Efficient, Constant-Round and Actively Secure MPC: Beyond the Three-Party Case

Nishanth Chandran* Juan Garay†‡ Payman Mohassel§‡ Satyanarayana Vusirikala¶

Abstract

While the feasibility of constant-round and actively secure MPC has been known for over two decades, the last few years have witnessed a flurry of designs and implementations that make its deployment a palpable reality. To our knowledge, however, existing concretely efficient MPC constructions are only for up to three parties.

In this paper we design and implement a new actively secure 5PC protocol tolerating two corruptions that requires 8 rounds of interaction, only uses fast symmetric-key operations, and incurs 60% less communication than the passively secure state-of-the-art solution from the work of Ben-Efraim, Lindell, and Omri [CCS 2016]. For example, securely evaluating the AES circuit when the parties are in different regions of the U.S. and Europe only takes 1.8s which is 2.6× faster than the passively secure 5PC in the same environment.

Instrumental for our efficiency gains (less interaction, only symmetric key primitives) is a new 4-party primitive we call *Attested OT*, which in addition to Sender and Receiver involves two additional "assistant parties" who will attest to the respective inputs of both parties, and which might be of broader applicability in practically relevant MPC scenarios. Finally, we also show how to generalize our construction to n parties with similar efficiency properties where the corruption threshold is $t \approx \sqrt{n}$, and propose a combinatorial problem which, if solved optimally, can yield even better corruption thresholds for the same cost.

1 Introduction

Secure multiparty computation (MPC) allows a group of parties with private inputs to compute a joint function of their inputs correctly, despite the potential misbehavior of some of them, and without revealing any information beyond what can be inferred from the outcome of the computation. Since the seminal results from the 1980s [Yao82, GMW87, BGW88, CCD88], which demonstrated the feasibility of general-purpose MPC for computing arbitrary functions, a large body of work has focused on improving both asymptotic and concrete efficiency of these feasibility results, in particular in the last few years. For constant-round MPC, which is the focus of this work, the main ingredient of most existing constructions is Yao's garbled circuit protocol [Yao82] and its multi-party variant [BMR90]. In the two-party case, Yao's original passively secure construction can be efficiently transformed into an actively secure one by applying the cut-and-choose paradigm (e.g., [MF06, LP07]) which has been extensively studied and optimized over the last decade or so (cf. [Woo07, LPS08, NO09, PSSW09, asS11, LP12, KasS12, HKE13, Lin13, MR13, asS13, LR14,

*Microsoft Research, India; Email: nichandr@microsoft.com

†Texas A&M University; Email: juan.a.garay@gmail.com

‡Work done in part while at Yahoo! Research.

§Visa Research; Email: payman.mohassel@gmail.com

¶University of Texas, Austin; Email: satya.vus@gmail.com. Work done while at Microsoft Research India.

Comparison of different MPC protocols				
Protocol	No. corr.	Comm.	Assumption	Adversary
[IKP10]	$t < \frac{n}{3}$	$\mathcal{O}(\kappa n^3 \lvert C \rvert)$	PRG	Malicious
[LPSY15]	$t < n$	$\mathcal{O}(\kappa^2 n^2 \lvert C \rvert)$	PRG+OT	Malicious
[BLO16b]	$t < n$	$\mathcal{O}(\kappa n^2 \lvert C \rvert)$	PRG+OT	Semihonest
[KRW17]	$t < n$	$\mathcal{O}(\frac{\kappa n^2 s \lvert C \rvert}{\log \lvert C \rvert})$	PRG+OT	Malicious
[HSSV17]	$t < n$	$\mathcal{O}(\frac{\kappa n^2 s \lvert C \rvert}{\log \lvert C \rvert})$	PRG+OT	Malicious
Ours	$t < \sqrt{n}$	$\mathcal{O}(\kappa n^2 t \lvert C \rvert)$	PRG+OT	Malicious

Table 1: Comm. complexity of constant-round MPC

HKK+14, LR15]), and even extended to the three-party case tolerating two corruptions [CKMZ14]. This paradigm, however, is a multiplicative factor of s more expensive than the passively secure variant in both computation and communication, where s denotes a statistical security parameter typically set to a value between 40 and 128, depending on the intended security level. Recent work by Mohassel *et al.* [MRZ15] and Ishai *et al.* [IKKP15] show that with only one corruption, actively secure 3PC can be obtained without this multiplicative overhead. In fact, they propose constructions that are as efficient as the passive Yao's garbled circuit protocol for two parties.

To our knowledge, prior to this work it was not known whether the same level of efficiency could be obtained beyond the three-party case. Even in the case of five parties with only two corruptions, the existing constant-round MPC constructions are either only secure against passive adversaries [BLO16b], or incur a multiplicative overhead (in the security parameter) in both computation and communication, as is the case with concurrent and independent works [KRW17, HSSV17]).

1.1 Our contributions

In this work, we design an actively secure 5PC protocol with security against two corruptions that requires 8 rounds of interactions, only uses fast symmetric-key operations (i.e., no use of Oblivious Transfer [OT]), and incurs 60% less communication compared to the state-of-the-art solution of Ben-Efraim *et al.* [BLO16b], which is only passively secure (against 4 corruptions). Instrumental in our construction is a new 4-party primitive we call *Attested OT* (AOT), which in addition to Sender and Receiver involves two additional "assistant parties" who will attest to the respective inputs of both parties.

We also show how to generalize our construction to a larger number of parties n with similar efficiency properties where the corruption threshold is $t \approx \sqrt{n}$. In fact, we also formulate a combinatorial problem that, if solved optimally, can yield even higher corruption thresholds than we currently obtain (our current solution requires the use of oblivious transfer for n parties; however a better assignment can eliminate the need for OT). With t denoting the number of corruptions (as a function of n), Comm. denoting communication complexity and κ, s denoting the computational and statistical security parameters, respectively, the communication complexity of our protocol and its comparison with other recent works on constant-round MPC is provided in Table 1. Note that the protocol by Ishai *et al.* [IKP10] is the only prior work with active security and asymptotic complexity close to ours, but which does not yield a 5PC with two corruptions (since $t < n/3$), and its concrete efficiency is not well-understood.

We have implemented our actively secure 5PC protocol (with up to two active corruptions; denoted by 5PC-M for brevity) as well as a simpler passively secure variant (denoted by 5PC-

SH), and compare their performance to the state-of-the-art implementation of [BLO16b] when run with five parties and passive security (with up to four corruptions). 5PC-SH requires 8× less communication, while 5PC-M incurs 60% less communication compared to [BLO16b]. For medium- to high-latency networks (i.e., machines across US and machines in the US and Europe, respectively), where constant-round protocols are more suitable, 5PC-SH evaluates the AES and SHA circuits $2.6 - 4.8\times$ faster than [BLO16b], while 5PC-M is a factor of $1.7 - 2.6\times$ faster than [BLO16b]. As a concrete example, securely evaluating the AES circuit with machines located across the US and Europe takes 5PC-M 1.88s, while the [BLO16b] protocol runs in 4.86s.

1.2 Technical overview

Our starting point is the actively secure 3PC protocol with abort by Mohassel *et al.* [MRZ15], whose idea was to designate one party P_3 as the evaluator and the other two parties (P_1, P_2) as circuit garblers. Since at most one party is corrupted, one garbler is always honest. Hence, they have both garblers generate the garbled circuit using a seed that they agree upon and have P_3 check equality of the garbled circuits before proceeding with garbled input generation and evaluation. This ensures honest garbled circuit generation, and with a little more work to get maliciously secure garbled input generation, they obtain an actively secure 3PC protocl with essentially no additional communication cost compared to semihonest 2PC using garbled circuits.

Generalizing this approach to the five-party case and beyond quickly runs into major technical challenges. Consider the following naïve generalization, where we designate, say, P_5 as the evaluator, and $P_1, \ldots, P_4$ as the garblers. We can have the garblers agree upon a seed s, and invidividually generate the garbled circuit using s and send this to P_5. P_5 would then check the equality of the circuits and rejects if they do not match. Now let us assume that P_5 somehow receives the garbled inputs for all parties (we will see that this has its own challenge). This approach fails since if the two corrupted parties are P_5 and one of the garblers, then the two of them combined learn both the seed (and therefore the garbled circuit secrets) and all other parties garbled inputs which they can combine to recover everyone's inputs.

A more promising approach is to have the garblers generate the garbled circuit in a distributed manner [BMR90, DI05, CKMZ14, BLO16b] so that no single garbler would learn the secrets, but then the challenge in this distributed setting is to obtain security against malicious garblers. Unfortunately, the existing solutions do not provide the concrete efficiency we are aiming for and incur a significant overhead (at least multiplicative in security parameter) compared to the semi-honest variant.

4-party malicious circuit garbling. To get around the above technical challenge, we design a new 4-party distributed garbling scheme with the properties that (i) if only one garbler is corrupted, then the garbled circuit is correct and its secrets remain hidden from the adversary, and (ii) if two garblers are corrupted, the garbled circuit remains correct but the adversary learns its secrets. This is sufficient for 5PC, since in the case of two corrupted garblers, the evaluator P_5 is guaranteed to be honest, and hence the only guarantee we need is the correctness of the garbled circuit.

Our starting point is a semi-honest 4-party distributed garbling scheme (4DG) in the same spirit as [DI05, CKMZ14, BLO16b] that takes place between the four garblers $P_1, \ldots, P_4$. We assume that all the randomness needed by P_i is generated using a random seed s_i. We now distribute these seeds among the four garblers $(P_1, \ldots, P_4)$ such that the seed generated by P_i is known to *two* other parties, and at the same time no single party has knowledge of all four seeds. In particular, the following assignment works where $\mathcal{S}_i$ denotes the set of indices of parties with knowledge of s_i: $\mathcal{S}_1 = \{1, 3, 4\}$, $\mathcal{S}_2 = \{2, 3, 4\}$, $\mathcal{S}_3 = \{1, 2, 3\}$, and $\mathcal{S}_4 = \{1, 2, 4\}$. The intuition is that all the

computation and communication generated based on each s_i can be perfomed by three parties and checked against each other for correctness. With at most two corruptions, at least one of the parties is honest and hence any malicious behavior is caught.

In principle, one can turn this idea into a compiler that transforms the semi-honest 4DG into a 4DG with malicious security tolerating two corruptions as discussed above. However, the resulting protocol would still not be as efficient as we want it to be. For example, this requires treating the many two-party OTs performed in the distributed garbling in a non-black-box way and checking the messages sent/received during the OTs among three designated parties, which is expensive.

Instead, we show how to replace each two-party OT in the semi-honest 4DG with a new protocol for four parties we call *Attested OT* (AOT), wherein one party is the sender, another is the receiver, and two other parties are "attesters" whose role is to check honest behavior by sender and receiver. We design such a protocol using only symmetric-key operations (i.e., commitments), and show that in the multiple-instance/batch setting (when many such OTs are performed), the amortized communication cost is that of sending two commitments and one decommitment. This gives us a protocol that is based solely on PRGs (in a black-box manner). In addition, we describe a specialized commitment with better efficiency, based on AES and secure in the "ideal cipher model" and also a batched version of our attested OT protocol (that additionally assumes the existence of collision-resistant hash functions). We note here, that instead of using attested OTs, one can also replace the OTs with OT extension protocols ([IKNP03]). However, some advantages of our attested OT protocol over OT extension are: a) fewer rounds (1 vs 3), b) less communication, and c) weaker hardness assumption.

As a result of these optimizations, we obtain a maliciously secure 4DG protocol with a very small overhead compared to the semi-honest approach. The garblers send the garbled circuits to P_5 for evaluation (with parties sending hashed copies of each other's shares to enable P_5 to check the correctness of the garbled circuit).

Garbled input generation. To enable P_5 to learn the garbled inputs, it is possible for us to have the parties peform the garbled input generation using a maliciously secure 5PC protocol since the cost is only proportional to the input size. However, doing so will be inefficient and would also require the use of public-key operations (which we wish to avoid). To obtain more efficient garbled input generation, we consider two separate cases: One for obtaining the garbled inputs for the garblers, and another for obtaining the evaluator's garbled input. In the former, each garbler P_i can generate the parts of the garbled circuit for which it has the seeds, but needs the other garblers' help to generate the missing parts. To do this, P_i secret-shares his input bit with the other garblers (who have the seed that P_i is missing). These garblers will compute the "garbled labels" on these shares and we show that these shares can be combined in a "homomorphic" manner to obtain P_i's input shares. This idea does not quite completely work and runs into subtleties, as a malicious P_5 colluding with one of the garblers can learn both labels corresponding to P_i's inputs. To defeat this, we have an additional step where the garblers mask their shares with secret-sharings of 0. To generate P_5's garbled input, we reduce the problem to the previous case, by having P_5 secret-share its input between at least three garblers. This almost works, except that in order to prevent the garblers from lying about their share of P_5's input, we require the garblers to commit to all the labels and have them open to the "correct" shares (this technique is similar to that used in [MRZ15] in the context of 3 parties).

Generalizing to more than five parties. We now present the high-level idea behind extending the above techniques to arbitrary n. The idea, as before, is to designate $n-1$ parties as garblers

and one party as the evaluator. The garblers will be given q seeds to PRFs such that: a) No $t-1$ of the garblers have all seeds; b) every pair of seeds is held by at least one garbler; and c) every seed is held by at least $t+1$ parties. The reason for this assignment is as follows: Requiring that no $t-1$ of the garblers have all seeds ensures that when $t-1$ garblers and the evaluator are corrupt, the privacy (and correctness) of the distributed garbled circuit is guaranteed which leads to security of the nPC. When every pair of seeds is held by at least one garbler, then this garbler can act as the "attester" in our AOT protocol described earlier, and hence we can replace standard OTs with AOTs (this is not a strict requirement but yields more efficient protocols). Finally, requiring that every seed is held by at least $t+1$ parties ensures that when t garblers are corrupt, there will be at least one honest party that computes the "right" message and hence the (honest) evaluator will never get an incorrect garbled circuit. The last condition is necessary only for actively secure nPC. Realizing the above requires us to obtain an assignment of q seeds to $n-1$ garblers with the above properties, which we call the (n, t, q)-*assignment problem*. We show how this can be done, in general, with $q \approx n$ and $t \approx \sqrt{n}$. We leave as an interesting open question to solve the assignment optimally.

1.3 Related work

As discussed above, a large body of work has studied efficiency of constand-round MPC based on the seminal works of [Yao82, BMR90], e.g., [DI05, BDNP08, GMS08, IKP10, BLO16b]. In the passive case, the first implementation of constant-round MPC is due to FairplayMP [BDNP08], and the state-of-the-art implementation is due to [BLO16b]. In the active case, the most efficient constructions are due to recent and concurrent work by Hazay *et al.* and Katz *et al.* [KRW17, HSSV17]. As discussed earlier, however, these constructions consider a dishonest majority and as a result have an additional multiplicative factor of overhead in security parameter compared to our solution. More efficient constructions for the 3-party case appeared in [CKMZ14, MRZ15, IKKP15]. We compare the asymptotic efficiency of our protocol with the works most related to ours (security with abort – in the case of active security) in Table 1.

In the case of MPC with round complexity proportional to the depth of the circuit, two different lines of research have been pursued, which we now briefly overview. In the cryptographic setting, building on the seminal work by Goldreich *et al.*[GMW87], offline-generated (authenticated) multiplication triplets are used to perform secure computation in a fast online phase [CHK+12, DPSZ12, DKL+13, KSS13], with a few recent works particularly focusing on the three-party case [AFL+16, FLNW16]. In the information-theoretic setting, building on [BGW88, CCD88], MPC is achieved using secret-sharing techniques, with several recent work focusing on better efficiency in the three-party case [BLW08, LDDAM12, ZSB13, CMF+14]. Finally, our seed sharing techniques can be seen as being similar in spirit to the notion of *replicated secret sharing* from [CDI05].

2 Preliminaries

We let κ denote the security parameter, and use $x \xleftarrow{\$} S$ to denote choosing a value uniformly at random from set S, and $||$ to denote concatenation of two strings. When denoting message spaces, we abuse notation and use M for unspecified message spaces that will be clear from the context.

Model and security definition. We will argue the security of our constructions in the simulation paradigm [GMW87, Can00, Can01]. For simplicty, we will follow Canetti's formulation for execution of multi-party cryptographic protocols [Can00], where the execution of a protocol by

a set of parties $P_1, \ldots, P_n$ proceeds in rounds, with inputs provided by an environment program denoted by $\mathcal{Z}$. Here we provide an abridged formulation of security in such framework. All parties are modelled as non-uniform interactive Turing machines (ITMs); further, we will be focusing on the case $n = 5$, for which we provide concrete performance measures, although our approach works for general n. An adversary $\mathcal{A}$, who interacts with and acts as instructed by the environment, at the beginning of the protocol "corrupts" a fraction of the parties (i.e., we consider *static* security); in the specific case $n = 5$, the adversary corrupts up to *two* of them. (See Section 7 for the achieved thresholds for arbitrary values of n.) These corrupted parties are under the control of the adversary, and can actively and arbitrarily deviate from the protocol specification. The environment receives the complete view of all adversarial parties in the interaction. At the end of the interaction, the environment outputs a single bit.

We now define two interactions. In the *real* interaction, the parties run a protocol Π in the presence of $\mathcal{A}$ and $\mathcal{Z}$, with input z, $z \in \{0,1\}^*$. Let $\mathsf{REAL}_{\pi,\mathcal{A},\mathcal{Z}}$ denote the binary distribution ensemble describing $\mathcal{Z}$'s output in this interaction. In the *ideal* interaction, parties send their inputs to an additional entity, a trusted *functionality* machine $\mathcal{F}$ that carries the desired computation truthfully. Let $\mathcal{S}$ (the *simulator*) denote the adversary in this idealized execution, and $\mathsf{IDEAL}_{\mathcal{F},\mathcal{S},\mathcal{Z}}$ the binary distribution ensemble describing $\mathcal{Z}$'s output after interacting with adversary $\mathcal{S}$ and ideal functionality $\mathcal{F}$.

A protocol Π is said to *securely realize* a functionality $\mathcal{F}$ if for every adversary $\mathcal{A}$ in the real interaction, there is an adversary $\mathcal{S}$ in the ideal interaction, such that no environment $\mathcal{Z}$, on any input, can tell the real interaction apart from the ideal interaction, except with negligible probability (in the security paramete κ). More precisely, if the two binary distribution ensembles above are computationally indistinguishable.

In this paper we will consider the secure computation of non-reactive functions (also known as *secure function evaluation*—SFE), represented by Boolean circuits (see below), and *allowing abort*, as instructed by the adversary. We will denote the ideal computation of $5-$ary function f with abort by $\mathcal{F}_{\text{SFE}}^{f}(\mathcal{P})$, where $\mathcal{P} = \{P_1, P_2, P_3, P_4, P_5\}$. Finally, protocols typically invoke other sub-protocols. In this framework the *hybrid model* is like a real interaction, except that some invocations of the sub-protocols are replaced by the invocation of an instance of an ideal functionality $\mathcal{F}$; this is called the "$\mathcal{F}$-hybrid model." We will perform such replacements, but some times, for the sake of efficiency, we will break away from modular/black-box composition rules, and thus it will be more convenient for us to express the security of components in a property-based fashion.

Cryptographic building blocks. Our constructions make use of a a pseudorandom function family, a collision-resistant hash function, and a secure (non-interactive) commitment scheme. Our protocol also makes use of variant of Oblivious Transfer we introduce in this paper, called *Attested OT*, which we describe in Section 3. We present the definitions and security of the primitives above in Appendix A.

Distributed circuit garbling schemes. In this paper we will follow the *circuit-garbling* approach to secure computation [Yao82], and in particular distributed multi-party garbling (cf. [BMR90, DI05]). First, we present some notation and correctness properties of garbling schemes, following Bellare *et al.* [BHR12]. Given the circuit representation of the function to be garbled f, a garbling scheme $\mathcal{G} = (\mathsf{Gb}, \mathsf{En}, \mathsf{De}, \mathsf{Ev}, f)$ consists of the following randomized functions:

- Garbling function $\mathsf{Gb}(f, 1^\kappa)$ outputs three strings (Gf, e, d);
- encoding function $\mathsf{En}(e, \cdot)$ that maps an initial input x to a garbled input $X = \mathsf{En}(e, x)$;

- evaluation function $\mathsf{Ev}(\mathsf{Gf}, \cdot)$ that maps every garbled input X to a garbled output $Y = \mathsf{Ev}(\mathsf{Gf}, X)$; and
- decoding function $\mathsf{De}(d, \cdot)$ that maps garbled output Y to a final output $y = \mathsf{De}(d, Y)$.

Bellare *et al.* [BHR12] formulate a series of properties for circuit-garbling schemes. In this paper we will be specifically interested in the following:

Definition 1. *We say that* $\mathcal{G} = (\mathsf{Gb}, \mathsf{En}, \mathsf{De}, \mathsf{Ev}, f)$ *is a* correct circuit garbling scheme *if for all functions* f*, and for all inputs* x *in the domain of* f*,* $\mathsf{De}(d, \mathsf{Ev}(\mathsf{Gf}, \mathsf{En}(e, x))) = f(x)$*, where* (Gf, e, d) *is the output of* $\mathsf{Gb}(f, 1^\kappa)$*.*

Several recent works on concretely efficient MPC (e.g., [CKMZ14, BLO16b]) use and instantiate circuit-garbling schemes that are computed by multiple parties in a distributed manner based on Oblivious Transfer. The functionality of such distributed garbling schemes is described in Figure 9 in Appendix C (our specific distributed garbling function will be described later). Our construction will be using the semi-honest distributed garbling protocol due to Ben-Efraim *et al.* [BLO16b] (which includes the free-XOR optimization) in order to obtain an actively secure distributed garbling protocol. At a very high level, their protocol allows parties to compute the distributed garbling function with no communication for all XOR gates and requires every party to perform (roughly) $2n$ bit-OTs and $8n$ string-OTs for every AND gate in the circuit. For more details, we refer the reader to [BLO16b]; however, the description of our version of the distributed garbling protocol will not assume prior familiarity with [BLO16b].

3 Attested Oblivious Transfer

While our MPC protocol generalizes to n parties (as described in Section 7), it will be easier to consider the specific case of 5-party MPC, where the adversary actively corrupts at most 2 parties at the beginning of the protocol; our experimental results will also focus on this specific case. We now define two specific 4−party functionalities – *Attested OT* (AOT) and *Batch Attested OT* (B-AOT). AOT can be viewed as an OT protocol between a sender and a receiver, with the additional help of two "assistant parties" who will attest to the respective inputs of both parties, while B-AOT, as its name indicates, is the combined/amortized version of AOTs of multiple instances. These functionalities will help us instantiate efficient malicious variants of OT in our distributed garbling process with less interaction and using only symmetric-key primitives. Throughout the following discussion, we will assume that the public commitment key (obtained by executing $\mathsf{ComGen}(1^\kappa)$) as well as the key for the collision-resistant hash function H (obtained when sampling H from $\mathcal{H}$) are publicly available to all parties.

3.1 Attested OT

The ideal functionality for Attested OT is presented in Figure 1. P_1 is the sender with input (m_0, m_1) and P_2 is the receiver with a bit value b. P_3 and P_4 are the *attesters*: they obtain copies of both P_1 and P_2's inputs and will help P_1 and P_2 perform the OT functionality. We present an AOT protocol secure against active corruptions in Figure 2 and prove security of the protocol in Lemma 2. When only considering a passive adversary, a much simpler information-theoretic AOT protocol with only one attester suffices. We describe this simple protocol in Figure 10 of Appendix C for completeness. In describing the functionality and protocols, we assume that attesters receive their copies of inputs from P_1 and P_2 in each execution. When invoked in our 4-party distributed garbling, however, the attesters obtain a random seed from P_1 and P_2 at the beginning of the

protocol and then use it to derive inputs to all future invocations without interaction. We prove that our protocol is secure against malicious adversaries by showing the lemma below (proof in Appendix B.

Functionality $\mathcal{F}_{\text{4AOT}}(P_1, P_2, \{P_3, P_4\})$

$\mathcal{F}_{\text{4AOT}}$ interacts with parties P_1, P_2, P_3, P_4 and the adversary $\mathcal{S}$, with P_1 and P_2 acting as sender and receiver, respectively, and P_3, P_4 as attesters.

- On input message $(\texttt{Sender}, sid, m_0, m_1)$ from P_1, where each $m_i \in \mathsf{M}$, record (m_0, m_1) and send $(\texttt{Sender}, sid, m_0, m_1)$ to P_3 and P_4 and $(\texttt{Sender}, sid)$ to the adversary. Ignore further $(\texttt{Sender}, sid, \cdot, \cdot)$ messages.
- On input message $(\texttt{Receiver}, sid, b)$ from P_2, where $b \in \{0, 1\}$, record b and send $(\texttt{Receiver}, sid, b)$ to P_3 and P_4 and $(\texttt{Receiver}, sid)$ to the adversary. Ignore further $(\texttt{Receiver}, sid, \cdot)$ inputs.
- On input message $(\texttt{Attester}, sid, m_0^j, m_1^j, b^j)$ from P_j, $j \in \{3, 4\}$, where each $m_i^j \in \mathsf{M}$, if $(\texttt{Sender}, sid, \cdot, \cdot)$ and $(\texttt{Receiver}, sid, \cdot)$ have not been recorded, ignore this message; otherwise, record (m_0^j, m_1^j, b^j) and send $(\texttt{Attester}, sid)$ to the adversary. Ignore further $(\texttt{Attester}, sid, \cdot, \cdot, \cdot)$ messages.
- On input message $(\texttt{Output}, sid)$ from the adversary, if $(m_0, m_1, b) \neq (m_0^3, m_1^3, b^3)$ or $(m_0, m_1, b) \neq (m_0^4, m_1^4, b^4)$, send $(\texttt{Output}, sid, \perp)$ to P_2; else send $(\texttt{Output}, sid, m_b)$ to P_2.
- On input message $(\texttt{Abort}, sid)$ from the adversary, send $(\texttt{Output}, sid, \perp)$ to P_2.

Figure 1: *The 4-party Attested OT ideal functionality $\mathcal{F}_{\text{4AOT}}$.*

Lemma 2. *Assuming* $(\mathsf{ComGen}, \mathsf{Com}, \mathsf{Open})$ *is a secure commitment scheme, protocol* Π_{4AOT} *securely realizes the* $\mathcal{F}_{4AOT}$ *functionality.*

3.2 Batch Attested OT

In our distributed garbling protocol, we need to perform a large batch of attested OT protocols (proportional to number of gates in the circuit). It turns out that we can optimize communication complexity of the protocol in the batch setting. In particular, P_3 and P_4 only need to send a hash of all commitments they needed to send to P_1. Furthermore, only one of them needs to send decommitments, while the other can just send the hash of all the decommitments concatenated as that is sufficient for checking the equality of the two. In Appendix C, $\mathcal{F}_{\text{B-4AOT}}$ (Figure 11) describes the functionality and $\Pi_{\text{B-4AOT}}$ (Figure 12) describes the corresponding protocol for this batch setting. The security of the batch assisted OT protocol (Lemma 3) below follows in a similar manner to the proof of Lemma 2 and is omitted here.

Lemma 3. *Assuming* $(\mathsf{ComGen}, \mathsf{Com}, \mathsf{Open})$ *is a secure commitment scheme, and* $H \leftarrow \mathcal{H}$ *is a collision resistant hash function, protocol* $\Pi_{\text{B-4AOT}}$ *securely realizes the* $\mathcal{F}_{\text{B-4AOT}}$ *functionality.*

3.3 Efficiency of Attested OT

Semi-honest case. Note that in the semi-honest case, first P_1 and P_2 send their inputs to P_3 and then P_3 sends the output to P_2. In case of bit OT this requires 4 bits of communication, but in our four-party garbling where P_1 and P_2 inputs can derived from a one-time communicated seed,

Protocol $\Pi_{\text{4AOT}}(P_1, P_2, \{P_3, P_4\})$

The protocol is carried out among P_1, P_2, P_3, P_4, with P_1 and P_2 acting as sender and receiver, respectively, and P_3, P_4 as attestors. Let $\mathsf{Commit} = (\mathsf{ComGen}, \mathsf{Com}, \mathsf{Open})$ be a secure noninteractive commitment scheme.

Input. P_1 holds m_0, m_1, and P_2 holds b.

Computation. Proceed as follows:

1. P_1 generates random values $r_0, r_1 \leftarrow \{0,1\}^*$ and computes $(\mathsf{Com}_0, \mathsf{Open}_0) := \mathsf{Com}(m_0; r_0)$ and $(\mathsf{Com}_1, \mathsf{Open}_1) := \mathsf{Com}(m_1; r_1)$. P_1 sends Com_0 and Com_1 to P_2 and sends r_0, m_0, r_1, m_1 to P_3 and P_4, who store them as $r_0^3, m_0^3, r_1^3, m_1^3$ and $r_0^4, m_0^4, r_1^4, m_1^4$, respectively.
2. P_2 sends b to P_3 and P_4 who store them as b^3 and b^4, respectively.
3. P_3 and P_4 exchange their copies of P_1 and P_2's inputs and the random values they receive from P_1.
   (a) If the values match, then for $i \in \{3,4\}$, P_i computes $(\mathsf{Com}_0^i, \mathsf{Open}_0^i)$ and $(\mathsf{Com}_1^i, \mathsf{Open}_1^i)$ using scheme Commit and its random values, and sends $(\mathsf{Com}_0^i, \mathsf{Com}_1^i)$ to P_2. (Wlog) P_3 also sends Open_b^i to P_2.
   (b) If the values do not match, i.e., $(m_0^3, m_1^3, b^3, r_0^3, r_1^3) \neq (m_0^4, m_1^4, b^4, r_0^4, r_1^4)$, they send a $\perp$ message to P_2 (denoting abort).

Output. P_2 checks the following and outputs $\perp$ if any of items are true: (i) It receives $\perp$ from P_3 or P_4; (ii) the three commitment pairs it has received from P_1, P_3, P_4 do not match; and (iii) $\mathsf{Open}(\mathsf{Com}_b^3, \mathsf{Open}_b^3) = \perp$ for $b^3 = b$. Otherwise, P_2 outputs $m_{b^3}^3 = \mathsf{Open}(\mathsf{Com}_b^3, \mathsf{Open}_b^3)$.

Figure 2: *The actively secure 4-party protocol for Attested OT.*

communicating *a single* bit per OT is sufficient. Similarly, the string attessted OT requires κ bits of communication. Also note that only one round of interaction would be sufficient when using seeds to derive inputs for attesters.

Malicious case. Similarly, in the malicious case, first P_1 and P_2 send their inputs and randomness for the commitments to P_3 and P_4. P_3 and P_4 exchange hashes of these values. But these steps can be eliminated in our distributed garbling since inputs can be derived from seeds. Finally P_1 sends commitments and P_3 sends decommitments, while P_3 and P_4 also send hashes of commitments which again is insignificant in the batch attested OT. So, the overall communication complexity is two commitments and one decommitment per OT. The computational cost is generating 6 commitments and roughly 2 hashings per OT. Also note that only one round of interaction is sufficient when using seeds to derive inputs for attesters.

4 Efficient and Actively Secure 5PC

We start by presenting our actively secure distributed garbling protocol, followed by the 5PC (with abort) protocol. It turns out that the protocol can be significantly simplified in the case of semi-honest adversaries, which might also be of practical interest. We conclude the section pointing out those simplifications.

Function $f^C_{\text{4GC}}(s_1, s_2, s_3, s_4)$

Inputs. All four parties hold the circuit C, security parameter κ and pseudorandom function family F. '`delta`', '`perm`' and '`key`' are known public strings. In addition, P_i, $i \in [4]$, has private input random seed $s_i \in \{0,1\}^\kappa$.

Computation. Proceed as follows:

1. For $i \in [4]$ do:
   - $R_i := F_{s_i}(\text{`}\texttt{delta}\text{'})$, $F_{s_i} \in \mathsf{F}$.
   - For every wire w in C that is not the output of an XOR gate, generate a random permutation bit $p^i_w := F_{s_i}(\text{`}\texttt{perm}\text{'}||w)$, and let $k^i_{w,0} := F_{s_i}(\text{`}\texttt{key}\text{'}||w||0)$ and $k^i_{w,1} := k^i_{w,0} \oplus R_i$. (These wires are set in this way in order to enable the free XOR technique.)
   - In a topological order, for every output wire w of an XOR gate with input wires u and v, set $p^i_w := p^i_u \oplus p^i_v$, $k^i_{w,0} := k^i_{u,0} \oplus k^i_{v,0}$ and $k^i_{w,1} := k^i_{w,0} \oplus R_i$.
2. For every w in C, set $p_w := \bigoplus_{i=1}^4 p^i_w$.
3. For every AND gate $g \in C$ with input wires u, v and output wire w, every $\alpha, \beta \in \{0,1\}$ and every $j \in [4]$, set:

$$g^j_{\alpha,\beta} := \Big(\bigoplus_{i=1}^4 F_{k^i_{u,\alpha}}(g||j) \oplus F_{k^i_{v,\beta}}(g||j) \Big) \oplus k^i_{w,0} \oplus \big(R_i \cdot ((p_u \oplus \alpha) \cdot (p_v \oplus \beta) \oplus p_w) \big) \tag{1}$$

Outputs.

- *(Public outputs)* Output to all parties $g^1_{\alpha,\beta}||\ldots||g^4_{\alpha,\beta}$, for every AND gate $g \in C$ and every $\alpha, \beta \in \{0,1\}$.
- *(Private outputs)* Output s_3, s_4 to P_1 and P_2, and s_1, s_2 to P_3 and P_4.

Figure 3: *The 4-party distributed garbling function.*

4.1 Actively secure distributed garbling scheme

Our garbling scheme secure against actively malicious adversaries builds on the passively secure distributed garbling protocol of Ben-Efraim *et al.* [BLO16b] (which includes the free-XOR optimization). At a very high level, we make three modifications to their protocol: First, in order to achieve active security, we ensure that each party's randomness and keys are generated using a small random seed, and that exactly two other parties learn the seed of each party. This enables two other parties to "check" every parties' computation. Second, we replace calls to each batch of two-party OTs in their protocol with calls to our 4-party Batch Assisted OT functionality $\mathcal{F}_{\text{B-4AOT}}$; this avoids the use of OT protocols altogether, and reduces the number of rounds of interactions to just one. Third, for each party's share of the garbled circuit, two other parties compute the same share and send it to the party missing that share. This ensures that at least one share is honestly generated and hence bad garbled circuits can be detected. Through these modifications, we obtain a more efficient 4-party distributed garbling protocol that will help us get a 5PC protocol secure against corrupted malicious parties. Our distributed garbling function is defined in Figure 3 (which is a tailored version of the $n-$party distributed garbling function from Figure 9). We now describe our distributed garbling scheme $\mathcal{G}$ (cf. Section 2) in more detail below:

Protocol $\Pi_{4\text{GC}}(C, \{P_1, P_2, P_3, P_4\})$

Inputs. All parties hold the circuit C, security parameter κ and pseudorandom function family F. '`delta`', '`perm`', '`key`', '`bitOT`' and '`strOT`' are known public strings. In addition, P_1 holds seeds $\{s_1, s_3, s_4\}$, P_2 holds seeds $\{s_2, s_3, s_4\}$, P_3 holds seeds $\{s_1, s_2, s_3\}$ and P_4 holds seed $\{s_1, s_2, s_4\}$, where all s_i are random seeds to F. Denote by $\mathcal{S}_i$ the set of indices of parties with knowledge of s_i, i.e., $\mathcal{S}_1 = \{1, 3, 4\}$, $\mathcal{S}_2 = \{2, 3, 4\}$, $\mathcal{S}_3 = \{1, 2, 3\}$, and $\mathcal{S}_4 = \{1, 2, 4\}$.

Keys and permutation bits. For $i \in [4]$, for all $j \in \mathcal{S}_i$, P_j performs the following:

- $R^{i,j} := F_{s_i}(\text{`}\texttt{delta}\text{'})$. Note, that if parties are honest, then $R^{i,j} = R^{i,\ell} = R_i$ for all $j, \ell \in \mathcal{S}_i$.
- For every wire w in C that is not the output of an XOR gate, generate a random permutation bit $p_w^{i,j} := F_{s_i}(\text{`}\texttt{perm}\text{'}||w)$, and set $k_{w,0}^{i,j} := F_{s_i}(\text{`}\texttt{key}\text{'}||w||0)$ and $k_{w,1}^{i,j} := k_{w,0}^{i,j} \oplus R^{i,j}$. Note, that $k_{w,\beta}^{i,j}$ denotes P_j's version of $k_{w,\beta}^i$, for $\beta \in \{0, 1\}$.
- In a topological order, for every wire that is the output of an XOR gate with input wires u and v, set $p_w^{i,j} := p_u^{i,j} \oplus p_v^{i,j}$, $k_{w,0}^{i,j} := k_{u,0}^{i,j} \oplus k_{v,0}^{i,j}$ and $k_{w,1}^{i,j} := k_{w,0}^{i,j} \oplus R^{i,j}$.

Computing $(p_u \oplus \alpha) \cdot (p_v \oplus \beta) \oplus p_w$, for $\alpha, \beta \in \{0, 1\}$.

- For every AND gate $g \in C$, denote the input wires by u, v and the output wire by w, and denote P_i's XOR share of the permutation bits by p_u^i, p_v^i, p_w^i, respectively. Once again, recall that $p_u^{i,j} = p_u^i, p_v^{i,j} = p_v^i, p_w^{i,j} = p_w^i$ for all honest $j \in \mathcal{S}_i$; i.e., $p_u^{i,j}$ denotes the value p_u^i as computed by $P_j, j \in \mathcal{S}_i$, while p_u^i denotes its true value. Our goal is to compute XOR shares of

$$p_u \cdot p_v = (\oplus_{i=1}^4 p_u^i) \cdot (\oplus_{i=1}^4 p_v^i) = (\oplus_{i=1}^4 p_u^i \cdot p_v^i) \oplus (\oplus_{i \neq j} p_u^i \cdot p_v^j).$$

- For all $i \in [4]$, for all $j \in \mathcal{S}_i$, P_j locally computes $p_u^{i,j} \cdot p_v^{i,j}$.
- For all $i, j \in [4]$, $i \neq j$, and all $g \in C$:
  - For all $\ell \in \mathcal{S}_i$, P_ℓ generates a random bit $r^{i,j,\ell} := F_{s_i}(\text{`}\texttt{bitOT}\text{'}||i||j||g)$.
  - For all $\ell \in \mathcal{S}_i \cap \mathcal{S}_j$, P_ℓ locally computes $(p_u^{i,\ell} \oplus r^{i,j,\ell}) \cdot p_v^{j,\ell}$. In our case, $|\mathcal{S}_i \cap \mathcal{S}_j| = 2$ and we will denote the indices of the parties in $\mathcal{S}_i \cap \mathcal{S}_j$ by $\ell_{i,j}^1, \ell_{i,j}^2$, where needed.
  - Parties invoke $\mathcal{F}_{4\text{AOT}}(P_s, P_r, \{P_{\ell_{i,j}^1}, P_{\ell_{i,j}^2}\})$, where $s \in \mathcal{S}_i - \mathcal{S}_j$, $r \in \mathcal{S}_j - \mathcal{S}_i$, and $\ell_{i,j}^1, \ell_{i,j}^2 \in (\mathcal{S}_i \cap \mathcal{S}_j)$: P_s inputs $(r^{i,j,s}, p_u^{i,s} \oplus r^{i,j,s})$, P_r inputs $p_v^{j,r}$, and $P_{\ell_{i,j}^z}$ inputs $(r^{i,j,\ell_{i,j}^z}, p_u^{j,\ell_{i,j}^z} \oplus r^{i,j,\ell_{i,j}^z}, p_v^{j,\ell_{i,j}^z})$, for $z = 1, 2$. P_r receives the output $(p_u^i \oplus r^{i,j}) \cdot p_v^j$ (or $\perp$).
- For all $i \in [4]$, all $j \in \mathcal{S}_i$, and all $g \in C$, P_j does the following:
  - P_j locally XORs the values it obtains from the computation above to compute $p_{uv}^{i,j}$, i.e., P_i's XOR share of $p_{uv} = p_u \cdot p_v$ as recorded by P_j.
  - Similarly, P_j locally computes P_i's XOR shares of

$$p_{uvw} = p_u \cdot p_v \oplus p_w \qquad p_{u\bar{v}w} = p_u \cdot \bar{p}_v \oplus p_w$$
$$p_{\bar{u}vw} = \bar{p}_u \cdot p_v \oplus p_w \qquad p_{\bar{u}\bar{v}w} = \bar{p}_u \cdot \bar{p}_v \oplus p_w$$

    where $\bar{p} = 1 - p$, for a bit p.

(Continued in Figure 5.)

Figure 4: *The 4-party distributed garbling protocol.*

Protocol $\Pi_{\text{4GC}}(C, \{P_1, P_2, P_3, P_4\})$ (cont'd)

Computing $R^i \cdot \big((p_u \oplus \alpha) \cdot (p_v \oplus \beta) \oplus p_w\big)$. For all $i, j \in [4]$, $i \neq j$, and all $g \in C$:

- For all $\ell \in \mathcal{S}_i$, P_ℓ generates a random κ-bit string $Q^{i,j,\ell} := F_{s_i}(\text{`}\texttt{strOT}\text{'}||i||j||g)$.
- For all $\ell \in \mathcal{S}_i \cap \mathcal{S}_j$, P_ℓ locally computes $(R^{i,\ell} \oplus Q^{i,j,\ell}) \cdot p^{j,\ell}_{uvw}$. Again, $R^{i,\ell}$ is the version of R^i held by P_ℓ.
- Parties invoke $\mathcal{F}_{\text{4AOT}}(P_s, P_r, \mathcal{S}_i \cap \mathcal{S}_j)$, where $s \in \mathcal{S}_i - \mathcal{S}_j$, $r \in \mathcal{S}_j - \mathcal{S}_i$, and $\ell \in (\mathcal{S}_i \cap \mathcal{S}_j)$: P_s inputs $(Q^{i,j,s}, R^{i,s} \oplus Q^{i,j,s})$, P_r inputs $p^{j,r}_{uvw}$, and P_ℓ inputs $(Q^{i,j,\ell}, R^{j,\ell} \oplus Q^{i,j,\ell}, R^{j,\ell})$. P_r receives the output $Q^{i,j,s} \oplus R^{i,s} \cdot p^{j,r}_{uvw}$, which is the same as $Q^{i,j} \oplus R^i \cdot p^j_{uvw}$ if parties are honest. The same is repeated for $p_{u\bar{v}w}, p_{\bar{u}vw}$, and $p_{\bar{u}\bar{v}w}$.
- Let $\rho^{i,j,\ell}_{w,\alpha,\beta}$ denote P_j's XOR share of $R^i \cdot \big((p_u \oplus \alpha) \cdot (p_v \oplus \beta) \oplus p_w\big)$, as recorded by P_ℓ.

Outputs. Let $\mathcal{R}_i$ denote the set of indices of seeds held by P_i, $i \in [4]$[a]. In other words, $\mathcal{R}_1 = \{1,3,4\}$, $\mathcal{R}_2 = \{2,3,4\}$, $\mathcal{R}_3 = \{1,2,3\}$, and $\mathcal{R}_4 = \{1,2,4\}$. For all $i \in [4]$, for all $g \in C$, and for all $\alpha, \beta \in \{0,1\}$:

- For all $j \in \mathcal{R}_i$, for $c \in [4]$, when $c = j$, P_i locally computes $F_{k^{j,i}_{u,\alpha}}(g||c) \oplus F_{k^{j,i}_{v,\beta}}(g||c) \oplus k^{j,i}_{w,0} \oplus \rho^{c,j,i}_{w,\alpha,\beta}$. When $c \neq j$, P_i computes $F_{k^{j,i}_{u,\alpha}}(g||c) \oplus F_{k^{j,i}_{v,\beta}}(g||c) \oplus \rho^{c,j,i}_{w,\alpha,\beta}$. As a result, P_i holds three of the four shares it needs to compute the garbled circuit.
- For $j = [4] - \mathcal{R}_i$, for all $\ell \in \mathcal{S}_j$, for $c \in [4]$
  1. When $c = j$, P_ℓ sends $F_{k^{j,\ell}_{u,\alpha}}(g||c) \oplus F_{k^{j,\ell}_{v,\beta}}(g||c) \oplus k^{j,\ell}_{w,0} \oplus \rho^{c,j,\ell}_{w,\alpha,\beta}$ to P_i. [b]
  2. When $c \neq j$, P_ℓ sends $F_{k^{j,\ell}_{u,\alpha}}(g||c) \oplus F_{k^{j,\ell}_{v,\beta}}(g||c) \oplus \rho^{c,j,\ell}_{w,\alpha,\beta}$ to P_i.
- If all three versions of each value P_i receives is the same it locally XORs the four values computed above for all $g \in C$ to obtain $g^i_{\alpha,\beta}$, and output $g^1_{\alpha,\beta}|| \ldots ||g^4_{\alpha,\beta}$ for all $\alpha, \beta \in \{0,1\}$ and all AND gates $g \in C$. Else, they output $\perp$.

[a] While sets $\mathcal{R}_i$ and $\mathcal{S}_i$ are the same in the 5PC case, they define different sets and would be different in the general case.

[b] For ease of composition we assume all parties send the complete value, but in fact two parties can only send the hash, which can be batched across all gates to save on communication.

Figure 5: *The 4-party distributed garbling protocol (continued from Fig. 4).*

- In Figure 3 the garbling function is $\mathsf{Gb}(1^\kappa, f)$ and Gf is the public output – i.e., $g^1_{\alpha,\beta}||\ldots||g^4_{\alpha,\beta}$, for every AND gate $g \in C$ and every $\alpha, \beta \in \{0,1\}$.
- For every input wire w corresponding to a party P_i's input bit b, define $p^j_w = F_{s_j}(\texttt{`perm'}||w)$, $p_w = \Sigma^4_{j=1} p^j_w$, $b' = b \oplus p_w$ and let the encoding function $\mathsf{En}(e, \cdot)$ be the concatenation of $F_{s_j}(\texttt{`key'}||w||0) \oplus F_{s_j}(\texttt{`delta'}).b'$, for all $j \in [4]$. That is, when $x = b, X = F_{s_j}(\texttt{`key'}||w||0) \oplus F_{s_j}(\texttt{`delta'}).b', \forall j \in [4]$.
- The evaluation function $\mathsf{Ev}(\mathsf{Gf}, \cdot)$ is the same function as in the semi-honest protocol of [BLO16b].
- Finally, for every output wire w, let $Y = \mathsf{Ev}(\mathsf{Gf}, X)$ be parsed as $k^1_w||k^2_w||k^3_w||k^4_w$. Now, if $k^i_w = k_{w,0} = F_{s_i}(\texttt{`key'}||w||0)$ for all $i \in [4]$, then set $\alpha_w = 0$, else set $\alpha_w = 1$ if $k^i_w = k_{w,0} = F_{s_i}(\texttt{`key'}||w||0) \oplus R_i$ for all $i \in [4]$. If neither of the two cases hold, then output $\perp$; otherwise, output $y = \alpha_w \oplus p_w$, where $p_w = \Sigma^4_{i=1} p^i_w$, and where $p^i_w = F_{s_i}(\texttt{`perm'}||w)$. Thus, the decoding function $\mathsf{De}(d, Y) = \alpha_w \oplus p_w$.

Ben-Efraim *et al.* [BLO16b] show that the garbling scheme $\mathcal{G}$, defined via the functions above is a correct garbling scheme. In addition, it is easy to see that when the 4 parties are honest, our protocol $\Pi_{\text{4GC}}(C, \{P_1, P_2, P_3, P_4\}$ described in Figures 4 and 5 computes the function in Figure 3 correctly (Definition 1) in the $\mathcal{F}_{\text{4AOT}}$-hybrid ($\mathcal{F}_{\text{B-4AOT}}$-hybrid) model. We leave this explicit corroboration for the full version of the paper.

4.2 The actively secure 5PC protocol

The 5PC protocol proceeds through the following steps detailed below. All parties hold the circuit C, security parameter κ, pseudorandom function family PRF, hash function H, and description of the commitment scheme. '`delta`', '`perm`', '`key`', '`bitOT`', '`strOT`' and '`rand`' are known public strings. P_i has a private input $x_i \in \{0,1\}^\ell$. The circuit $C(x_1, x_2, x_3, x_4, x_5)$ is modified into a circuit $C'(x_1, x_2||x'_2, x_3||x'_3, x_4||x'_4) = C(x_1, x_2, x_3, x_4, x'_2 \oplus x'_3 \oplus x'_4)$. The steps in the protocol are as follows:

1. **Seed distribution.** Parties P_1, P_2, P_3, P_4, known as the *garblers*, run a seed distribution phase in which each party picks a seed s_i and the seeds are distributed such that every party knows 3 seeds and every seed is held by 3 parties.

2. **Garbled input generation.** For garblers and evaluator:

   - *Garblers.* Consider an input wire w of party P_1 with a bit value b. The two labels corresponding to w are $k^1_{w,0}||k^2_{w,0}||k^3_{w,0}||k^4_{w,0}$ and $k^1_{w,1}||k^2_{w,1}||k^3_{w,1}||k^4_{w,1}$ and its permutation bit is $p_w = \oplus_{j\in[4]} p^j_w$. The goal is to let the evaluator P_5 learn $k^1_{w,b\oplus p_w}||\ldots||k^4_{w,b\oplus p_w}$.

     1. First, P_1 learns p^j_w for all $j \in [4]$ from the other parties (P_1 will check the correctness of these values by comparing the different versions of these values he receives). P_1 sets $b' = b \oplus p^j_w$.

     2. Observe that P_1 can compute $k^1_{w,b\oplus p_w}$, $k^3_{w,b\oplus p_w}$, and $k^4_{w,b\oplus p_w}$ on his own and send them to P_5. This is because P_1 knows seeds s_1, s_3, and s_4. We must also somehow enable P_5 to compute $k^2_{w,b\oplus p_w}$ (without any other party learning this value and P_5 itself learning anything else).

     3. To do this, P_1 will secret-share b' among P_2, P_3 and P_4; that is, they will receive b_2, b_3, b_4, respectively, such that $b_2 \oplus b_3 \oplus b_4 = b'$.

     4. P_ℓ, for $\ell \in \{2,3,4\}$ can compute $k^2_{w,b_\ell}[\ell] := F_{s_2}(\texttt{`key'}||w||0) \oplus F_{s_2}(\texttt{`delta'}) \cdot b_\ell$ as a share of $k^2_{w,b\oplus p_w}$. Through the secret sharing of b' into b_2, b_3 and b_4, we actually have that $k^2_{w,b_2}[2] \oplus k^2_{w,b_3}[3] \oplus k^2_{w,b_4}[4] = k^2_{w,b'} = k^2_{w,b\oplus p_w}$.

While the above steps work functionally, security breaks down, as P_5 colluding with one of the garblers can learn both wire labels. To prevent this, we have two garblers, P_1 and any other garbler, provide in addition secret sharings of 0 which each P_2, P_3 and P_4 add to their corresponding shares. These values will cancel out when combined, but will ensure that P_5 colluding with a garbler cannot learn anything else.

- *Evaluator.* To compute the garbled label for the evaluator (P_5)'s input, P_5 first secret-shares his input with P_1, P_2, P_3, P_4, and now these shares can be treated as inputs of P_1, P_2, P_3, P_4. However, a bit more care is needed to prevent $P_i, i \in [4]$ lying about their share of P_5's input. To prevent this, we have all parties provide commitments to all labels and have the corresponding party "open" the right label to the right share. This is quite similar to the technique used by Mohassel *et al.* [MRZ15].

5. **Distributed circuit garbling.** Parties execute the distributed circuit garbling protocol $\Pi_{\text{4GC}}(C, \{P_1, P_2, P_3, P_4\}$ from Figures 4 and 5. One party (say, P_1) sends the distributed garbled circuit to P_5, while other parties send a hash of the garbled circuit. P_5 accepts only if the hashes of the distributed garbled circuit match.

6. **Evaluation and output.** P_5 calls the $\mathsf{Ev}(\mathsf{Gf}, \cdot)$ procedure to evaluate the distributed garbled circuit Gf received. The output labels Y are sent to all parties. Every party runs the $\mathsf{De}(d, Y)$ to obtain the output of the computation, y.

The complete protocol, $\Pi_{\text{5PC}}(C, \{P_1, \ldots, P_5\})$, is described in Figure 6. The security of the protocol is proven in Theorem 4 below, the proof of which is given in Appendix B.

Theorem 4. *Assuming* $(\mathsf{ComGen}, \mathsf{Com}, \mathsf{Open})$ *is a secure commitment scheme, and* $H \xleftarrow{\$} \mathcal{H}$ *is a collision-resistant hash function, protocol* $\Pi_{\text{5PC}}(C, \{P_1, \ldots, P_5\})$ *securely realizes the functionality* $\mathcal{F}_{\text{SFE}}^{C}(\{P_1, \ldots, P_5\})$[1] *in the* $\mathcal{F}_{\text{4AOT}}$*-hybrid model.*

In practice, it is useful to execute the distributed circuit garbling protocol phase of the 5PC in our protocol before both the garbled input generation phases, so that the protocol can be split into a (slower) offline phase, that is independent of all inputs to the computation, and a (faster) online phase, that depends on inputs. In this case, our proof should be modified to use an adaptive notion of distributed garbling similar to what is defined in [BHR12], and can then be shown to be secure in the random oracle model.

4.3 Passively secure 5PC protocol

So far we have described our 5PC protocol against malicious adversaries. The protocol can be signifcantly simplified in the semi-honest case. We only point out the simplification we can make to the actively secure 5PC protocol but omit a full description. (i) First, all calls to $\mathcal{F}_{\text{4AOT}}$ can be replaced by the semi-honest variant described in Figure 10, which avoids the use of commitments and is much more communication efficient. In particular, the bit OTs only require communicating a single bit while the string OTs only communicate a single string. (ii) It is sufficient for one of the garblers (say, P_1) to compute the full garbled circuit an send it to P_5. The other parties can simply provide the necessary shares for computing the garbled circuit to P_1. Consequently, they also do not sent hashes of the garbled circuit to P_5. (iii) Some extra checks done in the garbled input generation can also be removed. For example, it is sufficient for only one party (as opposed to three) to send share of the permutation bits for input wires, and we can remove the commitments

[1] Recall that we slightly abuse notation, and mean security with abort.

Protocol $\Pi_{\text{5PC}}(C, \{P_1, \ldots, P_5\})$

Inputs. All parties hold the circuit C, security parameter κ and pseudorandom function family F. '`delta`', '`perm`', '`key`', '`bitOT`', '`strOT`' and '`rand`' are known public strings. Let $(\mathsf{ComGen}, \mathsf{Com}, \mathsf{Open})$ be a secure noninteractive commitment scheme, and H a collision-resistant hash function. In addition, P_i has a private input $x_i \in \{0,1\}^\ell$. The circuit $C(x_1, x_2, x_3, x_4, x_5)$ is modified into a circuit $C'(x_1, x_2||x_2', x_3||x_3', x_4||x_4') = C(x_1, x_2, x_3, x_4, x_2' \oplus x_3' \oplus x_4')$.

Seed distribution.

- P_1 and P_2 generate random seeds s_1 and s_2 respectively and send them to both P_3 and P_4. P_3 and P_4 exchange these two seeds and abort if they do not match. P_3 and P_4 send s_3 and s_4 to both P_1 and P_2. P_1 and P_2 exchange these two seeds and abort if they do not match.
- Denote by $\mathcal{S}_i$ the set of indices of parties with knowledge of s_i, i.e., $\mathcal{S}_1 = \{1,3,4\}$, $\mathcal{S}_2 = \{2,3,4\}$, $\mathcal{S}_3 = \{1,2,3\}$, and $\mathcal{S}_4 = \{1,2,4\}$.

Garbled input generation for $P_1, \ldots, P_4$. For all $i \in [4]$, for each input wire w corresponding to P_i with input value b, do the following:

- Let $j = [4] - \mathcal{R}_i$. For all $\ell \in \mathcal{S}_j$, P_ℓ computes $p_w^{j,\ell} := F_{s_j}(\text{`}\texttt{perm}\text{'}||w)$ and sends it to P_i. P_i checks that it receives the same value from all P_ℓ's. If so, simply denote the bit by p_w^j; if not, it sets $\mathsf{abort}_i := \mathsf{true}$. If $\mathsf{abort}_i = \mathsf{true}$, it sets b to a uniformly random value independent of its true input.
- P_i then sets $p_w := \oplus_{j=1}^4 p_w^j$ and $b' := b \oplus p_w$.
- Let $j = 4 - \mathcal{R}_i$. For all $\ell \in \mathcal{S}_j$, P_i generates random bits b_ℓ such that $\oplus_{\ell \in \mathcal{S}_j} b_\ell = b'$, and random strings $\beta_\ell \in \{0,1\}^\kappa$ such that $\oplus_{\ell \in \mathcal{S}_j} \beta_\ell = 0^\kappa$. P_i sends b_ℓ, β_ℓ to P_ℓ. Denote by j_1, j_2, j_3 the three indices in $\mathcal{S}_j$. P_{j_1} generates random strings $\gamma_{j_\ell} \in \{0,1\}^\kappa$ where $\oplus_{\ell=1}^3 \gamma_{j_\ell} = 0^\kappa$, and sends γ_{j_2} to P_{j_2} and γ_{j_3} to P_{j_3}.
- If $\mathsf{abort}_i = \mathsf{false}$, then for all $j \in \mathcal{R}_i$, P_i computes $k_{w,b'}^{j,i} := F_{s_j}(\text{`}\texttt{key}\text{'}||w||0) \oplus F_{s_j}(\text{`}\texttt{delta}\text{'}) \cdot b'$ and sends to P_5; otherwise, P_i sends $\perp$ to P_5.
- Let $j = [4] - \mathcal{R}_i$. For all $\ell \in \mathcal{S}_j$, if $\mathsf{abort}_\ell = \mathsf{false}$, P_ℓ computes $k_{w,b_j}^{i,j} := F_{s_j}(\text{`}\texttt{key}\text{'}||w||0) \oplus F_{s_j}(\text{`}\texttt{delta}\text{'}) \cdot b_\ell$ and sends $c_\ell := k_{w,b_j}^{i,j} \oplus \beta_\ell \oplus \gamma_\ell$ to P_5; otherwise P_ℓ sends $\perp$ to P_5.
- Finally, P_5 computes the label of wire w as the concatenation of $k_{w,b'}^{j,i}$ for all $j \in \mathcal{R}_i$ and $k_{w,b'}^i := \oplus_{\ell \in \mathcal{S}_j} c_j$ for $j = [4] - \mathcal{R}_i$ (or sets $\mathsf{abort}_5 := \mathsf{true}$ if it receives any $\perp$ message from any P_i, $i \in [4]$).

(Continued in Figure 7.)

Figure 6: *The 5PC protocol, secure against malicious adversaries.*

Protocol $\Pi_{\text{5PC}}(C, \{P_1, \ldots, P_5\})$ (cont'd)

Garbled input generation for P_5. For each input wire w in the original circuit C corresponding to P_5 with input value b, denote the corresponding input wires for P_2, P_3, P_4 in the modified circuit C' by w_2, w_3, w_4.

- P_5 generates random bits b_2, b_3, b_4 such that $b_2 \oplus b_3 \oplus b_4 = b$, and sends b_i to P_i, $i \in \{2,3,4\}$.
- For each of the three wires w_ℓ, for all $i \in [4]$, for $j \in \mathcal{R}_i$, each P_i computes $k^{j,i}_{w_\ell,e} := F_{s_j}(\text{`}\texttt{key}\text{'}||w_\ell||0) \oplus F_{s_j}(\text{`}\texttt{delta}\text{'}) \cdot e$ and $r^{j,i}_e := F_{s_j}(\text{`}\texttt{rand}\text{'}||w_\ell||e)$, for $e \in \{0,1\}$, and computes $(\mathsf{Com}^i_{j,w_\ell,e}, \mathsf{Open}^i_{j,w_\ell,e}) := \mathsf{Com}(k^{j,i}_{w_\ell,e}; r^{j,i}_e)$. It then sends the ordered pair $(\mathsf{Com}^i_{j,w_k,0}, \mathsf{Com}^i_{j,w_k,1})$ to P_5 (or $\perp$, if $\mathsf{abort}_i = \mathsf{true}$).
- P_5 verifies that all commitment pairs it receives from the other parties are consistent, i.e., all commitments derived from the same seeds are equal. If not, it sets $\mathsf{abort}_5 := \mathsf{true}$.
- For each wire w_ℓ where $\ell \in \{2,3,4\}$, P_ℓ performs the steps above for garbled input generation of non-P_5 parties to compute and send to P_5 garbled inputs for wire w_ℓ using the input value b_ℓ. The only difference compared to above is that instead of only sending $k^{j,\ell}_{w_\ell,b_\ell}$ for all $j \in \mathcal{S}_\ell$, P_ℓ sends $\mathsf{Open}^{j,\ell}_{w_\ell,b_\ell} = (k^{j,\ell}_{w_\ell,b_\ell}, r^{j,\ell}_e)$ to P_5. If $\mathsf{Open}(\mathsf{Open}^{j,\ell}_{w_\ell,b_\ell}\mathsf{Com}_{j,w_\ell,b_\ell}) = 0$, P_5 sets $\mathsf{abort}_i := \mathsf{true}$.

Distributed circuit garbling.

- $P_1, \ldots, P_4$ run the distributed circuit garbling protocol $\Pi_{\text{4GC}}(C', \{P_1, P_2, P_3, P_4\})$ using the seeds generated and distributed above as input. As a result, for all $i \in [4]$, P_i learns the garbled version of circuit C', call it GC' (or $\perp$). In addition, for all $j \in \mathcal{S}_i$, P_j learns s_i.
- P_1 sends GC' (or $\perp$) to P_5, while the other parties only send $H(GC')$ (or $\perp$). P_5 checks that all the GC' and the received hash values are consistent; otherwise sets $\mathsf{abort}_5 := \mathsf{true}$.

Evaluation and output. P_5 now has the correct garbled circuit GC', and garbled inputs (call this X) for all parties. It evaluates the garbled circuit using $\mathsf{Ev}(GC', X)$ to get Y and sends Y to all parties. All parties execute $y = \mathsf{De}(d, Y)$ to compute the output of the function, y[a].

[a]For ease of exposition, we assume that the evaluator does not receive any output. If we require the evaluator to obtain output, then similarly to what is done in the 2-party setting, the evaluator also receives hashes of both output labels along with the garbled circuit, thus allowing the evaluator to learn what bit the output label corresponds to.

Figure 7: *The 5PC protocol, secure against malicious adversaries (continued from Fig. 6).*

introduced for preventing malicious behavior in the garbled input generation for P_5. This yields a much more efficient semi-honest 5PC protocol, as shown in our experimental results in Section 6.1.

5 Efficiency Considerations

We start by discussing the communication efficiency of our 5PC protocol, followed by number of communication rounds, followed by a fast instantiation of a non-interactive commitment scheme.

Communication. In the 4-party distributed circuit garbling protocol, the main communication cost is due to calls to the AOT protocol. Specifically, the number of $\mathcal{F}_{\text{4AOT}}$ calls for bit-OTs is 12 per AND gate (since there are 4×3 pairwise OTs in the semi-honest distributed circuit garbling protocol), where each party plays the role of the sender in 3 and of the receiver in 3. Since every bit AOT has a communication of 3κ bits, this part gives us a total communication of $36\kappa|C|$ bits. The number of $\mathcal{F}_{\text{4AOT}}$ calls for the string-OTs is 36 per AND gate, borrowing all optimizations from [BLO16b], where each party is the sender in 9 and receiver in 9. Since the AOTs are performed for each gate in parallel, we also take advantage of our Batch AOT protocol to obtain better efficiency. In particular, the amortized communicaton cost of each AOT in the batch setting is two commitments and one decommitment. This yields 96 commitments and 48 decommitments (κ-bits each); i.e., 144κ bits of total communication per gate for the string OTs. Hence, the OT phase has a total communication of $180\kappa|C|$ bits.

To communicate the shares of the computed garbled circuit among the parties, each of the 4 parties must send 4 garbled circuit "shares" and each garbled gate requires 16κ bits of communication. This yields a total of $64\kappa|C|$ bits of communication. This sums up all communications that grow with the number of AND gates, yielding a total of roughly $244\kappa|C|$ bits of communication in total.

In contrast, the passively secure 5PC protocol of [BLO16b] which would perform a 5-party distributed circuit garbling, would require 20 bit OTs per gate and 60 string OTs per gate. Ignoring the communication cost of the bit OTs and assuming 4/3 hash values are communicated per string OT (since these are correlated OTs they can benefit from a 1/3 reduction in communication [BLO16b]), this yields 80κ bits of communication per gate. An additional communication of 500κ bits per gate is required for exchanging the garbled circuit (each of the 5 parties receives 5 of the full garbled circuits which at 20κ bits pergate). This results in a total of 580κ bits of communication per gate or roughly $580\kappa|C|$ bits of communication in total.

In summary, we obtain a 5PC protocol tolerating two malicious corrupted parties with 58% *better* communication complexity than the semi-honest 5PC protocol with security against four corruptions. We realize that the security guarantees are incomparable; yet, this indicates that one can achieve a tradeoff between number of corrupted parties and communication complexity.

Rounds. The protocol requires two initial rounds to exchange the seeds among the parties. This initial exchange, however, can be done only once and then used for multiple protocol execution; hence, we do not count it towards the total round complexity. Each set of Batch AOTs requires one round, hence a total of two. One additional round is required for exchanging shares of the garbled circuit, and another round for sending the garbled circuits to P_5. Three additional rounds are needed for P_5 to learn all the garbled inputs, and one extra round to send the results back to all parties. This yields a total of 8 rounds of interaction for the full 5PC protocol (ignoring the two rounds for exchanging the seeds).

A fast commitment scheme. While instantiating our bit and string commitments (required in the bit- and string-OT protocols), we observe that the (standard) commitment scheme with $\mathsf{Com}(m;r) = (c = H(m||r), d = m||r)$ (where H is a hash function, modeled as a random oracle), with c being the commitment and d the opening/decommitment, has an overhead of 2.44 microseconds per commitment (our protocol makes use of many commitments, proportional to the circuit size).

We reduce this overhead by constructing new bit and string commitment scheme based on block ciphers (e.g., AES), whose security holds in the ideal cipher model [Sha49, Bla06, HKT10]. At a high level, our commitment scheme is as follows. Let $F : \{0,1\}^\kappa \times \{0,1\}^\kappa \to \{0,1\}^\kappa$ denote a random permutation, parameterized by a key k—denoted $F_k(\cdot)$. We assume that all parties have access to k (and hence $F_k(\cdot)$). Our bit commitment scheme is then

$$\mathsf{Com}(b;r) \stackrel{\Delta}{=} (c = F_k(r) \oplus r \oplus b^\kappa,\ d = r||b),$$

where b^κ denotes bit b repeated κ times and r is chosen at random from $\{0,1\}^\kappa$. Hiding follows from the fact that the distribution of $F_k(r) \oplus r$ is indistinguishable from U_κ (the uniform distribution on κ bits), which follows simply from $F_k(\cdot)$ being a pseudorandom function and r being chosen at random. Regarding binding, an adversarial sender must find r, r' such that $F_k(r) \oplus r = \overline{F_k(r') \oplus r'}$, where $\overline{x}$ denotes the complement of x, in order to break it. If the adversary makes at most q queries to $F_k(\cdot)$, then one can show that the probability of finding such a pair is at most $\frac{q^2}{2^\kappa}$.

Now, a similar approach does not however work for string commitments — e.g., $F_k(r) \oplus r \oplus m$ for $m \in \{0,1\}^\kappa$ is not a secure commitment. However, we can show that

$$\mathsf{Com}(m;r) \stackrel{\Delta}{=} (c = F_k(r) \oplus r \oplus F_k(m) \oplus m,\ d = r||m)$$

is a secure commitment scheme using an argument similar to the above. This gives us a commitment scheme with an overhead of only 0.04 microseconds per commitment (roughly 62 times faster than the SHA256 based commitment scheme).

6 Implementation and Experiments

We implemented both our semi-honest as well as maliciously secure 5PC protocols (henceforth referred to as 5PC-SH and 5PC-M, for brevity) and ran various experiments for different circuit sizes and different latencies. Our code built upon the code for $n-$party semi-honest secure computation provided by Ben-Efraim *et al.* [BLO16a]. For all our experiments, we measured the times for the offline phase, which is independent of the inputs to the computation (distributed garbled circuit computation), and for the online phase, which is dependent on the inputs to the computation (garbled input generation, garbled circuit evaluation and output sharing), separately.

Platforms and parameters. We ran our experiments on Microsoft Azure Classic DS4_V2 VM instances (2.4 GHz Intel Xeon Processor with 8 cores) in three different configurations. All our experiments use hardware instructions AES-NI for the PRFs and SHA-256 for the hash function. For the first experiment, all instances were located in the Eastern US region. For the second experiment, the 5 instances were spread across East, Central, West, North Central, and South Central US regions, and for the third experiment, 3 garblers were located in the Western US region with a garbler and an evaluator located in Central Europe. As was reported in [BLO16b], network fluctuations account for almost all the variations in timings in our protocols.

Total Communication of 5 parties [MB]		
	AES	SHA-256
BLO	73.3	756
5PC-M	28.6	356
5PC-SH	9.3	112.2

Table 2: Communication comparison

We ran the protocol for 5 parties (semi-honest and malicious) on 2 different functions. In the first function, parties each hold 128-bit shares of key and input and the output of the computation was the AES function on the XOR of the party's inputs. The AES circuit we used had 6800 AND gates. In the second function, parties hold 300-bit values as input and the output of the computation was the SHA-256 function on the concatenation of all party's inputs. The SHA-256 circuit we used had 90,825 AND gates. Each experiment was performed 30 times and we computed the mean and standard deviation of all experiments. All numbers provided in the tables are in milliseconds with a 95% confidence interval. We compare all our results with the results obtained when executing the semi-honest protocol/code of [BLO16b, BLO16a] for 5 parties (secure against 4 semi-honest corruptions and henceforth referred to as BLO[2]) on the same platforms as our experiments. While our computation (as well as communication) can be parallelized across the 8 cores (across the garbling process) we only deployed mild parallelization and leave further optimizations to future work.

6.1 Experimental Evaluation

Asymptotically, we compared our protocols to maliciously secure MPC protocols that tolerate dishonest majoirty as well as dishonest minority (such as [IKP10]). However, to the best of our knowledge, the only available implementations of constant-round MPC tolerate dishonest majority (or are for the semi-honest case), and hence we compare the concrete efficiency only with these works.

Communication measurements. We first present the total communication (among all parties) of our protocols and a comparison with the numbers from [BLO16b]. As can be seen from Table 2, the total communication of 5PC-M is less than half that of BLO's communication (however, this protocol tolerates 4 semi-honest corruptions and could be optimized for 2 corruptions), while the communication in 5PC-SH is only $12 - 15\%$ that of BLO.

We present our experimental results on the three different network configurations listed above. While constant round protocols are not affected greatly affected by variation in latency of the network, since the overall computation time of our protocol is only a few hundred milliseconds or so, and our protocol has 8 rounds, latency variations affect our overall execution times a bit.

Low-latency network. In this network, all Azure Classic DS4_V2 VM instances were located in the Eastern US region with an average round-trip time of 2.7 milliseconds across all the instances (maximum time of 7.1 milliseconds). The average bandwidth as measured by the Iperf testing tool was 4.5 Gbps. We report the following times: offline execution time (OFT), which measures the wall clock time taken to execute the offline (distributed garbling) phase; the online execution time

[2]The protocol execution time of [BLO16b] does not vary with the number of corruptions.

5PC-M				
	OFT	ONT	TPT	CPUT
Garblers	198 ± 2	8 ± 1	206 ± 3	184 ± 2
Evaluator	50 ± 1	23 ± 1	74 ± 2	57 ± 2
5PC-SH				
	OFT	ONT	TPT	CPUT
Garblers	130 ± 2	8 ± 1	138 ± 2	114 ± 1
Evaluator	24 ± 1	23 ± 1	46 ± 2	36 ± 1
BLO				
	OFT	ONT	TPT	CPUT
All parties	118 ± 2	4 ± 1	122 ± 3	203 ± 2

Table 3: Execution times [ms]: AES circuit, low latency

5PC-M				
	OFT	ONT	TPT	CPUT
Garblers	2402 ± 21	9 ± 1	2411 ± 22	2715 ± 28
Evaluator	587 ± 11	148 ± 5	735 ± 16	632 ± 11
5PC-SH				
	OFT	ONT	TPT	CPUT
Garblers	1536 ± 14	9 ± 1	1545 ± 15	1328 ± 10
Evaluator	297 ± 17	150 ± 4	447 ± 21	363 ± 10
BLO				
	OFT	ONT	TPT	CPUT
All parties	994 ± 13	56 ± 1	1050 ± 14	2543 ± 17

Table 4: Execution times [ms]: SHA-256 circuit, low latency

(ONT), which measures the wall clock time taken to execute the online (input-specific) phase; total protocol time (TPT) which measures the wall clock time to execute the entire protocol; and the CPU time (CPUT), which measures the total time spent on computing across all cores (note that this time can sometimes be larger than TPT when there is great degree of parallelization in the implementation). The results are presented and compared in Tables 3 and 4.

As can be seen from the table, for the AES circuit, our 5PC-M is only 69% slower than BLO (which is only passively secure) in terms of TPT, with CPUT being even lesser in our case (indicating that compute time can be reduced in our protocol through greater parallelization which our protocol accommodates). Our 5PC-SH takes about the same time as BLO (due to the very fast network, the savings in communication is not visible and the BLO protocol takes advantage of parallelization in compute even though the CPU time in BLO is 78% more than in our protocol). For the SHA-256 circuit, our 5PC-M is only about 2.3 times slower than BLO, with roughly same CPU times, and our 5PC-SH is about 1.5 times slower than BLO, with about 52% of the CPU time, once again due to the parallelization optimizations performed in BLO.

Medium-latency network. For the second experiment, the 5 instances were spread across East, Central, West, North Central, South Central US regions with an average round-trip time of the

5PC-M				
	OFT	ONT	TPT	CPUT
Garblers	648 ± 48	39 ± 3	687 ± 51	182 ± 4
Evaluator	243 ± 52	84 ± 3	328 ± 55	60 ± 4
5PC-SH				
	OFT	ONT	TPT	CPUT
Garblers	441 ± 69	44 ± 3	485 ± 71	116 ± 2
Evaluator	111 ± 21	111 ± 17	223 ± 38	36 ± 2
BLO				
	OFT	ONT	TPT	CPUT
All parties	1177 ± 43	81 ± 3	1259 ± 46	207 ± 3

Table 5: Execution times [ms]: AES circuit, med. latency

5PC-M				
	OFT	ONT	TPT	CPUT
Garblers	3430 ± 99	38 ± 2	3468 ± 101	2707 ± 22
Evaluator	789 ± 197	288 ± 126	1077 ± 323	607 ± 10
5PC-SH				
	OFT	ONT	TPT	CPUT
Garblers	1937 ± 68	41 ± 2	1978 ± 70	1348 ± 12
Evaluator	526 ± 158	214 ± 5	740 ± 163	338 ± 5
BLO				
	OFT	ONT	TPT	CPUT
All parties	6007 ± 159	139 ± 2	6146 ± 161	2593 ± 14

Table 6: Execution times [ms]: SHA-256 circuit, med. latency

slowest link (East to West) being 92.8 milliseconds (maximum time of 974.6 milliseconds). The average bandwidth, again of the slowest link, as measured by the Iperf testing tool was 292 Mbps. The results are presented and compared in Tables 5 and 6. As communication becomes critical in medium-latency networks, our protocols perform significantly better than existing protocols in this domain. As can be seen from the table, for the AES circuit, our 5PC-M is actually 1.83 times faster than BLO (which, again, is only passively secure) in terms of total execution time; this is due to our better overall communication complexity as this factor dominates even in medium-latency networks . Our 5PC-SH is 2.6 times faster than BLO. For the SHA-256 circuit, our 5PC-M is once again 1.77 times faster than BLO, while our 5PC-SH is 3.1 times faster than BLO.

High-latency network. For the last experiment, 3 garblers were located in the Western US region and the remaining garbler and the evaluator located in North Europe. with an average round-trip time of the slowest link (West US to North Europe) being 142.6 milliseconds (maximum time of 153.2 milliseconds). The average bandwidth, again of the slowest link, as measured by the Iperf testing tool was 146 Mbps. The results[3] are presented and compared in Tables 7 and 8. For

[3]Network time is the huge dominating factor in this case. Since we measure average time of garblers, when one of the links between a garbler and evaluator is slow, the time of the evaluator is affected more than the average time of

5PC-M				
	OFT	ONT	TPT	CPUT
Garblers	1655 ± 104	220 ± 18	1875 ± 122	196 ± 2
Evaluator	1116 ± 45	316 ± 5	1432 ± 50	63 ± 2
5PC-SH				
	OFT	ONT	TPT	CPUT
Garblers	790 ± 96	110 ± 28	900 ± 124	120 ± 1
Evaluator	750 ± 32	264 ± 30	1014 ± 62	39 ± 1
BLO				
	OFT	ONT	TPT	CPUT
All parties	4556 ± 256	299 ± 10	4855 ± 266	249 ± 51

Table 7: Execution times [ms]: AES circuit, high latency

5PC-M				
	OFT	ONT	TPT	CPUT
Garblers	7529 ± 478	242 ± 32	7771 ± 510	2875 ± 32
Evaluator	2444 ± 214	453 ± 4	2897 ± 219	642 ± 14
5PC-SH				
	OFT	ONT	TPT	CPUT
Garblers	3722 ± 384	221 ± 23	3943 ± 407	1409 ± 17
Evaluator	3069 ± 396	455 ± 14	3524 ± 410	397 ± 12
BLO				
	OFT	ONT	TPT	CPUT
All parties	12957 ± 624	366 ± 18	13323 ± 642	2751 ± 16

Table 8: Execution times [ms]: SHA-256 circuit, high latency

AES, our 5PC-M is 2.6 times faster than BLO in total execution time and our 5PC-SH is 4.8 times faster than BLO. For SHA-256, our 5PC-M is 1.7 times faster than BLO, while 5PC-SH is 3.38 times faster than BLO.

7 The n-party case

So far, we have described our efficient MPC protocol for the case of five parties. It turns out that the ideas easily generalize to more than five parties as long as we can find appropriate seed distribution strategies that meet certain combinatorial properties. Next, we review these properties and various seed assignment strategies.

To generalize our approach to n-party MPC with at most t corrupted parties, similarly to the five-party case, we let P_n be the evaluator and have the remining $n-1$ parties simulate a q-party distributed garbling scheme (q-DG) to generate the garbled circuit used for evaluation. We focus on the generalization of the distributed garbling component of the MPC protocol and note that similar ideas can be used for the generalization of the garbled-input step.

the garblers, sometimes leading to a longer total time for the evaluator.

At a high level, this q-DG protocol needs to satisfy two main properties: (i) Produce a correct garbled circuit even if up to $t < n$ of the garblers are corrupted, and (ii) hide its randomness from the adversary (i.e., the randomness used to garble the circuit) even if up to $t-1$ garblers are corrupted. The adversary's corruption strategy can be split into two cases. If the adversary corrupts t garblers, but the evaluator remains honest, the first condition ensures that the garbled circuit is honestly generated and the honest evaluator (P_n) will evaluate the correct garbled circuit. If, on the other hand, the adversary corrupts P_n and $t-1$ of the garblers, the second property ensures that the corrupt evaluator does not learn the secrets of the garbled circuit and hence no information about the honest garblers' inputs is revealed.

To obtain a protocol that meets the above two properties, we follow the same approach as in the five-party case. In particular, we assume that all the randomness needed by the party i in the q-party distributed garbling is generated using a random seed s_i. Hence, we have q seeds $s_1, \ldots, s_q$ that need to be distributed among the $n-1$ garblers. This seed distribution step should satisfy the following properties:

1. **Privacy:** No $t-1$ garblers[4] should hold all seeds. This property is required for both the semi-honest and the malicious variants of our MPC protocol, and ensures that a corrupt evaluator does not learn an honest garbler's input to the computation.
2. **OT Attestation:** For every pair of seeds s_i, s_j, there should be a party P_k that holds both seeds. This party will play the role of attester in our AOT protocols, which we use as replacement for OT. We note that a more expensive variant of our construction without AOT can do without this condition.
3. **Correctness:** Every seed is held by at least $t+1$ parties. This property is only needed for the actively malicious case as it ensures that when each message of the semi-honest garbling protocol is received from $t+1$ parties, at least one of those messages was generated by an honest party and hence it must be the correct message.

7.1 The (n, t, q)-Assigment Problem

The above requirements yield an interesting combinatorial problem, which we call the (n, t, q)-*assignment problem*, for finding seed assignments that minimize n and q but maximize the corruption threshold t. It is easy to obeserve that the $(5, 2, 4)$-assignment we used in our 5PC protocol $(\{s_1, s_3, s_4\}, \{s_2, s_3, s_4\}, \{s_1, s_2, s_3\}\{s_1, s_2, s_4\})$ has all three properties. Next, we provide a general solution to the problem that works for all values of n, and also explore better assignments for particular values of n. A more thorough study of the problem is left as future work.

We describe a simple solution for the case $n = t^2 + 1$ and $q = t^2$ that meeds the Privacy and Correctness properties discussed above. Let $s_1, \ldots, s_q$ be the random seeds we need to assign to $n-1$ parties. We simply assign to P_i the seeds $s_{((i-1) \mod q)+1}, \ldots, s_{((i+t) \mod q)+1}$. The modular operation enforces that we "cycle around" when s_q is reached. It is easy to see that this assignment satisfies Privacy since even if the $t-1$ parties chosen not to have any overlap in their set of assigned seeds, they can only cover $(t-1)(t+1) = t^2 - 1 = q - 1$ seeds. Note that this assignment does not meet the OT Attestation condition and hence needs to be instantiated using standard OTs (with OT extension), where each OT message is computed by the $t+1$ parties who hold the seeds for either the sender or the receiver in that OT instance. Finally, it is easy to see that each seed s_i is held by $t+1$ parties, namely, $P_{i \mod q}, \ldots, P_{(i+t+1) \mod q}$.

[4] That's right, $t-1$ and not t, as we have to account for the possiblity of the evaluator being corrupted.

7.2 A Few Special Cases

The above assignment strategy is general but does not always yield the optimal assignment. For example, while it yields the optimal solution for $t = 2$ and $n = 5$, the best solution it yields for $t = 3$ is $n = 10$ and $q = 9$. We now show a simple alternative assignment where $q = 6$ suffices which implies fewer seeds and a more efficient distributed garbling protocol.

$(10, 3, 6)$-assignment (active security). Consider the first five seeds $s_1, \ldots, s_5$. There are $\binom{5}{4} = 5$ subsets of size four. We assign each subset to one of the first five parties and assign s_6 to P_6, P_7, P_8, P_9. As before, P_{10} is not a garbler. It is easy to see that no two parties hold all the seeds and each seed is held by at least four parties. This satisfies both the Privacy and Correctness conditions that are sufficient for actively secure MPC. This solution also generalizes easily to a solution with $q \approx n$ and $t \approx \sqrt{n}$.

$(7, 3, 6)$-assignment (semi-honest security). The assignment of the six seeds to the six garblers is as follows: $\{s_1, s_2, s_3\}, \{s_3, s_4, s_5\}, \{s_2, s_4, s_5\}, \{s_1, s_5, s_6\}, \{s_2, s_3, s_6\}, \{s_1, s_4, s_6\}$. This assignment meets the Privacy and OT Attestation conditions, yielding a very efficient semi-honest 7PC protocol that can fully benefit from our AOT protocols. In particular, it is easy to see that no two parties hold all six seeds and that every pair of seeds is held by at least one of the garblers.

Acknowledgments

We thank the anonymous reviewers of ACM CCS 2017 for providing useful feedback that helped improve the presentation of the paper.

References

[AFL+16] Toshinori Araki, Jun Furukawa, Yehuda Lindell, Ariel Nof, and Kazuma Ohara. High-throughput semi-honest secure three-party computation with an honest majority. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, pages 805–817. ACM, 2016.

[asS11] abhi shelat and Chih-Hao Shen. Two-output secure computation with malicious adversaries. In *Advances in Cryptology - EUROCRYPT 2011 - 30th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Tallinn, Estonia, May 15-19, 2011. Proceedings*, pages 386–405, 2011.

[asS13] abhi shelat and Chih-Hao Shen. Fast two-party secure computation with minimal assumptions. In *2013 ACM SIGSAC Conference on Computer and Communications Security, CCS'13, Berlin, Germany, November 4-8, 2013*, pages 523–534, 2013.

[BDNP08] Assaf Ben-David, Noam Nisan, and Benny Pinkas. Fairplaymp: a system for secure multi-party computation. In *Proceedings of the 15th ACM conference on Computer and communications security*, pages 257–266. ACM, 2008.

[BGW88] Michael Ben-Or, Shafi Goldwasser, and Avi Wigderson. Completeness theorems for non-cryptographic fault-tolerant distributed computation (extended abstract). In *STOC* [DBL88], pages 1–10.

[BHR12] Mihir Bellare, Viet Tung Hoang, and Phillip Rogaway. Foundations of garbled circuits. In *the ACM Conference on Computer and Communications Security, CCS'12, Raleigh, NC, USA, October 16-18, 2012*, pages 784–796, 2012.

[Bla06] John Black. The ideal-cipher model, revisited: An uninstantiable blockcipher-based hash function. In *Fast Software Encryption, 13th International Workshop, FSE 2006*, volume 4047 of *Lecture Notes in Computer Science*, pages 328–340. Springer, 2006.

[BLO16a] Aner Ben-Efraim, Yehuda Lindell, and Eran Omri. Implementation of protocol from blo16. https://github.com/cryptobiu/Semi-Honest-BMR, 2016.

[BLO16b] Aner Ben-Efraim, Yehuda Lindell, and Eran Omri. Optimizing semi-honest secure multiparty computation for the internet. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security, Vienna, Austria, October 24-28, 2016*, pages 578–590, 2016.

[BLW08] Dan Bogdanov, Sven Laur, and Jan Willemson. Sharemind: A framework for fast privacy-preserving computations. In *European Symposium on Research in Computer Security*, pages 192–206. Springer, 2008.

[BMR90] Donald Beaver, Silvio Micali, and Phillip Rogaway. The round complexity of secure protocols (extended abstract). In *Proceedings of the 22nd Annual ACM Symposium on Theory of Computing, May 13-17, 1990, Baltimore, Maryland, USA*, pages 503–513, 1990.

[Can00] Ran Canetti. Security and composition of multiparty cryptographic protocols. 13(1):143–202, 2000.

[Can01] Ran Canetti. Universally composable security: A new paradigm for cryptographic protocols. In *42nd Annual Symposium on Foundations of Computer Science, FOCS 2001, 14-17 October 2001, Las Vegas, Nevada, USA*, pages 136–145, 2001.

[CCD88] David Chaum, Claude Crépeau, and Ivan Damgård. Multiparty unconditionally secure protocols (extended abstract). In *STOC* [DBL88], pages 11–19.

[CDI05] Ronald Cramer, Ivan Damgård, and Yuval Ishai. Share conversion, pseudorandom secret-sharing and applications to secure computation. In *Theory of Cryptography, Second Theory of Cryptography Conference, TCC 2005, Cambridge, MA, USA, February 10-12, 2005, Proceedings*, pages 342–362, 2005.

[CHK+12] Seung Geol Choi, Kyung-Wook Hwang, Jonathan Katz, Tal Malkin, and Dan Rubenstein. Secure multi-party computation of boolean circuits with applications to privacy in on-line marketplaces. In *Cryptographers Track at the RSA Conference*, pages 416–432. Springer, 2012.

[CKMZ14] Seung Geol Choi, Jonathan Katz, Alex J. Malozemoff, and Vassilis Zikas. Efficient three-party computation from cut-and-choose. In *Advances in Cryptology - CRYPTO 2014 - 34th Annual Cryptology Conference, Santa Barbara, CA, USA, August 17-21, 2014, Proceedings, Part II*, pages 513–530, 2014.

[CMF+14] Koji Chida, Gembu Morohashi, Hitoshi Fuji, Fumihiko Magata, Akiko Fujimura, Koki Hamada, Dai Ikarashi, and Ryuichi Yamamoto. Implementation and evaluation of an efficient secure computation system using rfor healthcare statistics. *Journal of the American Medical Informatics Association*, 21(e2):e326–e331, 2014.

[DBL88] *Proceedings of the Twentieth Annual ACM Symposium on Theory of Computing, 2-4 May 1988, Chicago, Illinois, USA*. ACM, 1988.

[DI05] Ivan Damgård and Yuval Ishai. Constant-round multiparty computation using a black-box pseudorandom generator. In *Advances in Cryptology - CRYPTO 2005: 25th Annual International Cryptology Conference, Santa Barbara, California, USA, August 14-18, 2005, Proceedings*, pages 378–394, 2005.

[DKL+13] Ivan Damgård, Marcel Keller, Enrique Larraia, Valerio Pastro, Peter Scholl, and Nigel P Smart. Practical covertly secure mpc for dishonest majority–or: breaking the spdz limits. In *European Symposium on Research in Computer Security*, pages 1–18. Springer, 2013.

[DPSZ12] Ivan Damgård, Valerio Pastro, Nigel Smart, and Sarah Zakarias. Multiparty computation from somewhat homomorphic encryption. In *Advances in Cryptology–CRYPTO 2012*, pages 643–662. Springer, 2012.

[FLNW16] Jun Furukawa, Yehuda Lindell, Ariel Nof, and Or Weinstein. High-throughput secure three-party computation for malicious adversaries and an honest majority. pages 554–581. Springer, 2016.

[GMS08] Vipul Goyal, Payman Mohassel, and Adam Smith. Efficient two party and multi party computation against covert adversaries. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 289–306. Springer, 2008.

[GMW87] Oded Goldreich, Silvio Micali, and Avi Wigderson. How to play any mental game or a completeness theorem for protocols with honest majority. In *STOC*, pages 218–229. ACM, 1987.

[HKE13] Yan Huang, Jonathan Katz, and David Evans. Efficient secure two-party computation using symmetric cut-and-choose. In *CRYPTO (2)*, volume 8043 of *Lecture Notes in Computer Science*, pages 18–35. Springer, 2013.

[HKK+14] Yan Huang, Jonathan Katz, Vladimir Kolesnikov, Ranjit Kumaresan, and Alex J Malozemoff. Amortizing garbled circuits. In *International Cryptology Conference*, pages 458–475. Springer, 2014.

[HKT10] Thomas Holenstein, Robin Künzler, and Stefano Tessaro. Equivalence of the random oracle model and the ideal cipher model, revisited. *CoRR*, abs/1011.1264, 2010.

[HSSV17] Carmit Hazay, Peter Scholl, and Eduardo Soria-Vazquez. Low cost constant round mpc combining bmr and oblivious transfer. Cryptology ePrint Archive, Report 2017/214, 2017. http://eprint.iacr.org/2017/214.

[IKKP15] Yuval Ishai, Ranjit Kumaresan, Eyal Kushilevitz, and Anat Paskin-Cherniavsky. Secure computation with minimal interaction, revisited. In *CRYPTO (2)*, volume 9216 of *Lecture Notes in Computer Science*, pages 359–378. Springer, 2015.

[IKNP03] Yuval Ishai, Joe Kilian, Kobbi Nissim, and Erez Petrank. Extending oblivious transfers efficiently. In *Advances in Cryptology - CRYPTO 2003, 23rd Annual International Cryptology Conference, Santa Barbara, California, USA, August 17-21, 2003, Proceedings*, pages 145–161, 2003.

[IKP10] Yuval Ishai, Eyal Kushilevitz, and Anat Paskin. Secure multiparty computation with minimal interaction. In *CRYPTO*, volume 6223 of *Lecture Notes in Computer Science*, pages 577–594. Springer, 2010.

[KasS12] Benjamin Kreuter, abhi shelat, and Chih-Hao Shen. Billion-gate secure computation with malicious adversaries. In *Proceedings of the 21th USENIX Security Symposium, Bellevue, WA, USA, August 8-10, 2012*, pages 285–300, 2012.

[KRW17] Jonathan Katz, Samuel Ranellucci, and Xiao Wang. Authenticated garbling and efficient maliciously secure multi-party computation. Cryptology ePrint Archive, Report 2017/189, 2017. http://eprint.iacr.org/2017/189.

[KSS13] Marcel Keller, Peter Scholl, and Nigel P Smart. An architecture for practical actively secure mpc with dishonest majority. In *Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security*, pages 549–560. ACM, 2013.

[LDDAM12] John Launchbury, Iavor S Diatchki, Thomas DuBuisson, and Andy Adams-Moran. Efficient lookup-table protocol in secure multiparty computation. In *ACM SIGPLAN Notices*, volume 47, pages 189–200. ACM, 2012.

[Lin13] Yehuda Lindell. Fast cut-and-choose based protocols for malicious and covert adversaries. In *Advances in Cryptology - CRYPTO 2013 - 33rd Annual Cryptology Conference, Santa Barbara, CA, USA, August 18-22, 2013. Proceedings, Part II*, pages 1–17, 2013.

[LP07] Yehuda Lindell and Benny Pinkas. An efficient protocol for secure two-party computation in the presence of malicious adversaries. In *Advances in Cryptology - EUROCRYPT 2007, 26th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Barcelona, Spain, May 20-24, 2007, Proceedings*, pages 52–78, 2007.

[LP12] Yehuda Lindell and Benny Pinkas. Secure two-party computation via cut-and-choose oblivious transfer. *J. Cryptology*, 25(4):680–722, 2012.

[LPS08] Yehuda Lindell, Benny Pinkas, and Nigel P. Smart. Implementing two-party computation efficiently with security against malicious adversaries. In *Security and Cryptography for Networks, 6th International Conference, SCN 2008, Amalfi, Italy, September 10-12, 2008. Proceedings*, pages 2–20, 2008.

[LPSY15] Yehuda Lindell, Benny Pinkas, Nigel P. Smart, and Avishay Yanai. Efficient constant round multi-party computation combining BMR and SPDZ. In *CRYPTO (2)*, volume 9216 of *Lecture Notes in Computer Science*, pages 319–338. Springer, 2015.

[LR14] Yehuda Lindell and Ben Riva. Cut-and-choose yao-based secure computation in the online/offline and batch settings. In *International Cryptology Conference*, pages 476–494. Springer, 2014.

[LR15] Yehuda Lindell and Ben Riva. Blazing fast 2pc in the offline/online setting with security for malicious adversaries. In *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*, pages 579–590. ACM, 2015.

[MF06] Payman Mohassel and Matthew Franklin. Efficiency tradeoffs for malicious two-party computation, 2006.

[MR13] Payman Mohassel and Ben Riva. Garbled circuits checking garbled circuits: More efficient and secure two-party computation. In *Advances in Cryptology - CRYPTO 2013 - 33rd Annual Cryptology Conference, Santa Barbara, CA, USA, August 18-22, 2013. Proceedings, Part II*, pages 36–53, 2013.

[MRZ15] Payman Mohassel, Mike Rosulek, and Ye Zhang. Fast and secure three-party computation: The garbled circuit approach. In *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security, Denver, CO, USA, October 12-6, 2015*, pages 591–602, 2015.

[NO09] Jesper Buus Nielsen and Claudio Orlandi. *LEGO for Two-Party Secure Computation*, pages 368–386. Springer Berlin Heidelberg, Berlin, Heidelberg, 2009.

[PSSW09] Benny Pinkas, Thomas Schneider, Nigel P. Smart, and Stephen C. Williams. Secure two-party computation is practical. In *Advances in Cryptology - ASIACRYPT 2009, 15th International Conference on the Theory and Application of Cryptology and Information Security, Tokyo, Japan, December 6-10, 2009. Proceedings*, pages 250–267, 2009.

[Sha49] Claude E Shannon. Communication theory of secrecy systems. *Bell Labs Technical Journal*, 28(4):656–715, 1949.

[Woo07] David P. Woodruff. Revisiting the efficiency of malicious two-party computation. In *Advances in Cryptology - EUROCRYPT 2007, 26th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Barcelona, Spain, May 20-24, 2007, Proceedings*, volume 4515 of *Lecture Notes in Computer Science*, pages 79–96. Springer, 2007.

[Yao82] Andrew Chi-Chih Yao. Protocols for secure computations (extended abstract). In *FOCS*, pages 160–164. IEEE, 1982.

[ZSB13] Yihua Zhang, Aaron Steele, and Marina Blanton. Picco: a general-purpose compiler for private distributed computation. In *Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security*, pages 813–826. ACM, 2013.

A Preliminaries (cont'd)

In this section we present complementary preliminary material, including the definition of the cryptographic building blocks.

Definition 5. *A* pseudo-random function *(PRF) family $\mathcal{PRF}$ is a family of functions* $\mathsf{PRF} : K \times D \to R$. *The setup algorithm produces (on input 1^κ) a key k at random from K. Let $\mathsf{PRF}_k(\cdot)$ be the function parameterized by k. The security property requires that when k is chosen at random from K, no PPT adversary $\mathcal{A}$ can distinguish between $\mathsf{PRF}_k(\cdot)$ and a random function (with appropriate domain and range), when given oracle access to one of the functions.*

Definition 6. *A* collision-resistant hash function *(CRHF) family $\mathcal{H}$ is a family of functions $H : K \times D \to R$, where $|D| < |R|$. The setup algorithm produces (on input 1^κ) a key k at random from K. Let $H_k(\cdot)$ be the function parameterized by k. The security property requires that when k is chosen at random from K, no PPT adversary $\mathcal{A}$, given $\mathcal{H}$ and k, can produce x_0 and x_1 (with $x_0 \neq x_1$) such that $H_k(x_0) = H_k(x_1)$, except with negligible probability in κ.*

Definition 7. *A* (non-interactive) commitment scheme *(for a message space M) is a triple of algorithms* $(\mathsf{ComGen}, \mathsf{Com}, \mathsf{Open})$ *such that:*

- $CK \leftarrow \mathsf{ComGen}(1^\kappa)$, *where CK is the public commitment key;*
- *for $m \in \mathsf{M}$, $(c, d) \leftarrow \mathsf{Com}(m)$ is the commitment/decommitment pair for m. (We omit mentioning the public key CK when it is clear from the context.) When we wish to make explicit the randomness used by $\mathsf{Com}(\cdot)$, we write $\mathsf{Com}(m; r)$;*
- $\tilde{m} \leftarrow \mathsf{Open}(c, d)$, *where $\tilde{m} \in \mathsf{M} \cup \{\perp\}$, and where $\perp$ is returned if c is not a valid commitment to any message;*

satisfying the following properties:

- *Correctness: For any $m \in \mathsf{M}$, $\mathsf{Open}(\mathsf{Com}(m)) = m$.*
- *Hiding: For all $m_0, m_1 \in \mathsf{M}$ output by any PPT adversary $\mathcal{A}$, the distributions c_0 and c_1 are indistinguishable to $\mathcal{A}$, where $(c_b, d_b) \leftarrow \mathsf{Com}(m_b)$ for $b \in \{0, 1\}$, except with negligible probability.*
- *Binding: No PPT adversary $\mathcal{A}$ can produce (c, d_0, d_1, m_0, m_1) (with $m_0 \neq m_1$) such that $\mathsf{Open}(c, d_b) \to m_b$ for $b \in \{0, 1\}$, except with negligible probability.*

Oblivious Transfer. While our protocol does not make use of the oblivious transfer primitive, we present the primitive below for completeness (as the protocol of [BLO16b], which we modify, does). The oblivious transfer (OT) functionality is described in Figure 8; for a description of a protocol implementing the functionality, we refer the reader to [BLO16b].

B Proofs

LEMMA 3.1 *Assuming* $(\mathsf{ComGen}, \mathsf{Com}, \mathsf{Open})$ *is a secure commitment scheme, protocol Π_{4AOT} securely realizes the $\mathcal{F}_{4AOT}$ functionality.*

Proof. (Sketch) We shall prove this considering various corruption scenarios and providing simulator strategies for each. For any adversary $\mathcal{A}$ corrupting parties, we describe a simulator $\mathcal{S}$ interacting with the ideal functionality $\mathcal{F}_{4\text{AOT}}$. We first consider the case when only one party is corrupted. P_1 is the sender and P_2 is the receiver.

Functionality $\mathcal{F}_{\text{OT}}(P_1, P_2)$

$\mathcal{F}_{\text{OT}}$ interacts with parties P_1 and P_2 and the adversary $\mathcal{S}$, with P_1 and P_2 acting as sender and receiver, respectively.

Input.

- On input message $(\texttt{Sender}, sid, m_0, m_1)$ from P_1, where each $m_\beta \in \mathsf{M}$, record (m_0, m_1) and send $(\texttt{Sender}, sid)$ to the adversary. Ignore further $(\texttt{Sender}, \cdot, \cdot, \cdot)$ messages.
- On input message $(\texttt{Receiver}, sid, b)$ from P_2, where $b \in \{0, 1\}$, record b and send $(\texttt{Receiver}, sid)$ to the adversary. Ignore further $(\texttt{Receiver}, \cdot, \cdot)$ inputs.

Output. On input message $(\texttt{Output}, sid)$ from the adversary, send $(\texttt{Output}, sid, \texttt{success}, m_b)$ to P_2.

Figure 8: *The Oblivious Transfer ideal functionality* $\mathcal{F}_{\text{OT}}$.

- P_1 is corrupted: P_2, P_3 and P_4 are honest. $\mathcal{S}$ runs $\mathcal{A}$. It receives two tuples from $\mathcal{A}$, $(m_0^3, m_1^3, r_0^3, r_1^3)$ and $(m_0^4, m_1^4, r_0^4, r_1^4)$ intended for P_3 and P_4. If the tuples are not equal, $\mathcal{S}$ sends $\perp$ to the functionality, and simulates the honest parties aborting. If the tuples are the same, $\mathcal{A}$ will send $(\mathsf{Com}_0^1, \mathsf{Com}_1^1$ intended for P_2. $\mathcal{S}$ verifies that the commitments are correctly generated using the tuple it obtained earlier. If not, it sends an `Abort` message to the functionality; else, it submits $(m_0^3, m_1^3, b^3, r_0^3, r_1^3)$ as P_1's input to the functionality. This completes the simulator's description.

  Note that in the real execution, if P_3 and P_4 receive two sets of values $(m_0^3, m_1^3, r_0^3, r_1^3) \neq (m_0^4, m_1^4, r_0^4, r_1^4)$, then P_3 and P_4 will detect this and induce an abort in Step 3b of the protocol. This abort is independent of P_2's input b and identically distributed to $\mathcal{S}$'s abort. If P_3 and P_4 receive two sets of values that are equal, then P_3 and P_4 will generate and send $(\mathsf{Com}_0^1, \mathsf{Com}_1^1, \mathsf{Open}_{m_{b^2}^1}^1)$ to P_2 (as $m_0^3 = m_0^4$ and $m_1^3 = m_1^4$, when P_3 and P_4 are honest). If P_1 sends a different set of commitments $(\overline{\mathsf{Com}}_0^1, \overline{\mathsf{Com}}_1^1)$ in Step 1 of the protocol, then P_2 will detect this and abort. Once again, this abort is independent of b and identically distributed to $\mathcal{S}$'s abort. Now, suppose $(\overline{\mathsf{Com}}_0^1, \overline{\mathsf{Com}}_1^1) = (\mathsf{Com}_0^3, \mathsf{Com}_1^3)$ (i.e., the commitments to m_0^1 and m_1^1 by P_1, P_3 and P_4 are identical), then P_2 indeed receives the opening to Com_b^1 from P_3 and P_4 and hence outputs m_b.
- P_2 is corrupted: In this case, P_1, P_3 and P_4 are honest. $\mathcal{S}$ runs $\mathcal{A}$. It receives two bits from the adversary, b^3, b^4, intended for P_3 and P_4. If the two values are different, $\mathcal{S}$ sends `Abort` to the functionality and simulates the honest parties aborting. If the two bits are the same, it submits b^3 as P_2's input to the ideal functionality and receives $m_{b^3}^3$ from the functionality. It then generates two commitments/openings $(\mathsf{Com}_0^1, \mathsf{Com}_1^1)$, one committing to $m_{b^3}^3$, and another to a dummy value, say, 0 (permuted based on the bit b^3), and sends the two commitments to $\mathcal{A}$ on behalf of honest P_1. It then sends decommitment for $m_{b^3}^3$ on behalf of honest P_3. This completes the simulation.

  In the real execution, P_2 will receive commitments $(\mathsf{Com}_0^1, \mathsf{Com}_1^1)$ as well as the decommitment $\mathsf{Open}_{b^2}^1$ in Step 3a. By the hiding property of the commitment scheme, P_2 will learn no information about $m_{\bar{b}^2}^1$ and it can be replaced by a commitment to 0, making the real and simulated views indistinguishable.
- P_3 is corrupted: since P_1, P_2 are honest, $\mathcal{S}$ receives (m_0^3, m_1^3, b^3) from the functionality. It then generates fresh randomness r_0^3, r_1^3 and sends $(m_0^3, m_1^3, r_0^3, r_1^3)$ on behalf of P_1 and b^3 on behalf of P_2 to the adversary. If $\mathcal{A}$ sends a different tuple intended for honest P_4, $\mathcal{S}$ sends `Abort`

to the functionality and simulates the honest P_4 aborting. Else, it receives two commitments and a decommitment from $\mathcal{A}$ intended for P_2. If the commitments and/or decommitment are not consistent with the tuples it sent to $\mathcal{A}$ earlier, $\mathcal{S}$ sends `Abort` to the functionality. This completes the simulation.

In the real execution, if P_3 cheats by sending a different set of $(m_0^3, m_1^3, b^3, r_0^3, r_1^3)$ values in Step 3b, then P_4 will send a $\perp$ message to P_2, which is what our simulator does as well. If P_3 cheats by sending a different set of commitments to P_2 in Step 3a, then again P_2 will detect this as honest P_4 (and P_1) send honest versions of these commitments. If P_3 cheats by sending a different opening of the commitment $\mathsf{Com}_{b^3}^3$, then by the binding property of the commitment scheme, P_3 can indeed only open $\mathsf{Com}_{b^3}^3 = \mathsf{Com}_{b^2}^1$ to $m_{b^2}^1$ and hence P_2 will output $m_{b^2}^1$ (or abort) which is identical to $\mathcal{S}$'s behavior.

- P_4 is corrupted: This case is similar to when P_3 is corrupted.

Next, let us consider the case when two parties are corrupted: Note that our functionality in this case does not require privacy of inputs since corrupted attesters will learn both parties' inputs. It only guarantees that an honest P_2 will always output the right m_b (or abort); hence, the only interesting cases are when P_2 is honest, i.e., P_1 and P_3 are corrupted, or P_3 and P_4 are corrupted (other cases are symmetric).

- P_1 and P_3 are corrupted: P_2 and P_4 are honest. $\mathcal{S}$ runs $\mathcal{A}$. It receives a tuple from $\mathcal{A}$, $(m_0^4, m_1^4, r_0^4, r_1^4)$ intended from P_1 to P_4 and another tuple $(m_0^3, m_1^3, b^3, r_0^3, r_1^3)$ intended from P_3 to P_4. If the tuples do not hold the same values, $\mathcal{S}$ sends `Abort` to the functionality. , simulating the honest P_4 aborting. If the tuples are the same, $\mathcal{A}$ will send $(\mathsf{Com}_0^1, \mathsf{Com}_1^1$ intended for P_2. $\mathcal{S}$ verifies that the commitments are correctly generated using the tuple it obtained earlier. If not, it sends `Abort` to the functionality, inducing P_2's abort. Else, it submits $(m_0^4, m_1^4, b^4, r_0^4, r_1^4)$ as P_1's input to the functionality. This completes the simulator's description.

  Consider the real execution. Recall that P_4 is honest in this case. If P_3 sends a different set of values in Step 3b, then P_4 will send a $\perp$ message, and P_2 will abort the protocol. Now, if P_1 or P_3 send maliciously generated messages $(\overline{\mathsf{Com}}_0^i, \overline{\mathsf{Com}}_1^i)$ (for $i = 1, 3$, in Step 1 or Step 3a, respectively), then P_2 will detect this and output $\perp$ when P_4 sends the correct $(\mathsf{Com}_0^4, \mathsf{Com}_1^4)$ to P_1 in Step 3a. Similarly, if P_3 sends a maliciously generated message $(\bar{m}_{b^3}^3, \bar{r}_{b^3})$ (in Step 3a), then, by the binding property of the commitment scheme, $\bar{m}_{b^3}^3 = m_{b^2}^1$. Hence, P_2 always outputs $m_{b^2}^1$ or aborts. It is easy to see that the aborts are identically distributed to the simulation.
- P_3 and P_4 are corrupted: In this case, P_1 and P_2 are honest and the simulation is very similar to the case above where P_3 was corrupted with the only difference that $\mathcal{S}$ does not simulate an honest P_4 aborting since P_4 is not honest in this case.

□

THEOREM 4.1 *Assuming* $(\mathsf{ComGen}, \mathsf{Com}, \mathsf{Open})$ *is a secure commitment scheme, and* $H \xleftarrow{\$} \mathcal{H}$ *is a collision-resistant hash function, protocol* $\Pi_{\text{5PC}}(C, \{P_1, \ldots, P_5\})$ *securely realizes the functionality* $\mathcal{F}_{\text{SFE}}^C(\{P_1, \ldots, P_5\})$[5] *in the* $\mathcal{F}_{\text{4AOT}}$*-hybrid model.*

Proof. (Sketch) To prove our 5PC protocol $\Pi_{\text{5PC}}(C, \{P_1, \ldots, P_5\})$ secure, for any adversary $\mathcal{A}$ in the protocol, we describe a simulator $\mathcal{S}$ that interacts with the ideal functionality $\mathcal{F}_{\text{SFE}}^C(\{P_1, \ldots, P_5\})$. There are two main corruption scenarios to consider: (i) When two garblers are corrupted. In this case, without loss of generality we assume P_1 and P_2 are corrupted since the protocol is symmetric

[5]Recall that we slightly abuse notation, and mean security with abort.

with respect to the garblers; and (ii) when the evaluator P_5 and one of the garblers is corrupted. Similary, wlog, we assume P_1 and P_5 are corrupted.

P_1 and P_2 are corrupted. At a high level, in this case the evaluator is honest, and hence the only guarantee we need from the distributed circuit garbling is to generate a correct garbled circuit. We also require that the garblers' inputs are extractable from the garbled input generation process. These two properties combined will guarantee that we can describe a simulator that simulates any adversary corrupting P_1 and P_2. More details follow.

$\mathcal{S}$ runs $\mathcal{A}$. $\mathcal{S}$ receives two copies of each seed s_1, s_2 intended for honest parties P_3 and P_4 from $\mathcal{A}$. $\mathcal{S}$ checks whether the two copies are the same or not. If not, it sends an abort message to the functionality. Else $\mathcal{S}$ generates random seeds s_3 and s_4 on behalf of honest P_3 and P_4 and sends them to the adversary. It then generates random inputs x_3, x_4, x_5 for P_3, P_4, P_5 and uses them in the rest of the simulation.

During the garbled input generation $\mathcal{S}$ behaves as honest P_3, P_4, P_5 in most cases and using the random inputs and seeds it generated above, and sends an abort to the functionality if it detects any cheating, or if the adversary opens the commitments generated for P_5's garbled input generation to a different value than expected (this is indistinguishable from the real-world interaction due to the binding property of the commitment). The aborts are independent of the honest parties' inputs as the inputs are always XORed with three uniformly random pads one of which is held by an honest party. If there is no abort, for each input wire of P_1, $\mathcal{A}$ sends $F_{s_1}(\text{`}\texttt{key}\text{'}||w||0) \oplus F_{s_1}(\text{`}\texttt{delta}\text{'}) \cdot b'$ intended for P_5. Given that $\mathcal{S}$ has knowledge of all seeds, it can use it to derive b' and further derive P_1's input $b = b' \oplus p_w$. A similar strategy can be used to extract P_2's input. Denote these inputs by x'_1, x'_2.

In the distributed garbling stage, $\mathcal{S}$ behaves as honest P_3 and P_4 and instructs the functionality to abort if it detects any cheating, i.e., if messages intended for P_3 and P_4 are not consistent. If there is no abort, it is easy to see that the garbling function described in Figure 3 will be the output of honest parties. If the garbled circuit sent by $\mathcal{A}$ intended for P_5 (or its hash) is not the same as what the honest parties would have obtained, $\mathcal{S}$ sends abort to the functionality.

$\mathcal{S}$ then sends x'_1, x'_2 to the functionality. From the correctness of the garbling function, if there are no aborts, the garbled circuit evaluated by P_5 in the real protocol would evaluate to the same output as what the functionality returns.

P_1 and P_5 are corrupted. In this case, the evaluator is not honest but only one of the garblers is malicious. At a high level, after extracting the adversary's inputs for P_1 and P_5, the simulator obtains the output of the computation from the functionality and helps generate a fake circuit that always evaluates to that output, but is indistinguisable from the real garbled circuit in the adversary's view. Furthermore, it should be hard for the adversary (corrupted P_5) to produce any output label that translates to a different value than the hardcoded output in the fake garbled circuit. More details follow.

$\mathcal{S}$ runs $\mathcal{A}$. $\mathcal{S}$ receives two copies of seed s_1 intended for honest parties P_3 and P_4 from $\mathcal{A}$. $\mathcal{S}$ checks whether the two copies are the same or not. If not, it sends an abort message to the functionality. Else $\mathcal{S}$ generates random seeds s_2, s_3 and s_4 on behalf of honest P_2, P_3 and P_4 and sends s_3 and s_4 to the adversary. It then generates random inputs x_2, x_3 and x_4 for P_2, P_3 and P_4 and uses them in the rest of the simulation.

$\mathcal{S}$ extracts P_1's input in the distributed garbled input generation as in the previous case. Denote that by x'_1. Extracting P_5's input is somewhat similar. Note that $\mathcal{A}$ sends three XOR shares of its inputs to the garblers. At least two of these are honest parties and hence $\mathcal{S}$ obtains those shares.

The third share is extracted similarly to the input extraction for the garblers above since the process for garbling each share is similar and the three shares are XORed to obtain the extracted input x'_5.

$\mathcal{S}$ now calls the functionality with inputs x'_1, x'_5 for P_1 and P_5 and obtains $out = f(x'_1, x_2, x_3, x_4, x'_5)$. Next, the simulator who plays the role of honest parties P_2, P_3, P_4 needs to influence the distributed circuit garbling scheme to generate a fake garbled circuit that hard-codes *out* as its output when run on extracted inputs x'_1, x'_5 and the random inputs $\mathcal{S}$ generated on behalf of honest parties, while ensuring that this fake distributed circuit garbling is indistinguishable from the real distributed circuit garbling protocol from the adversary's point of view.

The idea behind this simulation is as follows and is similar to the one in [BLO16b] except in our case the adversary can be malicious on behalf of one garbler. $\mathcal{S}$ knows all the seeds generated above. Furthermore, it has full control of any randomness and garbled circuit shares generated using s_2 since only P_1 is corrupted among the garblers and he does not hold s_2, and $\mathcal{S}$ is playing the role of all three parties holding s_2. $\mathcal{S}$ participates in the distributed garbling as before for all intermediate gates. For the output gates, however, it needs to make sure that the labels corresponding to bits of *out* are always the labels encrypted in rows corresponding to evaluation using the extracted and random inputs $\mathcal{S}$ knows. Given that $\mathcal{S}$ has knowledge of all seeds, it knows what the corresponding label for *out* is (say, 0) and also knows the label corresponding to 1. For all such rows that encrypt the label 1, $\mathcal{S}$ can produce the one-time pads that are derived using s_2 (on behalf of honest P_2, P_3 and P_4) such that the encrypted label is flipped to the label for 0 instead. As a result, the generated garbled circuit will evaluate to *out*, and this process is indistinguishable from the adversary's point of view given that it only can decrypt one row of each table and other rows are indistinguishable from random given the semantic security of the encryption used for garbling and the fact that the adversary does not know all seeds.

$\mathcal{S}$ sends this fake garbled circuit (or its hash) on behalf of honest parties to $\mathcal{A}$ who controls the evaluator. Finally, $\mathcal{S}$ receives an output out' along with the corresponding output label. If it receives a different output than *out* from the adversary, it instructs the functionality to abort. The probability that P_5 can generate a different output label is negligible since in this fake garbled circuit only one label is decrypted and portions of the other label are encrypted using a seed that is not known to the adversary. As a result, the distributions in the real- and ideal-world interactions are indistinguishable. This completes the sketch of the proof. □

C Figures

Function f_{GC}^{C}

Inputs. All parties hold the circuit C, security parameter κ. In addition P_i holds the following private inputs:

1. A global difference string $R_i \in \{0,1\}^\kappa$ chosen at random;
2. For every wire w in C that is not the output of an XOR gate, a random permutation bit p_w^i and $k_{w,0}^i$ chosen at random from $\{0,1\}^\kappa$

Computation. Proceed as follows:

1. For $i \in [4]$, in a topological order, for every output wire w of an XOR gate with input wires u and v, set $p_w^i := p_u^i \oplus p_v^i$, $k_{w,0}^i := k_{u,0}^i \oplus k_{v,0}^i$ and $k_{w,1}^i := k_{w,0}^i \oplus R_i$.
2. For every w in C, set $p_w := \bigoplus_{i=1}^4 p_w^i$.
3. For every AND gate $g \in C$ with input wires u, v and output wire w, every $\alpha, \beta \in \{0,1\}$ and every $j \in [4]$, set:

$$g_{\alpha,\beta}^j := \Big(\bigoplus_{i=1}^{4} F_{k_{u,\alpha}^i}(g||j) \oplus F_{k_{v,\beta}^i}(g||j) \Big) \oplus k_{w,0}^i \oplus \big(R_i \cdot ((p_u \oplus \alpha) \cdot (p_v \oplus \beta) \oplus p_w) \big) \tag{2}$$

Outputs. Output to all parties $g_{\alpha,\beta}^1 || \ldots || g_{\alpha,\beta}^4$, for every AND gate $g \in C$ and every $\alpha, \beta \in \{0,1\}$.

Figure 9: *The distributed circuit-garbling function.*

Protocol $\Pi_{\text{SH4AOT}}(P_1, P_2, P_3)$

The protocol is carried out among P_1, P_2, P_3, with P_1 and P_2 acting as sender and receiver, respectively, and P_3 as the attester.

Input. P_1 holds m_0, m_1, and P_2 holds b.

Computation.

1. P_1 sends m_0, m_1 to P_3, and P_2 sends b to P_3.
2. P_3 sends m_b to P_2.

Output. P_2 outputs m_b.

Figure 10: *The passively secure 4-party protocol for Attested OT.*

Functionality $\mathcal{F}_{\text{B-4AOT}}(P_1, P_2, \{P_3, P_4\})$

$\mathcal{F}_{\text{B-4AOT}}$ interacts with parties P_1, P_2, P_3, P_4 and the adversary $\mathcal{S}$, with P_1 and P_2 acting as sender and receiver, respectively, and P_3, P_4 as attesters.

Input.

- On input message $(\texttt{Sender}, sid, \{m_{0,t}, m_{1,t}\}_{t\in[\ell]})$ from P_1, where each $m_{i,t} \in \mathsf{M}$, record $\{(m_{0,t}, m_{1,t})\}_{t\in[\ell]}$ and send $(\texttt{Sender}, sid, \{m_{0,t}, m_{1,t}\}_{t\in[\ell]})$ to P_3 and P_4 and $(\texttt{Sender}, sid)$ to the adversary. Ignore further $(\texttt{Sender}, ...)$ messages.
- On input message $(\texttt{Receiver}, sid, \{b_t\}_{t\in[\ell]})$ from P_2, where $b_t \in \{0, 1\}$, record $\{b_t\}_{t\in[\ell]}$ and send $(\texttt{Receiver}, sid, \{b_t\}_{t\in[\ell]})$ to P_3 and P_4 and $(\texttt{Receiver}, sid)$ to the adversary. Ignore further $(\texttt{Receiver}, ...)$ inputs.
- On input message $(\texttt{Attester}, sid, \{m^j_{0,t}, m^j_{1,t}, b^j_t\}_{t\in[\ell]})$ from P_j for $j \in \{3, 4\}$, where each $m^j_{i,t} \in \mathsf{M}$, record $\{(m^j_{0,t}, m^j_{1,t}, b^j_t)\}_{t\in[\ell]}$ and send $(\texttt{Attester}, sid)$ to the adversary. Ignore further $(\texttt{Attester}, ...)$ messages.

Output. On input message $(\texttt{Output}, sid)$ from the adversary, if $(m_{0,t}, m_{1,t}, b_t) \neq (m^3_{0,t}, m^3_{1,t}, b^3_t)$ or $(m_{0,t}, m_{1,t}, b_t) \neq (m^4_{0,t}, m^4_{1,t}, b^4_t)$ for any $t \in [\ell]$, send $(\texttt{Output}, sid, \perp)$ to P_2; else send $(\texttt{Output}, sid, \{m_{b_t,t}\}_{t\in[\ell]})$ to P_2.

Figure 11: *The 4-party Batch Attested OT ideal functionality* $\mathcal{F}_{\text{B-4AOT}}$.

Protocol $\Pi_{\text{B-4AOT}}(P_1, P_2, \{P_3, P_4\})$

The protocol is executed among P_1, P_2, P_3, P_4, with P_1 and P_2 acting as sender and receiver, respectively, and P_3, P_4 as attestors. Let $\mathsf{Commit} = (\mathsf{ComGen}, \mathsf{Com}, \mathsf{Open})$ be a secure noninteractive commitment scheme.

Inputs. P_1 holds $\{m^1_{0,t}, m^1_{1,t}\}_{t\in[\ell]}$, and P_2 holds $\{b^2_t\}_{t\in[\ell]}$.

1. P_1 generates random values $\{r_{0,t}, r_{1,t}\}_{t\in[\ell]} \leftarrow \{0,1\}^*$ and computes $(\mathsf{Com}^1_{0,t}, \mathsf{Open}_{0,t}) := \mathsf{Com}(m^1_{0,t}; r_{0,t})$, $(\mathsf{Com}^1_{1,t}, \mathsf{Open}_{1,t}) := \mathsf{Com}(m^1_{1,t}; r_{1,t})$. P_1 sends $\{\mathsf{Com}^1_{0,t}\}_{t\in[\ell]}$ and $\{\mathsf{Com}^1_{1,t}\}_{t\in[\ell]}$ to P_2 and sends $\{r_{0,t}, m_{0,t}, r_{1,t}, m_{1,t}\}_{t\in[\ell]}$ to P_3 and P_4, who store them as $\{r^3_{0,t}m^3_{0,t}, r^3_{1,t}, m^3_{1,t}\}_{t\in[\ell]}$ and $\{r^4_{0,t}, m^4_{0,t}, r^4_{1,t}, m^4_{1,t}\}_{t\in[\ell]}$, respectively.
2. P_3 and P_4 exchange hash of the values they received from P_1 i.e. $H(\{m^3_{0,t}, m^3_{1,t}, b^3_t, r^3_{0,t}, r^3_{1,t}\}_{t\in[\ell]})$ and $H(\{m^4_{0,t}, m^4_{1,t}, b^4_t, r^4_{0,t}, r^4_{1,t}\}_{t\in[\ell]})$.
   (a) If the values match, then for $i \in \{3,4\}$, P_i computes $(\mathsf{Com}^i_{0,t}, \mathsf{Open}^i_{0,t})$ and $(\mathsf{Com}^i_{1,t}, \mathsf{Open}^i_{1,t})$ using scheme Commit and random values $r^i_{0,t}$ and $r^i_{1,t}$ respectively, compute $H(\{\mathsf{Com}^i_{0,t}, \mathsf{Com}^i_{1,t}\}_{t\in[\ell]})$ and sends the hash value to P_2. (Wlog) P_3 also sends $\{\mathsf{Open}^3_{t,b^3_t}\}_{t\in[\ell]}$ to P_2.
   (b) If the hash values do not match, i.e., $H(\{m^3_{0,t}, m^3_{1,t}, b^3_t, r^3_{0,t}, r^3_{1,t}\}_{t\in[\ell]}) \neq H(\{m^4_{0,t}, m^4_{1,t}, b^4_t, r^4_{0,t}, r^4_{1,t}\}_{t\in[\ell]})$, they send $\perp$ message to P_2 (denoting abort).
3. P_2 checks the following and outputs $\perp$ if any of them is true: (i) it receives $\perp$ from P_3 or P_4; (ii) the hash of the set of three commitments pairs it has received from P_1, P_3, P_4 do not match; i.e., $H(\{\mathsf{Com}^1_{0,t}, \mathsf{Com}^1_{1,t}\}_{t\in[\ell]}) \neq H(\{\mathsf{Com}^3_{0,t}, \mathsf{Com}^3_{1,t}\}_{t\in[\ell]})$ or $H(\{\mathsf{Com}^1_{0,t}, \mathsf{Com}^1_{1,t}\}_{t\in[\ell]}) \neq H(\{\mathsf{Com}^4_{0,t}, \mathsf{Com}^4_{1,t}\}_{t\in[\ell]})$ (iii) $\mathsf{Open}(\mathsf{Com}^3_{t,b^3_t}, \mathsf{Open}^3_{t,b^3_t}) = \perp$ for $b^3_t = b^2_t$ and for any $t \in [\ell]$. Otherwise, P_2 outputs $\{m^3_{t,b^3_t}\}_{t\in[\ell]} \leftarrow \mathsf{Open}(\mathsf{Com}^3_{t,b^3_t}, \mathsf{Open}^3_{t,b^3_t})$.

Figure 12: *The 4-party protocol for Batch Attested OT.*