Efficient Publicly Verifiable 2PC over a Blockchain with Applications to Financially-Secure Computations

Ruiyu Zhu
Indiana University
Bloomington, IN
zhu52@iu.edu

Changchang Ding
Indiana University
Bloomington, IN
dingchan@iu.edu

Yan Huang
Indiana University
Bloomington, IN
yh33@iu.edu

ABSTRACT

We present a new efficient two-party secure computation protocol which allows the honest party to catch dishonest behavior (if any) with a publicly-verifiable, non-repudiable proof without sacrificing the honest party's secret. Comparing to the best existing protocol of its kind, ours requires a substantially simpler judge algorithm and is able to process circuit evaluator's input-wires two orders of magnitude faster. Further, we propose an automated, decentralized judge implemented as a blockchain smart-contract. As a killer application of combining our two-party PVC protocol with our decentralized judge, we proposed the concept of financially-secure computation, which can be useful in many practical scenarios where it suffices to consider rational adversaries. We experimentally evaluated our prototype implementation, demonstrated the 2PC protocol is highly efficient and the judge is very affordable to protect users against rational attackers.

CCS CONCEPTS

• **Security and privacy → Privacy-preserving protocols**; • **Theory of computation → Cryptographic protocols**.

KEYWORDS

secure computation, smart-contract, blockchain, rational adversary

ACM Reference Format:
Ruiyu Zhu, Changchang Ding, and Yan Huang. 2019. Efficient Publicly Verifiable 2PC over a Blockchain with Applications to Financially-Secure Computations. In *2019 ACM SIGSAC Conference on Computer and Communications Security (CCS '19), November 11–15, 2019, London, United Kingdom.* ACM, New York, NY, USA, 18 pages. https://doi.org/10.1145/3319535.3363215

1 INTRODUCTION

Secure computation is an important technique that has many real-world applications. It offers a practical way to replace a mutually trusted party with well-known mathematical hardness assumptions. Rigorous mathematical proofs assure all protocol participants that, except for a negligible probability, after running the protocol they cannot lose any extra information beyond what they expected and agreed a priori.

However, real-world use of secure computation protocols is hampered by the prohibitive cost of cryptography. This is especially the case for MPC protocols that need to withhold active attacks in settings without honest-majority. Typically, expensive, sophisticated cut-and-choose mechanisms are employed in these protocols, which incur a significant multiplicative factor (10–20x) overhead compared to the semi-honest version of those protocols.

The large performance overhead can be attributed to an overly pessimistic/simplistic threat model in the protocol design: (1) it assumes the adversary has nothing to lose hence can risk everything to break the security of these protocols; and (2) any leakage on the secret inputs is considered as rendering immeasurable losses. This, however, would not make sense in many realistic scenarios. For example, when two-parties play a gambling game with 2PC, an honest party's loss is clearly bounded by the amount he/she has already deposited. In business applications, the value of a company's sensitive data used in MPC can often be appraised with reasonable accuracy. In addition, the participants can be required to put down a security deposit that will be forfeited if cheating was caught, hence attack is not free. Unfortunately, current definition of security for MPC protocols doesn't account for these factors, but requires running the protocol with the worst scenario in mind: the good guy losses everything while the bad guy attacks for free.

In this work, we develop secure protocols that take the financial gain/loss of the attacker into account. We suggest a new notion of security for MPC protocols, called *financial security*, which can be useful in many business applications. We realized financially-security 2PC by combining a public verifiable covert (PVC) protocol with today's blockchain technology. In our design, the two parties will deposit some fund through smart-contract before running the 2PC protocol. Then, the PVC protocol is executed using properly configured (based on their deposits and the appraised value of the secret inputs) security parameters. In event of cheating being caught, a smart-contract transaction will be executed on behalf of the honest party to reclaim the attacker's deposit. In essence, the *judge* of the PVC protocol is a blockchain smart-contract.

Simple and intuitive as the basic idea above sounds to be, this work exposes several challenges that were not considered before in prior studies. First, there are no quantitative notion of financial gain/loss in existing formal models and definitions of secure computation. Second, the cost of the judge algorithm has to be sufficiently low so that it can actually run as smart-contract transactions over a well-established, reliable blockchain network.

Unfortunately, the complexity of the judge algorithm was not a focus in the design of all existing PVC protocols, which used a monolithic algorithm to carry out an enormous amount of computation. On the other hand, due to many practical constraints (see Section 2.2), smart-contracts are only capable of executing small-scale computations at affordable costs. Take the most recent PVC protocol [22] as an example, even for a simple AES circuit (6800 ANDs), the gas cost of the judge's verify will be at least $6800 \times 4 \times G_{\text{AES}}$ (merely for re-garbling the circuit) plus $128 \times (8 \times G_{\text{ECMULT}} + 5 \times G_{\text{ECADD}})$ (only to replay the 128 base PVW-OT, each of which requires 8 elliptic curve multiplications and 5 elliptic curve additions). On Ethereum, by far the most popular platform supporting smart-contracts, the gas cost of computing a single AES cipher, G_{AES}, is $340K$ (using a solidity implementation derived from the reference implementation of AES cipher in C). The gas cost of ECC multiplication and addition are $G_{\text{ECMULT}} = 40K$ and $G_{\text{ECADD}} = 0.5K$, respectively [40]. Hence, the total gas cost would amounts to at least 9.2 billion, which is worth roughly $7176 USD at the time of writing this paper! Even if the SHA3 EVM instructions are used in place of the expensive AES function for garbling (though sacrificing runtime performance by a factor of 10 or so), the base OT part alone will still require at least 41 million gas, which is way above the threshold (8 million gas) allowed in current Ethereum blocks. It would be theoretically possible to divide a long computation into multiple smaller chunks so that it can be included in multiple Ethereum blocks. However, in reality, such hacks are extremely impractical because of the heavy use of long-term storage (1000x more expensive than volatile memory) to keep intermediate values across different blocks, and also because of the need for uploading (linearly) many code chunks sliced from the dynamic instruction trace of the long computation.

1.1 Contributions

New Methodology. We propose the concept of *financial security.* A financially secure 2PC guarantees that the expected financial gain of a cheating adversary must be smaller than a preset threshold (which can be negative), where an adversary's expected financial gain is calculated from the appraised value of the honest party's input, the adversaries security deposit, and the cheating deterrence rate of the PVC protocol. Comparing to the conventional definition of security for 2PC, the new notion allows to reduce the amount of heavyweight cryptography based on a quantitative analysis on each party's stake in the 2PC protocol.

New protocol design and implementation. We propose a financially-secure 2PC protocol by combining public-verifiable covert (PVC) secure computation with an efficient, decentralized, automatic judge realized as an Ethereum smart-contract. We have overcome the tight budget of today's smart-contracts by designing a simpler PVC protocol (with a much simplified judge), as well as an efficient succinct proof search mechanism to further reduce the judge's cost. The proof searching process uses only logarithmic rounds in the size of the circuit and guarantees the honest party to obtain a constant-size irrefutable proof. As a side benefit, our judge doesn't need to know the circuit to verify the proof, hence is application-independent. We developed a prototype implementation of our protocol and experimentally demonstrated the substantial performance advantage of our protocol, and the practicality/affordability of penalizing a cheating adversary via smart-contracts, (therefore, it makes no sense for any rational attacker to ever try to attack).

As a part of this work, we proposed a cut-and-choose based PVC 2PC protocol. Our protocol features a significantly simpler judge that only needs to verify one garbled circuit (GC), one signed GC-hash, and one sender non-repudiable OT (consisting of verifying 3 ECMULT + 1 ECADD + two signed messages). In case of normal execution, our protocol is able to offer $1/n$-deterrence with only n calls to a *sender non-repudiable OT* (whose cost is comparable to the signed-OT) and a single call to a standard correlated-OT-extension protocol (containing only $l + s$ OTs where l is the evaluator's input length and s is the statistical security parameter), hence $O(n + l)$ bandwidth cost overall. In comparison, the best existing protocol [22] requires signing and verifying messages of length $O(n \cdot (l + s))$. Thus, our PVC protocol is not only simpler but also asymptotically more efficient, hence may be of independent interest.

1.2 Related Work

MPC with Covert Adversaries. Canetti and Ostrovsky [13] considered secure computation against *honest-looking* adversaries who may deviate from protocol specification only if the deviation *cannot* be detected. The current notion of covert secure computation was first formalized by Aumann and Lindell, who also presented a cut-and-choose-based solution in the two-party setting [7]. Their construction was then improved, and also extended to work in the multiparty setting without honest majority by Goyal et al. [19].

Publicly Verifiable Secure Computation. Merely *detecting* misbehavior, especially by a party who itself can be questionable, may not be enough to intimidate the attackers. So Asharov and Orlandi proposed the concept of *publicly verifiable covert* (PVC) secure computation and realized it using a signed-OT primitive [6]. A PVC protocol enables the honest party to prove to the public of the cheater's misbehavior without sacrificing its own secret input. Kolesnikov and Malozemoff presented a more efficient construction of PVC based on signed OT-extension [29].

In a recent concurrent work, Hong et al. provided the first efficient implementation of two-party PVC protocol [22]. Their protocol doesn't use signed OT-extension but requires the circuit generator to run instances of OT-extension protocol with the same random seeds used to generate the corresponding garbled circuits. In comparison, we developed a different mechanism to prevent selective-OT attacks which is even simpler and more efficient in both practical and asymptotic sense. Like their protocol, our PVC protocol of Figure 7 is also *defamation-free* (thus "non-halting detection accurate"). In addition, we focus on realizing the judge as automatic, decentralized smart-contracts. We also proposed financially-secure 2PC as an application of our PVC protocol, both of which are beyond the scope of their paper.

Secure Computation with a Blockchain. Research on leveraging Bitcoin network for MPC was initially focused on solving the fairness issues [3, 11, 27, 31]. The goal was to ensure that a party who aborts prematurely with the output should pay a monetary penalty to the honest parties. In addition, [3] also suggested linking MPC inputs and outputs to Bitcoin transactions to prevent cheating on MPC input and enforcing the MPC outcome, two issues out of the scope of the conventional MPC research. [33] proposed the secure cash distribution with penalty primitive that can handle many stateful secure computations such as auctions and games. Kiayias et al. [27] further formalized the interaction between MPC and Blockchain as standard ITMs (Interactive Turing Machines) and discussed the applicability of composition theorem in these scenarios. [32] proposed a method to achieve fairness for polynomially many instances of secure computations with low amortized cost. Researchers have applied and improved these techniques to enable decentralized poker as a killer application [12, 15, 31].

In contrast, our work focuses on leveraging blockchain smart-contracts to realize an affordable, automated, decentralized judge for public verifiable financially-secure computations.

Refereed Delegation of Computations. The idea of allowing individual independent parties to commit and contest each other deterministic computation was first proposed by Canetti et al. [14] and later popularized in the work of VerSum [38] and Arbitrum [24]. Similar to this work, these schemes can also resolve a dispute by producing a constant publicly verifiable proof of computational faults in logarithmic rounds. However, they used this idea for verifiable computation purpose, hence the focus was on a generic architecture to allow verifying arbitrary computations. Contrasting with these works, we target at a specific task (garbling) in the context of improving GC-based secure computation protocols, which allows us to consider a much simpler but more efficient execution model.

Rational Cryptography. There have been a large body of research papers on rational cryptography [4, 17, 18, 21, 28]. Although those works also assumed and exploited that the adversaries have certain "rational", our work is very different from those works in many aspects of the problem context: (1) Our focus is on improving the performance of GC-based cut-and-choose protocols for two-party secure computation in presence of rational adversaries; whereas those works were about MPC protocols based on rational secret sharing; (2) We reveal and address the practical challenges in realizing an automated and decentralized judge using smart-contracts over a real-world blockchain network; (3) In our model, the adversary is simply rationalized to weigh the legitimate selling price of an honest party's input data against the expected penalty of caught cheating, whereas prior works were about a completely different assumption that the adversary is interested in learning the secret inputs of the other players while preventing others from learning their own secrets; (4) We don't consider the fairness problem (but several existing blockchain-based fairness solutions can be used with our protocol to address the concern); whereas fairness is a main goal in most of those prior works.

2 PRELIMINARIES

We present some useful building blocks that appear in our solutions. A list of common variables is summarized in Figure 1.

2.1 Incremental Collision-Resistant Hash

Incremental collision-resistant hash was first proposed by Bellare et al. [8] to speed up cryptographic processing of a file undergoing active updates, or many large files whose differences are small. The key idea was that if a file has already been processed, then another file similar to the processed one could be finished with a cost that only depends on the size of the difference (as opposed to the entire file).

s	The statistical security parameter. Typically, $s = 40$
κ	The computational security parameter. Typically $\kappa = 128$
n	The total number of GCs involved in cut-and-choose.
l	The length of each party's input and the circuit output, i.e., we assume $f : \{0,1\}^l \times \{0,1\}^l \mapsto \{0,1\}^l$.
$[\![\cdot]\!]_\cdot$	For a party P and a message m, $[\![m]\!]_P \overset{\text{def}}{=\!=} (m, \mathsf{Sign}_{pk_P}(m))$. When $[\![m]\!]_P$ is treated as a non-repudiable claim of the value of m by P, we implicitly assume $\mathsf{Sign}_{pk_P}(m)$ is a signature over a non-ambiguous claim on the value of m including necessary meta-data.
R_i	The XOR difference between a wire's 0-label and 1-label in the i^{th} GC.
$\mathsf{wid}_\mathsf{x}(j)$	The index of the wire associated with x_j, the j-th bit of P_1's input x.
$\mathsf{wid}_\mathsf{y}(j)$	The index of the wire associated with y_j, the j-th bit of P_2's input y.
$\mathsf{wid}_\mathsf{r}(j)$	The index of the wire associated with r_j, the j-th bit of P_2's augmented input r.
$\mathsf{wid}_\mathsf{z}(j)$	The index of the wire associated with z_j, the j-th bit of the output z.
f, f_i	f is the circuit description file; f_i is the i^{th} line of f.
F, F_i	F is the garbled circuit of f; F_i is the i^{th} garbled gate in F.
IH_i^M	The incremental hash of the first i blocks of a message M.
IH_i^f	The incremental hash of the first i lines of the circuit file f.
IH_i^F	The incremental hash of the first i garbled gates of the GC F.
$\{\!\{\cdot\}\!\}$	$\{\!\{v\}\!\}$ denotes a pair of signed values $([\![v_1]\!]_{P_1}, [\![v_2]\!]_{P_2})$ where v_1 and v_2 are, resp., P_1's and P_2's non-repudiable claims on the value of the same variable v.
IDENTICAL	IDENTICAL($\{\!\{v\}\!\}$) evaluates to whether $v_1 = v_2$.
PEER	PEER($\{\!\{v\}\!\}$) evaluates to v_1 if it was P_2 calling the function and v_2 if it was P_1 calling.

Figure 1: Table of Notations.

We will use a restricted version of incremental collision-resistant hash that only considers "append" operation. Thus, we define an incremental collision-resistant hash scheme $\mathcal{H}$ as a tuple of efficient algorithms $(Gen, H, IncH)$ where (Gen, H) constitutes a collision-resistant hash scheme, and for all $M' = M\|\delta$, and $s \leftarrow Gen(1^\kappa)$ for computational parameter κ,

$$H^s(M') = IncH^s(H^s(M), \delta)$$

and the cost of $IncH$ is only polynomial in the length of δ.

In this paper, we show how incremental collision-resistant hash and the hash-then-sign method can be applied in a fresh context, to allow an honest party to efficiently rebut a false claim from its dishonest peer about a computation. The rebuttal has only logarithmic round in the size of the computation and generates a proof that costs only constant computation for a judge to verify.

2.2 Blockchain and Smart-Contracts

A blockchain is an open, decentralized ledger system operated by volunteers all over the world connected by the Internet. The consensus on records in the ledger is reached either through proof-of-work, proof-of-stake, or a mix of both. When its ledger is used to record balances of accounts, a blockchain essentially realizes a cryptocurrency. Bitcoin [1, 34] and Ethereum [2, 40] are two most valuable blockchains in market capitalization.

A major advantage of Ethereum over Bitcoin is its support of smart-contracts, which allows to run Turing-complete computations specified by EVM (Ethereum Virtual Machine, a stack-based RISC architecture) assemblies to manage its transactions. However, the support of Turing-complete smart-contracts comes with a heavy burden to Ethereum volunteers, e.g., to permanently remember its source code and to re-run the same computation by all Ethereum miners who are actively mining (and also when establishing a new Ethereum node). To limit the impact of smart-contracts on Ethereum's throughput and scalability, EVM introduced a stringent pricing scheme [40] on every instruction and storage (short-term or long-term) used by its computations. For example, it charges 21000 gas base rate plus 68 gas/byte for each transaction and additional 32000 gas for contract creation, 5000 gas/word (1 word = 32 bytes) for modifying long-term variables, 3 gas to perform an addition, 30 gas plus 6 gas/word for SHA-3, and 3 gas/word for short-term storage. To execute a smart-contract transaction, a user must set enough fund aside to drive the smart-contract execution till the fund depletes.

When a transaction is submitted, a *gas-price* needs to be specified, e.g., 4×10^9 wei/gas which allows the transaction to be included within 2 minutes, indicating the user is willing to pay 4 Gwei per gas spent in executing this transaction. The higher gas-price one offers, the quicker it gets included on the chain. Since 1 Ether is 10^{18} wei and 1 Ether is worth \$195 USD at the time of writing this paper, 1 gas converts to 7.8×10^{-7} USD. We note that both the gas-price and Ether/USD conversion rate fluctuate over time.

2.3 Garbled Circuits

In this work, we use a slightly modified definition of secure garbling scheme from the one proposed by Bellare et al. [10] (and enhanced by Frederiksen et al. [16] for verifiability), to emphasize the use of *seed*s and *determinism* of the garbling algorithm. We define garbling scheme $\mathcal{G}$ as a tuple of efficient deterministic algorithms (Gb, Ev, En, De, Ve) that have the following syntax:

- $X := \mathsf{En}(e, x)$ runs the *encoder* En with encoding information e and an input x to produce X (i.e., the encoding of x);
- $(F, e, d) := \mathsf{Gb}(1^\kappa, f, \mathit{Seed})$: runs a deterministic *garbler* Gb on input function f and a uniform *Seed* to produce a garbled circuit F, the encoding and decoding information e, d (where κ is a computational parameter);
- $Y := \mathsf{Ev}(F, X)$ runs the *evaluator* Ev on garbled circuit F and encoded input X to produce the encoded output Y;
- $y := \mathsf{De}(d, Y)$ runs *decoder* De with decoding information d on encoding Y to decode Y into plaintext y.
- $b := \mathsf{Ve}(f, \mathit{Seed}, F, d)$ runs the *verifier* Ve and outputs a bit indicating if F and d are indeed the outputs of $\mathsf{Gb}(1^\kappa, f, \mathit{Seed})$.

Note that unlike the syntax used by Frederiksen et al. [16], we don't need e to be an input to Ve because in our protocol e will never be given to the evaluator and can be deterministically reproduced from *Seed*. Interesting properties of garbling include correctness, privacy, obliviousness, authenticity, and verifiability.

- *Correctness:* For all f, all $(F, e, d) := \mathsf{Gb}(1^\kappa, f, \mathit{Seed})$ and all x,
$$\mathsf{De}(d, \mathsf{Ev}(F, \mathsf{En}(e, x))) = f(x).$$
- *Privacy:* There exists an efficient $\mathcal{S}$ so that for all f and all x,
$$\left\{(F, X, d) \;:\; \begin{array}{c}(F, e, d) := \mathsf{Gb}(1^\kappa, f, \mathit{Seed}) \\ X := \mathsf{En}(e, x)\end{array}\right\} \approx \left\{\mathcal{S}(1^\kappa, f, f(x))\right\}.$$
- *Obliviousness:* There exists an efficient $\mathcal{S}$ so that for all f and x,
$$\left\{(F, X) \;:\; \begin{array}{c}(F, e, d) := \mathsf{Gb}(1^\kappa, f, \mathit{Seed}), \\ X := \mathsf{En}(e, x)\end{array}\right\} \approx \left\{\mathcal{S}(1^\kappa, f)\right\}.$$
- *Authenticity:* For all efficient adversary $\mathcal{A}$, all f and all x:
$$\Pr\left(\begin{array}{c} Y \neq \mathsf{Ev}(F, X) \\ \textbf{and} \\ \mathsf{De}(d, Y) \neq \bot \end{array} \;:\; \begin{array}{c}(F, e, d) := \mathsf{Gb}(1^\kappa, f, \mathit{Seed}) \\ X := \mathsf{En}(e, x) \\ Y \leftarrow \mathcal{A}(1^\kappa, f, F, X)\end{array}\right) \leq \mathbf{negl}(\kappa).$$
- *Verifiability:* For all efficient adversary $\mathcal{A}$, all f and all x,
$$\Pr\left(\begin{array}{c}\mathsf{De}(d, \mathsf{Ev}(F, \mathsf{En}(e, x))) \neq f(x) \\ \textbf{and } \mathsf{Ve}(f, \mathit{Seed}, F, d) = 1\end{array} \;:\; (F, e, d, \mathit{Seed}) \leftarrow \mathcal{A}(1^\kappa, f)\right) \leq \mathbf{negl}(\kappa).$$

When applying a garbling scheme for two-party secure computation purpose, one party, called the circuit generator, will run the Gb function and send the garbled circuit F to the other party, known as the circuit evaluator. In addition, the circuit generator will run En to translate its own plaintext input into their encodings that are also known as wire-labels and send them to the evaluator. The parties will run oblivious transfer protocols for the evaluator to obtain the encodings of its own plaintext input. Once all circuit input encodings are available at the evaluator's end, the evaluator will run the Ev algorithm to obtain Y, the encoding of the output. Finally, depending on who will learn the plaintext result, Y will be either decoded to plaintext by the evaluator, or sent back to and decoded at the circuit generator's side.

In presence of adversaries who could deviate from protocol specification (which will allow a malicious generator to learn extra information of the evaluator), a common cryptographic treatment, called *cut-and-choose*, is to require the generator to produce multiple garbled circuits using different seeds for the same public function f, so that some random garbled circuits will be checked to gain confidence over the circuit generator's honest behavior and only the rest are evaluated for computing the result. Note that garbled circuits are typically very large. Hence, to save bandwidth, usually only short collision-resistant hashes of the garbled circuits are transmitted for verification purpose [19]. In our case, the syntax of Ve becomes $b := \mathsf{Ve}(f, \mathit{Seed}, h, d)$, and the inequality in the definition of verifiability becomes,

$$\Pr\left(\begin{array}{c}\mathsf{De}(d, \mathsf{Ev}(F, \mathsf{En}(e, x))) \neq f(x) \\ \textbf{and } h = H(F) \\ \textbf{and } \mathsf{Ve}(f, \mathit{Seed}, h, d) = 1\end{array} \;:\; (F, e, d, \mathit{Seed}) \leftarrow \mathcal{A}(1^\kappa, f)\right) \leq \mathbf{negl}(\kappa).$$

where $h = H(F)$ with H being a collision-resistant hash (and optionally, incremental only if generating succinct proof of misbehavior is desired).

In practice, garbling schemes can be efficiently implemented using hardware-accelerated assembly instructions either in the random oracle model [9, 41] or under standard assumptions only [20].

2.4 Oblivious Transfer

Oblivious transfer is an important enabling primitive for MPC protocols. A composable actively-secure string OT, which allows an OT receiver to obliviously select one out of two messages from the OT sender, can be built from various computational hardness assumptions such as DDH and lattice LWE [36]. To avoid expensive public-key operations, OT-extension protocols were proposed to amortize the cost of a few expensive public-key operations over polynomially many OT instances, first in the semi-honest model [23], then in the malicious model [5, 26]. In this work, we use a variant of OT-extension called Correlated OT-extension (see Figure 5 and Figure 2) first seen used in [35] for the same purpose of preventing selective failure attacks on the evaluator's inputs.

3 APPROACH OVERVIEW

We summarize the key ideas of the PVC protocol (Section 3.1), the short proof generation and verification mechanism (Section 3.2). We will conclude this section with a brief, intuitive explanation of the financial-security threat model (Section 3.3).

3.1 Publicly Verifiable Covert 2PC

Our PVC protocol is based on cut-and-choosing garbled circuits (GC) but lifts most of the heavyweight due to cut-and-choose by considering only covert adversaries. A covert attacker will cheat only if cheating is caught with a probability less than certain threshold, say $1/n$. So it suffices to generate n GCs, open $n - 1$ of them for verification purpose and evaluate only one GC. If any GC fails the verification, the honest party obtains a non-repudiable proof than can be checked by the rest of the world without risking its own secret input. Nevertheless, the cut-and-choose-based PVC protocol involves multiple garbled circuits and is still subject to some consistency-related attacks. For example, to guess about the evaluator's secret input, a malicious circuit generator could run the OT with some input wire-labels different from those produced by garbling. This is also known as the selective failure attacks on OT inputs, or selective-OT attacks for short.

Correlated-OT + Lightweight Circuit Augmentation. To thwart such attacks, our PVC protocol (Figure 7) only makes a *single* call the standard correlated-OT $\mathcal{F}_{\mathsf{CorrOT}}$ (Figure 5) to translate the circuit evaluator's input y into input wire-labels on the (only) evaluation-GC (Step (4) of Figure 7). As Figure 2 shows, $\mathcal{F}_{\mathsf{CorrOT}}$ can be efficiently realized by slightly modifying Keller-Orsini-Scholl's actively-secure OT-extension protocol. In our protocol, no signatures are needed on any part of the transcript of running this correlated-OT. Instead, the evaluator only needs to check the consistency of a few values obtained from $\mathcal{F}_{\mathsf{CorrOT}}$ and simply abort in case any check fails (Step (4)). A key insight behind this design is that it is okay to not accuse a cheating correlated-OT sender's misbehavior during OT if this misbehavior will either lead to an abort

(1) **(Base OTs)** R picks $(\kappa + s)$ pairs of κ-bit uniform seeds $\{(\mathbf{k}_0^i, \mathbf{k}_1^i)\}_{i\in[\kappa+s]}$. S uniformly picks a bit matrix $H \in \{0,1\}^{\kappa\times(\kappa+s)}$ and a string $\Delta' \in \{0,1\}^{\kappa+s}$ such that $\Delta = H\Delta'$, where Δ is S's input to the correlated OT, and H has rank κ. For all $i \in [\kappa + s]$, R and S call $\mathcal{F}_{\mathsf{OT}}$ as the sender and receiver, respectively, so that S learns $\mathbf{k}^i_{\Delta'_i}$.

(2) **(OT Extension)** R computes $\forall i \in [\kappa + s]$, $\mathbf{T0}^i := \mathsf{PRG}(\mathbf{k}_0^i)$, $\mathbf{T1}^i := \mathsf{PRG}(\mathbf{k}_1^i)$, $\mathbf{u}^i := \mathbf{T0}^i \oplus \mathbf{T1}^i \oplus \mathbf{x}'$ where $\mathbf{T0}^i, \mathbf{T1}^i, u^i, \mathbf{x}' \in \{0,1\}^{l+\kappa+s}$ and $\mathbf{x}'$ is formed by appending $\kappa + s$ random bits picked by R to its own l-bit OT choice vector $\mathbf{x}$. R sends $\{\mathbf{u}^i\}_{i\in[\kappa+s]}$ to S. S computes $\mathbf{T}^i := \mathsf{PRG}(\mathbf{k}^i_{\Delta'_i}) \in \{0,1\}^{l+\kappa+s}$ and $\mathbf{q}^i := \mathbf{T}^i \oplus \Delta'_i \mathbf{u}^i$. Let $\mathbf{q}_j$ be the j^{th} row of the $l\times(\kappa+s)$ bit matrix $[\mathbf{q}^1, \ldots, \mathbf{q}^{\kappa+s}]$, and $\mathbf{T0}_j$ be the j^{th} row of the $l\times(\kappa+s)$ bit matrix $\mathbf{T0} = [\mathbf{T0}^1, \ldots, \mathbf{T0}^{\kappa+s}]$.

(3) **(Correlation Check)** S picks $l + \kappa + s$ uniform $(\kappa + s)$-bit vectors $\{\chi_i\}_{i\in[l+\kappa+s]}$ and sends them to R. R sends $\chi := \sum_{j=1}^{l+\kappa+s} \mathbf{x}'_j \chi_j$ and $t := \sum_{j=1}^{l+\kappa+s} (\mathbf{T0}_j \odot \chi_j)$ (where $\odot$ is bitwise AND of two equal-length strings) to S. S computes $q := \sum_{j=1}^{l+\kappa+s} (\mathbf{q}_j \odot \chi_j)$ and verifies $t = q + \chi\Delta'$.

(4) **(Compress and Output)** S sends H, $\{\mathbf{v}_j := m_j \oplus H\mathbf{q}_j\}_{j\in[l]}$ to R and outputs $\perp$. R outputs $\{m_{x_j} := \mathbf{v}_j \oplus H\,\mathbf{T0}_j\}_{j\in[l]}$.

Figure 2: Realize $\mathcal{F}_{\mathsf{CorrOT}}$ in the $\mathcal{F}_{\mathsf{OT}}$-hybrid model.

or eventually be accused in a later stage with a probability greater than what the adversary can tolerate, all in a way independent of the honest party's input. This is indeed the case in our protocol: (1) errors in any of the GCs will be caught with an irrefutable proof with probability greater than $1 - 1/n$; (2) selective failure attacks on input wire-labels to the correlated-OT will cause protocol to abort except for a negligible probability, because correlated-OT guarantees that every pair of the sender's input wire-labels to the OT has to share the same XOR-difference.

One may still worry about a malicious correlated-OT sender's ability to guess the OT receiver's input by calling $\mathcal{F}_{\mathsf{CorrOT}}$ with a corrupted $R_{\mathcal{I}}$ but with some 0-labels in $\{L^0_{\mathcal{I},\mathsf{wid}_\mathsf{y}(j)}\}_{j\in[l]}$ carefully crafted such that $\left\{L^{\hat{y}_j}_{\mathcal{I},\mathsf{wid}_\mathsf{y}(j)}\right\}_{j\in[l]}$ are all valid wire-labels (where $\hat{y}_j$ denotes the attacker's guess of y_j). This attack would be particularly worrisome when l is small or the entropy in y is low so it is easy to guess y. To this end, we introduced a preparation step (Step (0) of Figure 7) where the function f is augmented into f', which has s (a small statistical security parameter, e.g., 40) auxiliary input-wires for input r (whose value will be picked uniformly by the evaluator) and also treats the wires of y and r as evaluator's output-wires (so ordered-pairs of wire-label hashes on these wires are sent to the evaluator as part of the decoding information). After the correlated-OT is done, the evaluator will ensure that all wire-labels representing y and r obtained from $\mathcal{F}_{\mathsf{CorrOT}}$ are consistent with the decoding information on these wires. This lightweight mechanism reduces the success rate of the aforementioned attack to 2^{-s} (as $r \in \{0,1\}^s$). We remark that the decoding information on the wires of y is also indispensable, because otherwise an attacker can easily flip the 0-label and 1-label on a bit of y without ever being detected. Moreover, it is important to augment f for all garbled circuits (as opposed to only the evaluation-circuit), because otherwise there is no guarantee that the decoding information on those output-wires is valid (if it could never be checked).

Finally, we note that the correlated-OT used for translating the evaluator's secret input bits are executed only *after* the garbled circuits are sent. This allows the OT to be done only once as compared to n times in prior protocols, and dismisses the need to sign and examine the transcript of the correlated-OT. However, it poses a challenge in proving the protocol secure: a simulator will not be able to learn $f(x, y)$ at the time of circuit garbling any more. To obtain a rigorous proof of security, we introduced l "RIGHT" gates, ►, in f', which always returns its right input, to derive $f(x, y)$ as x ► $f(x, y)$. A ► gate is implemented using $\wedge$ and $\oplus$: a ► $b = ((a \oplus b) \wedge \overline{a \oplus b}) \oplus b$. These seemingly redundant ► gates helps in our security proof: For the case when the evaluator is corrupted, it allows the simulator $\mathcal{S}$ to garble the evaluation-circuit with all ►s replaced by "LEFT" gates, ◄, which always return its left input. The fact that these replacements can't be noticed by P_2 depends on not only the privacy and obliviousness properties of secure garbling, but also the fact that $\mathcal{S}$ only needs to obliviously modify the garbled rows of the AND gate (by flipping the right input before AND-ing) in every ► to turn a ► into a ◄.

Sender Non-Repudiable OT. We use seed-based deterministic garbling and use n instances of OT to allow the evaluator to obliviously select $n - 1$ GCs as check-GCs whose seeds are obliviously transferred. To ensure a cheating circuit generator be indicted for sending corrupted GCs, we use *Sender Non-Repudiable OT*, $\mathcal{F}_{\text{SNR-OT}}$, to transfer the GC seeds. $\mathcal{F}_{\text{SNR-OT}}$ provides conceptually the same functionality as signed-OT of [6], except that $\mathcal{F}_{\text{SNR-OT}}$ is defined completely through an ideal two-party *stateful* functionality (see Figure 4). This approach allows a cleaner description without relying on the less standard notion of EU-CMRA (Existentially Unforgeable under Chosen Message and Randomness Attacks) signatures as required by [6] or EU-CMPRA (Existentially Unforgeable under Chosen Message and Partial Randomness Attacks) signatures as in [29]. Note that the **Verify** function of $\mathcal{F}_{\text{SNR-OT}}$ is an *authorized* interface that requires *no interaction.* We call it authorized because it only responds non-trivially to queries that are signed (thus permitted) by the OT receiver. This authorization requirement is critical security-wise as it can prevent an OT sender to learn the receiver's choice by simply querying the **Verify** function. It is also important for the **Verify** function to be non-interactive: we want a judge to be able to verify an OT receiver's claim without any help from a potentially malicious OT sender (who can refuse to assist in the proof). Finally, it is easy to see that by definition, there is no chance that a malicious OT receiver can defame an honest sender by tricking **Verify** to output 1 on some (b', m') different from those actually used in the corresponding **Execute-OT**. Our realized $\mathcal{F}_{\text{SNR-OT}}$ with a protocol almost the same as that of signed-OT [6], except that we only use a standard EU-CMA signature scheme. We describe and prove our sender non-repudiable OT protocol in Section 4.1.

The Judge. Figure 8 describes the judge's verification procedure. To verify that a circuit generator has sent inconsistent $Seed_\imath$ and $h_\imath$, it suffices for the judge to verify that $Seed_\imath$ indeed comes from the $\imath^{th}$ OT but $Seed_\imath$ cannot produce the $d_\imath, h_\imath$ that was received earlier from P_1. The simplicity of this judge fits squarely in our plan to run the judge as affordable smart-contracts. To the best of our knowledge, this is the leanest judge for a PVC protocol by far.

Comparison with [6], [29], and [22]. Here we restrain the comparison to the cost of processing evaluator's inputs. Both [6] and [29] employed the wire-splitting technique, except that [6] used $(l \cdot n^2)$ instances of PVW-based signed-OT whereas [29] is more efficient because it uses n instances of signed-OT-Extension, each of which contains $(l \cdot n)$ OTs. Most recently, [22] did not use wire-splitting but proposed to let the OT sender to run each of the n instances of OT-extension (each contains l OT) with the same randomness used to generate the corresponding GC and sign its entire transcript. So it amounts to $O(nl + ns)$ bandwidth overall due to an $O(s)$ additive overhead from OT-extension. This, however, leaves the burden of replaying the entire the OT-extension to not only the evaluator, but also the judge. In contrast, our PVC protocol uses only one instance of the standard OT-extension-based correlated-OT, which boils down to $O(l + s)$ total bandwidth, and requires no signing and verifying the transcript of the correlated-OT. And our judge is completely relieved of verifying anything about the correlated-OT.

[6]	[29]	[22]	Ours
$O(l \cdot n^2)$	$O((l + s)n^2)$	$O(nl + ns)$	$O(l + s)$

Figure 3: Bandwidth cost for thwarting selective-OT

3.2 Affordable Judge and Succinct Proofs

Even though our judge has been greatly simplified, the tight budget of smart-contracts would still make it infeasible to execute the judge (whose cost is linear in circuit size because of the Ve algorithm) as a monolithic computation on Ethereum. In this work, we propose an efficient protocol for the honest party to derive a constant-size, non-repudiable proof of the adversary's misbehavior that cost the judge

Public Input: The sender's and receiver's public keys pk_S, pk_R.
Execute-OT:
(1) Receive (tag, m_0, m_1) where $m_0, m_1 \in \{0,1\}^n$ from S, and (tag, b) where $b \in \{0,1\}$ from R.
(2) Store (tag, b, m_b) and send m_b to R.
Verify: Upon receiving $[\![tag', (b', m')]\!]_R$, return 1 *iff*

$$\textbf{Verify}_{pk_S}\left([\![tag', (b', m')]\!]_R\right) = 1, tag' = tag, b' = b, m' = m_b.$$

Figure 4: The Ideal Sender Non-Repudiable OT $\mathcal{F}_{\text{SNR-OT}}$.

Execute-OT: Upon receiving $\left(\Delta, \{m_i\}_{i\in[l]}\right)$ where $\Delta, m_i \in \{0,1\}^\kappa \in \{0,1\}^\kappa$ from S and $\mathbf{x}$ where $\mathbf{x} \in \{0,1\}^l$ from R, send $\{m_i \oplus \mathbf{x}_i\Delta\}_{i\in[l]}$ to R.

Figure 5: The Correlated OT functionality $\mathcal{F}_{\text{CorrOT}}$.

constant gas to verify. We emphasize that by "proof", we always refer to a *proof of misbehavior of the adversary*. It is important that the proof generation protocol is *complete* and *sound*. Informally, completeness means that the honest party can always obtain such a short proof, and soundness means that a dishonest party can never derive a valid proof to defame the honest party.

We observe that in our PVC protocol (Figure 7), the only point when cheating is considered being *caught* is in Step (3). At that point, an honest evaluator must have discovered that $\mathrm{Ve}(f', Seed_\imath, d_\imath, h_\imath) = 0$, where $Seed_\imath$ comes from the $\imath^{th}$ non-repudiable OT and $(d_\imath, h_\imath)$ were received earlier at the end of Step (2). We note all the checks in Figure 8 are deterministic functions over $(f', Seed_\imath, d_\imath, h_\imath)$ whose values come with non-repudiable evidences: (1) f' has been signed in Step (0); (2) value of $Seed_\imath$ is irrefutable thanks to $\mathcal{F}_{\textsf{SNR-OT}}$; (3) non-repudiation of $d_\imath, h_\imath$ received in Step (2) is backed by the EU-CMA signature scheme. To reduce the judge's work in verifying the checks, especially the one involving computing Ve, we require the two parties to compute the checks *for* the judge and contesting each other along the way with non-repudiable claims about the intermediate states of the computation. Moreover, they hash-then-sign the computation using incremental cryptography. Therefore, once they use binary search to find the first intermediate state that they disagree upon, the honest party can submit the adversary's claim about the previous state (which has to be true) and the adversary's claim about the current state (which is false) to the judge. The judge only need to replay a single step to decide if cheating has occurred in the PVC protocol.

As a warmup, consider the case when the parties had a dispute on the value of a very long message consisting of n blocks. For example, let $\{m_i\}_{i=1}^n$ be an n-block message where m_i denotes its i^{th} block, and initially the parties agree on the incremental hash of the entire message, i.e., $\mathit{IH}_n = \mathit{IH}(\{m_i\}_{i=1}^n)$, and each party has IH_n signed by its peer. If at some point the attacker claimed a value $\tilde{\mathit{IH}}_k \neq \mathit{IH}_k$, then they will contest each other on the value of $\mathit{IH}_{(k+n)/2}$. Since the adversary cannot provide a value of $\mathit{IH}_{(k+n)/2}$ that is consistent both with its own claims of IH_k and IH_n, they will continue this contesting process until they end up with an $\imath$ such that they agree on $\mathit{IH}_{\imath+1}$ but not $\mathit{IH}_\imath$. At this point, the honest party already obtained a non-repudiable proof, i.e., the adversary's non-repudiable claims on $\mathit{IH}_\imath$ and $\mathit{IH}_{\imath+1}$. Thanks to the collision-resistance of the hash, the attacker cannot find a presage of $\mathit{IH}_{\imath+1}$. This proof can be verified by a single call to IncH: $\mathit{IH}_{\imath+1} = \mathit{IncH}(\mathit{IH}_\imath, m_{\imath+1})$, whose cost is a constant. It is easy to see that this binary search can produce such a proof in $O(\log n)$ rounds for disputes on static messages.

The idea also applies to resolving disputes on dynamic, deterministic computations such as $\mathrm{Ve}(f, Seed, d, h)$. However, the procedure turns out to be much more complicated due to some subtle issues. We model the garbled circuit as a sequence of garbled ANDs and XORs whose order and wire-connections are determined by the circuit file f', whose content has been incrementally hashed and signed. Both parties should have also incrementally hashed the sequence of garbled gates in order. To generate a short proof by contesting the execution of Ve, the parties must have already agreed on the values of f' (otherwise the PVC won't even start), *Seed* (otherwise, a proof of misbehavior would have obtained from $\mathcal{F}_{\textsf{SNR-OT}}$), and d, h (otherwise, a proof would simply be the signed message $[\![d, h]\!]_{P_1}$ received in Step (2)). That being said, they will begin with searching for the first garbled gates whose contents they disagree. Once this gate is found, say with index $\imath$, they need to declare values of the two input wire-labels, the garbled rows, the output wire-label, and the circuit file description of this gate. Depending on the adversary's claim, the honest party will sequentially check the following cases and must end up in one of them. For the first two cases, the honest party easily obtains a succinct proof.

(1) This particular gate was garbled wrong.
(2) The hash of the garbled rows and output wire-label was wrong.
(3) They disagree on the input wire-labels. In this case, they have already agreed on the incremental hash of all garbled gates up to gate-$(\imath - 1)$ (because gate-$\imath$ is the first garbled gate they disagreed). Since the sequence of garbled gates are in topological order (thus the input wire-labels must have appeared as output wire-labels of a previous gate), a proof of misbehavior can be produced using the binary search method explained earlier using the hash of the sequence of garbled gates.
(4) They disagree on the circuit description of this gate. In this case, since the description is just one line in the circuit file, a proof can thus be generated with the binary search method using the hash of the circuit file.

For situation (1) described above, one might worry that verifying the correctness of a garbled gate can still be considered too expensive. Indeed, re-garbling an AND costs at least $340\text{K} \times 4 = 1.36\text{M}$ gas. However, we note that it is easy to subdivide the work of garbling an AND into 4 AES calls, and further into 10 basic AES rounds. Therefore, with 6 more rounds of contesting to zoom into the AND garbling, we can further reduce the judge's effort to one basic AES round, which costs only 34K gas.

It is not hard to prove by induction that if the adversary cooperates in running the proof search procedure to the end, then the honest party will always obtain a constant proof of misbehavior in logarithmic rounds and the proof can be verified by the judge using constant resource. Nevertheless, an adversary can be uncooperative, either by staying silent or through sending meaningless replies, leaving the honest party unable to derive a short proof alone. We finally tackle these two situations as below.

What if the adversary stays silent and refuses to cooperate? We resolve this issue by requiring the parties to deposit certain amount of fund in advance and always communicate through an Ethereum smart-contract transaction, called `sendmsg`, which takes a party's messages as input, ensures the sequence numbers of consecutive messages are sequentially incremented, and records the identity of the most recent replying party. If the adversary keeps silent for more than a threshold $n_{timeout}$ number of blocks, the honest party can take the silent adversary's security deposit by invoking the judge with a `Timeout` proof. `sendmsg` transactions are very cheap: it only keeps two variables and checks one integer field (the sequence number), and do nothing about its input otherwise. The inputs to `sendmsg` will be uploaded on the blockchain, which is essentially a public, sequenced, authenticated channel between the two parties. Therefore, an adversary who refuse to cooperate with the judge cannot gain any financial advantage.

Handling Adversary's Non-sense Replies. To reduce cost, `sendmsg` doesn't really examine the contents of the replies, thus cannot decide whether a reply "makes sense". To address this issue, we formalized the proof search procedure as a finite state automaton with transitions well defined over the responses (also called *records*) from both parties. Each state of this automaton must be entered with a valid record of multiple key-value pairs, where the validity of a record is defined with respect to the state and the set of keys in the record, e.g., a valid record for state 1 must contain exactly the set of keys named as key_1, key_2, key_3. So the proof search procedure can be encoded as transition rules of this FSA. Once we turn each of the transition rules into a small transaction that checks the validity of a message based on information of the two states the rule connects, any meaningless message from the adversary immediately becomes a proof of misbehavior that can be verified by the judge.

DoS Attacks. Due to the use of blockchain, an honest party running our protocol would be subject to DoS attacks. This is a vulnerability generally suffered by all other blockchain-based dispute resolving schemes. On the other hand, a DoS attack is relatively hard to launch due to the open design of public blockchain networks. Typically, the parties are recommended to reserve backup channels to access the blockchain only in case of emergency. Since the backup channels are not exposed, it is harder to block them beforehand.

3.3 Financially-Secure 2PC

The net effect of executing a two-party PVC protocol can be modeled by a zero-sum game — what the adversary expects to gain (from cheating) is exactly what the honest party would lose.[1] Intuitively, a secure two-party computation is *financially-secure* if, regardless of the adversary's behavior, comparing to a (simulated) adversary in the ideal world, the real-world adversary cannot gain any extra financial advantage. This security notion will be useful in working with *rational* adversaries, for whom it makes no sense to attack if the expected extra gain from cheating is negative.[2]

Backed by today's blockchain networks like Ethereum, we realize financially-secure 2PC by requiring the party who runs as the circuit generator to make a security deposit d_{gen} before starting the PVC protocol.[3] If it is caught on cheating in the protocol execution, the honest party simply invokes the smart-contract (judge) with the proof it obtained during the protocol execution. Once verified, the deposit d_{gen} will be transferred to the honest party. Let the asset of the evaluator's input data be v_{ev} and assume our PVC protocol configured at $(1 - 1/n)$-deterrence is used. Then the 2PC is financially-secure if $v_{ev}/n - d_{gen} \cdot (1 - 1/n) \leq 0$. I.e., it is easy to achieve financial-security by setting $n \geq 1 + v_{ev}/d_{gen}$. For example, if the evaluator's input is worth \$1000 (i.e., he/she is willing to sell its data for \$1000) and the generator is willing to put down \$100 deposit, it suffices to generate 11 copies of GCs in the PVC protocol to guarantee financial-security. Compared with the concept of covert security, financial security enables direct and quantitative justification on the choice of the deterrence parameter n.

[1] We will not need to consider smart-contract fees when analyzing this zero-sum game (see the discussion on smart-contract fees).

[2] Even if the expected profit is negative, some adversary may still choose to go against the odds hoping to profit from deviation. However, such adversary is no longer considered rational. Analyzing their behavior involves modeling their risk preference, which is out of the scope of this paper. However, the concern can be alleviated by setting the threshold of the adversary's expected gain to a negative number.

[3] In our PVC protocol, the circuit evaluator has no way to cheat for a profit, so no deposit from them is needed for this purpose. However, each party needs to deposit a separate d_{fee} (referred as Type-II fee below) to cover the potential transaction fees in case the judge is needed to resolve a dispute later.

A remark on the threat model. We consider scenarios where each party knows how much its input is worth and is willing to sell their data upfront for the amount of money. However, we do not assume one's data is worth to oneself as much as to the adversary. We note that it suffices to set the cut-and-choose parameter n such that the honest data owner is happy, because even if the data is worth more to the adversary, it is considered acceptable for the adversary to buy the data directly from the data owner at the price set by the honest owner then profits from the purchased data. In other words, the adversary cannot gain any extra financial advantage when running our protocol comparing to executing an ideal model protocol.

What about the smart-contract fees? Per our discussion above, it is a zero-sum game between the two protocol participants if we ignore smart-contract transaction fees. In reality, all Ethereum smart-contracts are executed with a fee. There are two types of transaction fees involved in our protocol. Type-I fee covers the transactions (e.g., newSession, deposit and conclude) which always occur regardless of the adversary's behavior; whereas Type-II fee covers those transactions that occur only if the adversary cheats.

We note that Type-I fee cannot affect our game analysis because it is a sunk cost independent of the adversary's behavior. Regarding Type-II fee, in our protocol, both parties need to deposit d_{fee} that is sufficient to cover both parties' expenses in case a dispute is needed, but so Type-II fee is always charged towards the cheater. Thus, Type-II fee will not affect our game analysis, either. Therefore, after all the blockchain fees are taken into account, it can still be modeled by a zero-sum game.

Thanks to our enhancement techniques, we have experimentally shown that Type-I fee is only 482K gas, or \$0.38 USD, so normal protocol executions are very affordable. We show that Type-II fee is also reasonably low (at most 5.04M gas, or \$3.93 USD for a billion-gates circuit) for large scale applications.

Will the judge really be executed? One of the most interesting phenomena we expect from this design is that the judge never need to be invoked because no rational adversaries will attack a properly configured PVC protocol. However, it is still critically important to develop and deploy an affordable smart-contract to handle the disputes so that all rational adversaries will stay away from cheating. Note that these smart-contracts are executed only in very rare situations, their run-time efficiency is less of a concern as long as it is reasonable to intimidate rational attackers.

4 SECURE PUBLICLY-VERIFIABLE 2PC

We first describe the sender non-repudiable OT protocol which is indispensable in catching misbehavior in a publicly verifiable and defamation-free manner (Section 4.1). Then, we show the construction and security proof of our publicly-verifiable 2PC protocol (Section 4.2).

4.1 Sender Non-repudiable OT

The ideal $\mathcal{F}_{\text{SNR-OT}}$ functionality needed by our PVC protocol was given in Figure 4. $\mathcal{F}_{\text{SNR-OT}}$ can be implemented based on the DDH-based construction proposed by Peikert et al. [36]. We describe our realization of $\mathcal{F}_{\text{SNR-OT}}$ in Figure 6, whose basic idea is quite similar to Asharov and Orlandi's signed-OT [6]. However, our approach does not rely on the less standard EU-CMRA signatures.

Public Input: Public keys pk_S and pk_R.
Initialize: S and R agree on a *tag* for this protocol instance.
Setup: Let $\mathbb{G}$ be a cyclic group of order q. R uniformly picks $g_0, g_1 \leftarrow \mathbb{G}$, $\alpha \leftarrow \mathbb{Z}_q$, computes $h_0 := g_0^{\alpha}, h_1 := g_1^{\alpha-1}$, and proves through $\mathcal{F}_{\text{ZKPoK}}^{DDH}$ that $(g_0, h_0, g_1, h_1 \cdot g_1)$ is a DDH-tuple.
Execute-OT:
(1) R, with choice bit b, uniformly picks $r \leftarrow \mathbb{Z}_q$ and computes and sends $g := g_b^r, h := h_b^r$ to the OT Sender (S).
(2) S, with input messages $m_0, m_1 \in \mathbb{G}$, verifies that $g \neq 1$ and $h \neq 1$ (aborts otherwise), then computes (u_0, w_0) and (u_1, w_1) where $\forall i \in \{0,1\}$, $u_i = g_i^s \cdot h_i^t, v_i = g^s \cdot h^t$, and $w_i = v_i \cdot m_i$ for some uniform $s, t \in \mathbb{Z}_q$ that S randomly picks. S sends $[\![tag, g, h, g_0, g_1, h_0, h_1, (u_0, w_0, u_1, w_1)]\!]_S$ to R.
(3) R verifies

$$\mathbf{Verify}_{pk_S}\left([\![tag, g, h, g_0, g_1, h_0, h_1, (u_0, w_0, u_1, w_1)]\!]_S\right) = 1$$

and the values of $(tag, g, h, g_0, g_1, h_0, h_1)$ in the signed message are as they were actually used in previous steps. Then R outputs $w_b \cdot u_b^{-r}$.
Verify:
Any party V, who obtains

$$\left([\![tag', (b', m'), r]\!]_R, [\![tag, g, h, g_0, g_1, h_0, h_1, (u_0, w_0, u_1, w_1)]\!]_S\right),$$

should output 1 *iff* all checks below pass:
(1) $\mathbf{Verify}_{pk_R}\left([\![tag', (b', m'), r]\!]_R\right) = 1$ and $tag' = tag$;
(2) $\mathbf{Verify}_{pk_S}\left([\![tag, g, h, g_0, g_1, h_0, h_1, (u_0, w_0, u_1, w_1)]\!]_S\right) = 1$;
(3) $g = g_{b'}^r, h = h_{b'}^r$, and $m' = w_{b'} \cdot u_{b'}^r$.

Figure 6: Realize $\mathcal{F}_{\text{SNR-OT}}$ in $\mathcal{F}_{\text{ZKPoK}}^{DDH}$-hybrid model.

Theorem 4.1. *The protocol in Figure 6 securely realize $\mathcal{F}_{\text{SNR-OT}}$ in presence of malicious adversaries.*

Proof. Our protocol in Figure 6 doesn't alter the original DDH-based PVW-OT [36] except for requiring the sender to sign some of its outgoing messages. Therefore, the proof that **Execute-OT** interface is securely realized is essentially the same as the security proof of PVW-OT.

Next, we show the **Verify** interface is also securely realized. Since the **Verify** interface does not require interaction. Its security proof of **Verify** boils down to showing a completeness and a soundness property.

- *Completeness.* It is trivial to verify that our instantiation of **Verify** always return 1 on signed input from an honest receiver R. In the ideal/real model paradigm, completeness implies that **Verify** in both models always outputs 1 on honest inputs.
- *Soundness.* If a malicious receiver R received (b, m_b) when executing the protocol but submits to **Verify**,

$$\left([\![tag', (b', m'), r']\!]_R, [\![tag, g, h, g_0, g_1, h_0, h_1, (u_0, w_0, u_1, w_1)]\!]_S\right),$$

where $(b', m') \neq (b, m_b)$, then **Verify** cannot return 1 no matter if $b = b'$ or not. We prove this by contradiction.
  - If $b = b'$ and **Verify** returns 1, then check-condition (3) in Figure 6 implies $g_b^r = g = g_{b'}^{r'} = g_b^{r'}$, so it must be $r = r'$, hence $m = m'$, which contradicts to the fact that $(b', m') \neq (b, m_b)$.
  - If $b = \bar{b'}$ and **Verify** returns 1, then check-condition (3) implies $g_b^r = g = g_{\bar{b}}^{r'}$ and $h_b^r = h = h_{\bar{b}}^{r'}$. In case $b = 0$, we have $h_0^r = g_0^{\alpha r} = (g_0^r)^{\alpha} = (g_1^{r'})^{\alpha} = (g_1^{\alpha})^{r'} = h_1^{r'} = g_1^{(\alpha-1)r'}$, which implies $g_1^{r'} = g_0^r = 1$, contradicting to the fact that the sender S had ensured that $g = g_0^r \neq 1$ when executing OT. Similarly, we can show $b = 1$ contradicts to the fact that $h = h_0^r \neq 1$.

This completes the soundness proof. In the ideal/real model paradigm proof, soundness implies that **Verify** in both models always output 0 on corrupted inputs. □

4.2 The Two-Party PVC Protocol

Assume P_1, the garbler holding input $x \in \{0,1\}^l$, and P_2, the evaluator holding input $y \in \{0,1\}^l$, want to compute and send $z = f(x, y)$ (but nothing else) to P_2. Here f is a public function represented as a fully unrolled sequence of gates. Without loss of generality, weassume x, y, and z each has l bits.

We model H_R, H_L as distinct random oracles which are used to derive the secret XOR difference (between 0- and 1-labels) and the initial input wire-labels, resp. Let s, κ, $\mathsf{wid}_\mathsf{x}(j)$, $\mathsf{wid}_\mathsf{y}(j)$, $\mathsf{wid}_\mathsf{r}(j)$, $\mathsf{wid}_\mathsf{z}(j)$ be as was defined in Figure 1. Let $[\![m]\!]_{P_1}$ denote a pair $(m, \mathsf{Sign}_{pk_{P_1}}(m))$. Depending on the context, $[\![m]\!]_{P_1}$ can serve as an irrefutable evidence of P_1's claim on the value of m. When used for this purpose, we assume $[\![m]\!]_{P_1}$ is a signature over a non-ambiguous claim on the value of m including necessary meta-data such as the names of the variables and timestamps as message sequence number.

Our PVC protocol is formally specified in Figure 7. The judge for our protocol is specified in Figure 8. The constructions make black-box calls to $\mathcal{F}_{\text{SNR-OT}}$ (Sender Non-Repudiable OT, see Figure 4) and $\mathcal{F}_{\text{CorrOT}}$ (Correlated OT, see Figure 5). The key ideas and intuitions behind our design was given in Section 3.1. We assume the output is revealed only to P_2 but it isn't hard to modify it to support other arrangements of the output.

Theorem 4.2. *The protocol in Figure 7 is a publicly verifiable secure two-party computation protocol with $(1-1/n)$-deterrence against covert adversaries.*

Proof. We first show the protocol of Figure 7 securely computes $f(x, y)$ in presence of covert adversary with deterrence $1 - \frac{1}{n}$.

When P_1 is corrupted. Let $\mathcal{S}$ be an efficient simulator that runs P_1 as a subroutine, interacts with the ideal 2PC functionality $\mathcal{F}_{\text{2PC}}$ in P_1's role. $\mathcal{S}$ interacts with P_1 using the protocol of Figure 7 except for the following changes:

(1) In Step (1), $\mathcal{S}$ extracts $Seed_i$ and w_i for $i \in [n]$ and computes $(d_i', h_i') := \mathrm{Gb}(1^{\kappa}, f', Seed_i)$.

Inputs: The public function $f : \{0,1\}^l \times \{0,1\}^l \mapsto \{0,1\}^l$. P_1 holds $x \in \{0,1\}^l$ and P_2 holds $y \in \{0,1\}^l$. Two random oracles H_R, H_L.
Outputs: P_1 outputs nothing; P_2 outputs $f(x,y)$.

Protocol:

(0) **[Augment f and Sign]** Augment the function f to f' such that $f'(x,(y,r)) = (x \blacktriangleright f(x,y), y, r)$. Namely, f' takes $x \in \{0,1\}^l$ from P_1, $(y,r) \in \{0,1\}^{l+s}$ from P_2, and outputs $(x \blacktriangleright f(x,y), y, r)$ to P_2. Then both parties hash and sign the circuit file of f', exchange and verify each other's signature on this circuit, and abort if the signature fails to verify or the circuits are not identical.

Remark 1. *The "RIGHT" gate, $\blacktriangleright$, will be realized using ANDs and XORs. We need it for proof of security: In the case of corrupted P_2, the simulator who doesn't know x can obliviously change the $\blacktriangleright$ gate into a "LEFT" gate $\blacktriangleleft$ in the evaluation-circuit and set $x = z$ (where $z = f(x,y)$ was obtained from the ideal 2PC functionality) to allow P_2 to output $f(x,y)$. (see Section 4.2)*

Remark 2. *We intentionally designed (y,r) to be both P_2's input and output, for circuit verification reasons (so that P_2 can carry out the necessary checks later in Step (3) and Step (4) to thwart selective OT attacks).*

(1) **[OT c]** P_2 picks a uniform integer $\mathcal{I} \in [n]$ which is the index of the evaluation circuit. Let $c \in \{0,1\}^n$ such that $c_i = 0$ for all $i \neq \mathcal{I}$ and $c_i = 1$ when $i = \mathcal{I}$. P_1 picks uniform $\{w_i\}_{i\in[n]}$ where $w_i \in \{0,1\}^\kappa$. P_1 and P_2 run n instances of $\mathcal{F}_{\text{SNR-OT}}$. In the i^{th} $\mathcal{F}_{\text{SNR-OT}}$, P_1 sends to $\mathcal{F}_{\text{SNR-OT}}$ $\left(\text{“}i^{th}\text{-OT”}, (Seed_i, w_i)\right)$ and P_2 as the receiver sends $\mathcal{F}_{\text{SNR-OT}}$ $\left(\text{“}i^{th}\text{-OT”}, c_i\right)$. As a result, P_2 learns $\{Seed_i\}_{i\in[n], i\neq\mathcal{I}}$ and $w_\mathcal{I}$.

(2) **[Garble]** For all $i \in [n]$, P_1 computes $(F_i, e_i, d_i) := \mathsf{Gb}(1^\kappa, f', Seed_i)$ and $h_i := H(F_i)$ where H is a collision-resistant hash and e_i can be parsed into $\left(R_i, \left\{L^0_{i,\mathsf{wid_x}(j)}, L^0_{i,\mathsf{wid_y}(j)}\right\}_{j\in[l]}, \left\{L^0_{i,\mathsf{wid_r}(j)}\right\}_{j\in[s]}\right)$. Note that for all $i \in [n]$, $R_i = H_R(Seed_i)$; $\forall i \in [n]$,
$$L^0_{i,\mathsf{wid_x}(j)} = H_L(Seed_i, \mathsf{wid_x}(j)),\ \forall j \in [l]; \qquad L^0_{i,\mathsf{wid_y}(j)} = H_L(Seed_i, \mathsf{wid_y}(j)),\ \forall j \in [l]; \qquad L^0_{i,\mathsf{wid_r}(j)} = H_L(Seed_i, \mathsf{wid_r}(j)),\ \forall j \in [s],$$
which are, respectively, the XOR-difference between a 0-label and a 1-label on the same wire, 0-labels of P_1's inputs, 0-labels of P_2's inputs, and 0-labels of the s augmented wires. Then P_1 sends $\left\{[\![(d_i,h_i)]\!]_{P_1}\right\}_{i\in[n]}$ to P_2.

(3) **[Verify check-circuits]** For every $i \neq \mathcal{I}, 0 \le i < n$, P_2 verifies that d_i, h_i can be produced by $\mathsf{Gb}(1^\kappa, f, Seed_i)$. If there exists an $\imath \neq \mathcal{I}$ such that $d_\imath, h_\imath$ cannot be produced by $\mathsf{Gb}(1^\kappa, f, Seed_\imath)$, then P_2 halts and submits $\left(\imath, [\![(d_\imath,h_\imath)]\!]_{P_1}, \left[\!\left[\text{“}\imath^{th}\text{-OT”}, (c_\imath, Seed_\imath)\right]\!\right]_{P_2}\right)$ to Judge.

(4) **[OT y]** P_2 sends $\mathcal{I}$ and $w_\mathcal{I}$ to P_1, who validates $w_\mathcal{I}$ with its own copy of $w_\mathcal{I}$. P_2 samples uniform $r \in \{0,1\}^s$. The two parties run a $\mathcal{F}_{\text{CorrOT}}$ (of $l+s$ OTs) where P_1 as the sender sends $\left(R_\mathcal{I}, \left\{L^0_{\mathcal{I},\mathsf{wid_y}(j)}\right\}_{j\in[l]}, \left\{L^0_{\mathcal{I},\mathsf{wid_r}(j)}\right\}_{j\in[s]}\right)$ to $\mathcal{F}_{\text{CorrOT}}$ and P_2 as the receiver sends $\left(\{y_j\}_{j\in[l]}, \{r_j\}_{j\in[s]}\right)$ to $\mathcal{F}_{\text{CorrOT}}$. As a result, P_2 receives $\left(\left\{L^{y_j}_{\mathcal{I},\mathsf{wid_y}(j)}\right\}_{j\in[l]}, \left\{L^{r_j}_{\mathcal{I},\mathsf{wid_r}(j)}\right\}_{j\in[s]}\right)$. P_2 verifies that the $\left\{L^{y_j}_{\mathcal{I},\mathsf{wid_y}(j)}\right\}_{j\in[l]}$ and $\left\{L^{r_j}_{\mathcal{I},\mathsf{wid_r}(j)}\right\}_{j\in[s]}$ obtained from $\mathcal{F}_{\text{CorrOT}}$ is consistant with the decoding information $d_\mathcal{I}$ and aborts otherwise.

(5) **[Verify evaluation-circuit]** P_1 sends $F_\mathcal{I}$ to P_2, who verifies $H(F_\mathcal{I}) = h_\mathcal{I}$ where H is a collision-resistant hash.

(6) **[Evaluate]** P_1 sends $\left\{L^{x_j}_{\mathcal{I},\mathsf{wid_x}(j)}\right\}_{j\in[l]}$ to P_2. Then P_2 computes and outputs the first entry of
$$z \overset{\text{def}}{=\!=} \mathsf{De}\left(d_\mathcal{I}, \mathsf{Ev}\left(F_\mathcal{I}, \left\{L^{x_j}_{\mathcal{I},\mathsf{wid_x}(j)}, L^{y_j}_{\mathcal{I},\mathsf{wid_y}(j)}\right\}_{j\in[l]}, \left\{L^{r_j}_{\mathcal{I},\mathsf{wid_r}(j)}\right\}_{j\in[s]}\right)\right).$$

Figure 7: Our publicly-verifiable two-party computation protocol with $(1-1/n)$-deterrence in the $\mathcal{F}_{\text{SNR-OT}}$-hybrid model.

Upon receiving $\left(\imath, [\![(d_\imath,h_\imath)]\!]_{P_1}, \left[\!\left[\text{“}\imath^{th}\text{-OT”}, (c_\imath, Seed_\imath)\right]\!\right]_{P_2}\right)$, check

(1) $\mathbf{Verify}_{pk_{P_1}}\left([\![(d_\imath,h_\imath)]\!]_{P_1}\right) = 1$;

(2) $\mathcal{F}_{\text{SNR-OT}}.\mathbf{Verify}\left(\left[\!\left[\text{“}\imath^{th}\text{-OT”}, (c_\imath, Seed_\imath)\right]\!\right]_{P_2}\right) = 1$ and $c_\imath = 0$;

(3) $\mathsf{Ve}(f', Seed_\imath, d_\imath, h_\imath) = 0$.

and output $\mathsf{P_1}$-Cheats if they all hold, and $\mathsf{P_2}$-Cheats otherwise.

Figure 8: The Judge algorithm for our PVC 2PC.

(2) In Step (2), $\mathcal{S}$ receives $[\![(d_i,h_i)]\!]_{P_1}$. Let J be the set of indexes i such that $(d'_i, h'_i) \neq (d_i, h_i)$.

- If $|J| \ge 2$, then $\mathcal{S}$ sends Blatant Cheat to $\mathcal{F}_{\text{2PC}}$ and
$$\left(\imath, [\![(d_\imath,h_\imath)]\!]_{P_1}, \left[\!\left[\text{“}\imath^{th}\text{-OT”}, (0, Seed_\imath)\right]\!\right]_{P_2}\right)$$
to P_1 (for $\imath$ uniformly picked in J) and aborts.
- If $J = \{\imath\}$, then $\mathcal{S}$ sends Cheat to $\mathcal{F}_{\text{2PC}}$. If $\mathcal{F}_{\text{2PC}}$ returns Corrupted, $\mathcal{S}$ sends to P_1
$$\left(\imath, [\![(d_\imath,h_\imath)]\!]_{P_1}, \left[\!\left[\text{“}\imath^{th}\text{-OT”}, (0, Seed_\imath)\right]\!\right]_{P_2}\right)$$
and aborts. Otherwise, $\mathcal{S}$ sets $Attack := 1$ and $\mathcal{I} = \imath$.
- If $J = \emptyset$, $\mathcal{S}$ sends Honest to $\mathcal{F}$. $\mathcal{S}$ sets $Attack := 0$ and uniformly samples $\mathcal{I} \in [n]$.

(3) In Step (4), $\mathcal{S}$ sends $\mathcal{I}$ and $w_{\mathcal{I}}$ to P_1. Then $\mathcal{S}$ uniformly samples $\{r_j\}_{j\in[s]}$ and executes $\mathcal{F}_{\mathsf{CorrOT}}$ with P_1 by setting $y_j = 0$ for $j \in [l]$. $\mathcal{S}$ also extracts P_1's inputs from $\mathcal{F}_{\mathsf{CorrOT}}$.
   - If *Attack* $= 0$, $\mathcal{S}$ aborts if P_1's inputs extracted are not $\left(R_{\mathcal{I}}, \left\{L^0_{\mathcal{I},\mathsf{wid_y}(j)}\right\}_{j\in[l]}, \left\{L^0_{\mathcal{I},\mathsf{wid_r}(j)}\right\}_{j\in[s]}\right)$.
   - If *Attack* $= 1$, $\mathcal{S}$ behaves the same as an honest P_2.

(4) In Step (6): $\mathcal{S}$ receives $\left\{L^{x_j}_{\mathcal{I},\mathsf{wid_x}(j)}\right\}_{j\in[l]}$ from P_1.
   - If *Attack* $= 1$, $\mathcal{S}$ recovers $\hat{f}$ (which models the leakage of an ideal covert 2PC in event of a successful attack) from $\left\{L^{x_j}_{\mathcal{I},\mathsf{wid_x}(j)}\right\}_{j\in[l]}, F_{\mathcal{I}}, d_{\mathcal{I}}$ and P_1's actual input to $\mathcal{F}_{\mathsf{CorrOT}}$. Then $\mathcal{S}$ sends $\hat{f}$ to $\mathcal{F}$.
   - If *Attack* $= 0$, $\mathcal{S}$ extracts x from $\left\{L^{x_j}_{\mathcal{I},\mathsf{wid_x}(j)}\right\}_{j\in[l]}$, $Seed_{\mathcal{I}}$ and $F_{\mathcal{I}}$, then sends x to $\mathcal{F}$.

   Finally, $\mathcal{S}$ outputs whatever P_1 outputs.

We now argue that the joint distribution of the outputs of $\mathcal{S}$ and P_2 in the ideal model is indistinguishable from that of the outputs of the malicious P_1 and an honest P_2 in the real model. We prove this by considering a series of hybrid experiments in each of which a different simulator is used.

- **Hybrid$_1$**: This is the real model execution. The simulator S_1 in **Hybrid$_1$** knows y and interacts with P_1 using the protocol of Figure 7 as an honest P_2.
- **Hybrid$_2$**: $\mathcal{S}_2$ is the same as $\mathcal{S}_1$ in **Hybrid$_1$**, except that:
  (1) In Step (1), $\mathcal{S}_2$ extracts $Seed_{\mathcal{I}}$, thus can learn whether $F_{\mathcal{I}}$ is correct.
  (2) In Step (4), $\mathcal{S}_2$ aborts if $F_{\mathcal{I}}$ is correct but P_1's inputs $R_{\mathcal{I}}$ and $\{L^0_{\mathcal{I},j}\}_{j\in[l+s]}$ to $\mathcal{F}_{\mathsf{CorrOT}}$ is not consistent with $Seed_{\mathcal{I}}$.

  We claim **Hybrid$_1$** $\approx$ **Hybrid$_2$** because
  – If a different $R'_{\mathcal{I}}$ was used in place of $R_{\mathcal{I}}$ in $\mathcal{F}_{\mathsf{CorrOT}}$, then $\mathcal{S}_1$ will abort when verifying $\{L^0_{\mathcal{I},\mathsf{wid_r}(j)} \oplus r_j R_{\mathcal{I}}\}_{j\in[s]}$ against $\{H(L^0_{\mathcal{I},\mathsf{wid_r}(j)}), H(L^1_{\mathcal{I},\mathsf{wid_r}(j)})\}_{j\in[s]}$ contained in $d_{\mathcal{I}}$. Because each r_j is uniformly sampled, the probability that P_1 guess all s bits of r is at most 2^{-s}.
  – If P_1 used a correct $R_{\mathcal{I}}$ but a corrupted $L^0_{\mathcal{I},\mathsf{wid_y}(j)}$ for some j, no matter what $\mathcal{S}_1$'s choice bit is, the value $\mathcal{S}_1$ obtained from $\mathcal{F}_{\mathsf{CorrOT}}$ cannot be consistent with the pair of hashes $H(L^0_{\mathcal{I},\mathsf{wid_r}(j)}), H(L^1_{\mathcal{I},\mathsf{wid_r}(j)})$ contained in $d_{\mathcal{I}}$. So is it the case for a corrupted $L^0_{\mathcal{I},\mathsf{wid_r}(j)}$. Note that P_1 cannot swap the 0-label and 1-label either, because $\mathcal{S}_1$ can always detect this using the semantic value and the corresponding hash in $d_{\mathcal{I}}$. Therefore, $\mathcal{S}_2$ and $\mathcal{S}_1$ will abort at the same time.
- **Hybrid$_3$**: $\mathcal{S}_3$ is the same as $\mathcal{S}_2$, except that $\mathcal{S}_3$ extracts P_1's input x and outputs $f(x,y)$ if the evaluation-circuit is correct. The indistinguishably of **Hybrid$_2$** and **Hybrid$_3$** comes from the correctness of the garbling scheme.
- **Hybrid$_4$**: $\mathcal{S}_4$ is the same as $\mathcal{S}_3$, except that $\mathcal{S}_4$ can interact with the ideal $\mathcal{F}_{\mathsf{2PC}}$ as follows.
  (1) In Step (1), $\mathcal{S}_4$ extracts $Seed_i$ and w_i and recomputes $(d'_i, h'_i) := \mathsf{Gb}(1^\kappa, f', Seed_i)$ for $i \in [n]$. Let J be the set of indexes i such that $(d'_i, h'_i) \neq (d_i, h_i)$.
  – If $|J| \geq 2$, then $\mathcal{S}_4$ sends blatantCheat to $\mathcal{F}_{\mathsf{2PC}}$ and
  $$\left(\imath, [\![(d_\imath, h_\imath)]\!]_{P_1}, \left[\!\!\left[\text{“}\imath^{th}\text{-OT”}, (0, Seed_\imath)\right]\!\!\right]_{P_2}\right)$$
  to P_1 (for $\imath$ randomly picked in J) and aborts.
  – If $J = \{\imath\}$, then $\mathcal{S}_4$ sends Cheat to $\mathcal{F}_{\mathsf{2PC}}$. If $\mathcal{F}_{\mathsf{2PC}}$ returns Corrupted, $\mathcal{S}$ sends P_1
  $$\left(\imath, [\![(d_\imath, h_\imath)]\!]_{P_1}, \left[\!\!\left[\text{“}\imath^{th}\text{-OT”}, (0, Seed_\imath)\right]\!\!\right]_{P_2}\right)$$
  and aborts. Otherwise, $\mathcal{S}_4$ sets *Attack* $:= 1$ and $\mathcal{I} := \imath$.
  – If $J = \emptyset$, $\mathcal{S}_4$ sends Honest to $\mathcal{F}_{\mathsf{2PC}}$. $\mathcal{S}_4$ sets *Attack* $:= 0$ and uniformly samples $\mathcal{I} \in [n]$.
  (2) In Step (6): $\mathcal{S}_4$ receives $\left\{L^{x_j}_{\mathcal{I},\mathsf{wid_x}(j)}\right\}_{j\in[l]}$ from P_1.
  – If *Attack* $= 1$, S_4 recovers $\hat{f}$ from $\left\{L^{x_j}_{\mathcal{I},\mathsf{wid_x}(j)}\right\}_{j\in[l]}, F_{\mathcal{I}}, d_{\mathcal{I}}$ and P_1's input to $\mathcal{F}_{\mathsf{CorrOT}}$. Then $\mathcal{S}_4$ sends $\hat{f}$ to $\mathcal{F}_{\mathsf{2PC}}$.
  – If Attack $= 0$, $\mathcal{S}_4$ extracts x from $\left\{L^{x_j}_{\mathcal{I},\mathsf{wid_x}(j)}\right\}_{j\in[l]}$ and $F_{\mathcal{I}}$, then sends x to $\mathcal{F}_{\mathsf{2PC}}$.

  $\mathcal{S}_4$ outputs whatever P_1 outputs.

  We claim **Hybrid$_3$** $\approx$ **Hybrid$_4$** because:
  – If S_4 sends Honest to $\mathcal{F}_{\mathsf{2PC}}$, then both $\mathcal{S}_3$ and the actual P_2 in **Hybrid$_4$** will output $f(x,y)$.
  – If $\mathcal{S}_4$ sends blatantCheat to $\mathcal{F}_{\mathsf{2PC}}$, then both $\mathcal{S}_3$ and the actual P_2 in **Hybrid$_4$** will halt with
  $$\left(\imath, [\![(d_\imath, h_\imath)]\!]_{P_1}, \left[\!\!\left[\text{“}\imath^{th}\text{-OT”}, (0, Seed_\imath)\right]\!\!\right]_{P_2}\right)$$
  as output for a randomly chosen $\imath \in J$.
  – If $\mathcal{S}_4$ sends Cheat to $\mathcal{F}_{\mathsf{2PC}}$, then $\mathcal{S}_3$ will abort and output
  $$\left(\imath, [\![(d_\imath, h_\imath)]\!]_{P_1}, \left[\!\!\left[\text{“}\imath^{th}\text{-OT”}, (0, Seed_\imath)\right]\!\!\right]_{P_2}\right)$$
  with $1 - \frac{1}{n}$ probability. Meanwhile, the actual P_2 in **Hybrid$_4$** will abort and output the witness with the same probability. Furthermore, both $\mathcal{S}_3$ and the actual P_2 in **Hybrid$_4$** will output $\hat{f}(y)$ for $\frac{1}{n}$ probability where $\hat{f}$ is specified by P_1.
- **Hybrid$_5$**: $\mathcal{S}_5$ is the same as $\mathcal{S}_4$, except that it uses $y_i = 0$ for $i \in [l]$ instead of P_2's real input y. $\mathcal{S}_5$ is exactly the simulator $\mathcal{S}$ we described at the beginning.

  We claim **Hybrid$_5$** $\approx$ **Hybrid$_4$** because: P_1's views in **Hybrid$_4$** and **Hybrid$_5$** are indistinguishable since $\mathcal{F}_{\mathsf{CorrOT}}$ is ideal and y is never used by $\mathcal{S}_4$ in **Hybrid$_4$** except being given to $\mathcal{F}_{\mathsf{CorrOT}}$.

By this series of indistinguishable experiments we have shown that the ideal world execution and the real world execution are indistinguishable. Thus our protocol is secure against corrupted P_1.

When P_2 is corrupted. Let $\mathcal{S}$ be an efficient simulator that runs the corrupted P_2 as a subroutine and interacts with $\mathcal{F}_{\mathsf{2PC}}$ in the ideal model in P_2's role. $\mathcal{S}$ behaves like an honest P_1 in the protocol of Figure 7 except for the following changes:

(1) In Step (1), $\mathcal{S}$ extracts P_2's choice string c to $\mathcal{F}_{\mathsf{SNR\text{-}OT}}$. Let $C = \{i | c_i = 1\}$, i.e., the set of indices i such that $c_i = 1$.

(2) In Step (2), for all $i \notin C$, $\mathcal{S}$ generates (F_i, e_i, d_i) honestly. For all $i \in C$, $\mathcal{S}$ computes $(\hat{F}_i, \hat{e}_i, \hat{d}_i)$ by garbling the function

$$\hat{f}'(x,(y,r)) = \left(\perp, (x \blacktriangleleft f(x,y), y, r)\right)$$

and computes $\hat{h}_i = H(\hat{F}_i)$. So,

$$\mathsf{De}\left(\hat{d}_i, \mathsf{Ev}\left(\hat{F}_i, \mathsf{En}\left(\hat{e}_i, (x,(y,r))\right)\right)\right) = (x,(y,r))$$

Then S sends $[\![(d_i, h_i)]\!]_{P_1}$ for all $i \notin C$ and $\left[\!\!\left[(\hat{d}_i, \hat{h}_i)\right]\!\!\right]_{P_1}$ for all $i \in C$ to P_2.

(3) In Step (4), $\mathcal{S}$ receives $(\mathcal{I}, w_{\mathcal{I}})$ and verifies that they are consistent with his/her own record and $\mathcal{I} \in C$. Then $\mathcal{S}$ runs $\mathcal{F}_{\mathsf{CorrOT}}$ as described in the protocol and extracts y.

(4) In Step (5), $\mathcal{S}$ sends the $\hat{F}_{\mathcal{I}}$ to P_2.

(5) In Step (6), $\mathcal{S}$ submits y to $\mathcal{F}_{\mathsf{2PC}}$ and receives $z := f(x,y)$ from $\mathcal{F}_{\mathsf{2PC}}$. S sends $\left\{L^{z_j}_{\mathcal{I},\mathsf{wid}_{\mathsf{x}}(j)}\right\}_{j\in[l]}$ to P_2. Finally, $\mathcal{S}$ outputs whatever P_2 outputs.

We argue that the joint distribution of the outputs of $\mathcal{S}$ and P_1 in the ideal model is indistinguishable from that of the outputs of the malicious P_2 and an honest P_1 in the real model. Since P_1 has no output in both ideal and real models, we only need to focus on the view of P_2. We will prove this using a series of hybrid experiments.

- **Hybrid$_1$**: This is the real model execution. The simulator $\mathcal{S}_1$ interacts with P_2 through the protocol of Figure 7 using P_1's input x.
- **Hybrid$_2$**: $\mathcal{S}_2$ is the same as $\mathcal{S}_1$, except $\mathcal{S}_2$ aborts if $\mathcal{I} \notin C$ in Step (4).

  We claim **Hybrid$_1$** $\approx$ **Hybrid$_2$**, because P_2 can't find the valid $w_{\mathcal{I}}$ where $\mathcal{I} \notin C$ except unless P_2 guessed $w_{\mathcal{I}}$ correctly. Therefore, with all but a negligible probability, $\mathcal{S}_2$ aborts in **Hybrid$_2$** if and only if $\mathcal{S}_1$ also aborts in **Hybrid$_1$**.
- **Hybrid$_3$**: $\mathcal{S}_3$ is the same as $\mathcal{S}_2$, except:
  (1) In Step (2), $\mathcal{S}_3$ replaces (F_i, e_i, d_i) with $(\hat{F}_i, \hat{e}_i, \hat{d}_i)$ for all $i \in C$.
  (2) In Step (4), $\mathcal{S}_3$ extracts y from $\mathcal{F}_{\mathsf{CorrOT}}$.
  (3) In Step (6), $\mathcal{S}_3$ computes $z = f(x,y)$ and sends $\left\{L^{z_j}_{\mathcal{I},\mathsf{wid}_{\mathsf{x}}(j)}\right\}_{j\in[l]}$.

  We claim **Hybrid$_2$** $\approx$ **Hybrid$_3$** because:
  – P_2 can't distinguish $(\hat{F}_i, \hat{e}_i, \hat{d}_i)$ from (F_i, e_i, d_i) because: (1) the garbling scheme is oblivious; and (2) garbled $\blacktriangleright$ gates are indistinguishable from garbled $\blacktriangleleft$ gates. (Note that for any $a, b \in \{0,1\}$, $a \blacktriangleright b = ((a \oplus b) \wedge \overline{a \oplus b}) \oplus b$. Let $\tilde{\wedge}$ be a corrupted AND that always and the first input bit with a flipped value of its second input bit. It is easy to verify that $a \blacktriangleleft b = ((a \oplus b) \tilde{\wedge} \overline{a \oplus b}) \oplus b$. Because the garblings of $\wedge$ and $\tilde{\wedge}$ are indistinguishable and the wire connections of the circuits for $\blacktriangleright$ and $\blacktriangleleft$ are identical, garbled $\blacktriangleright$ and $\blacktriangleleft$ are also indistinguishable.)
  – Since $(\hat{F}_i, \hat{e}_i, \hat{d}_i)$ corresponds to the identity function, the decoded results in Step (6) of **Hybrid$_3$** are the same as that in **Hybrid$_2$**.
- **Hybrid$_4$**: $\mathcal{S}_4$ is the same as $\mathcal{S}_3$, except that $\mathcal{S}_4$ interacts with the ideal functionality $\mathcal{F}_{\mathsf{2PC}}$: In Step (6), $\mathcal{S}_4$ sends y to $\mathcal{F}_{\mathsf{2PC}}$, who sends $z := f(x,y)$ back; then $\mathcal{S}_4$ outputs whatever P_2 outputs.

  Hybrid$_3$ $\approx$ **Hybrid$_4$** because the ideal functionality $\mathcal{F}_{\mathsf{2PC}}$ computes $f(x,y)$ correctly.
- **Hybrid$_5$**: $\mathcal{S}_5$ is the same as $\mathcal{S}_4$, except that $\mathcal{S}_5$ sets $x := 0$ instead of using P_1's actual input x. $\mathcal{S}_5$ is the simulator $\mathcal{S}$ we gave earlier at the beginning of the proof for the corrupted P_2 case.

  Hybrid$_4$ $\approx$ **Hybrid$_5$** because $\mathcal{S}_4$ doesn't use x in **Hybrid$_4$**.

By this series of indistinguishable experiments we have shown that the ideal and the real model executions are indistinguishable. Thus our protocol is secure against corrupted P_2.

Public Verifiability. It is easy to see that the proofs obtained from our PVC protocol is publicly verifiable.

(1) *Completeness.* By definition, P_1 cheats if he/she sent some d_i, h_i and $Seed_i$ through $\mathcal{F}_{\mathsf{SNR\text{-}OT}}$ such that $\mathsf{Ve}(f', Seed_\imath, d_\imath, h_\imath) = 0$ where Ve is the public verification algorithm of the garbling scheme. The judge checks exactly this fact, as well as the validity of $Seed_i$ through $\mathcal{F}_{\mathsf{SNR\text{-}OT}}$. So whenever inconsistent $d_i, h_i, Seed_i$ were sent, the judge must output "P$_1$-Cheats".

(2) *Soundness.* Note that $[\![d_i, h_i]\!]_{P_1}$ is signed by an EU-CMA secure signature scheme. By the security of EU-CMA and the definition of $\mathcal{F}_{\mathsf{SNR\text{-}OT}}$'s **Verify** function, P_2 cannot forge the values of $d_i, h_i, Seed_i$ and still pass the **Verify** functions of $\mathcal{F}_{\mathsf{SNR\text{-}OT}}$ and the signature scheme. Therefore, the judge must always output "P$_2$-Cheats" on receiving forged $(d_i, h_i, Seed_i)$.

□

5 AUTOMATED DECENTRALIZED JUDGE

Our goal here is to develop a lightweight, decentralized judge that can automatically verify proofs and punish the cheater. We explained the intuitions behind this part of the system in Section 3.2. Here, we focus on the formal presentation and details.

5.1 Search Short Proofs of Peer's Misbehavior

The proof search procedure is triggered by the event that P_2 obtains evidence $\left(\imath, [\![(d_\imath, h_\imath)]\!]_{P_1}, \left[\!\!\left[\text{“}\imath^{th}\text{-OT”}, (c_\imath, Seed_\imath)\right]\!\!\right]_{P_2}\right)$. We model the proof searching process with a variant of finite state automaton whose states are parameterized by a *tag*, which is a signed tuple of values (including a state id and a variable number of values needed by that state) that both parties have agreed. Each parameterized state is entered with a *record*, which is a tag followed by one or

Algorithm 1 The short proof generation algorithm

```
fun genProofInductive (record)
    halfRec_1 := next(record)
    halfRec_2 := msgExchange(halfRec_1)
    if halfRec_1.stateTag ≠ half_rec_2.stateTag then
        return (ErrMsg, record, halfRec_2)
    proof := genProofBase(halfRec_2)
    if proof ≠ ⊥ then
        return proof
    else
        record' := merge(halfRec_1, halfRec_2)
        return genProofInductive(record')
```

more new (signed) claims. Namely,

$$tag \stackrel{\text{def}}{=} (id, tagval_1, tagval_2, \dots)$$
$$record \stackrel{\text{def}}{=} (tag, recval_1, recval_2, \dots),$$

with the number and semantics of the values specifically defined for each state. Both parties must run a deterministic function genProofInductive of Algorithm 1 to navigate the states: first, they compute a half-record (i.e., the tag and its own claim on a set of values) for the next state (Line 2); then they exchange their half-record and check if they agree on the tag of next state (Line 3). If not, the honest party immediately obtains a proof (Line 5) because next is deterministic function over the same record. Otherwise, the honest party calls genProofBase to see if he/she can already output a proof in this state (Line 8) and proceeds to a recursive call to genProofInductive with the freshly merged record only if a proof wasn't found yet (Line 11). We will describe in detail how we design the states and their transition rules for resolving disputes on incrementally hashed messages (Section 5.2) and the computation of Ve (Section 5.3).

The msgExchange function is realized as a lightweight Ethereum transaction `sendmsg`. This binds both parties to faithfully run the proof search algorithm to the very end, at which point an irrefutable proof of misbehavior has to be derived by the honest party. `sendmsg` never examines the contents of the messages but only (1) ensures the message ids are indeed sequential, (2) remembers the identity of the last responder and the block-depth (as a timestamp), and (3) uploads the messages to the blockchain. If the adversary doesn't reply within a threshold number of block, the honest party will invoke `verifyPrf` transaction with a Timeout proof.

If the adversary replies nonsensical messages, the honest party can immediately output the two consecutive messages, the second of which makes no sense according to the deterministic next function, as a proof. This proof will be sent to and judged by an appropriate smart-contract transaction who cares only about how the next function for a particular state is computed. Therefore, nonsense messages senders will be penalized.

Of the three checks of Figure 8 by the judge, the first one involves only a small constant computation, the second check boils down to three conditional checks defined by SNROT's **Verify** (see Figure 6), each of which involves only small constant computation. So, in the rest of this section, we will focus on describing how to generate the proof from contradicting claims on the computation of the third check, which inflicts linear work on the judge if treated naïvely.

Notation. All values appear in a record are signed with an EUCMA-secure signature scheme. See Figure 1 for meanings of $f, f_i, F, F_i, IH^f_i, IH^F_i, \{\!\{\cdot\}\!\}$, identical and peer.

Circuit File Format. We specify f in a circuit file format similar to the one used by many MPC prototypes [37], except for a few customary enhancements for efficient dispute resolution purpose. It allows 4 types of gates: INPUT, OUTPUT, AND, and XOR. Each line specifies exactly one gate. An INPUT gate is specified as a line "`INPUT`". E.g., "`INPUT`" appearing on the 10th line of the circuit file means wire-10 is an input-wire of the circuit f. An OUTPUT gate is specified as a line "`OUTPUT wire#`". E.g., "`OUTPUT 8`" means wire-8 is an output-wire of f. An AND gate is specified as a line "`AND wire# wire#`". E.g., "`AND 8 9`" appearing on the 10th line in the file means wire-10 is the output of AND-ing wire-8 and wire-9. Similarly to ANDs are the XOR gates specified.

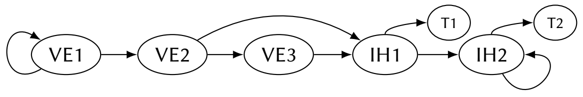

Figure 9: The state transition diagram

We illustrated the state transition diagram in Figure 9. IH1 and IH2 models the proof search process where the two parties have disagreed on an incrementally signed (long) message (Section 5.2). The message can either be the circuit file or a sequence of garbled gates. States VE1, VE2, VE3 handles proof searching from a disagreeing result of running the Ve algorithm. Note that both VE2 and VE3 can directly transit to IH1 when it is sure if there is a problem in the garbled circuit F, or the circuit description f. Except for state VE1, it is possible that a valid proof is obtained in all states (through their respective genProofBase function), thus terminating the automaton. Next, we describe the next and the genProofBase functions in ML-like pseudocode, which makes extensive use of pattern matching. We also borrow the "@" syntax from Haskell to conveniently pattern matching both a value and the value's component values.

5.2 Warmup: Proofs from Disagreeing Hashes

When an adversary cheats, it can disagree with the honest party on some message M that is too long to send to a smart-contract to verify. E.g., F and f used by Ve can be very long. So we use incremental hashes of the message, i.e., IH^M. The process can be described as a state machine with the following parametric states.

5.2.1 State IH1. Let IH^M_i be the (incremental) hash of M from block 1 to block i, and M_i be the i^{th} block of M. Entering this state on record $\left((\mathsf{IH1}, i_s, i_e, IH^M_{i_e}, \{\!\{M_{i_s}\}\!\}), \{\!\{IH^M_{i_s}\}\!\}\right)$, the parties must have agreed on the value of $IH^M_{i_e}$ but disagree on M_{i_s} with $i_s \le i_e$.

```
12: fun next $\left((\mathsf{IH1}, i_s, i_e, IH^M_{i_e}, \{\!\{M_{i_s}\}\!\}), \{\!\{IH^M_{i_s}\}\!\}\right)$
13:     | identical($\{\!\{IH^M_{i_s}\}\!\}$) ⇒ return $((\mathsf{T1}, IH^M_{i_s}, \{\!\{M_{i_s}\}\!\}), IH^M_{i_s-1})$
14:     | ¬identical($\{\!\{IH^M_{i_s}\}\!\}$) ⇒
15:         return $\left((\mathsf{IH2}, i_s, i_e, \{\!\{IH^M_{i_s}\}\!\}, IH^M_{i_e}), IH^M_{(i_s+i_e)/2}\right)$
```

So if they also agree on $IH^M_{i_s}$ (Line 13), they only need to go to state T1 where the cheater will be caught since it cannot provide the information to reproduce the faked $IH^M_{i_s}$. Note that T1 is a terminal state where a proof is guaranteed to be derived. In case they disagree on $IH^M_{i_s}$ (Line 14), they will enter state IH2, where they use binary search to pin down whose claim on $IH^M_{i_s}$ was wrong.

In state IH1, the honest party can directly output a proof if i_s happens to be the same as i_e. Recall that they already agree on $IH^M_{i_e}$, hence the cheater can be caught with the simple evidence that $IH^M_{i_e} \neq \text{peer}(IH^M_{i_s})$:

```
16: fun genProofBase $\left((\mathsf{IH1}, i_s, i_e, IH^M_{i_e}, \{\!\{M_{i_s}\}\!\}), \{\!\{IH^M_{i_s}\}\!\}\right)$
```

```
17:     | $i_s = i_e$ && $\textsc{peer}(\{\!\{IH^M_{i_s}\}\!\}) \neq IH^M_{i_e} \Rightarrow$ return (IH-Prf1, $rec$)
```

Given the proof (IHErr1, *rec*), the judge can check by extracting $i_s, i_e, IH^M_{i_s}, IH^M_{i_e}$ from *rec* and verifying the same predicates on which the proof was produced (Line 17).

If the parties enter the terminating state T1, they must agree on $IH^M_{i_s}$ but not on M_{i_s}. Therefore, the honest party can output (IH-Prf2, *rec*) as the proof:

```
18: fun genProofBase $rec$ @ $\left((\mathsf{T1}, IH^M_{i_s}, \{\!\{M_{i_s}\}\!\}), IH^M_{i_s-1}\right)$
19:     return (IH-Prf2, $rec$)
```

The judge can check this proof by extracting $\{\!\{M_{i_s}\}\!\}, IH^M_{i_s-1}, IH^M_{i_s}$ from *rec* and verifying $\textsc{IncH}\left(IH^M_{i_s-1}, \textsc{peer}(\{\!\{M_{i_s}\}\!\})\right) \neq IH^M_{i_s}$.

5.2.2 *State* IH2.

Entering this state with record

$$\left((\mathsf{IH2}, i_s, i_e, \{\!\{IH^M_{i_s}\}\!\}, IH^M_{i_e}), \{\!\{IH^M_{(i_s+i_e)/2}\}\!\}\right),$$

the parties agree on indices i_s, i_e (where $i_s < i_e$) and $IH^M_{i_e}$ but disagree on $IH^M_{i_s}$. They also proclaim their own values of $IH^M_{(i_s+i_e)/2}$, which may or may not be the same. Its next function can branch into three cases.

```
20: fun next $\left((\mathsf{IH2}, i_s, i_e, \{\!\{IH^M_{i_s}\}\!\}, IH^M_{i_e}), \{\!\{IH^M_{(i_s+i_e)/2}\}\!\}\right)$
21:     | $i_s + 1 = i_e \Rightarrow$ return $((\mathsf{T2}, \{\!\{IH^M_{i_s}\}\!\}, IH^M_{i_e}), M_{i_e})$
22:     | identical($\{\!\{IH^M_{(i_s+i_e)/2}\}\!\}$) $\Rightarrow$
23:         let $mid := (i_s + i_e)/2$;  $mid' := (i_s + mid)/2$ in
24:         return $((\mathsf{IH2}, i_s, mid, \{\!\{IH^M_{i_s}\}\!\}, IH^M_{mid}), IH^M_{mid'})$
25:     | $\neg$ identical($\{\!\{IH^M_{(i_s+i_e)/2}\}\!\}$) $\Rightarrow$
26:         let $mid := (i_s + i_e)/2$;  $mid' := (mid + i_e)/2$ in
27:         return $((\mathsf{IH2}, mid, i_e, \{\!\{IH^M_{mid}\}\!\}, IH^M_{i_e}), IH^M_{mid'})$
```

If $i_s + 1 = i_e$, then the disagreement between the two parties can be verified by a single call to IncH, so they will enter state T2 where the honest party obtains a proof as the cheater can't reproduce the correct IH_{i_e} from its corrupted IH_{i_s}. If the gap between i_s and i_e is bigger than 1, the parties will re-enter state IH2 halving the search range in a way based on if their claims of $IH_{(i_s+i_e)/2}$ match or not.

In state T2, the honest party simply output the record as a proof.

```
28: fun genProofBase $rec$ @ $\left((\mathsf{T2}, \{\!\{IH^M_{i_e-1}\}\!\}, IH^M_{i_e}), M_{i_e}\right)$
29:     return (IH-Prf3, $rec$)
```

A judge can verify this proof by checking

$$\textsc{IncH}\left(\textsc{peer}(\{\!\{IH^M_{i_e-1}\}\!\}), M_{i_e}\right) \neq IH^M_{i_e}.$$

5.3 Proofs from Disagreeing Claims on Ve

Let n be the size of circuit f'. To resolve a dispute on the result of Ve, the parties will enter state VE1 with record

$$\left((1, 0, n, IH^F_0, \{\!\{IH^F_n\}\!\}), \{\!\{IH^F_{n/2}\}\!\}\right)$$

with disagreeing claims on IH^F_n while having already agreed on $Seed_\iota$ (otherwise, a valid proof should have been obtained from SNROT's **Verify**). The states and transitions involved in disputing Ve are given below.

5.3.1 *State* VE1.

Entering this state with record

$$\left((\mathsf{VE1}, i_s, i_e, IH^F_{i_s}, \{\!\{IH^F_{i_e}\}\!\}), \{\!\{IH^F_{(i_s+i_e)/2}\}\!\}\right),$$

the parties agree on the indices i_s, i_e (with $i_s < i_e$) and $IH^F_{i_s}$ but disagree on $IH^F_{i_e}$. They also proclaim their respective $IH^F_{(i_s+i_e)/2}$, which may or may not match.

```
30: fun next $\left((\mathsf{VE1}, i_s, i_e, IH^F_{i_s}, \{\!\{IH^F_{i_e}\}\!\}), \{\!\{IH^F_{(i_s+i_e)/2}\}\!\}\right)$
31:     | $i_s + 1 = i_e \Rightarrow$ return $((\mathsf{VE2}, i_e, IH^F_{i_s}, \{\!\{IH^F_{i_e}\}\!\}), (f_{i_e}, F_{i_e}))$
32:     | identical($\{\!\{IH^F_{(i_s+i_e)/2}\}\!\}$) $\Rightarrow$
33:         let $mid := (i_s + i_e)/2$;  $mid' := (mid + i_e)/2$ in
34:         return $\left((\mathsf{VE1}, mid, i_e, IH^F_{mid}, \{\!\{IH^F_{i_e}\}\!\}), IH^F_{mid'}\right)$
35:     | $\neg$ identical($\{\!\{IH^F_{(i_s+i_e)/2}\}\!\}$) $\Rightarrow$
36:         let $mid := (i_s + i_e)/2$;  $mid' := (i_s + mid)/2$ in
37:         return $\left((\mathsf{VE1}, i_s, mid, IH^F_{i_s}, \{\!\{IH^F_{mid}\}\!\}), IH^F_{mid'}\right)$
```

If $i_s + 1 = i_e$, the parties have identified the first gate that caused the trouble. So they will enter state VE2 to investigate whose claim on $IH^F_{i_e}$ was wrong. Otherwise, they re-enter state VE1 with the search range reduced by half in a way depending on if their respective $IH^F_{(i_s+i_e)/2}$ match or not.

5.3.2 *State* VE2.

Entering this state with record

$$\left((\mathsf{VE2}, i, IH^F_{i-1}, \{\!\{IH^F_i\}\!\}), \{\!\{f_i\}\!\}, \{\!\{F_i\}\!\}\right),$$

the parties agree that i is the index of the first gate whose hash (over the garbled rows for AND gate, a wire-label if it is an AND, XOR or INPUT gate, and the decoding information if it is an OUTPUT gate) they have disagreed.

If both parties agree on f_i, then they extract the indices of the two input-wires of the i^{th} gate and proceed to state VE3. Note that the index $i - 1$ needs to be carried over to VE3 so that it can be used to bound the range (as the ending index) of the gates about which matching claims were made. If they disagree on f_i (Line 42), they will go to state IH1 with f_i, where they will find out who lied on the circuit description.

```
38: fun next $\left((\mathsf{VE2}, i, IH^F_{i-1}, \{\!\{IH^F_i\}\!\}), \{\!\{f_i\}\!\}, \{\!\{F_i\}\!\},\right)$
39:     | identical($\{\!\{f_i\}\!\}$) $\Rightarrow$
40:         let $(i_l, i_r)$ := inputWireIndices($f_i$) in
41:         return $((\mathsf{VE3}, i_l, i_r, i-1, IH^F_{i-1}, f_i, \{\!\{F_i\}\!\}), F_{i_l}, F_{i_r})$
42:     | $\neg$identical($\{\!\{f_i\}\!\}$) $\Rightarrow$ return $((\mathsf{IH1}, i, n, IH^f_n, \{\!\{f_i\}\!\}), IH^f_i)$
```

At VE2, an honest party outputs a proof in two cases: (1) if its peer can't provide the preimage of its (forged) IH^F_i (Line 44); (2) if gate-i is an INPUT gate, so L_i can be computed directly from $Seed_\iota$.

```
43: fun genProofBase $rec$ @ $\left((\mathsf{VE2}, i, (IH^F_{i-1}, \{\!\{IH^F_i\}\!\})), f_i, F_i\right)$
44:     | IncH($IH^F_{i-1}, F_i$) $\neq$ peer($\{\!\{IH^F_i\}\!\}$) $\Rightarrow$ return (VE-Prf1, $rec$)
45:     | gateType($f_i$) = INPUT $\Rightarrow$ return (VE-Prf2, $F_i$, $Seed_\iota$)
```

It is trivial for the judge to verify these proofs: just check the predicate in the corresponding conditions under which the proofs were produced.

5.3.3 State VE3. Entering state VE3 with record

$$\left(\mathsf{VE3}, i, j, e, IH^F_e, f_k, \{\!\{F_k\}\!\}), \{\!\{F_i\}\!\}, \left\{\!\!\left\{F_j\right\}\!\!\right\}\right),$$

the parties have agreed on IH^F_e with $i \le e$, and f_k where i, j are the indices of the input-wires of gate-k^{th}, but disagreed on F_k or L_k. To proceed, the parties inspect the actual wire-labels corresponding to i or j. If they disagree on any of the two wire-labels, they will go to state IH1 with specification of the gate, which defines the disagreeing input wire-labels.

```
46: fun NEXT $\left((\mathsf{VE3}, i, j, e, IH^F_e, f_k, \{\!\{F_k\}\!\}), \{\!\{F_i\}\!\}, \left\{\!\!\left\{F_j\right\}\!\!\right\}\right)$
47:     | $\neg$IDENTICAL($\{\!\{F_i\}\!\}$) $\Rightarrow$ return ((IH1, $i, e, IH^F_e, \{\!\{F_i\}\!\}), IH^F_i$)
48:     | $\neg$IDENTICAL($\left\{\!\!\left\{F_j\right\}\!\!\right\}$) $\Rightarrow$ return ((IH1, $j, e, IH^F_e, \left\{\!\!\left\{F_j\right\}\!\!\right\}), IH^F_j$)
```

In state VE3, an honest party finds a proof if the two parties end up agreeing on both L_i and L_j (which can be derived from F_i and F_j). In this case, it must be true that $\text{GARBLE}(f_k, L_i, L_j) \neq (\text{PEER}(\{\!\{F_k\}\!\}))$. Namely, if gate-$k$ is an XOR, then it must be that $L_i \oplus L_j \neq L_k$; if gate-k is an AND, then it must be that $\text{Gb}(L_i, L_j) \neq F_k$; if gate-$k$ is an OUTPUT gate, then it must be the case that its decoding information $(H(L^0_i), H(L^1_i)) \neq F_k$; (Note that gate-$k$ can't be an INPUT gate because if this was true, a valid proof would have already been generated at Line 45 before entering VE3.) In any case, it suffices to output *rec* as the proof.

```
49: fun GENPROOFBASE rec @ $\left((\mathsf{VE3}, i, j, (e, IH^F_e, f_k, \{\!\{F_k\}\!\})), F_i, F_j\right)$
50:     let $L_i$ = OUTPUTWIRELABEL($F_i$)
51:         $L_j$ = OUTPUTWIRELABEL($F_j$)
52:     in if GARBLE($f_k, L_i, L_j$) $\neq$ PEER($\{\!\{F_k\}\!\}$)
53:        then return (VE-Prf3, rec)
```

To verify this type of proofs, the judge simply checks the predicate in the condition (Line 52) under which the proof was produced.

5.4 Judge as a Smart-Contract

Our judge is a smart-contract that offers a number of functions for both parties to call to execute transactions. The life cycle of the smart-contract is described by a number of *stage*s depicted in Figure 10. An instance of the contract is created per instance of 2PC protocol. Once created, the contract stays in the INIT stage waiting for commitment of the security deposit. Once the `deposit` transactions are executed, the contract enters the ENGAGED stage. Only after this point should the parties start public verifiable secure computation protocol.

If the secure computation protocol finishes normally, both parties will call `conclude` to get back their deposits and be mutually released. If a party doesn't call `conclude`, its peer can still reclaim its deposit after a certain period of time (measured by Ethereum block depth).

If the evaluator catches a misbehavior in the PVC 2PC protocol in Step (3), it will call `initDispute` to move the contract forward to the MSG EXCHANGE stage where both parties are obligated to cooperate in searching a short proof via running GENPROOFINDUCTIVE. Finally, any party can submit a proof to move the contract to the JUDGING stage, where the cheater get penalized and honest party gets compensated, then finally end in the MUTUAL RELEASE stage. In this case, there will be at most logarithmic number of `sendmsg` transactions plus exactly one `verifyPrf` transaction.

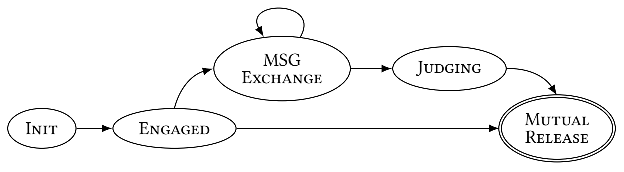

Figure 10: Lifecycle of the Judge as a Smart-Contract

6 APPLICATION: FINANCIALLY-SECURE 2PC

We propose *financially secure computation* as an interesting application enabled by PVC and blockchain technologies.

Financially Secure Computation in the Ideal Model. We can augment the ideal covert 2PC functionality $\mathcal{F}^{\text{covert}}_{\text{2PC}}$ [7] to define ideal financially secure computations $\mathcal{F}^{\text{financial}}_{\text{2PC}}$. We introduce three changes to $\mathcal{F}^{\text{covert}}_{\text{2PC}}$ to define $\mathcal{F}^{\text{financial}}_{\text{2PC}}$:

(1) When a protocol instance is created, $\mathcal{F}^{\text{financial}}_{\text{2PC}}$ receives security deposits d_1, d_2 from party P_1 and P_2, resp.
(2) At the **Input** step, in addition to receiving inputs x_1, x_2 from P_1, P_2 resp., $\mathcal{F}^{\text{financial}}_{\text{2PC}}$ also receives transfers of v_1, v_2 from P_1, P_2, resp., where v_i is the asset valuation of x_i.
(3) Upon receiving Cheat_i from P_i,
    - With probability p, P_i is caught cheating and $\mathcal{F}^{\text{financial}}_{\text{2PC}}$ transfers $(d_1 + d_2)$ to $P_{\bar{i}}$ in addition to transferring v_i to P_i.
    - With probability $(1-p)$, P_i succeeds in cheating and $\mathcal{F}^{\text{financial}}_{\text{2PC}}$ transfers $v_1 + v_2$ to P_i and refunds d_1, d_2 to P_1, P_2, resp.

So the *expected financial gain* of a cheating party P_i in the ideal model execution is $(1-p)\cdot v_{\bar{i}} - p\cdot d_i$. Since p is an adjustable parameter of $\mathcal{F}^{\text{covert}}_{\text{2PC}}$ and $\{d_i, v_i\}_{i\in[2]}$ are application specific parameters, it is easy to setup p such that $(1-p)\cdot v_{\bar{i}} - p\cdot d_i < 0$. Namely, the expected financial gain of any cheating P_i is negative. Finally, we also note that an ideal financially secure computation can be viewed as a zero-sum game between the two computation participants.

Realize $\mathcal{F}^{\text{financial}}_{\text{2PC}}$. We can realize $\mathcal{F}^{\text{financial}}_{\text{2PC}}$ using our PVC 2PC protocol of Section 4 with a judge implemented as an efficient blockchain smart-contract. More specifically, thanks to the asymmetry of the garbled circuit protocol, we only need P_1 to make a security deposit before starting the protocol. If the PVC protocol finishes without P_2 obtaining any evidence of cheating in Step (3), P_1 can finalize the contract to get its security deposit refunded. Otherwise, P_2 will submit its proof to the smart-contract to take P_1's deposit. To a rational adversary who can attack in arbitrary ways as long as the expected financial gain is positive, it is guaranteed (by the security of PVC and the blockchain network) that it cannot profit from attacking the real model protocol.

In practice, the blockchain network charges a fee for every smart-contract transaction. In our design, the judge fee incurred by the dispute resolution will be paid out of the pocket of the loser. So, we require both parties to put down an extra deposit to cover the cost of a potential dispute. This also helps to restrain a malicious P_2 from defaming an honest P_1: upon receiving a false proof, the judge will decide the proof submitter to be the cheater and transfer its security deposit to its peer.

Implications. Since it doesn't make sense for a rational adversary to attack the protocol, the relatively slower and more expensive dispute arbitration process would never need to run in practice. However, we stress that it is critically important to construct efficient smart-contract transactions to intimidate potential adversaries from attacking. As a result, the performance of the arbitration is far from as critical as the PVC protocol itself, so long as it is affordable if it ever needs to run. This is also why the techniques given in Section 5.4 is indispensable: if the parties can't afford the cost of the decentralized judge, the adversary could attack without penalty.

Comparison with maliciously-secure 2PC. There have been many works on efficient maliciously-secure 2PC protocols, using either authenticated garbling [25, 39, 42] or batched cut-and-choose [30, 43–45]. In essence, our financially-secure 2PC trades security for performance. However, since there is no single best malicious-secure 2PC protocol for all scenarios, the concrete performance benefit of our financially-secure 2PC over a malicious 2PC will depend on many factors including concrete security guarantees, network & hardware conditions, and even the application circuits. Hence, a fair comparison requires careful case-specific analysis and experiments. One may worry that the cost of interacting with a blockchain would offset the performance savings. However, we stress this will not be the case because (1) first, the dispute phase that involves almost all blockchain transactions wouldn't actually happen to rational adversaries; and (2) second, the only remaining blockchain transaction is the initial deposit transaction, which would require a substantial confirmation time (e.g., 180 seconds). However, this waiting period can either be accomplished in advance, or happen concurrently with the garbled circuit protocol execution, thus having nearly zero impact on overall performance.

7 IMPLEMENTATION AND EVALUATION

We implemented our PVC 2PC protocol in C++ and the judge in Solidity. We measured the performance of the PVC protocol using Google Compute Engine n1-standard-1 instances (1 vCPU, 3.75 GB memory). The LAN (WAN) setting used in our experiments has 2 Gbps (200 Mbps) bandwidth and 0.2 ms (40 ms) round-trip latency. We set security parameters $s = 40, \kappa = 128$.

7.1 PVC 2PC

Figure 11 shows the performance of our protocol running under different configurations. We used AES circuit as the benchmark application. Each AES cipher consists of 6800 ANDs, 26816 XORs, 256 INPUTs and 128 OUTPUTs. After our augmentation, it has 6928 ANDs, 27072 XORs, 296 INPUTs and 296 OUTPUTs. We included the lines for $n = 1$ for easier comparison with the semi-honest garbled circuit execution.

As expected, the impact of deterrence parameter n on bandwidth is hardly observable. This is because only one GC is actually sent, whereas the added bandwidth due to the additional SNROTs and GC-hashes are negligible in comparison with the size of an actual GC. On the other hand, most of run-time is spent on computing/verifying the incremental hash. For example, when $n = 20$, we observed the time spent on computing the hashes is $3.297 \times 20 = 65.9s$, which is already 93% of the overall time. Note that as n grows, the cost gap of our PVC protocol between the LAN and the WAN settings shrinks quickly. When n is 20 or larger, the cost differences are already very small.

	n	LAN (s)	WAN (s)	Bandwidth (MB)
1 AES	1	0.036	0.08	0.21
	5	0.17	0.30	0.21
	20	0.69	0.82	0.21
100 AES	1	3.64	4.43	21.1
	5	17.6	18.4	21.1
	20	70.0	70.9	21.1
1000 AES	1	36	48	211
	5	176	195	211
	20	696	717	211

Figure 11: Performance of $(1 - 1/n)$-deterrence PVC 2PC

Dispute-Unfriendly			Dispute-Friendly		
Garble	Hash	Garble & Hash	Garble	Hash	Garble & Hash
0.20 s	1.14 s	1.33 s	0.20 s	3.29 s	3.49 s

Figure 12: Generating and hashing a circuit of 100 AES

To allow efficient proof search, we used an incremental hash constructed using SHA-3, i.e., $H^1 = \mathsf{SHA3}(m_1), H^{i+1} = \mathsf{SHA3}(H^i, m_{i+1})$ for all $i \geq 1$. We also run a hash for every XOR gate. Experiments show that this makes GC hashing 2.7x slower than regular SHA256-based hashing (Figure 12). For reference, SHA256 costs 16.09 CPU cycles/byte, which is close to SHA-3 (13.37 cycles/byte), according to openssl. We stress that our slowdown is caused by the extra data (H^i) going through the SHA-3 hashing. Therefore, this slowdown factor can be reduced by decreasing the length ratio between H^i and m_i. In both cases, we observe that hashing garbled circuit is 5.7–16x slower than garbling, hence clearly the bottleneck.

Comparison with [22]. The main difference between our PVC protocol and theirs lies in P_2's input processing. [22]'s PVC runs n instances of regular actively-secure OT-extension protocol, signs all the traffic, and also requires the circuit evaluator to replay $n - 1$ of them and verify the signatures. Moreover, they required a judge to fully replay one instance of the OT-extension and verify the signature of the transcript. In contrast, our protocol only uses a single instance of correlated OT protocol and there is no need to replay it by the evaluator, nor the judge. As a result, we can process the evaluator's input 45–200x faster on LAN (14–60x on WAN) at run-time while incurring no cost on the judge for the correlated-OT (Figure 13). The $680K$ gas/wire for their judge is estimated by $2G_{\text{AES}} = 2 \times 340K$ as they need 2 AES per/wire. This cost could be reduced by replacing AES with SHA3, but at the cost of slowing down OT-extension's speed by an order-of-magnitude due to the speed gap between SHA3 (13.37 CPU cycles/byte) and AESNI (1.18 CPU cycles/byte).

7.2 Decentralized Judge

Figure 14 lists the fees in Ethereum Gas and USD for all transactions in our smart-contract. Note that the gas costs we reported in the table are the worst case costs for every transaction. The `newSession`

	n	LAN (wires/s)	WAN (wires/s)	Bandwidth (bytes/wire)	Judge's Cost (gas/wire)
[22]	5	86.1K	68.4K	160	>680K
	20	18.3K	15.0K	640	
Ours	5	**3.93M**	**0.97M**	**37**	**0**
	20				

Figure 13: Processing the evaluator's input-wires

Transaction	Gas	USD
`newSession`	363K	0.2835
`deposit`	50K	0.0389
`conclude`	69K	0.0535
`initDispute`	100K	0.0783
$\texttt{sendmsg}_{\mathsf{VE1}}$	77K	0.0598
$\texttt{sendmsg}_{\mathsf{VE2}}$	77K	0.0603
$\texttt{sendmsg}_{\mathsf{VE3}}$	82K	0.0643
$\texttt{sendmsg}_{\mathsf{IH1}}$	79K	0.0616
$\texttt{sendmsg}_{\mathsf{IH2}}$	76K	0.0597
$\texttt{sendmsg}_{\mathsf{T1}}$	79K	0.0615
$\texttt{sendmsg}_{\mathsf{T2}}$	78K	0.0605

All fees in USD are calculated assuming \$195 USD/ETH and $4 \cdot 10^{-9}$ ETH/gas observed at the time of writing this paper.

Transaction		Gas	USD
`verifyPrf`	VE1-ErrMsg	90K	0.0704
	VE2-ErrMsg	97K	0.0754
	VE3-ErrMsg	93K	0.0724
	IH1-ErrMsg	89K	0.0696
	IH2-ErrMsg	91K	0.0713
	IH-Prf1	72K	0.0564
	IH-Prf2	72K	0.0562
	IH-Prf3	72K	0.0561
	VE-Prf1	70K	0.0547
	VE-Prf2	76K	0.0592
	VE-Prf3	96K	0.0701
	OT-Prf1	104K	0.0814
	OT-Prf2	104K	0.0814
	OT-Prf3	109K	0.0847
	Timeout	54K	0.0424

Figure 14: Gas costs of smart-contract transactions

transaction initializes an instance of the smart-contract with a session ID. The `deposit` transaction engages the parties by synchronize basic information about the protocol like the circuit size and the hash of f. The `conclude` transaction gets the parties mutually released when no dispute occurs. If the PVC protocol runs with no dispute, these are the only three transactions to be executed, which total at 482K gas at most, or roughly \$0.38 USD.

In case an arbitration is needed, the `initDispute` transaction gets the parties synchronized on a few values (e.g., the channel timer, the $Seed_\iota$ etc) commonly needed for the rest of the transactions during dispute. The `sendmsg` transactions enable the two parties to exchange messages when searching for a succinct proof (Step (3) of Algorithm 1). Because of the variation in the message format, there are 7 different `sendmsg` transactions, one for each of the 7 states. Finally, the dispute will be wrapped up by a single `verifyPrf` transaction, which can be any of the 15 types of proofs given on the right part of Figure 14. We stress that, out of these transactions, only `sendmsg` needs to be executed more than once.

Although a dispute will never have to happen when running the protocol with rational adversaries, it is easy to estimate an upper-bound of a dispute. Take a billion-gates (2^{30}) circuit PVC protocol as an example, it will require at most $G_{\texttt{initDispute}} + 30G_{\texttt{sendmsg}_{\mathsf{VE1}}} + G_{\texttt{sendmsg}_{\mathsf{VE2}}} + G_{\texttt{sendmsg}_{\mathsf{VE3}}} + G_{\texttt{sendmsg}_{\mathsf{IH1}}} + 30G_{\texttt{sendmsg}_{\mathsf{IH2}}} + G_{\texttt{verifyPrf}} <$ 5.04M gas, or \$3.93 USD per party to resolve a dispute. That is, each party only needs to deposit $\$3.93 \times 2 = \$7.86$ USD so we are sure that the loser can pay all arbitration fees. Also, if the timeout threshold in the dispute process is set to 12 blocks (roughly 3 minutes), then the arbitration can be done in $3 \times (63 \times 2 + 2) = 384$ minutes, or roughly 6.4 hours in the worst case. Since arbitration occurs very rarely, we consider that delay to be acceptable in many scenarios. After all, honest players will not lose money.

Finally, we measured that it costs 6M and 5M gas to upload the smart-contract and its library to the Ethereum blockchain (roughly \$9 USD in total). However, we note this needs to be done *only once* for all instances of the PVC protocol and all users in the world.

8 CONCLUSION

We improved the art of designing two-party PVC protocols. Most importantly, our protocol allows an efficient, practically affordable, decentralized judge. We propose the concept of financial security, which brings up the performance of 2PC by weighing the assessed values of secrets against the amount of money a potential adversary has deposited. We hope our approach would popularize MPC for business applications.

ACKNOWLEDGMENTS

We thank Google for the generous free credits for Google Cloud Platform used in our experiments. This work is partly supported by NIH award 1U01EB023685-01.

REFERENCES

[1] (accessed May 11, 2019). Bitcoin.org. https://bitcoin.org/.
[2] (accessed May 11, 2019). Ethereum.org. https://ethereum.org/.
[3] Marcin Andrychowicz, Stefan Dziembowski, Daniel Malinowski, and Lukasz Mazurek. 2014. Secure multiparty computations on bitcoin. In *IEEE Symposium on S&P*.
[4] Gilad Asharov, Ran Canetti, and Carmit Hazay. 2011. Towards a game theoretic view of secure computation. In *EUROCRYPT*.
[5] Gilad Asharov, Yehuda Lindell, Thomas Schneider, and Michael Zohner. 2015. More efficient oblivious transfer extensions with security for malicious adversaries. In *EUROCRYPT*.
[6] Gilad Asharov and Claudio Orlandi. 2012. Calling out cheaters: Covert security with public verifiability. In *ASIACRYPT*.
[7] Yonatan Aumann and Yehuda Lindell. 2010. Security against covert adversaries: Efficient protocols for realistic adversaries. *Journal of Cryptology* (2010).
[8] Mihir Bellare, Oded Goldreich, and Shafi Goldwasser. 1994. Incremental cryptography: The case of hashing and signing. In *CRYPTO*.
[9] Mihir Bellare, Viet Tung Hoang, Sriram Keelveedhi, and Phillip Rogaway. 2013. Efficient garbling from a fixed-key blockcipher. In *IEEE Symposium on S&P*.
[10] Mihir Bellare, Viet Tung Hoang, and Phillip Rogaway. 2012. Foundations of garbled circuits. In *ACM CCS*.
[11] Iddo Bentov and Ranjit Kumaresan. 2014. How to use bitcoin to design fair protocols. In *CRYPTO*.
[12] Iddo Bentov, Ranjit Kumaresan, and Andrew Miller. 2017. Instantaneous decentralized poker. In *ASIACRYPT*.
[13] Ran Canetti and Rafail Ostrovsky. 1999. Secure computation with honest-looking parties: What if nobody is truly honest?. In *ACM STOC*.
[14] Ran Canetti, Ben Riva, and Guy N Rothblum. 2011. Practical delegation of computation using multiple servers. In *ACM CCS*.
[15] Bernardo David, Rafael Dowsley, and Mario Larangeira. 2017. Kaleidoscope: An Efficient Poker Protocol with Payment Distribution and Penalty Enforcement. *IACR Cryptology ePrint Archive* (2017).
[16] Tore Frederiksen, Jesper Nielsen, and Claudio Orlandi. 2015. Privacy-free garbled circuits with applications to efficient zero-knowledge. In *EUROCRYPT*.
[17] Georg Fuchsbauer, Jonathan Katz, and David Naccache. 2010. Efficient rational secret sharing in standard communication networks. In *TCC*.
[18] Juan Garay, Jonathan Katz, Ueli Maurer, Björn Tackmann, and Vassilis Zikas. 2013. Rational protocol design: Cryptography against incentive-driven adversaries. In *IEEE FOCS*.
[19] Vipul Goyal, Payman Mohassel, and Adam Smith. 2008. Efficient two party and multi party computation against covert adversaries. In *EUROCRYPT*.
[20] Shay Gueron, Yehuda Lindell, Ariel Nof, and Benny Pinkas. 2015. Fast garbling of circuits under standard assumptions. In *ACM CCS*.
[21] Joseph Halpern and Vanessa Teague. 2004. Rational secret sharing and multiparty computation. In *ACM STOC*.

[22] Cheng Hong, Jonathan Katz, Vladimir Kolesnikov, Wen-jie Lu, and Xiao Wang. 2019. Covert Security with Public Verifiability: Faster, Leaner, and Simpler. In *EUROCRYPT*.

[23] Yuval Ishai, Joe Kilian, Kobbi Nissim, and Erez Petrank. 2003. Extending oblivious transfers efficiently. In *CRYPTO*.

[24] Harry Kalodner, Steven Goldfeder, Xiaoqi Chen, S Matthew Weinberg, and Edward W Felten. 2018. Arbitrum: Scalable, private smart contracts. In *USENIX Security Symposium*.

[25] Jonathan Katz, Samuel Ranellucci, Mike Rosulek, and Xiao Wang. 2018. Optimizing authenticated garbling for faster secure two-party computation. In *CRYPTO*.

[26] Marcel Keller, Emmanuela Orsini, and Peter Scholl. 2015. Actively secure OT extension with optimal overhead. In *CRYPTO*.

[27] Aggelos Kiayias, Hong-Sheng Zhou, and Vassilis Zikas. 2016. Fair and robust multi-party computation using a global transaction ledger. In *EUROCRYPT*.

[28] Gillat Kol and Moni Naor. 2008. Games for exchanging information.. In *ACM STOC*.

[29] Vladimir Kolesnikov and Alex J Malozemoff. 2015. Public verifiability in the covert model (almost) for free. In *ASIACRYPT*.

[30] Vladimir Kolesnikov, Jesper Buus Nielsen, Mike Rosulek, Ni Trieu, and Roberto Trifiletti. 2017. DUPLO: unifying cut-and-choose for garbled circuits. In *ACM CCS*.

[31] Ranjit Kumaresan and Iddo Bentov. 2014. How to use bitcoin to incentivize correct computations. In *ACM CCS*.

[32] Ranjit Kumaresan and Iddo Bentov. 2016. Amortizing secure computation with penalties. In *ACM CCS*.

[33] Ranjit Kumaresan, Tal Moran, and Iddo Bentov. 2015. How to use bitcoin to play decentralized poker. In *ACM CCS*.

[34] Satoshi Nakamoto. 2008. Bitcoin: A Peer-to-Peer Electronic Cash System. (2008).

[35] Jesper Nielsen, Thomas Schneider, and Roberto Trifiletti. 2017. Constant-Round Maliciously Secure 2PC with Function-Independent Preprocessing Using LEGO. In *NDSS*.

[36] Chris Peikert, Vinod Vaikuntanathan, and Brent Waters. 2008. A framework for efficient and composable oblivious transfer. In *CRYPTO*.

[37] Stefan Tillich and Nigel Smart. 2016 (accessed May 11, 2019). Circuits of Basic Functions Suitable For MPC and FHE. https://homes.esat.kuleuven.be/~nsmart/MPC/.

[38] Jelle van den Hooff, M Frans Kaashoek, and Nickolai Zeldovich. 2014. Versum: Verifiable computations over large public logs. In *ACM CCS*.

[39] Xiao Wang, Samuel Ranellucci, and Jonathan Katz. 2017. Authenticated garbling and efficient maliciously secure two-party computation. In *ACM CCS*.

[40] Gavin Wood et al. 2019. Ethereum: A secure decentralised generalised transaction ledger. *Ethereum project yellow paper* Byzantium Version (2019).

[41] Samee Zahur, Mike Rosulek, and David Evans. 2015. Two halves make a whole. In *EUROCRYPT*.

[42] Ruiyu Zhu, Darion Cassel, Amr Sabry, and Yan Huang. 2018. NANOPI: extreme-scale actively-secure multi-party computation. In *ACM CCS*.

[43] Ruiyu Zhu and Yan Huang. 2017. JIMU: faster LEGO-based secure computation using additive homomorphic hashes. In *International Conference on the Theory and Application of Cryptology and Information Security*. Springer, 529–572.

[44] Ruiyu Zhu, Yan Huang, and Darion Cassel. 2017. Pool: scalable on-demand secure computation service against malicious adversaries. In *ACM CCS*.

[45] Ruiyu Zhu, Yan Huang, Jonathan Katz, and Abhi Shelat. 2016. The cut-and-choose game and its application to cryptographic protocols. In *USENIX Security*.