Fast and Secure Three-party Computation: The Garbled Circuit Approach

Payman Mohassel* Mike Rosulek† Ye Zhang‡

September 23, 2015

Abstract

Many deployments of secure multi-party computation (MPC) in practice have used information-theoretic three-party protocols that tolerate a *single, semi-honest* corrupt party, since these protocols enjoy very high efficiency.

We propose a new approach for secure three-party computation (3PC) that improves security while maintaining practical efficiency that is competitive with traditional information-theoretic protocols. Our protocol is based on garbled circuits and provides security against a single, *malicious* corrupt party. Unlike information-theoretic 3PC protocols, ours uses a constant number of rounds. Our protocol only uses inexpensive symmetric-key cryptography: hash functions, block ciphers, pseudorandom generators (in particular, no oblivious transfers) and has performance that is comparable to that of Yao's (semi-honest) 2PC protocol.

We demonstrate the practicality of our protocol with an implementation based on the JustGarble framework of Bellare et al. (S&P 2013). The implementation incorporates various optimizations including the most recent techniques for efficient circuit garbling. We perform experiments on several benchmarking circuits, in different setups. Our experiments confirm that, despite providing a more demanding security guarantee, our protocol has performance comparable to existing information-theoretic 3PC.

1 Introduction

Secure multi-party computation (MPC) allows a set of parties to compute a function of their joint inputs without revealing any information beyond the output of the function they compute. MPC has found numerous applications not only enabling various privacy-preserving tasks on sensitive data, but also removing a single point of attack by allowing for distribution of secrets and trust while maintaining the original functionality. Since the seminal work of [Yao86, GMW87] showing its feasibility in the two-party and multi-party settings, MPC has been the subject of extensive research, focusing on bettering security and efficiency.

The case of three-party computation (3PC) where the adversary corrupts at most one party (honest majority) is an important special case that has received particular attention. It has been the subject of active research, implementation and optimization in frameworks such as VIFF [Gei07], Sharemind [BLW08], ShareMonad [LDDAM12, LADM14] and MEVAL [CMF+14]. These protocols have been used in a wide range of applications such as statistical data analysis [BTW12], and email filtering [LADM14]. They have also been deployed in practice for online beet auctions [BCD+09] and for financial data analysis [BTW12]. A main reason for popularity of 3PC with-one-corruption is the simplicity and efficiency of the resulting protocols. In particular, protocols designed in this setting can be significantly more efficient than their two-party counterparts (or dishonest majority protocols in general) since they are commonly based on secret-sharing schemes and hence only require arithmetic operations that are considered faster than cryptographic ones.

However, the secret-sharing-based solutions have several drawbacks. In particular, the *round complexity* of these protocols is proportional to the circuit-depth of the computation being performed, which can be high in practice. Also, to the best of our knowledge, With the exception of [IKHC14], existing implementations

*Yahoo Labs, pmohassel@yahoo-inc.com.

†Oregon State University, rosulekm@eecs.oregonstate.edu, supported by NSF award CCF-1149647.

‡Google Inc., zhye@google.com, most of the work done while an Intern at Yahoo Labs! and a PhD student at Penn State.

are only secure against *semi-honest* adversaries. Traditionally, one may be willing to settle for semi-honest security given that security against *active cheating* (malicious adversaries) has a reputation of requiring significant overhead. Our work shows that this impression need not be true, and that malicious security can in fact be obtained with little to no overhead over semi-honest security in the 3-party setting.

1.1 Our Contributions

We design a new protocol for 3PC with one corruption based on Garbled Circuits (GC) [Yao82, LP09, BHR12b]. Our protocol is constant-round and secure against a malicious adversary that corrupts one party. Unlike the standard approach of applying cut-and-choose techniques for compiling GC-based protocols into malicious 2PC, we show that in the setting of 3PC with one corruption one can avoid the cut-and-choose paradigm and achieve malicious security at a cost similar to semi-honest two-party constructions. We also avoid the use of public-key operations such as Oblivious Transfer.

We prove our protocol secure in the Universal Composability (UC) model, but avoid the use of expensive UC-secure primitives due to the honest-majority setting. The only cryptographic tools we require are a secure garbling scheme and a non-interactive (standalone-secure) commitment scheme, both of which can be instantiated using symmetric key primitives.

Our protocol does not achieve fairness, and we leave it open to design a protocol with similar level of efficiency that also achieves fairness (a feasible goal in the honest majority setting).

We implement our protocol by enhancing the implementation of JustGarble [BHKR13] in various ways and incorporating the state-of-the-art "half-gates" garbling scheme of [ZRE15]. We further reduce communication (which our experiments show to be the bottleneck), by a factor of two using a hashing technique described in Section 3.4. We run experiments evaluating benchmarking circuits such as AES/MD5/SHA1/SHA256, and with different communication techniques turned on/off. Our experimental results confirm that our construction is competitive with prior work in the same setting while achieving the stronger malicious security. They also confirm that communication remains the major bottleneck in GC-based constructions even in the three-party setting. We also explore a motivating application we call distributed credential encryption service, that naturally lends itself to an offline pre-processing stage. Our experiments show that the online phase can be very fast.

1.2 Related Work

The most relevant line of work to ours are MPC constructions with an honest majority. Starting with seminal work of [BOGW88, CCD88] a large body of work has studied round and communication complexity of such protocols. A main building block for achieving security against a malicious adversarie in these constructions is verifiable secret sharing (VSS) [BOGW88, RBO89]. While these constructions are quite efficient and avoid cryptographic operations, their practical efficiency and the constant factors are not fully examined. The one implementation of 3PC with malicious security we know of is [IKHC14]. Their work proposes an approach for compiling a semi-honest 3PC into a malicious one with a small overhead (we discuss the overhead in more detail in the experiment section). The other existing implementations we know of are based on customized 3PC frameworks provided in [BLW08, LDDAM12, ZSB13, CMF+14] which only provide security against semi-honest adversaries. We provide a more detailed comparison with this line of work in the experiment section. Another customized 3PC based on garbled circuits, using the cut-and-choose paradigm and distributed garbling, was introduced in [CKMZ14]. Their protocol considers the stronger two-corruption setting and is naturally more expensive.

The more general multiparty and constant-round variant of Yao's garbled circuit was also studied in both the semi-honest setting [BMR90], and the malicious setting [DI05, IKP10]. An implementation exists [BDNP08] for the semi-honest case. These protocols are conceptually based on garbled circuits but require a particular instantiation of garbled circuits that expands the wire-labels through secret-sharing. We leave it as interesting open work to investigate whether recent optimizations to standard garbled circuits can be similarly applied to these protocols, and to compare the practical efficiency of malicious-secure variants.

In concurrent and independent work, Ishai et al. [IKKPC15] describe efficient, constant-round secure computation protocols for 3 and 4 parties, tolerating 1 malicious corruption. Both their protocol and ours use as a starting point the protocol of Feige et al. [FKN94] in the *private simultaneous messages* (PSM)

setting, which is in turn based on Yao's garbled circuit construction. The two protocols ([IKKPC15] and ours) use incomparable techniques to strengthen the PSM protocol against one malicious participant, and achieve a different mix of properties. In the 3-party setting, [IKKPC15] achieve a 2-round protocol whose cost is essentially that of 3 garbled circuits, whereas our protocol requires 3 rounds (in its random oracle instantiation) and has cost of 1 garbled circuit. In the 4-party setting, [IKKPC15] achieve guaranteed output delivery as well.

Fast implementation of malicious 2PC and MPC in the dishonest majority setting include cut-and-choose solutions based on garbled circuits [LPS08, KS12, FN13, AMPR14], OT-based solutions [NNOB12, LOS14], and implementations in the pre-processing models [DKL$^+$12, DKL$^+$13]. These protocols resists a larger fraction of coalition of corrupted parties than ours, but are significantly less efficient.

1.3 Organization

The building blocks used in our protocols such as a garbling scheme, commitment schemes and coin-tossing are all defined and described in Section 2. Our main construction and its security proof are described in Section 3. Our implementation, experimental results, and comparison with other implementations can be found in Section 4. We discuss the distributed encryption service application in Section 5.

2 Preliminaries

2.1 Secure MPC: UC Framework

We define security of multi-party computation using the framework of Universal Composition (UC) [Can01]. We give a very brief overview here, and refer the reader to [Can01] for all of the details.

An execution in the UC framework involves a collection of (non-uniform) interactive Turing machines. In this work we consider an adversary that can statically (i.e., at the beginning of the interaction) corrupt at most one party. We consider security against *active adversaries*, meaning that a corrupt party is under complete control of the adversary and may deviate arbitrarily from the prescribed protocol. The *parties* exchange messages according to a protocol. Protocol inputs of uncorrupted parties are chosen by an *environment* machine. Uncorrupted parties also report their protocol outputs to the environment. At the end of the interaction, the environment outputs a single bit. The adversary can also interact arbitrarily with the environment — without loss of generality the adversary is a *dummy* adversary which simply forwards all received protocol messages to the environment and acts in the protocol as instructed by the environment.

Security is defined by comparing a real and ideal interaction. Let $\text{REAL}[\mathcal{Z}, \mathcal{A}, \pi, k]$ denote the final (single-bit) output of the environment $\mathcal{Z}$ when interacting with adversary $\mathcal{A}$ and honest parties who execute protocol π on security parameter k. This interaction is referred to as the **real** interaction involving protocol π.

In the **ideal** interaction, parties run a "dummy protocol" in which they simply forward the inputs they receive to an uncorruptable *functionality* machine and forward the functionality's response to the environment. Hence, the trusted functionality performs the entire computation on behalf of the parties. Let $\text{IDEAL}[\mathcal{Z}, \mathcal{S}, \mathcal{F}, k]$ denote the output of the environment $\mathcal{Z}$ when interacting with adversary $\mathcal{S}$ and honest parties who run the dummy protocol in presence of functionality $\mathcal{F}$ on security parameter k.

We say that a protocol π **securely realizes** a functionality $\mathcal{F}$ if for every adversary $\mathcal{A}$ attacking the real interaction (without loss of generality, we can take $\mathcal{A}$ to be the dummy adversary), there exists an adversary $\mathcal{S}$ (called a simulator) attacking the ideal interaction, such that for all environments $\mathcal{Z}$, the following quantity is negligible (in k):

$$\left| \Pr\left[\text{REAL}[\mathcal{Z}, \mathcal{A}, \pi, k] = 1\right] - \Pr\left[\text{IDEAL}[\mathcal{Z}, \mathcal{S}, \mathcal{F}, k] = 1\right] \right|.$$

Intuitively, the simulator must achieve the same effect (on the environment) in the ideal interaction that the adversary achieves in the real interaction. Note that the environment's view includes (without loss of generality) all of the messages that honest parties sent to the adversary as well as the outputs of the honest parties. Thus the definition captures both the information that an adversary can learn about honest parties' inputs as well as the effect that an adversary can have on the honest parties' outputs. In a secure protocol these capabilities cannot exceed what is possible in the ideal interaction.

Target functionality The code of the functionality $\mathcal{F}$ implicitly defines all of the properties that comprise the security required of a protocol π. In Figure 1 we define the ideal functionality $\mathcal{F}_f$ for secure 3-party computation of a function f.

Input collection. On message (INPUT, x_i) from a party P_i ($i \in \{1, 2, 3\}$), do the following: if a previous $(\text{INPUT}, \cdot)$ message was received from P_i, then ignore. Otherwise record x_i internally and send $(\text{INPUTFROM}, P_i)$ to the adversary.

Computation. After all 3 parties have given input, compute $y = f(x_1, x_2, x_3)$. If any party is corrupt, then send (OUTPUT, y) to the adversary; otherwise send (OUTPUT, y) to all parties.

Unfair output. On message DELIVER from the adversary, if an identical message was received before, then ignore. Otherwise send (OUTPUT, y) to all honest parties.

Figure 1: Ideal functionality $\mathcal{F}_f$ for secure 3-party computation of a function f.

In particular, $\mathcal{F}_f$ provides "security with abort" (i.e., unfair output) in which the adversary is allowed to learn its output from the functionality before deciding whether the uncorrupted parties should also receive their output.

Contrasting active and semi-honest security When one party (say, P_1) is actively corrupt, it may send unexpected messages to an honest party (say, P_3). This may have the effect that P_3's view leaks extra information about the other honest party P_2. We note that this situation is indeed possible in our protocol (for example, P_1 can send to P_3 the seed used to generate the garbled circuit, which allows everyone's inputs to be computable from P_3's view). However, we emphasize that this is **not** a violation of security in the 1-out-of-3 corruption case.

A protocol with malicious security must let the honest parties handle "unexpected" messages appropriately. The security definition only considers what effect such "unexpected" messages have on the *final output* of an honest party, but not the effect they have on the *view* of an honest party. More precisely, if P_3 is honest, then only his final protocol output is given to the environment, while his entire view is not. A participant who hands its entire view over to the environment must be at least semi-honest corrupt, but in our case only one party (P_1 in our example) is assumed to be corrupt at all. We leave as an open problem to achieve security in the presence of one active and one semi-honest party (simultaneously), with comparable efficiency to our protocol.

We also emphasize that our primary point of comparison is against existing 3PC protocols that tolerate 1 *semi-honest* corruption. In these protocols, a single, *actively corrupt* participant can violate privacy of others' inputs and integrity of others' final outputs, often completely undetectably. So while 1-out-of-3 active security has some limitations, it is a significantly stronger guarantee than 1-out-of-3 semi-honest security.

2.2 Garbling Scheme

We employ the abstraction of garbling schemes [BHR12b] introduced by Bellare *et al.* Below is a summary of garbling scheme syntax and security:

A **garbling scheme** is a four-tuple of algorithms $\mathcal{G} = (\mathsf{Gb}, \mathsf{En}, \mathsf{De}, \mathsf{Ev})$ where: Gb is a randomized garbling algorithm that transforms function f into a triplet (F, e, d), where F is the garbled circuit, e is encoding information, and d is decoding information. En is an encoding algorithm that maps input x into garbled input via $X = \mathsf{En}(e, x)$. De is a decoding algorithm that maps the garbled output Y into plaintext output $y = \mathsf{De}(d, Y)$. Ev is the algorithm that on garbled input X and garbled circuit F, produces garbled output $Y = \mathsf{Ev}(F, X)$.

The **correctness** property of a garbling scheme is that, for all (F, e, d) in the support of $\mathsf{Gb}(1^k, f)$ and all inputs x, we have $\mathsf{De}(d, \mathsf{Ev}(F, \mathsf{En}(e, x))) = f(x)$, where k denotes the security parameter.

We require a **projective** scheme, meaning that e is an $n \times 2$ matrix of wire labels, and the encoding algorithm En has the structure $\mathsf{En}(e, x) = (e[1, x_1], e[2, x_2], \ldots, e[n, x_n])$.

A scheme satisfies **privacy** if there exists a simulator $\mathcal{S}$ for which following two processes induce indistinguishable output distributions:

$\underline{M_1(1^k, f, x)}$:
$(F, e, d) \leftarrow \mathsf{Gb}(1^k, f)$
$X \leftarrow \mathsf{En}(e, x)$
return (F, X, d)

$\underline{M_2(1^k, f, x)}$:
$(F, X, d) \leftarrow \mathcal{S}(1^k, f, f(x))$
return (F, X, d)

In other words, the tuple (F, X, d) contains no information beyond $f(x)$.

A scheme satisfies **authenticity** if the following property holds. Given (F, X) as in the output of M_1 above, it is with only negligible probability that a poly-time adversary can generate $\tilde{Y} \neq \mathsf{Ev}(F, X)$ such that $\mathsf{De}(d, \tilde{Y}) \neq \perp$. In other words, without d, it is not possible to give a *valid* garbled output other than Y obtained from Ev.

Our protocol requires an additional "soft decoding" function $\widetilde{\mathsf{De}}$ that can decode garbled outputs without the decoding information d. The soft-decoding function must satisfy $\widetilde{\mathsf{De}}(\mathsf{Ev}(F, \mathsf{En}(e, x))) = f(x)$ for all (F, e, d) in the support of $\mathsf{Gb}(1^k, f)$. Note that both $\widetilde{\mathsf{De}}$ and De can decode garbled outputs, but the *authenticity* security property holds only with respect to De — that is, $\widetilde{\mathsf{De}}$ can in principle be "fooled" when given maliciously crafted garbled outputs. However, if the garbled circuit and garbled input are honestly generated, the output of $\widetilde{\mathsf{De}}$ will be correct. In our protocol, we will let the evaluator of the garbled circuit obtain input using $\widetilde{\mathsf{De}}$, while the generators of the garbled circuit will use De (to protect against a corrupt party who tries to falsify the garbled outputs). In practice, we can achieve soft decoding in typical garbling schemes by simply appending the truth value to each output wire label. The true decoding function De will still verify the entire wire labels to guarantee authenticity.

2.3 Non-Interactive Commitment

We require a non-interactive commitment scheme (in the common random string model). Let crs denote the common random string and let $(\mathsf{Com}_{crs}, \mathsf{Chk}_{crs})$ be a non-interactive commitment scheme for n-bit messages. The Com_{crs} algorithm takes an n-bit message x and random coins r as input, and outputs a commitment C and the corresponding opening σ. We write $\mathsf{Com}_{crs}(x)$ as shorthand for the distribution $\mathsf{Com}_{crs}(x; r)$ induced by uniform choice of r. We require the following properties of the scheme:

- **Correctness:** for all crs, if $(C, \sigma) \leftarrow \mathsf{Com}_{crs}(x)$ then $\mathsf{Chk}_{crs}(C, \sigma) = x$.
- **Binding:** For all poly-time adversaries A, it is with negligible probability (over uniform choice of crs) that $A(crs)$ outputs (C, σ, σ') such that $\mathsf{Chk}_{crs}(C, \sigma) \neq \mathsf{Chk}_{crs}(C, \sigma')$ and $\perp \notin \{\mathsf{Chk}_{crs}(C, \sigma), \mathsf{Chk}_{crs}(C, \sigma')\}$.
- **Hiding:** For all poly-time adversaries A, all crs, and all $x, x' \in \{0,1\}^n$, the following difference is negligible:
$$\left| \Pr_{(C,\sigma) \leftarrow \mathsf{Com}_{crs}(x)}[A(C) = 1] - \Pr_{(C,\sigma) \leftarrow \mathsf{Com}_{crs}(x')}[A(C) = 1] \right|.$$
Since we quantify over all crs and A together (not just a random crs), it is not necessary to give crs to A in this definition. The definition also implies that the crs can be used for many commitments.

Instantations In the random oracle model, commitment is simple via $(C, \sigma) = (H(x\|r), x\|r) = \mathsf{Com}_{crs}(x; r)$. The crs can in fact be empty.

In the standard model, we can use a multi-bit variant of Naor's commitment [Nao91]. For n-bit strings, we need a $crs \in \{0,1\}^{4n}$. Let $G : \{0,1\}^n \to \{0,1\}^{4n}$ be a pseudorandom generator, and let $pad : \{0,1\}^n \to \{0,1\}^{4n}$ be the function that prepends $3n$ zeroes to its argument. Then the commitment scheme is:

- $\mathsf{Com}_{crs}(x; r)$: set $C = G(r) + crs \cdot pad(x)$, with arithmetic in $GF(2^{4n})$; set $\sigma = (r, x)$.
- $\mathsf{Chk}_{crs}(C, \sigma = (r, x))$: return x if $C = G(r) + crs \cdot pad(x)$; otherwise return $\perp$.

Security of this construction closely follows the original proof of Naor's construction, but is provided for completeness in Appendix A.

3 Our Protocol

In this section we present a new and efficient 3PC protocol that is secure against 1 malicious corruption. Its complexity is essentially the same as that of (semi-honest-secure) two-party Yao's protocol.

3.1 High Level Overview

Our starting point is Yao's protocol based on garbled circuits. In that protocol, one party generates a garbled circuit and the other evaluates it. The two parties use oblivious transfer to allow the evaluator to receive the garbled encoding of his input.

Yao's protocol is secure against a malicious evaluator, but secure against only a semi-honest garbler. Our 3-party protocol can be thought of as splitting the role of the garbler between two parties (while keeping the evaluator a single party). When only one party is corrupt, then at least one of the garbling parties is honest, and we can leverage that fact to protect against one malicious garbler.

In more detail, we let P_1 and P_2 agree on a random tape r and run Yao's protocol as garbler with P_3 as evaluator, with both instances using random tape r. By using the same random tape in Yao's protocol, P_1 and P_2 are expected to send identical messages to P_3 in every step, and P_3 can abort if this is not the case. Then, security against a malicious P_3 follows from the security of Yao's protocol — it is really P_3 attacking a single instance of 2-party Yao's protocol in this case. Security against a malicious P_1 or P_2 follows from the fact that Yao's protocol is secure against a garbler who runs the protocol honestly (even on adversarially chosen random tape). In our protocol, the only options for malicious P_1 or P_2 are to run Yao's 2-party protocol honestly or else cause P_3 to abort (by disagreeing with the honest garbler).

Obtaining Garbled Inputs This overview captures the main intuition, but it does not address the issue of garbled inputs. Indeed, P_1 and P_2 have their own private inputs and so must at some point send different messages in order to affect the final output. To address this, we have P_1 and P_2 commit to all of the input wire labels for the circuit. For each wire, the two commitments are randomly permuted. P_1 and P_2 will generate these commitments using the same randomness, so their correctness is guaranteed using the same reasoning as above. Then P_1 and P_2 can simply open the appropriate commitments corresponding to their input bits (note that the position of the commitments does not leak their inputs).

P_3's garbled input is handled using an oblivious transfer in Yao's protocol, but we are able to avoid OT altogether in our 3-party setting. We garble the circuit $f'(x_1, x_2, x_3, x_4) = f(x_1, x_2, x_3 \oplus x_4)$, so that x_3, x_4 are an additive secret sharing of P_3's logical input. We have P_3 send x_3 to P_1 and x_4 to P_2, so that P_1/P_2 can open the corresponding commitments to garbled inputs. Since at most one of $\{P_1, P_2\}$ is corrupt, an adversary can learn nothing about P_3's logical input. To ensure that P_1/P_2 open the correct commitments in this step (i.e., they do not flip bits in P_3's input), we have them both reveal the random ordering of the relevant commitments (which can be checked against each other by P_3).

Other Optimizations To make the commitment non-interactive, we can either assume a random oracle commitment scheme, or else a common random string (CRS). In the latter case, P_3 can choose the CRS himself (we use a commitment scheme that is hiding for *all* CRS values, not just for a uniformly chosen one).

Both P_1 and P_2 use a common random tape r to run Yao's protocol as garbler. We can actually have P_1 choose r himself; surprisingly there is no problem in a corrupt P_1 choosing r arbitrarily. In the case that P_1 is corrupt, the security proof needs to apply the binding security of the commitment scheme and correctness property of garbled circuits. Binding holds with respect to malicious senders (in particular, senders who choose r arbitrarily and use $\mathsf{PRF}(r, \cdot)$ to derive randomness to run the $\mathsf{Com}_{crs}(\cdot)$ algorithm). Correctness of garbled circuits holds with probability 1, i.e., for all (F, e, d) in the support of $\mathsf{Gb}(1^k, f')$. The fact that P_2 must be honest if P_1 is corrupt guarantees that the (F, e, d) used in the protocol is indeed in the support of Gb, so we can apply the correctness of the garbling scheme.

3.2 Detailed Protocol Description

We denote the three parties in the protocol by P_1, P_2 and P_3, and their respective inputs by x_1, x_2, and x_3^*. Their goal is to securely compute the function $y = f(x_1, x_2, x_3^*)$. For convenience we define the related function

$$f'(x_1, x_2, x_3, x_4) = f(x_1, x_2, x_3 \oplus x_4).$$

For simplicity we assume that $|x_i| = |y| = m$. The protocol is as follows. All communication between parties is on private point-to-point channels.

In what follows we assume that all parties learn the same output y, but it is easy to modify the protocol such that each party learns a different output (i.e., a 3-output function). In particular, P_3 can return to each of P_1 and P_2 the garbled values for the portion of the output wires corresponding to their own output, while the "soft-decoding" procedure is constrained to work only for P_3's output wires (concretely, the cleartext truth values are appended only to the output wires for P_3's outputs).[1]

1. P_3 samples a random crs for the commitment scheme and randomly secret-shares his input x_3^* as $x_3^* = x_3 \oplus x_4$. He sends x_3 to P_1 and x_4 to P_2 and broadcasts crs to both parties.

2. P_1 chooses random PRF seed $r \leftarrow \{0,1\}^k$ and sends it to P_2 (see the discussion in the previous section).

3. Both P_1 and P_2 do the following, independently, and obtaining *all* required randomness via $PRF(r, \cdot)$:

   (a) Garble the circuit f' via $\mathsf{Gb}(1^\lambda, f') \to (F, e, d)$.

   (b) Commit to all $4m$ input wire labels in the following way. Sample $b \leftarrow \{0,1\}^{4m}$. Then for all $j \in [4m]$ and $a \in \{0,1\}$, generate the following commitment:

   $$(C_j^a, \sigma_j^a) \leftarrow \mathsf{Com}_{crs}(e[j, b[j] \oplus a])$$

   Both P_1 and P_2 send the following values to P_3:[2]

   $$(b[2m+1 \cdots 4m], F, \{C_j^a\}_{j,a}).$$

   P_3 will abort if P_1 and P_2 report different values for these items.

4. P_1 and P_2 additionally reveal garbled inputs to P_3 in the following way (now P_1 and P_2 are sending different messages). For $j \in [m]$:

   (a) P_1 sends decommitment $\sigma_j^{x_1[j] \oplus b[j]}$ to P_3.

   (b) P_2 sends decommitment $\sigma_{m+j}^{x_2[j] \oplus b[m+j]}$ to P_3.

   (c) P_1 sends decommitment $\sigma_{2m+j}^{x_3[j] \oplus b[2m+j]}$ to P_3.

   (d) P_2 sends decommitment $\sigma_{3m+j}^{x_4[j] \oplus b[3m+j]}$ to P_3.

5. P_3 assembles the garbled input as follows. For $j \in [4m]$, P_3 computes $X[j] = \mathsf{Chk}_{crs}(C_j^{o[j]}, \sigma_j^{o[j]})$, for the appropriate $o[j]$. If any call to Chk returns $\perp$, then P_3 aborts. Similarly, P_3 knows the values $b[2m+1 \cdots 4m]$, and aborts if P_1 or P_2 did not open the "expected" commitments $C_{2m+j}^{x_3[j] \oplus b[2m+j]}$ and $C_{3m+j}^{x_4[j] \oplus b[3m+j]}$ corresponding to the garbled encodings of x_3 and x_4.

   P_3 runs $Y \leftarrow \mathsf{Ev}(F, X)$ and broadcasts Y to all parties.

6. At this point, P_1 and P_2 can compute $y = \mathsf{De}(d, Y)$. If $y \neq \perp$, then they output y, otherwise they abort. Also, P_3 can compute and output $y = \widetilde{\mathsf{De}}(Y)$, where $\widetilde{\mathsf{De}}$ is the "soft decoding" function described in Section 2.2.

[1] The resulting protocol will achieve a slightly weaker notion of security, since a corrupt P_3 can choose to make only one of $\{P_1, P_2\}$ abort. This notion is known as *security with selective abort*. [GL05]

[2] Since $b[2m+1 \cdots 4m]$ are given to P_3, we can actually take them to be all zeroes and eliminate them altogether. Still, to keep the treatment of all garbled inputs consistent in the notation, we continue with $b \in \{0,1\}^{4m}$.

3.3 Security Proof

Theorem 1. *The protocol in Section 3.2 securely realizes the $\mathcal{F}_f$ functionality against adversaries who actively corrupt at most one of the parties.*

Proof. First consider the case where P_1 is corrupted (the case of P_2 is essentially symmetric). We show that the real and ideal interactions are indistinguishable to all environments, in a sequence of hybrid interactions. The required simulator is built up implicitly in the hybrid sequence below. Recall that the environment's view consists of messages sent from honest parties to the adversary in the protocol, as well as the final outputs of the honest parties.

$\mathcal{H}_0$: This hybrid is identical to the real interaction, except that we repackage the various components of the interaction. A simulator plays the role of honest P_2 and P_3, receiving their inputs x_2 and x_3^* from the environment and running the protocol on their behalf.

$\mathcal{H}_1$: In the previous hybrid, the simulator runs the protocol on behalf of P_2 and hence knows the value b chosen in step (3b). Assuming P_3 does not abort in step (5), then P_1 must have succesfully opened commitments $C_j^{o[j]}$ for some string $o \in \{0,1\}^m$. At this point in $\mathcal{H}_1$ we have the simulator compute $x_1 = o \oplus b[1 \cdots m]$. The simulator also simulates an instance of the $\mathcal{F}_f$ functionality. It sends x_1, x_2, x_3^* to the simulated $\mathcal{F}_f$ (recall that at this point in the sequence of hybrids the simulator is receiving the other parties' inputs x_2 and x_3^*). As the simulator does nothing with anything computed by $\mathcal{F}_f$, there is no difference in the environment's view. Note that the simulator is still using x_2 and x_3^* at this point to run the protocol on behalf of P_2 and P_3.

$\mathcal{H}_2$: In the previous hybrid, the simulator runs the protocol on behalf of P_2 and hence knows what was committed in all of the C_j^a commitments.

We modify the simulator to abort if the simulated P_3 accepts the opening of any commitment (in step 5) to a value other than what was originally committed. The crs is chosen uniformly by P_3, so the commitment scheme's binding property guarantees that the probability of this abort occurring is negligible, so the hybrids are indistinguishable.[3]

Conditioned on this additional abort not happening, we can see that the garbled input X used by P_3 has been computed as:

$$X = \mathsf{En}(e, x_1 \| x_2 \| x_3 \| x_4),$$

where x_1 was the value extracted by the simulator in step (5) as explained above.

Further note that, as long as P_3 doesn't abort in step (3), we have that the values (F, e, d) are in the support of $\mathsf{Gb}(1^k, f')$. The garbling scheme's correctness property holds for all such (F, e, d), and it does not matter that the random coins used in Gb were influenced by P_1's selection of r in step (2).

$\mathcal{H}_3$: Same as above, except that in step (6), when the honest parties hand their output to the environment, the simulator instead hands the environment the value y computed by the simulated $\mathcal{F}_f$. By the correctness condition mentioned above, we have that if the simulator doesn't abort on behalf of P_3 in step (5) then $\mathsf{De}(d, \mathsf{Ev}(F, X)) = \widetilde{\mathsf{De}}(\mathsf{Ev}(F, X)) = f(x_1, x_2, x_3^*)$, where the first two expressions are what the simulated P_2 and P_3 would have output, and the final expression is what $\mathcal{F}_f$ computes. Hence hybrids $\mathcal{H}_2$ and $\mathcal{H}_3$ induce identical views on the environment.

$\mathcal{H}_4$: Same as above, except that instead of computing Y via $\mathsf{Ev}(F, X)$ in step (5), the simulator computes Y such that $\mathsf{De}(d, Y) = y$, where y is the output obtained from $\mathcal{F}_f$ (this is easy in all practical garbling schemes, when the simulator knows all the randomness used to generate the garbled circuit). By the correctness of the garbling scheme, this is an equivalent way to compute the same value, so the change has no effect on the environment's view.

[3] Note that we still need binding to hold against malicious senders. The commitments are not necessarily generated honestly; instead they are generated by having P_1 (maliciously) choose r which is expanded via $\mathsf{PRF}(r, \cdot)$ to provide randomness to call Com_{crs}.

$\mathcal{H}_5$: Note that in hybrid $\mathcal{H}_4$, the garbled circuit does not have to be evaluated during the simulation, hence the garbled input X is not used. But generating X was the only place x_4 (the secret share of x_3^*) was used. The other share x_3 of x_3^* is sent to P_1. In $\mathcal{H}_5$ we modify the simulator to send a random x_3 to P_1 in step (1). The change has no effect on the environment's view.

The final hybrid implicitly defines our protocol's simulator. It sends a random share x_3 to P_1 in step (1); it aborts in step (5) if P_1 has violated the binding property of any commitment; otherwise it extracts $x_1 = o \oplus b[1 \cdots m]$ and sends it to the ideal functionality $\mathcal{F}_f$. It receives y, and in step (5) sends Y to P_1 such that $\mathsf{De}(d, Y) = y$. The effect on the environment in this ideal interaction is indistinguishable from the real interaction, by the arguments in the above sequence of hybrids.

Next, we consider a corrupt P_3:

$\mathcal{H}_0$: As before, we consider a simulator playing the role of honest P_1 and P_2 running on their inputs. The environment receives the final outputs of the simulated P_1 and P_2.

$\mathcal{H}_1$: Same as above, except for the following change. The simulator will run an instance of $\mathcal{F}_f$. In step (1) of the protocol, the simulator will receive x_3, x_4 from P_3, set $x_3^* = x_3 \oplus x_4$, then send x_1, x_2, and x_3^* to the instance of $\mathcal{F}_f$. This is merely an internal change, since in this hybrid the simulator does not yet use the outputs of $\mathcal{F}_f$ in any way. Hence, the two hybrids induce identical views for the environment.

$\mathcal{H}_2$: Same as above, except that the simulated P_1 and P_2 use uniform randomness rather than pseudorandomness in step (2). The hybrids are indistinguishable by the security of the PRF and the fact that the PRF seed r is chosen uniformly by P_1 and P_2 in step (2).

$\mathcal{H}_3$: Same as above, except for the following change. In step (3), when the simulator is generating the C_j^a commitments, it knows in advance which ones will be opened. These are the commitment $C_j^{o[j]}$ where $o = b \oplus x_1 \| \cdots \| x_4$.

We modify the simulator to first choose random $o \leftarrow \{0,1\}^{4m}$ which index the commitments that will be opened, and then solve for $b = o \oplus x_1 \| \cdots \| x_4$ in step (3b). Note that the simulator indeed has all of the x_i values at this time. Then the simulator commits to dummy values for the commitments which will not be opened. The hybrids are indistinguishable by the hiding property of the commitment scheme (which holds with respect to *all* values of crs). Note that the simulation now does not use all of the garbled input encoding information e; rather, it only uses $X = \mathsf{En}(e, x_1 \| \cdots x_4)$.

$\mathcal{H}_4$: In step (6) in the previous hybrid, the simulated P_1 and P_2 will abort if $\mathsf{De}(d, \tilde{Y}) = \perp$, where $\tilde{Y}$ is the message sent by P_3 in step (5). We modify the simulator so that it aborts if $\tilde{Y} \neq \mathsf{Ev}(F, X)$, which is what P_3 is supposed to send. Note that the simulator indeed knows F and all of X at this point.

By the authenticity property of the garbling scheme, it is only with negligible probability that P_3 (who is not given decoding information d) would produce $\tilde{Y} \neq \mathsf{Ev}(F, X)$ such that $\mathsf{De}(d, \tilde{Y}) \neq \perp$. Hence, the two hybrids are indistinguishable.

$\mathcal{H}_5$: Conditioned on the simulator not aborting in step (6), the correctness of the garbling scheme guarantees that simulated P_1 and P_2 will output $y = f(x_1, x_2, x_3^*)$. Hence, instead of handing the environment the outputs of these simulated P_1/P_2, we have the simulator instruct $\mathcal{F}_f$ to release output to honest parties if the simulator hasn't aborted in step (6), and give the outputs from $\mathcal{F}_f$ directly to the environment. Again, this has no effect on the environment's view.

$\mathcal{H}_6$: Same as above, except for the following change. Note that throughout the simulation in $\mathcal{H}_5$, the simulator uses F, d, but only $X = \mathsf{En}(e, x_1 \| \cdots \| x_4)$ due to the previous hybrids. In particular, it does not use the other parts of e. We modify the simulator to generate (F, X, d) using the simulator of the garbling scheme, rather than the standard Gb, En. The simulator requires y which the simulator knows already in step (1). The hybrids are indistinguishable by the security of the garbling scheme.

The simulator implicit in hybrid $\mathcal{H}_6$ defines our final simulator. It extracts $x_3^* = x_3 \oplus x_4$ in step (1) and sends it to $\mathcal{F}_f$, receiving output y in return. It then generates a simulated garbled circuit/input (F, X) using y. In

step (3) it chooses random string o and commits to the entries of X as $C_j^{o[j]}$, while committing to dummy values in the other commitments. In step (4) it opens the commitments indexed by o. After receiving $\tilde{Y}$ from P_3 in step (5), it checks whether $\tilde{Y} = \mathsf{Ev}(F, X)$; if so, then it instructs $\mathcal{F}_f$ to deliver output to the honest parties. □

3.4 Reducing Communication

We can reduce the total communication by essentially half, as follows: Instead of both P_1 and P_2 sending the very long (identical) message to P_3 in step 3, we can have only P_1 send this message while P_2 simply sends the hash of this message, under a collision-resistant hash function. P_3 can then simply check the hash received from P_2 against the message received from P_1. While this reduces total communication size, it does not reduce total communication *latency* of the protocol in the most common scenario where P_1 and P_2 communicate with P_3, simultaneously.

To improve on this, we have P_1 and P_2 treat the message they send to P_3 as a string S, divided into equal halves $S = S_1||S_2$. We then have P_1 send S_1 and $H(S_2)$ and P_2 send $H(S_1)$ and S_2 to P_3. This still enables P_3 to retrieve S and also check that P_1 and P_2 agree on a common S. This variant not only reduces total communication by half, but also the communication latency in the scenario that P_1 and P_2 run at the same time.

4 Implementation and Experimental Validation

4.1 Implementation

Our implementation is written in C++11 with STL support. For an efficient implementation of a circuit garbling scheme, we used as a starting point the JustGarble library [BHKR13], an open-source library licensed under GNU GPL v3 license. We also used the MsgPack 0.5.8 library to serialize/deserialize data and used the openssl lib (version 1.0.1e-fips) for our implementation of SHA-256. We implement the commitment scheme needed for our protocol using SHA-256 as a random oracle.

In our implementation, P_3 initializes itself by first reading the circuit description file from the disk. The description file is in JustGarble's SCD format. Then, P_3 listens on a port via a socket. When P_i $(i = 1, 2)$ connect to the port, P_3 creates a new thread for this connection. The rest of communication/interaction between P_3 and P_i will be within this thread.

Then, P_1 and P_2 connect with each other to negotiate a shared seed and use it to generate a garbled circuit. We modify JustGarble to support a shared seed for the randomness needed in the garbling. As a result, the garbled circuits generated by P_1 and P_2 are identical.[4]

Communication Reduction Techniques. To reduce communication/serialization costs, we add several optimizations to JustGarble. First, we enable the *free-XOR* support [KS08] in JustGarble and also modify its code to make NOT gates free (no communication and computation) since the original JustGarble implementation treats NOT gates like other non-XOR gates. Additionally, we incorporated support for the recent *half-gate* garbling technique of [ZRE15] which reduces sizes of garbled non-XOR gates to two ciphertexts (a 33% reduction compared to the best previous technique).

Instead of sending each garbled gate ciphertext individually over a socket, which would significantly slow down communication due to overheads, we serialize multiple gates into a larger buffer with a size below the max socket size, and send it to the server who deserializes the received data. To further reduce communication size, we have P_1 send the first half of the serialized garbled circuit as usual, but only send a hash of the second half, and have P_2 do the reverse. P_3 can put together the two unhashed halves to construct the whole garbled circuits, and uses the hashed halves to check equality of the two garbled circuits. This technique reduces communication by a factor of two but requires more work by all parties in form of hashing the garbled

[4]While doing so, we identified a small bug in JustGarbled. Specifically, in the garble() function, the encryption key *garblingContext.dkCipherContext.K.rd_key* is used without initialization which resulted in different garbled circuits even when using the same seed.

circuits. Note that hashing operation increases computation cost. In fact, hashing the garbled circuit using SHA256 is more expensive than garbling the circuit itself which uses AES-NI instructions.[5] Nevertheless the reduction in communication cost (which is the bottleneck) makes this a net win as our experiments show.

4.2 Experiments

The experiments were conducted on Amazon EC2 Cloud Computing instances. The instances are of type `t2.micro` with 1GB memory, and Intel Xeon E5-2670 2.5Ghz CPU with AES_NI and SSE4.2 instruction sets enabled. The instances are interconnected using Amazon's network. We also use the iperf v3.1b3 tool to measure the maximum achievable bandwidth of t2.micro instances. The bandwidth is 1Gbits/s. The operating system was Red Hat Enterprise Linux 7.0 (kernel 3.10.0-123.8.1.el7.x86_64). We run each experiment 20 times and report the average and standard deviation for our measurements.

We used 4 different circuits for benchmarking our implementation: AES-128 (with key expansion), SHA-1, MD-5 and SHA-256. Besides the circuit for AES-128 which was provided with JustGarble, the description file for the other circuits were obtained from `http://www.cs.bris.ac.uk/Research/CryptographySecurity/MPC/`. We converted these files into JustGarble's SCD format using the code provided in [AMPR14].

The AES-128 circuit takes a 128-bit key and a 128-bit message as input. In our experiments, P_1 and P_2 independently choose 128-bit keys K_1 and K_2 and set $K = K_1 \oplus K_2$ to be the encryption key; P_3 provides the 128-bit message as input. Hence, the total input length to the AES-128 circuit is 384 and the output length is 128. The input/output sizes for all other circuits and the number of AND/XOR/NOT gates in each is presented in Table 1. Note that none of the circuits we used contains OR gates and since XOR gates and NOT gates are free in our implementation, the ultimate cost driver is the number of AND gates.

Circuit	AND	NOT	XOR	Input size	Output size
AES-128	7200	0	37638	384	128
MD-5	29084	34627	14150	512	128
SHA-1	37300	45135	24166	512	160
SHA-256	90825	103258	42029	512	256

Table 1: Number of AND/OR/NOT gates , and input/output sizes for each circuit. Input/output sizes are in bits.

In our first experiment, the client and server communicate via sockets whose max size is set to 10KB. The results are shown in Table 2. The computation time for P_1/P_2 measures the time that P_1/P_2 need to generate the garbled circuit, compute the commitments, serialize the data and hash half of the data. The computation time for P_3 measures the time P_3 needs to deserialize the data, compare and verify the garbled circuits, and evaluate them. The network time measures the time spent by P_1, P_2 and P_3 to send/receive data over socket.

Circuit	Network	P_3 Comp.	P_1/P_2 Comp.
AES-128	13.30 ± 0.97	2.30 ± 0.46	1.60 ± 0.59
MD-5	29.05 ± 1.12	9.05 ± 0.38	5.85 ± 0.66
SHA-1	36.60 ± 2.63	11.50 ± 0.80	7.65 ± 0.89
SHA-256	78.46 ± 2.62	31.18 ± 0.57	21.09 ± 1.90

Table 2: Half-gates and hashing technique enabled; times in milliseconds.

Table 3 shows the size of parties' communication during the same experiment.

Next, to examine the effect of the half-gate technique on the performance, we disable it (i.e. only the single row-reduction is enabled) and run the same experiments. The results are shown in Tables 4 and 5. As expected, the half-gate technique has a major effect on reducing the total running time as well the bandwidth of the protocol. The size of communication increases by 50% which is expected since garbled tabled sizes increase from 2 two 3.

[5] AES-NI provides high performance in the case where our usage of AES does not involve re-keying. One might be tempted to use AES (perhaps modeled as an ideal cipher) to construct a collision-resistant hash function in a way that avoids frequent re-keying. Unfortunately, negative results of [BCS05] show such a construction to be impossible.

Circuit	P_3 (KB)	P_1/P_2 (KB)
AES-128	1.19	752.37
MD-5	2.05	1276.16
SHA-1	2.78	1732.75
SHA-256	6.11	3752.25

Table 3: Communication sizes for experiments of Table 2.

Circuit	Network	P_3 Comp.	P_1/P_2 Comp.
AES-128	17.80 ± 1.45	3.10 ± 0.30	2.20 ± 0.41
% diff	+34.57%	+34.78%	+37.50%
MD-5	41.19 ± 2.20	15.45 ± 0.80	12.93 ± 0.53
% diff	+65.66%	+70.71%	+121.02%
SHA-1	51.74 ± 3.04	19.85 ± 0.79	16.48 ± 1.13
% diff	+64.90%	+72.61%	+115.42%
SHA-256	116.10 ± 4.00	50.10 ± 1.22	44.40 ± 2.01
% diff	+67.96%	+60.68%	+110.53%

Table 4: Half-gates disabled; times in milliseconds. Percentages show difference from Table 2.

Circuit	P_3 (KB)	P_1/P_2 (KB)
AES-128	1.77	1104.74
% diff	+48.74%	+46.83%
MD-5	3.05	1885.66
% diff	+48.78%	+47.76%
SHA-1	4.16	2561.82
% diff	+49.64%	+47.85%
SHA-256	9.17	5618.85
% diff	+50.08%	+49.75%

Table 5: Communication sizes for experiments of Table 4. Percentages show difference from Table 2

Next, we turn the half-gate technique back on and turn-off the hash technique and re-run the experiments. Results are shown in Tables 6 and 7. Again, this reconfirms the significant effect of the hashing technique on both the total running time and the communication size. Note that the computation times reduces when we turn off the hash technique which is natural since we avoid the extra hashing by all parties. But given that serializing and sending/receiving messages is the bottleneck, the net effect of the hashing optimization is positive.

Circuit	Network	P_3 Comp.	P_1/P_2 Comp.
AES-128	20.89 ± 1.24	1.30 ± 0.46	0.88 ± 0.40
% diff	+27.39%	-43.48%	-45%
MD-5	45.15 ± 0.97	4.20 ± 0.40	3.35 ± 0.58
% diff	+12.55%	-53.59%	-42.74%
SHA-1	57.50 ± 1.68	5.50 ± 0.59	4.65 ± 0.70
% diff	+14.03%	-52.17%	-39.22%
SHA-256	125.93 ± 2.35	15.72 ± 0.75	12.90 ± 0.68
% diff	+10.30%	-49.58%	-38.83%

Table 6: Hash technique disable. times in milliseconds. Percentages show difference from Table 2

Circuit	P_3 (KB)	P_1/P_2 (KB)
AES-128	2.33	1447.58
% diff	+95.80%	+92.40%
MD-5	4.03	2485.63
% diff	+96.59%	+94.77%
SHA-1	5.50	3380.84
% diff	+97.84%	+95.11%
SHA-256	12.15	7437.82
% diff	+98.85%	+98.22%

Table 7: Communication size for experiments of Table 6. Percentages show difference from Table 2

4.3 Comparison

3PC with one corruption As mentioned earlier, the most relevant protocols to ours are those designed in the same setting of 3PC with one corruption, or honest majority in general. The MPC constructions based on verifiable secret sharing [BOGW88, RBO89] achieve the same security as our construction (malicious security), and are asymptotically very efficient, as they require $O(poly(n))$ bits of communication per multiplication gate where n is number of parties, and the polynomial is relatively small, and these protocols also avoid cryptographic operations. However, to the best of our knowledge, no implementation of these constructions exists, so it is hard to provide a concrete comparison. It is a very valuable direction to explore practical efficiency of these constructions even in the three-party setting, and compare their efficiency and scalability with our construction in various scenarios.

The 3PC constructions with experimental results reported include VIFF [Gei07], Sharemind [BLW08], PICCO [ZSB13], ShareMonad [LDDAM12, LADM14], and [IKHC14]. With the exception of [IKHC14], these protocols only provide security against semi-honest adversaries. In contrast, our protocol is secure against malicious adversaries in the same setting, but demonstrates efficiency that is comparable to these semi-honest counterparts.

Admittedly, an accurate/fair comparison is not possible, given that each implementation runs in a different environment, using different language/libraries and networking, memory, and CPU specifications. For example, our machines' memory/CPU specifications seems fairly modest compared to the prior work and we do not take advantage of any parallelism. But to demonstrate that efficiency of our construction (for boolean circuits) is comparable to other implemented constructions with only semi-honest security, a single execution of an AES block requires 232 ms in Sharemind [LTW13], 14.3 ms in ShareMonad [LDDAM12], and 18 ms in our implementation (See Table 2). The construction of [IKHC14] which achieves malicious security (similar to ours) did not include an implementation of any boolean circuits circuit but based on the provided experimental numbers, their construction requires a factor of k (the security parameter) more communication and a factor of 3 more computation compared to secret-sharing-based semi-honest 3PC implementations.

Protocols with Dishonest Majority The bulk of other existing implementations with malicious security are focused on the dishonest majority setting. For example, there is large body of work on making Yao's two-party protocol secure against malicious adversaries with several implementations available [LPS08, KS12, FN13, AMPR14]. When considering a single execution, these protocols are at least a multiplicative factor of security parameter (80) more expensive than semi-honest Yao's protocol. But as seen earlier, our protocol has complexity that is fairly comparable to a single run of semi-honest and hence outperforms single-execution malicious 2PC protocols by a wide margin.

Note that when running many instances of malicious 2PC for the same function, there are recent results [LR14, HKK+14] that noticeably reduce the amortized multiplicative factor. In general, our comparisons are only for a single execution of the constructions. It is interesting to compare efficiency of the existing implementations in batch execution of the same of different setting, and to introduce new techniques for better batch 3PC.

We also emphasize that the above comparison is only fair for Boolean circuits, as secret-sharing-based protocols are often superior to garbled-circuit based ones for arithmetic circuits given that they directly implement multiplication/addition gates.

Another line of work [DKL+12, DKL+13] consider multi-party computation with dishonest majority in the pre-processing model where the goal is to implement a very fast online phase after a relatively expensive pre-prceossing stage for generating authenticated triplets. Similar to malicious 2PC, the total cost (offline + online) of these protocols significantly higher than ours (several seconds), but a direct comparison is not warranted give the difference in security guarantees and their focus on a fast online phase. We discuss this line of work again when considering the credential encryption service application where online/offline separation is natural.

5 Application: Distributed Credential Encryption Service

An interesting application of our work and an initial motivation for it was the design of a *distributed credential encryption service*. The goal of this service is to keep credentials such as hashed passwords encrypted at rest, in order to prevent offline dictionary attacks when user databases are compromised. At the same time, the service should be easily retrofitted into existing authentication systems, in which an authentication server computes or receives a hashed password in the clear. Hence, the goal is not to use crypto to protect the computation of a hashed password, but to simply protect the static database of hashed passwords.

A (non-distributed) credential encryption service receives a hashed password from the authentication server as input, which it encrypts to return a ciphertext that is stored in a database. To authenticate a user, the authentication server sends to the encryption service the hash of the purported password along with the correct ciphertext obtained from the database. The encryption service authenticates the user by decrypting the ciphertext and checking whether the result matches the hashed password that is given. Overall, the encryption service stores a secret encryption key and provides encryption/decryption functionality. The authentication server's database remains encrypted; authenticating a user requires interaction with the encryption service and hence cannot be done offline in a dictionary attack in the event that the authentication database is compromised.

In order to distribute trust and avoid a single point of attack we replace the encryption service by three servers running our 3PC protocol. P_1 and P_2 each hold a 128-bit random strings K_1 and K_2, while P_3 receives $h(passwd)$ from a log-in service to encrypt using the key $K = K_1 \oplus K_2$, i.e. to compute $AES_K(h(passwd))$. Our 3PC construction guarantees that even if the encrypted database is compromised, *and* one of the servers is compromised and even controlled by the adversary, the encryption key is not compromised and user credentials are not susceptible to offline dictionary attacks.

Note that unlike the general 3PC scenario, in the credential encryption service we can consider further optimizations. For example, since the computation involves the same circuit (e.g., AES) each time, we can in an offline phase (i.e. in the background or through a different channel without a latency restriction) have P_1 and P_2 garble many circuits separately using randomness they agree on and transmit them to P_3. Furthermore, given that P_1, P_2's inputs remain the same across multiple executions, they can also incorporate their garbled inputs to those circuits. A subtle issue is that for this approach to work, we have to assume that the garbling scheme is adaptively secure [BHR12a], a stronger assumption than we need for our main protocol.

The online phase then consists of P_3 obtaining the labels for his input $h(passwd)$, taking a pair of garbled circuits/commitments that are the same from a pile computed earlier and evaluating it. Our experiments show that the online running time of the protocol is on average $\mathbf{1.34 \pm 0.47}$ ms, making the protocol feasible for real-world deployment.

The SPDZ protocols and followup works [DKL+12, DKL+13], also focus on optimizing the online case at the cost of the offline phase, and achieve fast online performance in the order of milliseconds for AES circuit (though higher than ours), but have a much slower offline phase compared to us. Given than the offline phase is not reusable in either ours or SPDZ construction and has to repeated on a regular basis, our protocol seems more suitable for this application given the faster offline phase. On the other hand, the SPDZ protocols achieve stronger security by handling two corruptions.

Acknowledgements

We wish to thank Yan Huang and several anonymous reviewers for many helpful suggestions to improve the writeup. We are also appreciative of Yuval Ishai and Ranjit Kumaresan for bringing some related work to our attention.

References

[AMPR14] Arash Afshar, Payman Mohassel, Benny Pinkas, and Ben Riva. Non-interactive secure computation based on cut-and-choose. In *Advances in Cryptology–EUROCRYPT 2014*, pages 387–404. Springer, 2014.

[BCD+09] Peter Bogetoft, Dan Lund Christensen, Ivan Damgård, Martin Geisler, Thomas Jakobsen, Mikkel Krøigaard, Janus Dam Nielsen, Jesper Buus Nielsen, Kurt Nielsen, Jakob Pagter, et al. Secure multiparty computation goes live. In *Financial Cryptography and Data Security*, pages 325–343. Springer, 2009.

[BCS05] John Black, Martin Cochran, and Thomas Shrimpton. On the impossibility of highly-efficient blockcipher-based hash functions. In Ronald Cramer, editor, *EUROCRYPT 2005*, volume 3494 of *LNCS*, pages 526–541. Springer, May 2005.

[BDNP08] Assaf Ben-David, Noam Nisan, and Benny Pinkas. FairplayMP: a system for secure multi-party computation. In Peng Ning, Paul F. Syverson, and Somesh Jha, editors, *ACM CCS 08*, pages 257–266. ACM Press, October 2008.

[BHKR13] Mihir Bellare, Viet Tung Hoang, Sriram Keelveedhi, and Phillip Rogaway. Efficient garbling from a fixed-key blockcipher. In *Security and Privacy (SP), 2013 IEEE Symposium on*, pages 478–492. IEEE, 2013.

[BHR12a] Mihir Bellare, Viet Tung Hoang, and Phillip Rogaway. Adaptively secure garbling with applications to one-time programs and secure outsourcing. In Xiaoyun Wang and Kazue Sako, editors, *ASIACRYPT 2012*, volume 7658 of *LNCS*, pages 134–153. Springer, December 2012.

[BHR12b] Mihir Bellare, Viet Tung Hoang, and Phillip Rogaway. Foundations of garbled circuits. In Ting Yu, George Danezis, and Virgil D. Gligor, editors, *ACM CCS 12*, pages 784–796. ACM Press, October 2012.

[BLW08] Dan Bogdanov, Sven Laur, and Jan Willemson. Sharemind: A framework for fast privacy-preserving computations. In Sushil Jajodia and Javier López, editors, *ESORICS 2008*, volume 5283 of *LNCS*, pages 192–206. Springer, October 2008.

[BMR90] Donald Beaver, Silvio Micali, and Phillip Rogaway. The round complexity of secure protocols. In *Proceedings of the twenty-second annual ACM symposium on Theory of computing*, pages 503–513. ACM, 1990.

[BOGW88] Michael Ben-Or, Shafi Goldwasser, and Avi Wigderson. Completeness theorems for non-cryptographic fault-tolerant distributed computation (extended abstract). In STOC 1988 [STO88], pages 1–10.

[BTW12] Dan Bogdanov, Riivo Talviste, and Jan Willemson. Deploying secure multi-party computation for financial data analysis (short paper). In *Proceedings of the 16th International Conference on Financial Cryptography and Data Security. FC'12*, pages 57–64, 2012.

[Can01] Ran Canetti. Universally composable security: A new paradigm for cryptographic protocols. In *42nd FOCS*, pages 136–145. IEEE Computer Society Press, October 2001.

[CCD88] David Chaum, Claude Crépeau, and Ivan Damgård. Multiparty unconditionally secure protocols (extended abstract). In STOC 1988 [STO88], pages 11–19.

[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*, pages 513–530. Springer, 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 'R' for healthcare statistics. *Journal of the American Medical Informatics Association*, 21(e2):e326–e331, 2014.

[DI05] Ivan Damgård and Yuval Ishai. Constant-round multiparty computation using a black-box pseudorandom generator. In *Advances in Cryptology–CRYPTO 2005*, pages 378–394. Springer, 2005.

[DKL+12] Ivan Damgård, Marcel Keller, Enrique Larraia, Christian Miles, and Nigel P. Smart. Implementing AES via an actively/covertly secure dishonest-majority MPC protocol. In Ivan Visconti and Roberto De Prisco, editors, *SCN 12*, volume 7485 of *LNCS*, pages 241–263. Springer, September 2012.

[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 *Computer Security–ESORICS 2013*, pages 1–18. Springer, 2013.

[FKN94] Uriel Feige, Joe Kilian, and Moni Naor. A minimal model for secure computation (extended abstract). In *26th ACM STOC*, pages 554–563. ACM Press, May 1994.

[FN13] Tore Kasper Frederiksen and Jesper Buus Nielsen. Fast and maliciously secure two-party computation using the GPU. In Michael J. Jacobson Jr., Michael E. Locasto, Payman Mohassel, and Reihaneh Safavi-Naini, editors, *ACNS 13*, volume 7954 of *LNCS*, pages 339–356. Springer, June 2013.

[Gei07] Martin Geisler. Viff: Virtual ideal functionality framework. *Homepage: http://viff. dk*, 2007.

[GG14] Juan A. Garay and Rosario Gennaro, editors. *CRYPTO 2014, Part II*, volume 8617 of *LNCS*. Springer, August 2014.

[GL05] Shafi Goldwasser and Yehuda Lindell. Secure multi-party computation without agreement. *Journal of Cryptology*, 18(3):247–287, July 2005.

[GMW87] Oded Goldreich, Silvio Micali, and Avi Wigderson. How to play any mental game or A completeness theorem for protocols with honest majority. In Alfred Aho, editor, *19th ACM STOC*, pages 218–229. ACM Press, May 1987.

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

[IKHC14] Dai Ikarashi, Ryo Kikuchi, Koki Hamada, and Koji Chida. Actively private and correct mpc scheme in $t < n/2$ from passively secure schemes with small overhead. Cryptology ePrint Archive, Report 2014/304, 2014. http://eprint.iacr.org/.

[IKKPC15] Yuval Ishai, Ranjit Kumaresan, Eyal Kushilevitz, and Anat Paskin-Cherniavsky. Secure computation with minimal interaction, revisited. In Rosario Gennaro and Matthew Robshaw, editors, *CRYPTO 2015*, volume 9216 of *LNCS*, pages 359–378. Springer, 2015.

[IKP10] Yuval Ishai, Eyal Kushilevitz, and Anat Paskin. Secure multiparty computation with minimal interaction. In Tal Rabin, editor, *CRYPTO 2010*, volume 6223 of *LNCS*, pages 577–594. Springer, August 2010.

[KS08] Vladimir Kolesnikov and Thomas Schneider. Improved garbled circuit: Free xor gates and applications. In *Automata, Languages and Programming*, pages 486–498. Springer, 2008.

[KS12] Benjamin Kreuter and Chih-hao Shen. Billion-gate secure computation with malicious adversaries. In *In USENIX Security*. Citeseer, 2012.

[LADM14] John Launchbury, Dave Archer, Thomas DuBuisson, and Eric Mertens. Application-scale secure multiparty computation. In *Programming Languages and Systems*, pages 8–26. Springer, 2014.

[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.

[LOS14] Enrique Larraia, Emmanuela Orsini, and Nigel P. Smart. Dishonest majority multi-party computation for binary circuits. In Garay and Gennaro [GG14], pages 495–512.

[LP09] Yehuda Lindell and Benny Pinkas. A proof of security of Yao's protocol for two-party computation. *Journal of Cryptology*, 22(2):161–188, April 2009.

[LPS08] Yehuda Lindell, Benny Pinkas, and Nigel P. Smart. Implementing two-party computation efficiently with security against malicious adversaries. In Rafail Ostrovsky, Roberto De Prisco, and Ivan Visconti, editors, *SCN 08*, volume 5229 of *LNCS*, pages 2–20. Springer, September 2008.

[LR14] Yehuda Lindell and Ben Riva. Cut-and-choose Yao-based secure computation in the online/offline and batch settings. In Garay and Gennaro [GG14], pages 476–494.

[LTW13] Sven Laur, Riivo Talviste, and Jan Willemson. From oblivious aes to efficient and secure database join in the multiparty setting. In *Applied Cryptography and Network Security*, pages 84–101. Springer, 2013.

[Nao91] Moni Naor. Bit commitment using pseudorandomness. *Journal of Cryptology*, 4(2):151–158, 1991.

[NNOB12] Jesper Buus Nielsen, Peter Sebastian Nordholt, Claudio Orlandi, and Sai Sheshank Burra. A new approach to practical active-secure two-party computation. In *Advances in Cryptology–CRYPTO 2012*, pages 681–700. Springer, 2012.

[RBO89] Tal Rabin and Michael Ben-Or. Verifiable secret sharing and multiparty protocols with honest majority (extended abstract). In *21st ACM STOC*, pages 73–85. ACM Press, May 1989.

[STO88] *20th ACM STOC*. ACM Press, May 1988.

[Yao82] Andrew Chi-Chih Yao. Protocols for secure computations (extended abstract). In *23rd FOCS*, pages 160–164. IEEE Computer Society Press, November 1982.

[Yao86] Andrew Chi-Chih Yao. How to generate and exchange secrets (extended abstract). In *27th FOCS*, pages 162–167. IEEE Computer Society Press, October 1986.

[ZRE15] Samee Zahur, Mike Rosulek, and David Evans. Two halves make a whole - reducing data transfer in garbled circuits using half gates. In Elisabeth Oswald and Marc Fischlin, editors, *EUROCRYPT 2015, Part II*, volume 9057 of *LNCS*, pages 220–250. Springer, April 2015.

[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 Security Proof For Naor Commitment Variant

In Section 2.3 we defined a multi-bit variant of Naor's commitment in the CRS model. For n-bit strings, we need a $crs \in \{0,1\}^{4n}$. Let $G : \{0,1\}^n \to \{0,1\}^{4n}$ be a pseudorandom generator, and let $pad : \{0,1\}^n \to \{0,1\}^{4n}$ be the function that prepends $3n$ zeroes to the front of its argument. Then the commitment scheme is:

- $\mathsf{Com}_{crs}(x; r)$: set $C = G(r) + crs \cdot pad(x)$, with arithmetic in $GF(2^{4n})$; set $\sigma = (r, x)$.
- $\mathsf{Chk}_{crs}(C, \sigma = (r, x))$: return x if $C = G(r) + crs \cdot pad(x)$; otherwise return $\perp$.

Theorem 2. *This commitment scheme satisfies the properties of binding and hiding given in Section 2.3.*

Proof. Hiding follows easily from the security of PRG G. In particular, for all crs and x, a commitment is generated as $C = G(r) + crs \cdot pad(x)$. Since r is chosen uniformly, the result is pseudorandom.

Binding follows from the following argument. An adversary succeeds in equivocating if and only if it can produce C, r, x, r', x' with $x \neq x'$ and:

$$\begin{aligned} & C = G(r) + crs \cdot pad(x) = G(r') + crs \cdot pad(x') \\ \iff & crs = [G(r) - G(r')](pad(x') - pad(x))^{-1} \end{aligned}$$

Since $x \neq x'$ and pad is injective, $pad(x') - pad(x) \neq 0$ indeed has a multiplicative inverse in the field.

Let us call a crs "bad" if there exists $r, r', x \neq x'$ satisfying the above property. Clearly an adversary's success probability in equivocation is bounded by $\Pr[crs \text{ is bad}]$.

There are at most 2^{2n} values of $G(r) - G(r')$ and at most 2^n values of $pad(x') - pad(x)$ (under the standard encoding of $GF(2^k)$ into $\{0,1\}^k$, we have $pad(x') - pad(x) = pad(x' \oplus x)$). Hence there are at most 2^{3n} bad values of crs. It follows that the probability of a random $crs \in \{0,1\}^{4n}$ being bad is at most $1/2^n$. □