTs, for various keylength settings and underlying public-key cryptosystems. We see improvements from 1.1–10.3×, depending on the number of OTs, the underlying public-key cryptosystem, and the size of the security parameter. Note that for a smaller number of OTs (such as 100), signed-OT is more efficient, which makes sense due to the overhead of OT extension and the need to compute the base OTs. However, as the number of OTs grows, we see that signed-OT extension is superior across the board.

Computational Cost. We now look at the *computational cost* of the two protocols. Let ξ denote the cost of a public-key operation (we assume exponentiations and signing take the same amount of time), and let ζ denote the cost of a symmetric-key operation (where we let ζ denote the cost of operating over κ bits; e.g., hashing a 2κ-bit value costs 2ζ). We assume all other operations are "free". This is obviously a very coarse analysis; however, it gives a general idea of the performance characteristics of the two approaches.

The cost of executing ℓ OTs on n-bit messages is $(14\ell + 12)\xi$ if $n \leq \tau$ and $(14\ell + 12)\xi + 2\ell\frac{n}{\kappa}\zeta$ if $n > \tau$. Signed-OT requires an additional $2\ell\xi$ operations

(for signing and verifying). We again set $n = 3\kappa$, and thus the cost of executing t signed-OTs is $\mathbf{(16t + 12)\xi}$ if $3\kappa \leq \tau$ and $\mathbf{(16t + 12)\xi + 6t\zeta}$ otherwise.

The cost of our signed-OT extension protocol for t OTs (where we assume $t > \kappa$ and we hash the input prior to signing in Step 4) is $\frac{\ell}{\kappa}t\zeta + (14\ell + 12)\xi + 2\ell\frac{t}{\kappa}\zeta + 6\ell\mu\frac{t}{\kappa}\zeta + 2\log\ell + 2t\frac{\ell+n+\kappa}{\kappa}\zeta + 2\xi$. As above, we set $\ell = 213$ and $\mu = 3$ for short security parameter, $\ell = 318$ and $\mu = 2$ for long security parameter, and $n = 3\kappa$. Thus, the cost of executing t signed-OTs is $\mathbf{(14\ell + 14)\xi + ((5 + 6\mu)\frac{\ell}{\kappa} + 8)t\zeta + 2\log\ell\zeta}$.

	1,000 OTs			10,000 OTs		
Security	sOT	sOT-ext	Improvement	sOT	sOT-ext	Improvement
Short (FFC)	16.0	3.1	5.1×	160.0	3.8	42.4×
Short (ECC)	5.3	1.1	4.9×	53.3	1.7	30.9×
Long (FFC)	144.1	40.2	3.6×	1440.1	40.7	35.4×
Long (ECC)	14.4	4.1	3.5×	144.1	4.5	31.9×

Fig. 8. Computation cost (in millions of "time units") of transferring the input wire labels for P_2 when using signed-OT (sOT) versus signed-OT extension (sOT-ext) for 1,000 and 10,000 OTs. We assume symmetric-key operations take 1 "time unit", FFC (resp., ECC) operations take 1000 (resp., 333) "time units" for the short security parameter, and FFC (resp., ECC) operations take 9000 (resp., 900) "time units" for the long security parameter [1].

Figure 8 presents a comparison of the computational cost of both approaches when executing 1,000 and 10,000 OTs, for various keylength settings and underlying public-key cryptosystems. Here we see that regardless of the number of OTs and public-key cryptosystem used, signed-OT extension is (often much) more efficient, and as the number of OTs increases so does this improvement. For as few as 1,000 OTs we already see a 3.5–5.1× improvement, and for 10,000 OTs we see a 30.9–42.4× improvement.

Comparing Covert, PVC, and Malicious Protocols. We now compare the *computation* cost of our PVC protocol in Fig. 6, using both signed-OT and signed-OT extension, with the covert protocol of Goyal et al. [11] and the malicious protocol of Lindell [17][7].

Figure 9 presents a comparison of the computation cost of our protocol using both signed-OT (Ours^{sOT}) and signed-OT extension ($\text{Ours}^{\text{sOT-ext}}$), as well as comparisons to the Goyal et al. protocol (GMS) and Lindell protocol (Lin). Due to lack of space, the detailed cost formulas appear in the full version. We fix $\kappa = 128$, $\lambda = \nu = 3$ (giving a deterrence factor of $\epsilon = 1/2$), and assume the

[7] Lindell's malicious protocol can also be adapted into a covert protocol; however, we found that the computation cost is much more than that of Goyal et al., at least for deterrence factor 1/2.

f	# inputs	# gates	$\frac{\text{GMS}}{\text{Ours}^{\text{sOT-ext}}}$	$\frac{\text{Ours}^{\text{sOT}}}{\text{Ours}^{\text{sOT-ext}}}$	$\frac{\text{Lin}}{\text{Ours}^{\text{sOT-ext}}}$
16384-bit Comp.	16,384	32,229	0.85–0.73	17.1–86.7	357.0–1887.2
Hamming 16000	16,000	97,175	0.90–0.79	11.0–67.0	224.7–1408.4
16×16 Matrix Mult.	8192	4,186,368	1.00–0.98	1.2–3.1	14.2–54.3
1024-bit Sum	1,024	2,977	0.71–0.61	6.7–10.2	166.6–258.2
1024-bit Mult.	1,024	6,371,746	1.00–0.99	1.0–1.2	10.1–13.9
1024-bit RSA	1,024	15,149,856,895	1.00–1.00	1.0–1.0	9.6–9.6

Fig. 9. Ratio of computation cost of various secure computation protocols with our signed-OT extension construction, using a deterrence factor of 1/2 for the covert and PVC protocols. *GMS* denotes the covert protocol of Goyal et al. [11], *Ours*$^{\text{sOT}}$ denotes the optimized Asharov-Orlandi protocol run using signed-OT, *Ours*$^{\text{sOT-ext}}$ denotes the same protocol using signed-OT extension, and *Lin* denotes Lindell's malicious protocol [17]. We let f denote the function being computed, *# inputs* denote the number of input bits required as input by P_2, and *# gates* denote the number of non-XOR gates in the resulting circuit. All circuit information is taken from the PCF compiler [14, Table5]. We report each ratio as a range; the first number uses $\xi = 125$ as the cost of public-key operations and the second number uses $\xi = 1250$, where we assume a symmetric-key operation costs $\zeta = 1$.

use of elliptic curve cryptography (and thus $\tau = 256$). We expect public-key operations to take between 125–1250× more than symmetric-key operations, depending on implementation details, whether one uses AES-NI, etc. This range is a very conservative estimate using the Crypto++ benchmark [2], experiments using OpenSSL, and estimated ratios of running times between finite field and elliptic curve cryptography [1].

When comparing against GMS, we find that Ours$^{\text{sOT-ext}}$ is slightly more expensive, due almost entirely to the larger number of base OTs in the signed-OT extension. We note that in practice, however, a deterrence factor of 1/2 may not be sufficient for a covert protocol but may be sufficient for a PVC protocol, due to the latter's ability to "name-and-shame" the perpetrator. When increasing the deterrence factor for the covert protocol to $\epsilon \approx .9$, the cost ratios favor Ours$^{\text{sOT-ext}}$. For example, for 16×16 matrix multiplication, the ratio becomes 3.60–3.53×, depending on the cost of public-key operations (versus 1.00–0.98×).

Comparing Ours$^{\text{sOT-ext}}$ with Ours$^{\text{sOT}}$, we find that the former is 1.0–86.7× more efficient, depending largely on the characteristics of the underlying circuit. For circuits with a large number of inputs but a relatively small number of gates (e.g., 16384-bit Comp., Hamming 16000, and 1024-bit Sum) this difference is greatest, which makes sense, as the cost of the OT operations dominates. The circuits for which the ratio is around 1.0 (e.g., 1024-bit RSA) are those that have a huge number of gates compared to the number of inputs, and thus the cost of processing the GC far outweighs the cost of signed-OT/signed-OT extension.

Finally, comparing Ours$^{\text{sOT-ext}}$ with Lin, the former is 9.6–1887.2× more efficient, again depending in a large part on the characteristics of the circuit. We see that for circuits with a large number of inputs this difference is starkest;

e.g., for the Hamming 16000 circuit, we get an improvement of 224.7–1408.4×. The reason we see such large improvements for these circuits is that Lin requires cut-and-choose oblivious transfer, which cannot take advantage of OT extension. Thus, the number of public-key operations is huge compared to the circuit size, and this cost has a large impact on the overall running time. Note, however, that even for circuits where the number of gates dominates, we still see a relatively significant improvement (e.g., 14.2–54.3× for 16×16 Matrix Mult.). These results demonstrate that for settings where public shaming is enough of a deterrent from cheating, Ours$^{\text{sOT-ext}}$ presents a better choice than malicious protocols.

Acknowledgments. The authors thank Michael Zohner for a brief discussion on the relative performance of public- and symmetric-key primitives, and the anonymous reviewers for helpful suggestions.

The authors acknowledge the Office of Naval Research and its support of this work under contract N00014-14-C-0113. Work of Alex J. Malozemoff was also supported by the Department of Defense (DoD) through the National Defense Science & Engineering Graduate (NDSEG) Fellowship.

References

1. The case for elliptic curve cryptography. https://www.nsa.gov/business/programs/elliptic_curve.shtml
2. Crypto++ 5.6.0 benchmarks. http://www.cryptopp.com/benchmarks.html
3. Afshar, A., Mohassel, P., Pinkas, B., Riva, B.: Non-interactive secure computation based on cut-and-choose. In: Nguyen, P.Q., Oswald, E. (eds.) EUROCRYPT 2014. LNCS, vol. 8441, pp. 387–404. Springer, Heidelberg (2014)
4. Andrychowicz, M., Dziembowski, S., Malinowski, D., Mazurek, L.: Secure multiparty computations on bitcoin. In: 2014 IEEE Symposium on Security and Privacy, pp. 443–458. IEEE Computer Society Press, May 2014
5. Asharov, G., Lindell, Y., Schneider, T., Zohner, M.: More efficient oblivious transfer and extensions for faster secure computation. In: ACM CCS 13, pp. 535–548 (2013)
6. Asharov, G., Lindell, Y., Schneider, T., Zohner, M.: More efficient oblivious transfer extensions with security for malicious adversaries. In: Oswald, E., Fischlin, M. (eds.) EUROCRYPT 2015. LNCS, vol. 9056, pp. 673–701. Springer, Heidelberg (2015)
7. Asharov, G., Orlandi, C.: Calling out cheaters: covert security with public verifiability. In: Wang, X., Sako, K. (eds.) ASIACRYPT 2012. LNCS, vol. 7658, pp. 681–698. Springer, Heidelberg (2012)
8. Aumann, Y., Lindell, Y.: Security against covert adversaries: efficient protocols for realistic adversaries. J. Cryptol. **23**(2), 281–343 (2010)
9. Barker, E., Barker, W., Burr, W., Polk, W., Smid, M.: Recommendation for key management – Part 1: General (Revision 3). NIST Special Publication 800–57, July 2012
10. Bentov, I., Kumaresan, R.: How to use bitcoin to design fair protocols. In: Garay, J.A., Gennaro, R. (eds.) CRYPTO 2014, Part II. LNCS, vol. 8617, pp. 421–439. Springer, Heidelberg (2014)
11. Goyal, V., Mohassel, P., Smith, A.: Efficient two party and multi party computation against covert adversaries. In: Smart, N.P. (ed.) EUROCRYPT 2008. LNCS, vol. 4965, pp. 289–306. Springer, Heidelberg (2008)

12. Ishai, Y., Kilian, J., Nissim, K., Petrank, E.: Extending oblivious transfers efficiently. In: Boneh, D. (ed.) CRYPTO 2003. LNCS, vol. 2729, pp. 145–161. Springer, Heidelberg (2003)
13. Kolesnikov, V., Schneider, T.: Improved garbled circuit: free XOR gates and applications. In: Aceto, L., Damgård, I., Goldberg, L.A., Halldórsson, M.M., Ingólfsdóttir, A., Walukiewicz, I. (eds.) ICALP 2008, Part II. LNCS, vol. 5126, pp. 486–498. Springer, Heidelberg (2008)
14. Kreuter, B., Mood, B., Shelat, A., Butler, K.: PCF: a portable circuit format for scalable two-party secure computation. In: 22nd USENIX Security Symposium, August 2013
15. Kreuter, B., Shelat, A., Shen, C.H.: Towards billion-gate secure computation with malicious adversaries. In: 21st USENIX Security Symposium, August 2012
16. Kumaresan, R., Bentov, I.: How to use bitcoin to incentivize correct computations. In: Ahn, G.J., Yung, M., Li, N. (eds.) ACM CCS 14, pp. 30–41, November 2014
17. Lindell, Y.: Fast cut-and-choose based protocols for malicious and covert adversaries. In: Canetti, R., Garay, J.A. (eds.) CRYPTO 2013, Part II. LNCS, vol. 8043, pp. 1–17. Springer, Heidelberg (2013)
18. Peikert, C., Vaikuntanathan, V., Waters, B.: A framework for efficient and composable oblivious transfer. In: Wagner, D. (ed.) CRYPTO 2008. LNCS, vol. 5157, pp. 554–571. Springer, Heidelberg (2008)
19. Pinkas, B., Schneider, T., Smart, N.P., Williams, S.C.: Secure two-party computation is practical. In: Matsui, M. (ed.) ASIACRYPT 2009. LNCS, vol. 5912, pp. 250–267. Springer, Heidelberg (2009)
20. Zahur, S., Rosulek, M., Evans, D.: Two halves make a whole. In: Oswald, E., Fischlin, M. (eds.) EUROCRYPT 2015. LNCS, vol. 9057, pp. 220–250. Springer, Heidelberg (2015)