Covert Security with Public Verifiability: Faster, Leaner, and Simpler

Cheng Hong
Alibaba Group
vince.hc@alibaba-inc.com

Jonathan Katz
University of Maryland
jkatz@cs.umd.edu

Vladimir Kolesnikov
Georgia Tech
kolesnikov@gatech.edu

Wen-jie Lu
University of Tsukuba
riku@mdl.cs.tsukuba.ac.jp

Xiao Wang
MIT and Boston University
wangxiao@northwestern.edu

November 14, 2018

Abstract

The notion of *covert security* for secure two-party computation serves as a compromise between the traditional semi-honest and malicious security definitions. Roughly, covert security ensures that cheating behavior is detected by the honest party with reasonable probability (say, 1/2). It provides more realistic guarantees than semi-honest security with significantly less overhead than is required by malicious security.

The rationale for covert security is that it dissuades cheating by parties that care about their reputation and do not want to risk being caught. Further thought, however, shows that a much stronger disincentive is obtained if the honest party can generate a publicly verifiable *certificate* of misbehavior when cheating is detected. While the corresponding notion of publicly verifiable covert (PVC) security has been explored, existing PVC protocols are complex and less efficient than the best-known covert protocols, and have impractically large certificates.

We propose a novel PVC protocol that significantly improves on prior work. Our protocol uses only "off-the-shelf" primitives (in particular, it avoids signed oblivious transfer) and, for deterrence factor 1/2, has only 20–40% overhead (depending on the circuit size and network bandwidth) compared to state-of-the-art *semi-honest* protocols. Our protocol also has, for the first time, *constant-size* certificates of cheating (e.g., 354 bytes long at the 128-bit security level).

As our protocol offers strong security guarantees with low overhead, we suggest that it is the best choice for many practical applications of secure two-party computation.

1 Introduction

Secure two-party computation allows two mutually distrusting parties $\mathsf{P_A}$ and $\mathsf{P_B}$ to evaluate a function of their inputs without requiring either party to reveal their input to the other. Traditionally, two security notions have been considered [8]. Protocols with *semi-honest security* can be quite efficient, but only protect against passive attackers who do not deviate from the prescribed protocol. *Malicious security*, in contrast, categorically prevents an attacker from gaining any advantage by deviating from the protocol; unfortunately, despite many advances over the past few years, protocols achieving malicious security are still noticeably less efficient than protocols with semi-honest security.

The notion of *covert security* [3] was proposed as a compromise between semi-honest and malicious security. Roughly, covert security ensures that while a cheating attacker may be successful with some small probability, the attempted cheating will fail *and be detected by the other party* with the remaining probability. The rationale for covert security is that it dissuades cheating by parties that care about their reputation and do not want to risk being caught. Covert security thus provides stronger guarantees than semi-honest security. It can also be achieved with better efficiency than malicious security [3, 10, 6, 18].

Nevertheless, the guarantee of covert security is not fully satisfactory. Covert security only ensures that when cheating is unsuccessful, the honest party detects the fact that cheating took place—but it provides no mechanism for the honest party to *prove* this fact to anyone else (e.g., a judge or the public) and, indeed, existing covert protocols do not provide any such mechanism. Thus, a cheating attacker only risks harming its reputation with one other party; even if the honest party publicly announces that it caught the other party cheating, the cheating party can simply counter that it is being falsely accused.

Motivated by this limitation of covert security, Asharov and Orlandi [2] proposed the stronger notion of *publicly verifiable* covert (PVC) security. As in the covert model, any attempted cheating is detected with some probability; now, however, when cheating is detected the honest party can generate a *publicly verifiable certificate* of that fact. This small change would have a significant impact in practice, as a cheating attacker now risks having its reputation publicly and permanently damaged if it is caught. Alternatively (or additionally), the cheating party can be brought to court and fined for its misbehavior; the parties may even sign a contract in advance that describes the penalties to be paid if either party is caught. Going further, the parties could execute a "smart contract" in advance of the protocol execution that would automatically pay out if a valid certificate of cheating is posted on a blockchain. All these consequences are infeasible in the original covert model and, overall, the PVC model seems to come closer to the original goal of covert security.

Asharov and Orlandi [2] mainly focus on feasibility; although their protocol is implementable, it is not competitive with state-of-the-art semi-honest protocols since, in particular, it requires a stronger variant of oblivious transfer (OT) called *signed OT* and thus is not directly compatible with OT extension. Subsequent work by Kolesnikov and Malozemoff [14] shows various efficiency improvements to the Asharov-Orlandi protocol, with the primary gain resulting from a new, dedicated protocol for signed-OT extension. (Importantly, signed-OT extension does not follow generically from standard OT extension, and so cannot take advantage of the most-efficient recent constructions of the latter.)

Unfortunately, existing PVC protocols [2, 14] seem not to have attracted much attention; for example, to the best of our knowledge, they have never been implemented. We suggest this is due to a number of considerations:

- **High overhead.** State-of-the-art PVC protocols still incur a significant overhead compared to known semi-honest protocols, and even existing covert protocols. (See Section 6.)

- **Large certificates.** Existing PVC protocols have certificates of size at least $\kappa \cdot |\mathcal{C}|$ bits, where κ is the (computational) security parameter and $|\mathcal{C}|$ is the circuit size.[1] Certificates this large are prohibitively expensive to propagate and are incompatible with some of the applications mentioned above (e.g., posting a certificate on a blockchain).

[1]We observe that the certificate size in [14] can be improved to $O(\kappa \cdot n)$ bits (where n is the parties' input lengths) by carefully applying ideas from the literature. In many cases, this is still unacceptably large.

- **Complexity.** Existing PVC protocols rely on signed OT, a non-standard primitive that is less efficient than standard OT, is not available in existing secure-computation libraries, and is somewhat complicated to realize (especially for signed-OT extension).

1.1 Our Contributions

In this work we put forward a new PVC protocol that addresses the issues mentioned above. Specifically:

- **Low overhead.** We improve on the efficiency of prior work by roughly a factor of $3\times$ for deterrence factor $1/2$, and even more for larger deterrence. (An exact comparison depends on a number of factors; we refer to Section 6 for a detailed discussion.) Strikingly, our PVC protocol (with deterrence factor $1/2$) incurs *only 20–40% overhead compared to state-of-the-art* **semi-honest** *protocols based on garbled circuits.*

- **Small certificates.** We achieve, for the first time, *constant-size* certificates (i.e., independent of the circuit size or the lengths of the parties' inputs). Concretely, our certificates are small: at the 128-bit security level, they are only 354 bytes long.

- **Simplicity.** Our protocol avoids entirely the need for signed OT, and relies only on standard building blocks such as (standard) OT and circuit garbling. We also dispense with the XOR-tree technique for preventing selective-failure attacks; this allows us to avoid increasing the number of effective OT inputs. This reduction in complexity allowed us to produce a simple and efficient (and, to our knowledge, the first) implementation of a PVC protocol.

Overview of the paper. In Section 2 we provide an overview of prior PVC protocols and explain the intuition behind the construction of our protocol. After some background in Section 3, we present the description of our protocol in Section 4 and prove security in Section 5. Section 6 gives an experimental evaluation of our protocol and a comparison to prior work.

2 Technical Overview

We begin by providing an overview of the approach taken in prior work designing PVC protocols. Then we discuss the intuition behind our improved protocol.

2.1 Overview of Prior Work

At a high level, both previous works constructing PVC protocols [2, 14] rely on the standard cut-and-choose paradigm [19] using a small number of garbled circuits, with some additional complications to achieve public verifiability. Both works rely crucially on a primitive called *signed OT*; this is a functionality similar to OT but where the receiver additionally learns the sender's signatures on all the values it obtains. Roughly, prior protocols proceed as follows:

1. Let λ be a parameter that determines the deterrence factor (i.e., the probability of detecting misbehavior). P_A picks random seeds $\{\mathsf{seed}_j\}_{j=1}^{\lambda}$ and P_B chooses a random index $\hat{\jmath} \in \{1, \ldots, \lambda\}$ that will serve as the "evaluation index" while the $j \neq \hat{\jmath}$ will be "check indices." The parties run signed OT using these inputs, which allows P_B to learn $\{\mathsf{seed}_j\}_{j\neq\hat{\jmath}}$ along with signatures of P_A on all those values.

2. $\mathsf{P_A}$ generates λ garbled circuits, and then sends signed commitments to those garbled circuits (along with the input-wire labels corresponding to $\mathsf{P_A}$'s input wires). Importantly, seed_j is used to derive the (pseudo)randomness for the jth garbling as well as the jth commitment.

   The parties also use signed OT so that $\mathsf{P_B}$ can obtain the input-wire labels for its inputs across all the circuits.

3. For all $j \neq \hat{\jmath}$, party $\mathsf{P_B}$ checks that the commitment to the jth garbled circuit is computed correctly based on seed_j and that the input-wire labels it received are correct; if this is not the case, then $\mathsf{P_B}$ can generate a certificate of cheating that consists of the inconsistent values plus their signatures.

4. Assuming no cheating was detected, $\mathsf{P_B}$ reveals $\hat{\jmath}$ to $\mathsf{P_A}$, who then sends the $\hat{\jmath}$th garbled circuit and the input-wire labels corresponding to its own inputs for that circuit. $\mathsf{P_B}$ can then evaluate the garbled circuit as usual.

Informally, we refer to the jth garbled circuit and commitment as the *jth instance of the protocol.* If $\mathsf{P_A}$ cheats in the jth instance of the protocol, then it is caught with probability at least $1 - \frac{1}{\lambda}$ (i.e., if j is a check index). Moreover, if $\mathsf{P_A}$ is caught, then $\mathsf{P_B}$ has a signed seed (which defines what $\mathsf{P_A}$ was supposed to do in the jth instance) and also a signed commitment to an incorrect garbled circuit or incorrect input-wire labels. These values allow $\mathsf{P_B}$ to generate a publicly verifiable certificate that $\mathsf{P_A}$ cheated.

As described, the protocol still allows $\mathsf{P_A}$ to carry out a selective-failure attack when transferring garbled labels for $\mathsf{P_B}$'s input wires. Specifically, it may happen that a malicious $\mathsf{P_A}$ corrupts a single input-wire label (used as input to the OT protocol) for the $\hat{\jmath}$th garbled circuit—say, the label corresponding to a '1' input on some wire. If $\mathsf{P_B}$ aborts, then $\mathsf{P_A}$ learns that $\mathsf{P_B}$'s input on that wire was equal to 1. Such selective-failure attacks can be prevented using the XOR-tree approach [19].[2] This approach introduces significant overhead because it increases the number of effective inputs, which in turn requires additional signed OTs. The analysis in prior work [3, 2, 14] shows that to achieve deterrence factor (i.e., probability of being caught cheating) 1/2, a replication factor of $\lambda = 3$ is needed. More generally, the deterrence factor as a function of λ and the XOR-tree expansion factor ν is $(1 - \frac{1}{\lambda}) \cdot (1 - 2^{-\nu+1})$.

Practical performance. Several aspects of the above protocol are relatively inefficient. First, the dependence of the deterrence factor on the replication factor λ is not optimal due to the XOR tree, e.g., to achieve deterrence factor 1/2 at least $\lambda = 3$ garbled circuits are needed (unless ν is impractically large); the issue becomes even more significant when a larger deterrence factor is desired. In addition, the XOR-tree approach used in prior work increases the effective input length by at least a factor of 3, which necessitates $3\times$ more signed OTs; recall these are relatively expensive since signed-OT extension is. Finally, prior protocols have large certificates. This seems inherent in the more efficient protocol of [14] due to the way they do signed-OT extension. (Avoiding signed-OT extension would result in a much less efficient protocol overall.)

2.2 Our Solution

The reliance of prior protocols on signed OT and their approach to preventing selective-failure attacks affect both their efficiency as well as the size of their certificates. We address both these

[2]For reasonable values of the parameters, the XOR-tree approach will be more efficient than a coding-theoretic approach [19].

issues in the protocol we design.

As in prior work, we use the cut-and-choose approach and have $\mathsf{P_B}$ evaluate one garbled circuit while checking the rest, and we realize this by having $\mathsf{P_A}$ choose seeds for each of λ executions and then allowing $\mathsf{P_B}$ to obliviously learn all-but-one of those seeds. One key difference in our protocol is that we utilize the seeds chosen by $\mathsf{P_A}$ not only to "derandomize" the garbled-circuit generation and commitments, but *also to derandomize the entire remainder of* $\mathsf{P_A}$*'s execution*, and in particular its execution of the OT protocol used to transfer $\mathsf{P_B}$'s input-wire labels to $\mathsf{P_B}$. This means that after $\mathsf{P_B}$ obliviously learns all-but-one of the seeds of $\mathsf{P_A}$, the rest of $\mathsf{P_A}$'s execution is entirely deterministic; thus, $\mathsf{P_B}$ can verify correct execution of $\mathsf{P_A}$ during the entire rest of the protocol for all-but-one of the seeds. Not only does this eliminate the need for signed OT for the input-wire labels, but it also defends against the selective-failure attack described earlier *without* the need to increase the effective input length at all.

As described, the above allows $\mathsf{P_B}$ to *detect* cheating by $\mathsf{P_A}$ but does not yet achieve public verifiability. For this, we additionally require $\mathsf{P_A}$ to sign its protocol messages; if $\mathsf{P_A}$ cheats, $\mathsf{P_B}$ can generate a certificate of cheating from the seed and the corresponding signed inconsistent transcript.

Thus far we have focused on the case where $\mathsf{P_A}$ is malicious. We must also consider the case of a malicious $\mathsf{P_B}$ attempting to frame an honest $\mathsf{P_A}$. We address this by also having $\mathsf{P_B}$ commit in advance to *its* randomness[3] for each of the λ protocol instances. The resulting commitments will be included in $\mathsf{P_A}$'s signature, and will ensure that a certificate will be rejected if it corresponds to an instance in which $\mathsf{P_B}$ deviated from the protocol.

Having $\mathsf{P_B}$ commit to its randomness also allows us to avoid the need for signed OT in the first step, when $\mathsf{P_B}$ learns all-but-one of $\mathsf{P_A}$'s seeds. This is because those seeds can be reconstructed from $\mathsf{P_B}$'s view of the protocol, i.e., from the transcript of the (standard) OT protocol used to transfer those seeds plus $\mathsf{P_B}$'s randomness. Having $\mathsf{P_A}$ sign the transcripts of those OT executions serves as publicly verifiable evidence of the seeds used by $\mathsf{P_A}$.

We refer to Section 4 for further intuition behind our protocol, as well as its formal specification.

3 Covert Security with Public Verifiability

Before defining the notion of PVC security, we review the (plain) covert model [3] it extends. We focus on the strongest formulation of covert security, namely the *strong explicit cheat* formulation. This notion is formalized via an ideal functionality that explicitly allows an adversary to specify an attempt at cheating; in that case, the ideal functionality allows the attacker to successfully cheat with probability $1 - \epsilon$, but the attacker is caught by the other party with probability ϵ; see Figure 1. (As in [2], we also allow an attacker to "blatantly cheat," which guarantees that it will be caught.) For simplicity, we adapt the functionality such that only a malicious $\mathsf{P_A}$ can possibly cheat, as this is what is achieved by our protocol. For conciseness, we refer to a protocol realizing this functionality (against malicious adversaries) as having *covert security with deterrence* ϵ.

The PVC model extends the above to consider a setting wherein, before execution of the protocol, $\mathsf{P_A}$ has generated keys (pk, sk) for a digital-signature scheme, with the public key pk known to $\mathsf{P_B}$. We *do not* require that (pk, sk) is honestly generated, or that $\mathsf{P_A}$ gives any proof of knowledge of the secret key sk corresponding to the public key pk. In addition, the protocol is augmented with two additional algorithms, Blame and Judge. The Blame algorithm is run by $\mathsf{P_B}$ when it outputs

[3]As an optimization, we have $\mathsf{P_B}$ commit to seeds, just like $\mathsf{P_A}$, and then use those seeds to generate the (pseudo)randomness to use in each instance. (This optimization is critical for realizing constant-size certificates.)

Functionality $\mathcal{F}$

$\mathsf{P_A}$ sends $x \in \{0,1\}^{n_1} \cup \{\perp, \mathsf{blatantCheat}, \mathsf{cheat}\}$ and $\mathsf{P_B}$ sends $y \in \{0,1\}^{n_2}$.

1. If $x \in \{0,1\}^{n_1}$ then compute $f(x,y)$ and send it to $\mathsf{P_B}$.
2. If $x = \perp$ then send $\perp$ to both parties.
3. If $x = \mathsf{blatantCheat}$, then send $\mathsf{corrupted}$ to both parties.
4. If $x = \mathsf{cheat}$ then:
   - With probability ϵ, send $\mathsf{corrupted}$ to both parties.
   - With probability $1-\epsilon$, send $(\mathsf{undetected}, y)$ to $\mathsf{P_A}$. Then wait to receive $z \in \{0,1\}^{n_3}$ from $\mathsf{P_A}$, and send z to $\mathsf{P_B}$.

Figure 1: Functionality $\mathcal{F}$ for covert security with deterrence ϵ for two-party computation of a function f.

$\mathsf{corrupted}$. This algorithm takes as input $\mathsf{P_B}$'s view of the protocol execution thus far, and outputs a certificate cert which is then sent to $\mathsf{P_A}$. The Judge algorithm takes as input $\mathsf{P_A}$'s public key pk, (a description of) the circuit $\mathcal{C}$ being evaluated, and a certificate cert, and outputs 0 or 1.

A protocol Π along with algorithms $\mathsf{Blame}, \mathsf{Judge}$ is said to be *publicly verifiable covert with deterrence* ϵ for computing a circuit $\mathcal{C}$ if the following hold:

Covert security: The protocol Π has covert security with deterrence ϵ. (Since the protocol includes the step of possibly sending cert to $\mathsf{P_A}$ if $\mathsf{P_B}$ outputs $\mathsf{corrupted}$, this ensures that cert itself does not violate privacy of $\mathsf{P_B}$.)

Public verifiability: If the honest $\mathsf{P_B}$ outputs cert in an execution of the protocol, then we know $\mathsf{Judge}(pk, \mathcal{C}, \mathsf{cert}) = 1$, except with negligible probability.

Defamation freeness: If $\mathsf{P_A}$ is honest, then the probability that a malicious $\mathsf{P_B}$ generates a certificate cert for which $\mathsf{Judge}(pk, \mathcal{C}, \mathsf{cert}) = 1$ is negligible.[4]

Remark: As in prior work on the PVC model, we assume the Judge algorithm learns the circuit $\mathcal{C}$ through some "out-of-band" mechanism; in particular, we do not include $\mathcal{C}$ as part of the certificate.

In some applications (such as the smart-contract example), it may indeed be the case that the party running the Judge algorithm is aware of the circuit being computed in advance. When this is not the case, a description of $\mathcal{C}$ must be included as part of the certificate. However, we stress that the description of a circuit may be much shorter than the full circuit; for example, specifying a circuit for computing the Hamming distance between two 10^6-bit vectors requires only a few lines of high-level code in modern secure-computation platforms even though the circuit itself may have millions of gates. Alternately, there may be a small set of commonly used "reference circuits" that can be identified by ID number rather than by their complete wiring diagram.

[4]Note that defamation freeness implies that the protocol is also non-halting detection accurate [3].

4 Our PVC Protocol

4.1 Preliminaries

We let $[n] = \{1, \ldots, n\}$. We use κ for the (computational) security parameter, but for compactness in the protocol description we let κ be an implicit input to our algorithms. For a boolean string y, we let $y[i]$ denote the ith bit of y.

We let Com denote a commitment scheme. We assume for simplicity that it is non-interactive, but this restriction can easily be removed. The decommitment decom is simply the random coins used during commitment. H is a hash function with 2κ-bit output length.

We say a party "uses randomness derived from seed" to mean that the party uses a pseudorandom function (with seed as the key) in CTR mode to obtain sufficiently many pseudorandom values that it then uses as its random coins. If $m_1, m_2, \ldots$ is a transcript of an execution of a two-party protocol (where the parties alternate sending the messages), the *transcript hash* of the execution is defined to be $\mathcal{H} = (H(m_1), H(m_2), \ldots)$.

We let Π_{OT} be an OT protocol realizing a parallel version of the OT functionality, as in Figure 2.

Functionality $\mathcal{F}_{\mathsf{OT}}$

Private inputs: $\mathsf{P_A}$ has input $\{(B_{i,0}, B_{i,1})\}_{i=1}^{n_2}$ and $\mathsf{P_B}$ has input $y \in \{0,1\}^{n_2}$.

1. Upon receiving $\{(B_{i,0}, B_{i,1})\}_{i=1}^{n_2}$ from $\mathsf{P_A}$ and y from $\mathsf{P_B}$, send $\{B_{i,y[i]}\}_{i=1}^{n_2}$ to $\mathsf{P_B}$.

Figure 2: Functionality $\mathcal{F}_{\mathsf{OT}}$ for parallel oblivious transfer.

Garbling. Our protocol relies on a (circuit) *garbling scheme*. For our purposes, a garbling scheme is defined by algorithms $(\mathsf{Gb}, \mathsf{Eval})$ having the following syntax:

- Gb takes as input the security parameter 1^κ and a circuit $\mathcal{C}$ with $n = n_1 + n_2$ input wires and n_3 output wires. It outputs *input-wire labels* $\{X_{i,0}, X_{i,1}\}_{i=1}^{n}$, a *garbled circuit* GC, and *output-wire labels* $\{Z_{i,0}, Z_{i,1}\}_{i=1}^{n_3}$.
- Eval is a deterministic algorithm that takes as input a set of input-wire labels $\{X_i\}_{i=1}^{n}$ and a garbled circuit GC. It outputs a set of output-wire labels $\{Z_i\}_{i=1}^{n_3}$.

Correctness is defined as follows: For any circuit $\mathcal{C}$ as above and any input $w \in \{0,1\}^n$, consider the experiment in which we first run $(\{X_{i,0}, X_{i,1}\}_{i=1}^{n}, \mathsf{GC}, \{Z_{i,0}, Z_{i,1}\}_{i=1}^{n_3}) \leftarrow \mathsf{Gb}(1^\kappa, \mathcal{C})$ followed by $\{Z_i\} := \mathsf{Eval}(\{X_{i,w[i]}\}, \mathsf{GC})$. Then, except with negligible probability, it holds that $Z_i = Z_{i,y[i]}$ and $Z_i \neq Z_{i,1-y[i]}$ for all i, where $y = \mathcal{C}(w)$.

A garbling scheme can be used by (honest) parties $\mathsf{P_A}$ and $\mathsf{P_B}$ to compute $\mathcal{C}$ in the following way: first, $\mathsf{P_A}$ computes $(\{X_{i,0}, X_{i,1}\}_{i=1}^{n}, \mathsf{GC}, \{Z_{i,0}, Z_{i,1}\}_{i=1}^{n_3}) \leftarrow \mathsf{Gb}(1^\kappa, \mathcal{C})$ and sends $\mathsf{GC}, \{Z_{i,0}, Z_{i,1}\}_{i=1}^{n_3}$ to $\mathsf{P_B}$. Next, $\mathsf{P_B}$ learns the input-wire labels $\{X_{i,w[i]}\}$ corresponding to some input w. (In a secure-computation protocol, $\mathsf{P_A}$ would send $\mathsf{P_B}$ the input-wire labels corresponding to its own portion of the input, while the parties would use OT to enable $\mathsf{P_B}$ to learn the input-wire labels corresponding to $\mathsf{P_B}$'s portion of the input.) Then $\mathsf{P_B}$ computes $\{Z_i\} := \mathsf{Eval}(\{X_{i,w[i]}\}, \mathsf{GC})$. Finally, $\mathsf{P_B}$ sets $y[i]$, for all i, to be the (unique) bit for which $Z_i = Z_{i,y[i]}$; the output is y.

We assume the garbling scheme satisfies the standard security definition [11, 16]. That is, we assume there is a simulator $\mathcal{S}_{\mathsf{Gb}}$ such that for all $\mathcal{C}, w$, the distribution $\left\{\mathcal{S}_{\mathsf{Gb}}(1^\kappa, \mathcal{C}, \mathcal{C}(w))\right\}$ is computationally indistinguishable from

$$\left\{(\{X_{i,0}, X_{i,1}\}_{i=1}^{n}, \mathsf{GC}, \{Z_{i,0}, Z_{i,1}\}_{i=1}^{n_3}) \leftarrow \mathsf{Gb}(1^\kappa, \mathcal{C}) \;:\; (\{X_{i,w[i]}\}, \mathsf{GC}, \{Z_{i,0}, Z_{i,1}\}_{i=1}^{n_3})\right\}.$$

As this is the "minimal" security notion for garbling, it is satisfied by garbling schemes including all state-of-the-art optimizations [15, 4, 21].

4.2 Our Scheme

We give a high-level description of our protocol below; a formal definition of the protocol is provided in Figure 3. The Blame algorithm is included as part of the protocol description (cf. Step 6) for simplicity. The Judge algorithm is specified in Figure 4.

We use a signature scheme $(\mathsf{Gen}, \mathsf{Sign}, \mathsf{Vrfy})$. Before executing the protocol, $\mathsf{P_A}$ runs Gen to obtain public key pk and private key sk; we assume that $\mathsf{P_B}$ knows pk before running the protocol. As noted earlier, if $\mathsf{P_A}$ is malicious then it may choose pk arbitrarily.

The main idea of the protocol is to run λ parallel instances of a "basic" garbled-circuit protocol that is secure against a semi-honest $\mathsf{P_A}$ and a malicious $\mathsf{P_B}$. Of these instances, $\lambda - 1$ will be checked by $\mathsf{P_B}$, while a random one (the $\hat{\jmath}$th) will be evaluated by $\mathsf{P_B}$ to learn its output. To give $\mathsf{P_B}$ the ability to verify honest behavior in the check instances, we make all the executions deterministic by having $\mathsf{P_A}$ use (pseudo)randomness derived from corresponding seeds $\{\mathsf{seed}_j^A\}_{j\in[\lambda]}$. That is, $\mathsf{P_A}$ will uniformly sample each seed seed_j^A and use it to generate (pseudo)randomness for its jth instance. Then $\mathsf{P_A}$ and $\mathsf{P_B}$ run an OT protocol Π_{OT} (with malicious security) that allows $\mathsf{P_B}$ to learn $\lambda - 1$ of those seeds. Since $\mathsf{P_A}$'s behavior in those $\lambda - 1$ instances is completely determined by $\mathsf{P_B}$'s messages and those seeds, it is possible for $\mathsf{P_B}$ to check $\mathsf{P_A}$'s behavior in those instances.

The above idea allows $\mathsf{P_B}$ to catch a cheating $\mathsf{P_A}$, but not to generate a publicly verifiable certificate that $\mathsf{P_A}$ has cheated. To add this feature, we have $\mathsf{P_A}$ sign the transcripts of each instance, including the transcript of the execution of the OT protocol by which $\mathsf{P_B}$ learned the corresponding seed. If $\mathsf{P_A}$ cheats in, say, the jth instance ($j \neq \hat{\jmath}$) and is caught, then $\mathsf{P_B}$ can output a certificate that includes $\mathsf{P_B}$'s view (including its randomness) in the execution of the jth OT protocol (from which seed_j^A can be recomputed) and the transcript of the jth instance, along with $\mathsf{P_A}$'s signature on the transcripts. Note that, given the randomness of both $\mathsf{P_A}$ and $\mathsf{P_B}$, the entire transcript of the instance can be recomputed and anyone can then check whether it is consistent with seed_j^A. We remark that nothing about $\mathsf{P_B}$'s inputs is revealed by a certificate since $\mathsf{P_B}$ uses a dummy input in all the check instances.

There still remains the potential issue of *defamation.* Indeed, an honest $\mathsf{P_A}$'s messages might be deemed inconsistent if $\mathsf{P_B}$ includes in the certificate fake messages different from those sent by $\mathsf{P_B}$ in the real execution. We prevent this by having $\mathsf{P_B}$ commit to its randomness for each instance at the beginning of the protocol, and having $\mathsf{P_A}$ sign those commitments. Consistency of $\mathsf{P_B}$'s randomness and the given transcript can then be checked as part of verification of the certificate.

As described, the above would result in a certificate that is linear in the length of $\mathsf{P_B}$'s inputs, since there are that many OT executions (in each instance) for which $\mathsf{P_B}$ must generate randomness. We compress this to a constant-size certificate by having $\mathsf{P_B}$ also generate its (pseudo)randomness from a short seed.

The above description conveys the main ideas of the protocol, though various other modifications are needed for the proof of security. We refer the reader to Figures 3 and 4 for the details.

Protocol Π_{pvc}

Private inputs: $\mathsf{P_A}$ has input $x \in \{0,1\}^{n_1}$ and keys (pk, sk) for a signature scheme. $\mathsf{P_B}$ has input $y \in \{0,1\}^{n_2}$ and knows pk.
Public inputs: Both parties also agree on a circuit $\mathcal{C}$ and parameters κ, λ.
Protocol:

1. $\mathsf{P_B}$ chooses uniform κ-bit strings $\{\mathsf{seed}^B_j\}_{j\in[\lambda]}$, sets $h_j \leftarrow \mathsf{Com}(\mathsf{seed}^B_j)$ for all j, and sends $\{h_j\}_{j\in[\lambda]}$ to $\mathsf{P_A}$.

2. $\mathsf{P_A}$ chooses uniform κ-bit strings $\{\mathsf{seed}^A_j, \mathsf{witness}_j\}_{j\in[\lambda]}$, while $\mathsf{P_B}$ chooses uniform $\hat{\jmath} \in [\lambda]$ and sets $b_{\hat{\jmath}} := 1$ and $b_j := 0$ for $j \neq \hat{\jmath}$.

   $\mathsf{P_A}$ and $\mathsf{P_B}$ run λ executions of Π_{OT}, where in the jth execution $\mathsf{P_A}$ uses $(\mathsf{seed}^A_j, \mathsf{witness}_j)$ as input, and $\mathsf{P_B}$ uses b_j as input and randomness derived from seed^B_j. Upon completion, $\mathsf{P_B}$ obtains $\{\mathsf{seed}^A_j\}_{j\neq\hat{\jmath}}$ and $\mathsf{witness}_{\hat{\jmath}}$. Let trans_j be the transcript of the jth execution of Π_{OT}.

3. For each $j \in [\lambda]$, $\mathsf{P_A}$ garbles $\mathcal{C}$ using randomness derived from seed^A_j. Denote the jth garbled circuit by GC_j, the input-wire labels of $\mathsf{P_A}$ by $\{A_{j,i,b}\}_{i\in[n_1],b\in\{0,1\}}$, the input-wire labels of $\mathsf{P_B}$ by $\{B_{j,i,b}\}_{i\in[n_2],b\in\{0,1\}}$, and the output-wire labels by $\{Z_{j,i,b}\}_{i\in[n_3],b\in\{0,1\}}$.

   $\mathsf{P_A}$ and $\mathsf{P_B}$ then run λ executions of Π_{OT}, where in the jth execution $\mathsf{P_A}$ uses $\{(B_{j,i,0}, B_{j,i,1})\}_{i=1}^{n_2}$ as input, and $\mathsf{P_B}$ uses y as input if $j = \hat{\jmath}$ and 0^{n_2} otherwise. The parties use seed^A_j and seed^B_j, respectively, to derive all their randomness in the jth execution. In this way, $\mathsf{P_B}$ obtains $\{B_{\hat{\jmath},i,y[i]}\}_{i\in[n_2]}$. We let $\mathcal{H}_j$ denote the transcript hash for the jth execution of Π_{OT}.

4. $\mathsf{P_A}$ computes commitments $h^A_{j,i,b} \leftarrow \mathsf{Com}(A_{j,i,b})$ for all j, i, b, and then computes the commitments $\mathsf{c}_j \leftarrow \mathsf{Com}\left(\mathsf{GC}_j, \{h^A_{j,i,b}\}_{i\in[n_1],b\in\{0,1\}}, \{Z_{j,i,b}\}_{i\in[n_3],b\in\{0,1\}}\right)$ for all j, where each pair $(h^A_{j,i,0}, h^A_{j,i,1})$ is randomly permuted. All randomness in the jth instance is derived from seed^A_j. Finally, $\mathsf{P_A}$ sends $\{\mathsf{c}_j\}_{j\in[\lambda]}$ to $\mathsf{P_B}$.

5. For each $j \in [\lambda]$, $\mathsf{P_A}$ computes $\sigma_j \leftarrow \mathsf{Sign}_{\mathsf{sk}}(\mathcal{C}, j, h_j, \mathsf{trans}_j, \mathcal{H}_j, \mathsf{c}_j)$ and sends σ_j to $\mathsf{P_B}$. Then $\mathsf{P_B}$ checks that σ_j is a valid signature for all j, and aborts with output $\perp$ if not.

6. For each $j \neq \hat{\jmath}$, $\mathsf{P_B}$ uses seed^A_j and the messages it sent to simulate $\mathsf{P_A}$'s computation in steps 3 and 4, and in particular computes $\hat{\mathcal{H}}_j, \hat{\mathsf{c}}_j$. It then checks that $(\hat{\mathcal{H}}_j, \hat{\mathsf{c}}_j) = (\mathcal{H}_j, \mathsf{c}_j)$. If the check fails for some $j \neq \hat{\jmath}$, then $\mathsf{P_B}$ chooses a uniform such j, outputs $\mathsf{corrupted}$, sends $\mathsf{cert} := (j, \mathsf{trans}_j, \mathcal{H}_j, \mathsf{c}_j, \sigma_j, \mathsf{seed}^B_j, \mathsf{decom}^B_j)$ to $\mathsf{P_A}$, and halts.

7. $\mathsf{P_B}$ sends $(\hat{\jmath}, \{\mathsf{seed}^A_j\}_{j\neq\hat{\jmath}}, \mathsf{witness}_{\hat{\jmath}})$ to $\mathsf{P_A}$, who checks that $\{\mathsf{seed}^A_j\}_{j\neq\hat{\jmath}}, \mathsf{witness}_{\hat{\jmath}}$ are all correct and aborts if not.

8. $\mathsf{P_A}$ sends $\mathsf{GC}_{\hat{\jmath}}$, $\{A_{\hat{\jmath},i,x[i]}\}_{i\in[n_1]}$, $\{h^A_{\hat{\jmath},i,b}\}_{i\in[n_1],b\in\{0,1\}}$ (in the same permuted order as before), and $\{Z_{\hat{\jmath},i,b}\}_{i\in[n_3],b\in\{0,1\}}$ to $\mathsf{P_B}$, along with decommitments $\mathsf{decom}_{\hat{\jmath}}$ and $\{\mathsf{decom}^A_{\hat{\jmath},i,x[i]}\}$. If $\mathsf{Com}(\mathsf{GC}_{\hat{\jmath}}, \{h^A_{\hat{\jmath},i,b}\}, \{Z_{\hat{\jmath},i,b}\}; \mathsf{decom}_{\hat{\jmath}}) \neq \mathsf{c}_{\hat{\jmath}}$ or $\mathsf{Com}(A_{\hat{\jmath},i,x[i]}; \mathsf{decom}^A_{\hat{\jmath},i,x[i]}) \notin \{h^A_{\hat{\jmath},i,b}\}_{b\in\{0,1\}}$ for some i, then $\mathsf{P_B}$ aborts with output $\perp$.

   Otherwise, $\mathsf{P_B}$ evaluates $\mathsf{GC}_{\hat{\jmath}}$ using $\{A_{\hat{\jmath},i,x[i]}\}_{i\in[n_1]}$ and $\{B_{\hat{\jmath},i,y[i]}\}_{i\in[n_2]}$ to obtain output-wire labels $\{Z_i\}_{i\in[n_3]}$. For each $i \in [n_3]$, if $Z_i = Z_{\hat{\jmath},i,0}$, set $z[i] := 0$; if $Z_i = Z_{\hat{\jmath},i,1}$, set $z[i] := 1$. (If $Z_i \notin \{Z_{\hat{\jmath},i,0}, Z_{\hat{\jmath},i,1}\}$ for some i, then abort with output $\perp$.) Output z.

Figure 3: Full description of our PVC protocol.

```
Algorithm Judge

Inputs: A public key $pk$, a circuit $\mathcal{C}$, and a certificate $\mathsf{cert}$.

  1. Parse $\mathsf{cert}$ as $(j, \mathsf{trans}_j, \mathcal{H}_j, \mathsf{c}_j, \sigma_j, \mathsf{seed}_j^B, \mathsf{decom}_j^B)$. Compute $h_j := \mathsf{Com}(\mathsf{seed}_j^B; \mathsf{decom}_j^B)$.
  2. If $\mathsf{Vrfy}_{pk}((\mathcal{C}, j, h_j, \mathsf{trans}_j, \mathcal{H}_j, \mathsf{c}_j), \sigma_j) = 0$, output 0.
  3. Simulate an execution of $\Pi_{\mathsf{OT}}$ by $\mathsf{P_B}$, where $\mathsf{P_B}$'s input is 0, its randomness is derived from $\mathsf{seed}_j^B$, and $\mathsf{P_A}$'s messages are those included in $\mathsf{trans}_j$. Check that all of $\mathsf{P_B}$'s messages generated in this simulation are consistent with $\mathsf{trans}_j$; terminate with output 0 if not. Otherwise, let $\mathsf{seed}_j^A$ denote the output of $\mathsf{P_B}$ from the simulated execution of $\Pi_{\mathsf{OT}}$.
  4. Use $\mathsf{seed}_j^A$ and $\mathsf{seed}_j^B$ to simulate an honest execution of steps 3 and 4 of the protocol, and in particular compute $\hat{\mathcal{H}}_j, \hat{\mathsf{c}}_j$.
  5. Do:
      (a) If $(\hat{\mathcal{H}}_j, \hat{\mathsf{c}}_j) = (\mathcal{H}_j, \mathsf{c}_j)$ then output 0.
      (b) If $\hat{\mathsf{c}}_j \neq \mathsf{c}_j$ then output 1.
      (c) Find the first message for which $\hat{\mathcal{H}}_j \neq \mathcal{H}_j$. If this corresponds to a message sent by $\mathsf{P_A}$, output 1; otherwise, output 0.
```

Figure 4: The Judge algorithm.

4.3 Optimizations

Our main protocol is already quite efficient, but we briefly discuss some additional optimizations that can be applied.

Commitments in the random-oracle model. When standard garbling schemes are used, all the values committed during the course of the protocol have high entropy; thus, commitment to a string r can be done by simply computing $H(r)$ (if H is modeled as a random oracle) and decommitment requires only sending r.

Free Hash. Fan et al. [7] introduced a garbled-circuit optimization called *Free Hash* that provides a way to generate a hashed garbled circuit at lower cost than garbling followed by hashing. We can use this as part of generating $\mathsf{P_A}$'s commitment to a garbled circuit.

One technical note is that Free Hash by itself does not provide a way to *equivocate* the hash value, which is needed for a simulation-based proof of security against a malicious $\mathsf{P_B}$. However, we observe that in the random-oracle model such equivocation is easy to achieve by applying the random oracle H to the Free-Hash output.

Using correlated oblivious transfer. One optimization introduced by Asharov et al. [1] is using correlated OT for transferring $\mathsf{P_B}$'s input-wire labels when garbling is done using the free-XOR approach [15]. This optimization is compatible with our protocol in a straightforward manner.

Avoiding committing to the input-wire labels. In our protocol, we have $\mathsf{P_A}$ commit to its input-wire labels (along with the rest of the garbled circuit). This is done to prevent $\mathsf{P_A}$ from sending incorrect input-wire labels in the final step. We observe that this is unnecessary if the garbling scheme has the additional property that it is infeasible to generate a garbled circuit along with incorrect input-wire labels that result in a valid output when evaluated. (We omit a formal definition.) Many standard garbling schemes have this property.

5 Proof of Security

The remainder of this section is devoted to a proof of the following result:

Theorem 1. *Assume* Com *is computationally hiding/binding,* H *is collision-resistant, the garbling scheme is secure,* Π_{OT} *UC-realizes* $\mathcal{F}_{\mathsf{OT}}$*, and the signature scheme is existentially unforgeable under a chosen-message attack. Then protocol* Π_{pvc} *along with* Blame *as in step 6 and* Judge *as in Figure 4 is publicly verifiable covert with deterrence* $\epsilon = 1 - \frac{1}{\lambda}$*.*

Since our most efficient implementation relies on the random-oracle model anyway, we can use a universally composable OT protocol designed in the random-oracle model such as the Chou-Orlandi protocol [5]. Alternately, it suffices for the OT protocol to be secure under bounded parallel self composition.

Proof. We separately prove covert security with ϵ-deterrence (handling the cases where either $\mathsf{P_A}$ or $\mathsf{P_B}$ is corrupted), public verifiability, and defamation freeness.

Covert Security—Malicious $\mathsf{P_A}$

Let $\mathcal{A}$ be an adversary corrupting $\mathsf{P_A}$. We construct the following simulator $\mathcal{S}$ that holds pk and runs $\mathcal{A}$ as a subroutine, while playing the role of $\mathsf{P_A}$ in the ideal world interacting with $\mathcal{F}$:

1. Choose uniform κ-bit strings $\{\mathsf{seed}_j^B\}_{j\in[\lambda]}$, set $h_j \leftarrow \mathsf{Com}(\mathsf{seed}_j^B)$ for all j, and send $\{h_j\}_{j\in[\lambda]}$ to $\mathcal{A}$.
2. For all $j \in [\lambda]$, run Π_{OT} with $\mathcal{A}$, using input 0 and randomness derived from seed_j^B. In this way, $\mathcal{S}$ obtains $\{\mathsf{seed}_j^A\}_{j\in[\lambda]}$. Let trans_j denote the transcript of the jth execution.
3. For $j \in [\lambda]$, run an execution of Π_{OT} with $\mathcal{A}$, using input 0^{n_2} and randomness derived from seed_j^B. Let $\mathcal{H}_j$ denote the transcript hash of the jth execution.
4. Receive $\{\mathsf{c}_j\}_{j\in[\lambda]}$ from $\mathcal{A}$.
5. Receive $\{\sigma_j\}$ from $\mathcal{A}$. If any of the signatures are invalid, send $\perp$ to $\mathcal{F}$ and halt.
6. For all $j \in [\lambda]$, use seed_j^A and the messages sent previously to simulate the computation of an honest $\mathsf{P_A}$ in steps 3 and 4, and in particular compute $\hat{\mathcal{H}}_j, \hat{\mathsf{c}}_j$. Let J be the set of indices for which $(\hat{\mathcal{H}}_j, \hat{\mathsf{c}}_j) \neq (\mathcal{H}_j, \mathsf{c}_j)$.

   There are now three cases:

   - If $|J| \geq 2$ then send $\mathsf{blatantCheat}$ to $\mathcal{F}$, send $\mathsf{cert} := (j, \mathsf{trans}_j, \mathcal{H}_j, \mathsf{c}_j, \sigma_j, \mathsf{seed}_j^B, \mathsf{decom}_j^B)$ to $\mathcal{A}$ (for uniform $j \in J$), and halt.
   - If $|J| = 1$ then send cheat to $\mathcal{F}$. If $\mathcal{F}$ returns $\mathsf{corrupted}$ then set $\mathsf{caught} := \mathsf{true}$; if $\mathcal{F}$ returns $(\mathsf{undetected}, y)$, set $\mathsf{caught} := \mathsf{false}$. In either case, continue below.
   - If $|J| = 0$ then set $\mathsf{caught} := \perp$ and continue below.

0′. Rewind $\mathcal{A}$ and run steps 1′–6′ below until[5] $|J'| = |J|$ and $\mathsf{caught}' = \mathsf{caught}$.

[5]We use standard techniques [9, 17] to ensure that $\mathcal{S}$ runs in expected polynomial time; details are omitted for the sake of the exposition.

1′. Choose uniform $\hat{\jmath} \in [\lambda]$. For $j \neq \hat{\jmath}$, choose uniform κ-bit strings $\{\mathsf{seed}_j^B\}$ and set $h_j \leftarrow \mathsf{Com}(\mathsf{seed}_j^B)$. Set $h_{\hat{\jmath}} \leftarrow \mathsf{Com}(0^\kappa)$. Send $\{h_j\}_{j\in[\lambda]}$ to $\mathcal{A}$.

2′. For all $j \neq \hat{\jmath}$, run Π_{OT} with $\mathcal{A}$, using input 0 and randomness derived from seed_j^B. In this way, $\mathcal{S}$ obtains $\{\mathsf{seed}_j^A\}_{j\neq\hat{\jmath}}$. For the $\hat{\jmath}$th execution, use the simulator $\mathcal{S}_{\mathsf{OT}}$ for protocol Π_{OT}, thus extracting both $\mathsf{seed}_{\hat{\jmath}}^A$ and $\mathsf{witness}_{\hat{\jmath}}$. Let trans_j denote the transcript of the jth execution.

3′. For all $j \neq \hat{\jmath}$, run Π_{OT} with $\mathcal{A}$, using input 0^{n_2} and randomness derived from seed_j^B. For $j = \hat{\jmath}$, use the simulator $\mathcal{S}_{\mathsf{OT}}$ for protocol Π_{OT}, thus extracting $\{B_{\hat{\jmath},i,b}\}_{i\in[n_2],b\in\{0,1\}}$. Let $\mathcal{H}_j$ denote the transcript hash of the jth execution.

4′. Receive $\{\mathsf{c}_j\}_{j\in[\lambda]}$ from $\mathcal{A}$.

5′. Receive $\{\sigma_j\}$ from $\mathcal{A}$. If any of the signatures are invalid, then return to step 1′.

6′. For all $j \in [\lambda]$, use seed_j^A and the messages sent previously to simulate the computation of an honest $\mathsf{P_A}$ in steps 3′ and 4′, and in particular compute $\hat{\mathcal{H}}_j, \hat{\mathsf{c}}_j$. Let J' be the set of indices for which $(\hat{\mathcal{H}}_j, \hat{\mathsf{c}}_j) \neq (\mathcal{H}_j, \mathsf{c}_j)$.

If $|J'| = 1$ and $\hat{\jmath} \notin J'$ then set $\mathsf{caught}' := \mathsf{true}$. If $|J'| = 1$ and $\hat{\jmath} \in J'$ then set $\mathsf{caught}' := \mathsf{false}$. If $|J'| = 0$ then set $\mathsf{caught}' := \perp$.

7. If $|J'| = 1$ and $\mathsf{caught}' = \mathsf{true}$, then send $\mathsf{cert} := (j, \mathsf{trans}_j, \mathcal{H}_j, \mathsf{c}_j, \sigma_j, \mathsf{seed}_j^B)$ to $\mathcal{A}$ (where j is the unique index in J') and halt.

   Otherwise, send $(\hat{\jmath}, \{\mathsf{seed}_j^A\}_{j\neq\hat{\jmath}}, \mathsf{witness}_{\hat{\jmath}})$ to $\mathcal{A}$.

8. Receive GC, $\{A_i\}_{i\in[n_1]}$, $\{h_{i,b}^A\}_{i\in[n_1],b\in\{0,1\}}$, $\{Z_{i,b}\}_{i\in[n_3],b\in\{0,1\}}$, and the corresponding decommitments from $\mathcal{A}$. If any of the decommitments are incorrect, send $\perp$ to $\mathcal{F}$ and halt.

   Otherwise, there are two possibilities:

   - If $|J'| = 1$ and $\mathsf{caught}' = \mathsf{false}$, then use $\{B_{\hat{\jmath},i,b}\}_{i\in[n_2],b\in\{0,1\}}$ and the value y received from $\mathcal{F}$ to compute an output z exactly as an honest $\mathsf{P_B}$ would. Send z to $\mathcal{F}$ and halt.
   - If $|J'| = 0$, then compute an effective input $x \in \{0,1\}^{n_1}$ using $\mathsf{seed}_{\hat{\jmath}}^A$ and the input-wire labels $\{A_i\}_{i\in[n_1]}$. Send x to $\mathcal{F}$ and halt.

We now show that the joint distribution of the view of $\mathcal{A}$ and the output of $\mathsf{P_B}$ in the ideal world is computationally indistinguishable from the joint distribution of the view of $\mathcal{A}$ and the output of $\mathsf{P_B}$ in a real protocol execution. We prove this by considering a sequence of experiments, where the output of each is defined to be the view of $\mathcal{A}$ and the output of $\mathsf{P_B}$, and showing that the output of each is computationally indistinguishable from the output of the next one.

Expt$_0$. This is the ideal-world execution between $\mathcal{S}$ (as described above) and the honest $\mathsf{P_B}$ holding some input y, both interacting with functionality $\mathcal{F}$.

By inlining the actions of $\mathcal{S}, \mathcal{F}$, and $\mathsf{P_B}$, we may rewrite the experiment as follows:

1. Choose uniform κ-bit strings $\{\mathsf{seed}_j^B\}_{j\in[\lambda]}$, set $h_j \leftarrow \mathsf{Com}(\mathsf{seed}_j^B)$ for all j, and send $\{h_j\}_{j\in[\lambda]}$ to $\mathcal{A}$.

2. For all $j \in [\lambda]$, run Π_{OT} with $\mathcal{A}$, using input 0 and randomness derived from seed_j^B. Obtain $\{\mathsf{seed}_j^A\}_{j\in[\lambda]}$ as the outputs. Let trans_j denote the transcript of the jth execution.

3. For $j \in [\lambda]$, run an execution of Π_{OT} with $\mathcal{A}$, using input 0^{n_2} and randomness derived from seed_j^B. Let $\mathcal{H}_j$ denote the transcript hash of the jth execution.

4. Receive $\{\mathsf{c}_j\}_{j \in [\lambda]}$ from $\mathcal{A}$.

5. Receive $\{\sigma_j\}$ from $\mathcal{A}$. If any of the signatures are invalid, then $\mathsf{P_B}$ outputs $\perp$ and the experiment halts.

6. For all $j \in [\lambda]$, use seed_j^A and the messages sent previously to $\mathcal{A}$ to simulate the computation of an honest $\mathsf{P_A}$ in steps 3 and 4, and in particular compute $\hat{\mathcal{H}}_j, \hat{\mathsf{c}}_j$. Let J be the set of indices for which $(\hat{\mathcal{H}}_j, \hat{\mathsf{c}}_j) \neq (\mathcal{H}_j, \mathsf{c}_j)$.

   There are now three cases:

   - If $|J| \geq 2$, send $\mathsf{cert} := (j, \mathsf{trans}_j, \mathcal{H}_j, \mathsf{c}_j, \sigma_j, \mathsf{seed}_j^B)$ to $\mathcal{A}$ (for uniform $j \in J$). Then $\mathsf{P_B}$ outputs $\mathsf{corrupted}$ and the experiment halts.
   - If $|J| = 1$ then with probability ϵ set $\mathsf{caught} := \mathsf{true}$ and with the remaining probability set $\mathsf{caught} := \mathsf{false}$. If $\mathsf{caught} = \mathsf{true}$ then $\mathsf{P_B}$ outputs $\mathsf{corrupted}$ (but the experiment continues below in either case).
   - If $|J| = 0$ then set $\mathsf{caught} := \perp$ and continue below.

0′. Rewind $\mathcal{A}$ and run steps 1′–6′ below until $|J'| = |J|$ and $\mathsf{caught}' = \mathsf{caught}$ (using standard techniques [9, 17] to ensure the experiment runs in expected polynomial time).

   1′. Choose uniform $\hat{\jmath} \in [\lambda]$. For $j \neq \hat{\jmath}$, choose uniform κ-bit strings $\{\mathsf{seed}_j^B\}$ and set $h_j \leftarrow \mathsf{Com}(\mathsf{seed}_j^B)$. Set $h_{\hat{\jmath}} \leftarrow \mathsf{Com}(0^\kappa)$. Send $\{h_j\}_{j \in [\lambda]}$ to $\mathcal{A}$.

   2′. For all $j \neq \hat{\jmath}$, run Π_{OT} with $\mathcal{A}$, using input 0 and randomness derived from seed_j^B. Obtain $\{\mathsf{seed}_j^A\}_{j \neq \hat{\jmath}}$ as the outputs of these executions. For the $\hat{\jmath}$th execution, use the simulator $\mathcal{S}_{\mathsf{OT}}$ for protocol Π_{OT}, thus extracting both $\mathsf{seed}_{\hat{\jmath}}^A$ and $\mathsf{witness}_{\hat{\jmath}}$. Let trans_j denote the transcript of the jth execution.

   3′. For $j \neq \hat{\jmath}$, run an execution of Π_{OT} with $\mathcal{A}$ using input 0^{n_2} and randomness derived from seed_j^B. For $j = \hat{\jmath}$, use the simulator $\mathcal{S}_{\mathsf{OT}}$ for protocol Π_{OT}, thus extracting $\{B_{\hat{\jmath},i,b}\}_{i \in [n_2], b \in \{0,1\}}$. Let $\mathcal{H}_j$ denote the transcript hash of the jth execution.

   4′. Receive $\{\mathsf{c}_j\}_{j \in [\lambda]}$ from $\mathcal{A}$.

   5′. Receive $\{\sigma_j\}$ from $\mathcal{A}$. If any of the signatures are invalid, then return to step 1′.

   6′. For all $j \in [\lambda]$, use seed_j^A and the messages sent previously to simulate the computation of an honest $\mathsf{P_A}$ in steps 3′ and 4′, and in particular compute $\hat{\mathcal{H}}_j, \hat{\mathsf{c}}_j$. Let J' be the set of indices for which $(\hat{\mathcal{H}}_j, \hat{\mathsf{c}}_j) \neq (\mathcal{H}_j, \mathsf{c}_j)$.

   If $|J'| = 1$ and $\hat{\jmath} \notin J'$ then set $\mathsf{caught}' := \mathsf{true}$. If $|J'| = 1$ and $\hat{\jmath} \in J'$ then set $\mathsf{caught}' := \mathsf{false}$. If $|J'| = 0$ then set $\mathsf{caught}' := \perp$.

7. If $|J'| = 1$ and $\mathsf{caught}' = \mathsf{true}$, then send $\mathsf{cert} := (j, \mathsf{trans}_j, \mathcal{H}_j, \mathsf{c}_j, \sigma_j, \mathsf{seed}_j^B)$ to $\mathcal{A}$ (where j is the unique index in J') and halt.

   Otherwise, send $(\hat{\jmath}, \{\mathsf{seed}_j^A\}_{j \neq \hat{\jmath}}, \mathsf{witness}_{\hat{\jmath}})$ to $\mathcal{A}$.

8. Receive GC, $\{A_i\}_{i\in[n_1]}$, $\{h^A_{i,b}\}_{i\in[n_1],b\in\{0,1\}}$, $\{Z_{i,b}\}_{i\in[n_3],b\in\{0,1\}}$, and the corresponding decommitments from $\mathcal{A}$. If any of the decommitments are incorrect, then $\mathsf{P_B}$ outputs $\perp$ and the experiment halts.

   Otherwise, there are two possibilities:

   - If $|J'| = 1$ and $\mathsf{caught}' = \mathsf{false}$ then use $\{B_{\hat{\jmath},i,b}\}_{i\in[n_2],b\in\{0,1\}}$ and y to compute z exactly as in the protocol. $\mathsf{P_B}$ outputs z and the experiment halts.
   - If $|J'| = |J| = 0$, compute an effective input $x \in \{0,1\}^{n_1}$ using $\mathsf{seed}^A_{\hat{\jmath}}$ and the input-wire labels $\{A_i\}_{i\in[n_1]}$. Then $\mathsf{P_B}$ outputs $f(x,y)$ and the experiment halts.

Expt$_1$. Here we modify the previous experiment in the following way: Choose a uniform $\hat{\jmath} \in [\lambda]$ at the outset of the experiment. Then in step 6:

- If $|J| \geq 2$ then send $\mathsf{cert} := (j, \mathsf{trans}_j, \mathcal{H}_j, \mathsf{c}_j, \sigma_j, \mathsf{seed}^B_j)$ to $\mathcal{A}$ for uniform $j \in J \setminus \{\hat{\jmath}\}$. Then $\mathsf{P_B}$ outputs $\mathsf{corrupted}$ and the experiment halts.

- if $|J| = 1$ set $\mathsf{caught} := \mathsf{true}$ if $\hat{\jmath} \notin J$ and set $\mathsf{caught} := \mathsf{false}$ if $\hat{\jmath} \in J$.

Since $\hat{\jmath} \notin J$ with probability ϵ when $|J| = 1$, the outputs of **Expt**$_1$ and **Expt**$_0$ are identically distributed.

Expt$_2$. The previous experiment is modified as follows: In step 1, do not choose $\mathsf{seed}^B_{\hat{\jmath}}$. Instead, in step 1 set $h_{\hat{\jmath}} \leftarrow \mathsf{Com}(0^\kappa)$, and in steps 2 and 4 use true randomness in the $\hat{\jmath}$th execution of Π_{OT}.

It is immediate that the distribution of the output of **Expt**$_2$ is computationally indistinguishable from the distribution of the output of **Expt**$_1$.

Expt$_3$. We change the previous experiment in the following way: In steps 2 and 4, use $\mathcal{S}_{\mathsf{OT}}$ to run the $\hat{\jmath}$th instances of Π_{OT}. In doing so, extract all of $\mathcal{A}$'s inputs in those executions.

It follows from security of Π_{OT} that the distribution of the output of **Expt**$_3$ is computationally indistinguishable from the distribution of the output of **Expt**$_2$.

Expt$_{3a}$. Because steps $1'$–$4'$ in **Expt**$_3$ are identical to steps 1–4, we can "collapse" the rewinding and thus obtain the following experiment **Expt**$_{3a}$ that is statistically indistinguishable from **Expt**$_3$ (with the only difference occurring in case of an aborted rewinding in the latter):

1. Choose uniform $\hat{\jmath} \in [\lambda]$. For $j \neq \hat{\jmath}$, choose uniform κ-bit strings $\{\mathsf{seed}^B_j\}$ and set $h_j \leftarrow \mathsf{Com}(\mathsf{seed}^B_j)$. Set $h_{\hat{\jmath}} \leftarrow \mathsf{Com}(0^\kappa)$. Send $\{h_j\}_{j\in[\lambda]}$ to $\mathcal{A}$.

2. For all $j \neq \hat{\jmath}$, run Π_{OT} with $\mathcal{A}$, using input 0 and randomness derived from seed^B_j. Obtain $\{\mathsf{seed}^A_j\}_{j\neq\hat{\jmath}}$ as the outputs of these executions. For the $\hat{\jmath}$th execution, use the simulator $\mathcal{S}_{\mathsf{OT}}$ for protocol Π_{OT}, thus extracting both $\mathsf{seed}^A_{\hat{\jmath}}$ and $\mathsf{witness}_{\hat{\jmath}}$. Let trans_j denote the transcript of the jth execution.

3. For all $j \neq \hat{\jmath}$, run Π_{OT} with $\mathcal{A}$ using input 0^{n_2} and randomness derived from seed^B_j. For $j = \hat{\jmath}$, use the simulator $\mathcal{S}_{\mathsf{OT}}$ for protocol Π_{OT}, thus extracting $\{B_{\hat{\jmath},i,b}\}_{i\in[n_2],b\in\{0,1\}}$. Let $\mathcal{H}_j$ denote the transcript hash of the jth execution.

4. Receive $\{\mathsf{c}_j\}_{j\in[\lambda]}$ from $\mathcal{A}$.

5. Receive $\{\sigma_j\}$ from $\mathcal{A}$. If any of the signatures are invalid, then $\mathsf{P_B}$ outputs $\perp$ and the experiment halts.

6. For all $j \in [\lambda]$, use seed_j^A and the messages sent previously to $\mathcal{A}$ to simulate the computation of an honest $\mathsf{P_A}$ in steps 3 and 4, and in particular compute $\hat{\mathcal{H}}_j, \hat{\mathsf{c}}_j$. Let J be the set of indices for which $(\hat{\mathcal{H}}_j, \hat{\mathsf{c}}_j) \neq (\mathcal{H}_j, \mathsf{c}_j)$.

   There are now two cases:

   - If $|J| \geq 2$, or if $|J| = 1$ and $\hat{\jmath} \notin J$, then choose uniform $j \in J \setminus \{\hat{\jmath}\}$ and send $\mathsf{cert} := (j, \mathsf{trans}_j, \mathcal{H}_j, \mathsf{c}_j, \sigma_j, \mathsf{seed}_j^B)$ to $\mathcal{A}$. Then $\mathsf{P_B}$ outputs $\mathsf{corrupted}$ and the experiment halts.
   - If $|J| = 1$ and $\hat{\jmath} \in J$, or if $|J| = 0$, then continue below.

7. Send $(\hat{\jmath}, \{\mathsf{seed}_j^A\}_{j \neq \hat{\jmath}}, \mathsf{witness}_{\hat{\jmath}})$ to $\mathcal{A}$.

8. Receive GC, $\{A_i\}_{i\in[n_1]}$, $\{h_{i,b}^A\}_{i\in[n_1],b\in\{0,1\}}$, $\{Z_{i,b}\}_{i\in[n_3],b\in\{0,1\}}$, and the corresponding decommitments from $\mathcal{A}$. If any of the decommitments are incorrect, then $\mathsf{P_B}$ outputs $\perp$ and the experiment halts.

   Otherwise, there are two possibilities:

   - If $|J| = 1$ then $\mathsf{P_B}$ uses $\{B_{\hat{\jmath},i,b}\}_{i\in[n_2],b\in\{0,1\}}$ and y to compute z exactly as in the protocol. $\mathsf{P_B}$ outputs z and the experiment halts.
   - If $|J| = 0$, then compute an effective input $x \in \{0,1\}^{n_1}$ using $\mathsf{seed}_{\hat{\jmath}}^A$ and the input-wire labels $\{A_i\}_{i\in[n_1]}$. Then $\mathsf{P_B}$ outputs $f(x,y)$ and the experiment halts.

Expt$_4$. We modify the previous experiment as follows: In step 8, if $|J| = 0$ (and $\mathsf{P_B}$ has not already output $\perp$ in that step), use y to compute z exactly as in the protocol. Then $\mathsf{P_B}$ outputs z and the experiment halts.

Since $|J| = 0$, we know that $\mathsf{c}_{\hat{\jmath}}$ is a commitment to a correctly computed garbled circuit along with commitments to (correctly permuted) input-wire labels $\{A_{\hat{\jmath},i,b}\}$ and output-wire labels. Thus—unless $\mathcal{A}$ has managed to violate the commitment property of Com—if $\mathsf{P_B}$ does not output $\perp$ in this step it must be the case that the values GC, $\{A_i\}_{i\in[n_1]}$, $\{h_{i,b}^A\}_{i\in[n_1],b\in\{0,1\}}$, and $\{Z_{i,b}\}_{i\in[n_3],b\in\{0,1\}}$ sent by $\mathcal{A}$ in step 8 are correct. Moreover, since $|J| = 0$ the execution of Π_{OT} in step 4 was run honestly by $\mathcal{A}$ using correct input-wire labels $\{B_{\hat{\jmath},i,b}\}$. Thus, evaluating GC using $\{A_i\}_{i\in[n_1]}$ and $\{B_{\hat{\jmath},i,y[i]}\}$ yields a result that is equal to $f(x,y)$ as computed in **Expt**$_3$.

Since Com is computationally binding, this means that the distribution of the output of **Expt**$_4$ is computationally indistinguishable from the distribution of the output of **Expt**$_{3a}$.

Expt$_5$. Here we change the previous experiment in the following way: The computation in step 6 is done only for $j \in [\lambda] \setminus \{\hat{\jmath}\}$; let $\hat{J} \subseteq [\lambda] \setminus \{\hat{\jmath}\}$ be the set of indices for which $(\hat{\mathcal{H}}_j, \hat{\mathsf{c}}_j) \neq (\mathcal{H}_j, \mathsf{c}_j)$. Then:

- If $\hat{J} \neq \emptyset$ choose uniform $j \in \hat{J}$ and send $\mathsf{cert} := (j, \mathsf{trans}_j, \mathcal{H}_j, \mathsf{c}_j, \sigma_j, \mathsf{seed}_j^B)$ to $\mathcal{A}$. Then $\mathsf{P_B}$ outputs $\mathsf{corrupted}$ and the experiment halts.

- If $\hat{J} = \emptyset$ then run steps 7 and 8 as in **Expt**$_4$.

Letting J be defined as in $\mathbf{Expt}_4$, note that

$$|J| \geq 2 \text{ or } |J| = 1;\ \hat{\jmath} \notin J \Longleftrightarrow \hat{J} \neq \emptyset$$

and

$$|J| = 1, \hat{\jmath} \in J \text{ or } |J| = 0 \Longleftrightarrow \hat{J} = \emptyset.$$

Thus, the outputs of $\mathbf{Expt}_4$ and $\mathbf{Expt}_5$ are identically distributed.

$\mathbf{Expt}_6$. We now modify the previous experiment by running the $\hat{\jmath}$th instances of Π_{OT} honestly in steps 2 and 4, using input 1 in step 2 and input y in step 4.

It follows from security of Π_{OT} that the distribution of the output of $\mathbf{Expt}_6$ is computationally indistinguishable from the distribution of the output of $\mathbf{Expt}_5$.

$\mathbf{Expt}_7$. Finally, we modify the previous experiment so the $\hat{\jmath}$th instance of Π_{OT} in steps 2 and 4 uses pseudorandomness derived from a uniform seed $\mathsf{seed}^B_{\hat{\jmath}}$, and we compute $h_{\hat{\jmath}} \leftarrow \mathsf{Com}(\mathsf{seed}^B_{\hat{\jmath}})$.

It is immediate that the distribution of the output of $\mathbf{Expt}_7$ is computationally indistinguishable from the distribution of the output of $\mathbf{Expt}_6$.

Since $\mathbf{Expt}_7$ corresponds to a real-world execution of the protocol between $\mathcal{A}$ and $\mathsf{P_B}$ holding input y, this completes the proof.

Covert Security—Malicious $\mathsf{P_B}$

Let $\mathcal{A}$ be an adversary corrupting $\mathsf{P_B}$. We construct the following simulator $\mathcal{S}$ that runs $\mathcal{A}$ as a subroutine while playing the role of $\mathsf{P_B}$ in the ideal world interacting with $\mathcal{F}$:

0. Run Gen to generate keys (pk, sk), and send pk to $\mathcal{A}$.
1. Receive $\{h_j\}_{j\in[\lambda]}$ from $\mathcal{A}$.
2. Use the simulator $\mathcal{S}_{\mathsf{OT}}$ for protocol Π_{OT} to interact with $\mathcal{A}$. In this way, $\mathcal{S}$ extracts $\mathcal{A}$'s inputs $\{b_j\}_{j\in[\lambda]}$; let $J := \{j : b_j = 1\}$. As part of the simulation, return uniform κ-bit strings $\{\mathsf{seed}^A_j\}_{j\notin J}$ and $\{\mathsf{witness}_j\}_{j\in J}$ as output to $\mathcal{A}$.
3. For each $j \notin J$, run this step exactly as an honest $\mathsf{P_A}$ would. For each $j \in J$ do:
   - If $|J| = 1$ then let $\hat{\jmath}$ be the unique index in J. Use $\mathcal{S}_{\mathsf{OT}}$ to interact with $\mathcal{A}$ in the $\hat{\jmath}$th execution of Π_{OT}. In this way, $\mathcal{S}$ extracts $\mathcal{A}$'s input y for that execution. Send y to $\mathcal{F}$, and receive in return a value z. Compute

     $$(\{A_{\hat{\jmath},i}\}, \{B_{\hat{\jmath},i}\}, \mathsf{GC}_{\hat{\jmath}}, \{Z_{\hat{\jmath},i,b}\}) \leftarrow \mathcal{S}_{\mathsf{Gb}}(1^\kappa, \mathcal{C}, z),$$

     where we let $\{A_{\hat{\jmath},i}\}$ correspond to input wires of $\mathsf{P_A}$ and $\{B_{\hat{\jmath},i}\}$ correspond to input wires of $\mathsf{P_B}$. Return $\{B_{\hat{\jmath},i}\}$ as output to $\mathcal{A}$ from this execution of Π_{OT}.
   - If $|J| > 1$ then act as an honest $\mathsf{P_A}$ would but using true randomness.
4. For each $j \notin J$, compute c_j exactly as an honest $\mathsf{P_A}$ would. For each $j \in J$ do:
   - If $|J| = 1$ then compute $h^A_{\hat{\jmath},i,0} \leftarrow \mathsf{Com}(A_{\hat{\jmath},i})$ and let $h^A_{\hat{\jmath},i,1}$ be a commitment to the 0-string. Compute $\mathsf{c}_{\hat{\jmath}} \leftarrow \mathsf{Com}(\mathsf{GC}_{\hat{\jmath}}, \{h^A_{\hat{\jmath},i,b}\}, \{Z_{\hat{\jmath},i,b}\})$, where each pair $(h^A_{\hat{\jmath},i,0}, h^A_{\hat{\jmath},i,1}\}$ is in random permuted order.

- If $|J| > 1$ then compute c_j exactly as an honest $\mathsf{P_A}$ would but using true randomness.

Send $\{\mathsf{c}_j\}_{j\in[\lambda]}$ to $\mathcal{A}$.

5–6. Compute signatures $\{\sigma_j\}$ as an honest $\mathsf{P_A}$ would, and send them to $\mathcal{A}$.

7. If $|J| \neq 1$ then abort. Otherwise, receive $(\hat{\jmath}, \{\mathsf{seed}_j\}_{j\neq\hat{\jmath}}, \mathsf{witness}_{\hat{\jmath}})$ from $\mathcal{A}$ and verify these as an honest $\mathsf{P_A}$ would. (If verification fails, then abort.)

8. Send $\mathsf{GC}_{\hat{\jmath}}$, $\{A_{\hat{\jmath},i}\}$, $\{h^A_{\hat{\jmath},i,b}\}$ (in the same permuted order as before), and $\{Z_{\hat{\jmath},i,b}\}$ to $\mathcal{A}$, along with the corresponding decommitments. Then halt.

We show that the distribution of the view of $\mathcal{A}$ in the ideal world is computationally indistinguishable from its view in a real protocol execution. (Note that $\mathsf{P_A}$ has no output.) Let $\mathbf{Expt}_0$ be the ideal-world execution between $\mathcal{S}$ (as described above) and the honest $\mathsf{P_A}$ holding some input x, both interacting with functionality $\mathcal{F}$.

$\mathbf{Expt}_1$. Here we modify the previous experiment when $|J| = 1$ as follows. In step 3, compute

$$(\{A_{\hat{\jmath},i,b}\}, \{B_{\hat{\jmath},i,b}\}, \mathsf{GC}_{\hat{\jmath}}, \{Z_{\hat{\jmath},i,b}\}) \leftarrow \mathsf{Gb}(1^\kappa, \mathcal{C}),$$

and return the values $\{B_{\hat{\jmath},i,y[i]}\}$ as output to $\mathcal{A}$ from the simulated execution of Π_{OT} in that step. In steps 4 and 8, the values $A_{\hat{\jmath},i,x[i]}$ are used in place of $A_{\hat{\jmath},i}$.

It follows from security of the garbling scheme that the view of $\mathcal{A}$ in $\mathbf{Expt}_1$ is computationally indistinguishable from its view in $\mathbf{Expt}_0$.

$\mathbf{Expt}_2$. Now we change the previous experiment when $|J| = 1$ as follows: In step 3, compute $h^A_{\hat{\jmath},i,b} \leftarrow \mathsf{Com}(A_{\hat{\jmath},i,b})$ for all i, b. It follows from the hiding property of the commitment scheme that the view of $\mathcal{A}$ in $\mathbf{Expt}_2$ is computationally indistinguishable from its view in $\mathbf{Expt}_1$.

$\mathbf{Expt}_3$. This time, the previous experiment is modified by executing protocol Π_{OT} with $\mathcal{A}$ when $|J| = 1$ in step 3. Security of Π_{OT} implies that the view of $\mathcal{A}$ in $\mathbf{Expt}_3$ is computationally indistinguishable from its view in $\mathbf{Expt}_2$.

$\mathbf{Expt}_4$. The previous experiment is now modified in the following way. In step 2, also choose uniform $\{\mathsf{seed}^A_j\}_{j\in J}$ and $\{\mathsf{witness}^A_j\}_{j\notin J}$, and use pseudorandomness derived from $\{\mathsf{seed}^A_j\}_{j\in J}$ in steps 3 and 4 in place of true randomness. Also, in step 7 continue to run the protocol as an honest $\mathsf{P_A}$ would even in the case that $|J| \neq 1$.

It is not hard to show that when $|J| \neq 1$ then $\mathsf{P_A}$ aborts in $\mathbf{Expt}_4$ with all but negligible probability. Computational indistinguishability of $\mathcal{A}$'s view in $\mathbf{Expt}_4$ and $\mathbf{Expt}_3$ follows.

$\mathbf{Expt}_5$. Finally, we change the last experiment by executing protocol Π_{OT} in step 2. It follows from the security of Π_{OT} that the view of $\mathcal{A}$ in $\mathbf{Expt}_5$ is computationally indistinguishable from its view in $\mathbf{Expt}_4$.

Since $\mathbf{Expt}_5$ corresponds to a real-world execution of the protocol, this completes the proof.

Public Verifiability and Defamation Freeness

It is easy to check (by inspecting the protocol) that whenever an honest $\mathsf{P_B}$ outputs $\mathsf{corrupted}$ then it also outputs a valid certificate. Thus our protocol satisfies public verifiability. It is similarly easy to verify defamation freeness under the assumptions of the theorem. □

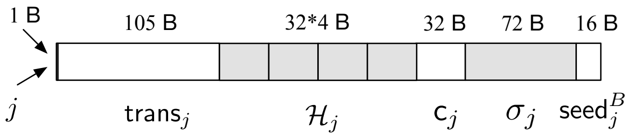
6 Implementation and Evaluation

We implemented our PVC protocol using the optimizations from Section 4.3 and state-of-the-art techniques for garbling [4, 21], oblivious transfer [5], and OT extension [13]. Our implementation uses SHA-256 for the hash function (modeled as a random oracle) and the standard ECDSA implementation provided by openssl as the signature scheme. We target $\kappa = 128$ in our implementation.

We evaluate our protocol in both LAN and WAN settings using the Alibaba Cloud. In the LAN setting, the network bandwidth is 1 Gbps and the latency is less than 1 ms; in the WAN setting, the bandwidth is 200 Mbps and the latency is 75 ms. In either setting, the machines running the protocol have 8 cores, each running at 2.5GHz. Due to pipelining, we never observe any issues with memory usage. All reported timing results are computed as the average of 10 executions.

6.1 Certificate Size

The size of the certificate in our protocol is independent of the circuit size or the lengths of the parties' inputs. The following figure gives a graphical decomposition of the certificate. (Note that since we instantiate Com by a random oracle as discussed in Section 4.3, we do not need to include an extra decommitment in the certificate.) In total, a certificate requires 354 bytes.



$\mathcal{H}_j$ contains 4 hash values, corresponding to a 4-round OT protocol obtained by piggybacking a 2-round OT-extension protocol with a 3-round base-OT protocol. The signature size varies from 70–72 bytes; we allocate 72 bytes for the signature so the total length of a certificate is fixed.

6.2 Comparison to Prior PVC Protocols

Because it enables signed-OT extension, the PVC protocol by Kolesnikov and Malozemoff [14] (the *KM15 protocol*) would be strictly more efficient than the original PVC protocol by Asharov and Orlandi [2]. We therefore focus our attention on the KM15 protocol. We compare our protocol to theirs in three respects.

Parameters. We briefly discuss the overhead needed to achieve deterrence factors larger than $\frac{1}{2}$ for each protocol. Recall that in the KM15 protocol the overall deterrence factor ϵ depends on both the garbled-circuit replication factor λ and the XOR-tree expansion factor ν as $\epsilon = (1-\frac{1}{\lambda})\cdot(1-2^{-\nu+1})$. For deterrence $\epsilon \approx 1 - \frac{1}{2^k}$, setting $\lambda = 2^{k+1}, \nu = k+2$ gives the best efficiency. In contrast, our protocol achieves this deterrence with $\lambda = 2^k, \nu = 1$, which means garbling half as many circuits and avoiding the XOR-tree approach altogether. For example, to achieve deterrence $\epsilon = 7/8$, our protocol garbles 8 circuits, whereas prior work would need to garble 16 circuits. Additionally, prior work would need to execute 5× as many OTs. (Plus, in prior work each OT is actually a *signed* OT, which is more expensive than standard OT; see next.)

Signed OT vs. standard OT. Signed OT induces higher costs than standard OT in terms of both communication and computation. As an illustration, fix the deterrence factor to 1/2. In that case our protocol runs OT extension twice, where each is used for n_2 OTs on κ-bit strings. Compared to this, the KM15 protocol needs to run $3n_2$ OTs on 2κ-bit strings. The total communication complexity of the OT step (for the input-wire labels) is $4\kappa n_2$ bits in our protocol, while in the KM15 protocol it is $3 * 2 * 3\kappa n_2 + 3 * 2.6\kappa n_2 = 25.8\kappa n_2$ bits, more than $6\times$ higher.

Moreover, signed OT also has a very high computational overhead:

- Signed-OT extension needs to use a wider matrix (by a factor of roughly $2.6\times$) compared to standard OT extension. Besides the direct penalty this incurs, a wider matrix means that the correlation-robust hash H cannot be based on fixed-key AES but must instead be based on a hash function like SHA-256. This impacts performance significantly.
- As part of signed-OT extension, $\mathsf{P_B}$ needs to reveal κ random columns in the matrix. Even with AVX operations, this incurs significant computational overhead.

Signed-OT extension [14] is complex, and we did not implement it in its entirety. However, we modified an existing (standard) OT-extension protocol to match the matrix width required by signed-OT extension; this can be used to give a conservative lower bound on the performance of signed-OT extension. Our results indicate that signed-OT extension requires roughly $5\times$ more computation than state-of-the-art OT extension.

Certificate size. In the KM15 protocol, the certificate size is at least $2\kappa \cdot n_2$ bits. Even for AES (with only 128-bit input length), this gives a certificate roughly $10\times$ larger than ours.

6.3 Comparing to Semi-Honest and Malicious Protocols

We believe our PVC protocol provides an excellent performance/security tradeoff that makes it the best choice for many applications of secure computation.

Performance. Our protocol is not much less efficient that the best known semi-honest protocols, and is significantly faster than the best known malicious protocols.

Security. The PVC model provides much more meaningful guarantees than the notion of semi-honest security, and may be appropriate for many (even if not all) applications of secure computation where full malicious security is overkill.

To support the first point, we compare the performance of our PVC protocol against state-of-the-art two-party computation protocols. The semi-honest protocol we compare against is a garbled-circuit protocol including all existing optimizations; for the malicious protocol we use the recent implementation of Wang et al. [20]. Our comparison uses the circuits listed in Table 1.

Circuit	n_1	n_2	n_3	$\lvert\mathcal{C}\rvert$
AES-128	128	128	128	6800
SHA-128	256	256	160	37300
SHA-256	256	256	256	90825
Sorting	131072	131072	131072	10223K
Integer mult.	2048	2048	2048	4192K
Hamming dist.	1048K	1048K	22	2097K

Table 1: Circuits used in our evaluation. The parties' input lengths are n_1 and n_2, and the output length is n_3. The number of AND gates in the circuit is denoted by $|\mathcal{C}|$.

Running time. In Table 2 we compare the running time of our protocol to that of a semi-honest protocol. From the table, we see that over a LAN our protocol adds at most 36% overhead except in two cases: AES and Hamming-distance computation. For AES, the reason is that the circuit is small and so the overall time is dominated by the base OTs. For Hamming distance, the total input size is equal to the number of AND gates in the circuit; therefore, the cost of processing the inputs becomes more significant.

In the WAN setting, our PVC protocol incurs only 17% overhead except for the Hamming-distance example (for a similar reason as above).

Circuit	LAN setting			WAN setting		
	Our PVC	Semi-honest	Slowdown	Our PVC	Semi-honest	Slowdown
AES-128	24.53 ms	15.31 ms	1.60×	960.4 ms	820.8 ms	1.17×
SHA-128	33.67 ms	24.69 ms	1.36×	1146 ms	976.8 ms	1.17×
SHA-256	48.43 ms	38.04 ms	1.27×	1252 ms	1080 ms	1.16×
Sort.	3468 ms	2715 ms	1.28×	13130 ms	12270 ms	1.07×
Mult.	1285 ms	1110 ms	1.16×	5707 ms	5462 ms	1.04×
Hamming	2585 ms	1550 ms	1.67×	11850 ms	6317 ms	1.69×

Table 2: Comparing the running times of our protocol and a semi-honest protocol in the LAN and WAN settings.

The comparison between our PVC protocol and the malicious protocol is shown in Table 3. As expected, our PVC protocol achieves much better performance, by a factor of 4–18×.

Circuit	LAN setting			WAN setting		
	Our PVC	Malicious [20]	Speedup	Our PVC	Malicious [20]	Speedup
AES-128	24.53 ms	157.3 ms	6.41×	960.4 ms	11170 ms	11.6×
SHA-128	33.67 ms	318.8 ms	9.47×	1146 ms	13860 ms	12.1×
SHA-256	48.43 ms	611.7 ms	12.6×	1252 ms	17300 ms	13.8×
Sort.	3468 ms	45130 ms	13.0×	13130 ms	197900 ms	15.1×
Mult.	1285 ms	17860 ms	13.9×	5707 ms	99930 ms	17.5×
Hamming	2586ms	11380 ms	4.40×	11850 ms	76280 ms	6.44×

Table 3: Comparing the running times of our protocol and a malicious protocol in the LAN and WAN settings.

Communication complexity. We also compare the communication complexity of our protocol to other protocols in a similar way; see Table 4. In this comparison we use the same semi-honest protocol as above, but use the more communication-efficient protocol by Katz et al. [12] as the malicious protocol. We see that, with the exception of the Hamming-distance example, the communication in our protocol is very close to the communication in the semi-honest case.

	AES-128	SHA-128	SHA-256	Sort.	Mult.	Hamming
Semi-honest	0.2218 MB	1.165 MB	2.800 MB	313.1 MB	128.0 MB	96.01 MB
Malicious [12]	3.545 MB	17.69 MB	42.95 MB	2953 MB	1228 MB	662.7 MB
Our PVC	0.2427 MB	1.205 MB	2.844 MB	325.1 MB	128.2 MB	144.2 MB

Table 4: Communication complexity of our protocol and other protocols.

6.4 Higher Deterrence Factors

Another important aspect of our protocol is how the performance is affected by the deterrence factor. Recall that the deterrence factor ϵ is the probability that a cheating party is caught, and in our protocol $\epsilon = 1 - \frac{1}{\lambda}$ where λ is the garbled-circuit replication factor. The performance of our protocol as a function of ϵ is shown in Table 5. We see that when doubling the value of λ, the running time of the protocol increases by only $\approx 20\%$ unless the circuit is very small (in which case the cost of the base OTs dominates the total running time). The running time when $\epsilon = 3/4$ (i.e., $\lambda = 4$) is still less than twice the running time of a semi-honest protocol.

		AES-128	SHA-128	SHA-256	Sort.	Mult.	Hamming
LAN	$\epsilon = 1/2, \lambda = 2$	24.53 ms	33.67 ms	48.43 ms	3468 ms	1285 ms	2586 ms
	$\epsilon = 3/4, \lambda = 4$	35.63 ms	45.92 ms	59.25 ms	3554 ms	1308 ms	3156 ms
	$\epsilon = 7/8, \lambda = 8$	46.62 ms	57.25 ms	71.31 ms	3954 ms	1396 ms	4856 ms
WAN	$\epsilon = 1/2, \lambda = 2$	960.4 ms	1146 ms	1252 ms	13130 ms	5707 ms	11850 ms
	$\epsilon = 3/4, \lambda = 4$	1112 ms	1375 ms	1700 ms	14400 ms	5952 ms	12899 ms
	$\epsilon = 7/8, \lambda = 8$	1424 ms	1912 ms	2436 ms	16130 ms	6167 ms	19840 ms

Table 5: Running time of our protocol for different ϵ, λ.

6.5 Scalability

Our protocol scales linearly in all parameters, and so can easily handle large circuits. To demonstrate this, we benchmarked our protocol with different input lengths, output lengths, and circuit sizes. The results are summarized in Figure 6.

	n_1 (μs/bit)	n_2 (μs/bit)	n_3 (μs/bit)	$\lvert\mathcal{C}\rvert$ (μs/gate)
LAN	0.20	0.88	0.23	0.29
WAN	0.61	3.13	0.62	1.10

Table 6: Scalability of our protocol. Initially, the input and output lengths are all 128 bits, and the circuit size is 1024 AND gates. We then gradually increase one of the input/output lengths or circuit size (while holding everything else constant) and record the running time. Since the dependence is linear in all cases, we report only the marginal cost (i.e., the slope) above.

References

[1] Gilad Asharov, Yehuda Lindell, Thomas Schneider, and Michael Zohner. More efficient oblivious transfer and extensions for faster secure computation. In *20th ACM Conf. on Computer and Communications Security (CCS)*, pages 535–548. ACM Press, 2013.

[2] Gilad Asharov and Claudio Orlandi. Calling out cheaters: Covert security with public verifiability. In *Advances in Cryptology—Asiacrypt 2012*, volume 7658 of *LNCS*, pages 681–698. Springer, 2012.

[3] Yonatan Aumann and Yehuda Lindell. Security against covert adversaries: Efficient protocols for realistic adversaries. *Journal of Cryptology*, 23(2):281–343, 2010.

[4] Mihir Bellare, Viet Tung Hoang, Sriram Keelveedhi, and Phillip Rogaway. Efficient garbling from a fixed-key blockcipher. In *2013 IEEE Symposium on Security & Privacy*, pages 478–492. IEEE, 2013.

[5] Tung Chou and Claudio Orlandi. The simplest protocol for oblivious transfer. In *Progress in Cryptology—Latincrypt 2015*, volume 9230 of *LNCS*, pages 40–58. Springer, 2015.

[6] Ivan Damgård, Martin Geisler, and Jesper Buus Nielsen. From passive to covert security at low cost. In *7th Theory of Cryptography Conference—TCC 2010*, volume 5978 of *LNCS*, pages 128–145. Springer, 2010.

[7] Xiong Fan, Chaya Ganesh, and Vladimir Kolesnikov. Hashing garbled circuits for free. In *Advances in Cryptology—Eurocrypt 2017, Part III*, volume 10212 of *LNCS*, pages 456–485. Springer, 2017.

[8] O. Goldreich. *Foundations of Cryptography, vol. 2: Basic Applications.* Cambridge University Press, Cambridge, UK, 2004.

[9] Oded Goldreich and Ariel Kahan. How to construct constant-round zero-knowledge proof systems for NP. *Journal of Cryptology*, 9(3):167–190, 1996.

[10] Vipul Goyal, Payman Mohassel, and Adam Smith. Efficient two party and multi party computation against covert adversaries. In *Advances in Cryptology—Eurocrypt 2008*, volume 4965 of *LNCS*, pages 289–306. Springer, 2008.

[11] Jonathan Katz and Rafail Ostrovsky. Round-optimal secure two-party computation. In *Advances in Cryptology—Crypto 2004*, volume 3152 of *LNCS*, pages 335–354. Springer, 2004.

[12] Jonathan Katz, Samuel Ranellucci, Mike Rosulek, and Xiao Wang. Optimizing authenticated garbling for faster secure two-party computation. In *Advances in Cryptology—Crypto 2018, Part III*, volume 10993 of *LNCS*, pages 365–391. Springer, 2018.

[13] Marcel Keller, Emmanuela Orsini, and Peter Scholl. Actively secure OT extension with optimal overhead. In *Advances in Cryptology—Crypto 2015, Part I*, volume 9215 of *LNCS*, pages 724–741. Springer, 2015.

[14] Vladimir Kolesnikov and Alex J. Malozemoff. Public verifiability in the covert model (almost) for free. In *Advances in Cryptology—Asiacrypt 2015, Part II*, volume 9453 of *LNCS*, pages 210–235. Springer, 2015.

[15] Vladimir Kolesnikov and Thomas Schneider. Improved garbled circuit: Free XOR gates and applications. In *35th Intl. Colloquium on Automata, Languages, and Programming (ICALP), Part II*, volume 5126 of *LNCS*, pages 486–498. Springer, 2008.

[16] Y. Lindell and B. Pinkas. A proof of security of Yao's protocol for two-party computation. *Journal of Cryptology*, 22(2):161–188, 2009.

[17] Yehuda Lindell. A note on constant-round zero-knowledge proofs of knowledge. *Journal of Cryptology*, 26(4):638–654, 2013.

[18] Yehuda Lindell. Fast cut-and-choose-based protocols for malicious and covert adversaries. *Journal of Cryptology*, 29(2):456–490, 2016.

[19] Yehuda Lindell and Benny Pinkas. An efficient protocol for secure two-party computation in the presence of malicious adversaries. In *Advances in Cryptology—Eurocrypt 2007*, volume 4515 of *LNCS*, pages 52–78. Springer, 2007.

[20] Xiao Wang, Samuel Ranellucci, and Jonathan Katz. Authenticated garbling and efficient maliciously secure two-party computation. In *24th ACM Conf. on Computer and Communications Security (CCS)*, pages 21–37. ACM Press, 2017.

[21] Samee Zahur, Mike Rosulek, and David Evans. Two halves make a whole - reducing data transfer in garbled circuits using half gates. In *Advances in Cryptology—Eurocrypt 2015, Part II*, volume 9057 of *LNCS*, pages 220–250. Springer, 2015.