Efficient Secure Two-Party Computation Using Symmetric Cut-and-Choose

Yan Huang[1], Jonathan Katz[1⋆], and David Evans[2⋆⋆]

[1] Dept. of Computer Science, University of Maryland
{jkatz,yhuang}@cs.umd.edu
[2] Dept. of Computer Science, University of Virginia
evans@cs.virginia.edu

Abstract. Beginning with the work of Lindell and Pinkas, researchers have proposed several protocols for secure two-party computation based on the *cut-and-choose* paradigm. In current instantiations of this approach, one party generates κ garbled circuits; some fraction of those are "checked" by the other party, and the remaining fraction are evaluated.

We introduce here the idea of *symmetric* cut-and-choose protocols, in which *both* parties generate κ circuits to be checked by the other party. The main advantage of our technique is that κ can be reduced by a factor of 3 while attaining the same statistical security level as in prior work. Since the number of garbled circuits dominates the costs of the protocol, especially as larger circuits are evaluated, our protocol is expected to run up to 3 times faster than existing schemes. Preliminary experiments validate this claim.

1 Introduction

Secure two-party computation was shown to be feasible in the late 1980s [35, 8]. But it is only in the past 10 years that the research community has devoted significant efforts toward making such protocols *practical*. Work in this direction was spurred by the Fairplay paper [25], which implemented Yao's protocol for two-party computation with security in the semi-honest setting. More recent work [10, 12, 11] has shown that Yao's protocol (in combination with other techniques) can be surprisingly efficient when semi-honest security is sufficient.

More desirable, of course, is to achieve security against malicious adversaries. While this is known to be feasible, in principle, using generic zero knowledge [8], a generic approach of this sort does not currently seem likely to result in efficient protocols even if specialized zero-knowledge proofs (as suggested in [15]) are used. The first technique to be explored for making efficient two-party computation protocols secure against malicious adversaries was the *cut-and-choose* paradigm. In that approach, roughly speaking, one party generates κ garbled circuits (where κ is a statistical security parameter); some fraction of those are "checked" by

[⋆] This work was supported by DARPA and NSF award #1111599.

[⋆⋆] This work was supported by AFOSR and NSF award #1111781.

the other party—who aborts if any misbehavior is detected—and the remaining fraction are evaluated with the results being used to derive the final output (we return to the exact mechanism for doing so in the next section). Cut-and-choose was used in a relatively naive way in [25] to give inverse-polynomial security. (In fact, the approach taken was later shown to be flawed [26, 16].) A rigorous analysis of the cut-and-choose paradigm was first given by Lindell and Pinkas [21], and their work was followed by numerous others exploring variations of this technique and their application to (ever more) efficient secure two-party computation [34, 24, 30, 32, 23, 33, 18].

In parallel with the above, other efficient approaches to achieving "full" malicious security in the two-party setting have also been explored. Approaches based on the IPS compiler [14] appear to have good asymptotic complexity [20], but seem challenging to implement (indeed, we are not aware of any implementations); other approaches [29, 5, 4] have round complexity proportional to the depth of the circuit being evaluated. Another direction is to explore weaker security guarantees [1, 26, 13], still against arbitrary malicious behavior. In the remainder of this paper we restrict our attention to protocols achieving the strongest notion of malicious security.

The critical question regarding the cut-and-choose approach is: *how many garbled-circuit copies (namely, κ) are needed to ensure some desired security level*? The value of κ has the greatest impact on the efficiency of cut-and-choose protocols, especially as larger circuits C are evaluated. The computational/communication complexity of such protocols is $O(\kappa \cdot k \cdot |C|) + \mathsf{poly}(n, k, s)$, where k is a cryptographic security parameter and n is the input length. Since $|C| \gg k, n$ (typical values are $k \approx 128$ and $n < 1000$, while $|C| \approx 10^9$ in [18]), the importance of minimizing κ is clear.

1.1 Prior Work

In previous applications of the cut-and-choose paradigm, one party (say, P_1) acts as the garbled-circuit generator and the other (P_2) acts as the garbled-circuit evaluator; assume for simplicity that only P_2 gets output. If the oblivious-transfer (OT) protocol used is secure against malicious adversaries, the main issue is to ensure correctness of P_2's output. (Note, however, that correctness is closely connected with privacy, since P_1 can potentially carry out a *selective failure* attack in which the output of P_2 is correlated with P_2's input, in a way which would not be possible in an ideal evaluation of the function.) Toward that end, P_2 checks some number c of the κ circuits generated by P_1 to make sure they were constructed correctly. Assuming they were, the remaining $\kappa - c$ garbled circuits are evaluated by P_2, who then outputs the *majority* value of those circuits' results on each output wire. (This informal description omits various details, since we wish to focus on the cut-and-choose aspect of the protocols.)

From the above we see that a malicious P_1 can successfully cheat if they generate b "bad" garbled circuits and (1) none of those bad garbled circuits is among the c garbled circuits checked by P_1, and (2) of the remaining $\kappa - c$ garbled circuits being evaluated, half or more are bad. Doing the analysis, prior work [21, 23] culminating in the work of Shelat and Shen [33] shows that using κ garbled circuits yields security level $2^{-0.32\kappa}$. Moreover, this bound was shown to be the best possible for a certain class of cut-and-choose approaches [33].

1.2 Our Contribution

We recast the cut-and-choose approach in a *symmetric* setting, where each party generates κ garbled circuits to be checked by the other party. In doing so, we are motivated by work of Mohassel and Franklin [26] (see also [13]) who show how symmetric creation/evaluation of garbled circuits (but without any cut-and-choose) can be used to achieve security with only one bit of "disallowed" leakage against malicious adversaries. Here we show how to extend their approach to achieve the standard (i.e., "full") notion of malicious security.

After checking each other's garbled circuits, each party in our protocol evaluates the remaining garbled circuits of the other party, and then the results of both parties' evaluations are securely "combined" to yield the final output. Informally, a party outputs a value v for some output wire of the circuit if and only if at least one of their own garbled circuits, and *at least one of the garbled circuits generated by the other party*, evaluate to v on that wire. Since an honest party always generates correct garbled circuits, our analysis shows that correctness holds as long as *at least one* of the evaluated circuits provided by the other party is correct. (This is in contrast to one-sided cut-and-choose, where a *majority* of the evaluated circuits must be correct.) Thus, a malicious party can successfully cheat only if they generate exactly $\kappa - c$ "bad" garbled circuits, and none of those is checked by the other party. Setting $c = \kappa/2$ (which minimizes the cheating probability), the probability of successful cheating is $\binom{\kappa}{\kappa/2}^{-1} = 2^{-\kappa + O(\log \kappa)}$. We can therefore achieve the same security level as previous work while reducing[3] κ by roughly a factor of 3.

As an added advantage, our protocol naturally supports having both parties receive output (an explicit concern of [33]), with no performance penalty if only one party should learn the output.

In concurrent work, Lindell [19] shows a different approach that achieves $2^{-\kappa}$ security using κ circuits generated by only one of the parties.

1.3 Outline of the Paper

In Section 2 we review the cryptographic building blocks used in our protocol. We provide an overview of the protocol in Section 3 along with some intuition for why it is secure. In Section 4 we provide a formal description of our protocol, and we prove security in Section 5. In Appendix A we give some preliminary experimental results showing that we outperform the recent work of [18].

[3] To be clear: in our protocol *each party* generates κ garbled circuits and so the total number of garbled circuits is 2κ. However, since this work is done in parallel by the two parties—in addition to whatever parallel processing is available on each user's own machine—and since the communication is symmetric, the "wall-clock time" of our protocol is expected to improve on previous protocols by up to a factor of 3.

2 Notation and Building Blocks

For simplicity, we describe our protocol using concrete (rather than asymptotic) notation. Nevertheless, it should be clear that our protocol can be cast in an asymptotic setting without difficulty.

Let $\mathbb{G}$ be a group of prime order q with generator g. We assume the computational Diffie-Hellman (CDH) problem is hard in $\mathbb{G}$. We let H be a hash function that will be treated in the analysis as a random oracle. We let Com be a commitment scheme.

We use the standard definitions of secure two-party computation [7].

2.1 Naor-Pinkas Oblivious Transfer

In our protocol we do not use oblivious transfer as a "black box," but instead rely on specific details of the OT protocol used. Although several candidate OT protocols could be used, for concreteness and efficiency we use an OT protocol due to Naor and Pinkas [27] which we now describe.

Say we have a sender holding inputs $x_0, x_1 \in \{0,1\}^*$, and a receiver holding input $b \in \{0,1\}$. In the first round, the sender chooses random $C \leftarrow \mathbb{G}$ and sends C to the other party. The receiver picks $k \leftarrow \mathbb{Z}_q$, defines $h^0 = g^k$ and $h^1 = C/g^k$, and sends $h = h^b$ to the sender. In turn, the sender chooses $r \leftarrow \mathbb{Z}_q$ and sends $g^r, H(h^r) \oplus x_0, H((C/h)^r) \oplus x_1$ to the other party. The receiver recovers x_b by computing $(g^r)^k$ and using the appropriate component of the sender's final message. Note that several independent OTs can share the same first message C.

This OT protocol is simulatable for a malicious receiver under the CDH assumption in the random oracle model. It achieves privacy (but is not known to be simulatable) against a malicious sender, and this suffices for our purposes. A variant of this protocol requires the receiver to give a (standard) perfect witness-indistinguishable proof of knowledge of $\log_g h$ or $\log_g(C/h)$ after sending h. We use this variant in our analysis since it simplifies the proof.

2.2 Garbled Circuits

We use a modification of standard garbled circuits [35]. Fix a function $f : \{0,1\}^n \times \{0,1\}^n \to \{0,1\}^n$. We abstract the construction/evaluation of a garbled circuit for f via algorithms $\mathsf{GenGC}, \mathsf{EvalGC}$ with the following properties. GenGC is a randomized algorithm that takes as input $2n$ input-wire labels $v_1^0, v_1^1, \ldots, v_n^0, v_n^1 \in \mathbb{G}$ (corresponding to the first input of f), $2n$ input-wire labels $v_{n+1}^0, v_{n+1}^1, \ldots, v_{2n}^0, v_{2n}^1 \in \{0,1\}^n$ (corresponding to the second input of f), and $2n$ output-wire labels $w_1^0, w_1^1, \ldots, w_n^0, w_n^1 \in \mathbb{Z}_q$. It outputs a garbled circuit GC. Deterministic algorithm EvalGC takes as input GC and $2n$ input-wire labels $v_1, \ldots, v_{2n}$; it outputs n values $b_1 \| w_1, \ldots, b_n \| w_n$, with $b_1, \ldots, b_n \in \{0,1\}$. Note that EvalGC explicitly outputs wire labels in addition to bits.

Correctness requires that for any set of input/output-wire labels, any garbled circuit GC output by $\mathsf{GenGC}\left(\{v_i^0, v_i^1\}_{i=1}^{2n}, \{w_i^0, w_i^1\}_{i=1}^{n}\right)$, and any $x, y \in \{0,1\}^n$

with $z = f(x, y)$, we have

$$\mathsf{EvalGC}\left(\mathsf{GC}, \{v_i^{x_i}\}_{i=1}^n, \{v_i^{y_i}\}_{i=n+1}^{2n}\right) = z_1 \| w_1^{z_1}, \ldots, z_n \| w_n^{z_n}.$$

Security requires a simulator SimGC such that for all x, y with $z = f(x, y)$, any $v_1^{x_1}, \ldots, v_n^{x_n} \in \mathbb{G}$, $v_{n+1}^{y_1}, \ldots, v_{2n}^{y_n} \in \{0,1\}^n$, and $w_1^0, w_1^1, \ldots, w_n^0, w_n^1 \in \mathbb{Z}_q$, the distribution

$$\left\{ \begin{array}{c} v_1^{1-x_1}, \ldots, v_n^{1-x_n} \leftarrow \mathbb{G}; \\ v_{n+1}^{1-y_1}, \ldots, v_{2n}^{1-y_n} \leftarrow \{0,1\}^n; \\ \mathsf{GC} \leftarrow \mathsf{GenGC}\left(\{v_i^0, v_i^1\}_{i=1}^{2n}, \{w_i^0, w_i^1\}_{i=1}^n\right) \end{array} : \left(\mathsf{GC}, \{v_i^{x_i}\}_{i=1}^n, \{v_{n+i}^{y_i}\}_{i=1}^n\right) \right\}$$

is computationally indistinguishable from

$$\left\{\mathsf{GC} \leftarrow \mathsf{SimGC}\left(x, z, \{v_i^{x_i}\}_{i=1}^n, \{v_{n+i}^{y_i}\}_{i=1}^n, \{w_i^{z_i}\}_{i=1}^n\right) : \left(\mathsf{GC}, \{v_i^{x_i}\}_{i=1}^n, \{v_{n+i}^{y_i}\}_{i=1}^n\right)\right\}.$$

In particular, this means (informally) that (1) given GC, $\{v_i^{x_i}\}_{i=1}^n$, and $\{v_i^{y_{n+i}}\}_{i=1}^n$, no information is leaked about $\{w_i^{1-z_i}\}_{i=1}^n$ where $z = f(x, y)$, and (2) this holds regardless of how the $\{v_i^{x_i}\}_{i=1}^n, \{v_{n+i}^{y_i}\}_{i=1}^n$ are chosen (as long as the other input-wire labels are random). These properties are not standard, but are easily seen to hold by modifying the construction/proof from [22].

Note: We always let input wires $1, \ldots, n$ denote the inputs of the party generating the circuit. Thus, technically, P_1 generates garbled circuits for the function f, and P_2 generates garbled circuits for the function $f'(y, x) \overset{\text{def}}{=} f(x, y)$.

2.3 Verifiable Secret Sharing

We use a notion of (non-interactive) verifiable secret sharing (VSS) that is weaker than the usual one in the literature. For our purposes, a t-out-of-κ VSS scheme comprises three algorithms $\mathsf{Share}, \mathsf{Vrfy}, \mathsf{Rec}$ with the following functionality:

- Share takes input $s \in \mathbb{Z}_q$ and outputs κ shares $w_1, \ldots, w_\kappa \in \mathbb{Z}_q$ and additional information pub.
- Vrfy takes as input the information pub, an index i, and a candidate share $w_i \in \mathbb{Z}_q$. It outputs a bit, with 1 denoting validity.
- Rec takes as input pub and t indices/shares $\{(i_j, w_{i_j})\}_{j=1}^t$. It outputs a value $s \in \mathbb{Z}_q$.

We require that for any $s \in \mathbb{Z}_q$, any $w_1, \ldots, w_\kappa, \mathsf{pub}$ output by $\mathsf{Share}(s)$, and any $i_1, \ldots, i_t \subset [\kappa]$, we have $\mathsf{Vrfy}(\mathsf{pub}, i, w_i) = 1$ and $\mathsf{Rec}(\mathsf{pub}, \{(i_j, w_{i_j})\}_{j=1}^t) = s$.

We define a secrecy requirement for an honest dealer, and a verifiability requirement for honest receivers. Secrecy requires hardness of recovering a random secret s given pub and at most $t-1$ shares. Formally, the following should be small for all efficient algorithms $\mathcal{A}$ and any $i_1, \ldots, i_{t-1}$:

$$\Pr[s \leftarrow \mathbb{Z}_q; (\mathsf{pub}, w_1, \ldots, w_\kappa) \leftarrow \mathsf{Share}(s) : \mathcal{A}(\mathsf{pub}, w_{i_1}, \ldots, w_{i_{t-1}}) = s].$$

Verifiability requires that the dealer cannot generate pub and two different sets of valid shares that reconstruct to different secrets. Formally, the following is small for all efficient algorithms $\mathcal{A}$:

$$\Pr\left[\big(\mathsf{pub}, \{(i_j, w_j)\}_{j=1}^t, \{(i'_j, w'_j)\}_{j=1}^t\big) \leftarrow \mathcal{A}\right.$$

$$\left. : \begin{array}{c} \forall j : \mathsf{Vrfy}(\mathsf{pub}, i_j, w_j) = 1 \\ \bigwedge \forall j : \mathsf{Vrfy}(\mathsf{pub}, i'_j, w'_j) = 1 \\ \bigwedge \mathsf{Rec}(\mathsf{pub}, \{(i_j, w_j)\}_{j=1}^t) \neq \mathsf{Rec}(\mathsf{pub}, \{(i'_j, w'_j)\}_{j=1}^t) \end{array} \right].$$

Feldman VSS [6] satisfies the above properties under the discrete-logarithm assumption.

3 High-Level Description of the Protocol

At a high level, the protocol proceeds in the following stages:

1. **Generate garbled circuits:** Each party generates κ garbled circuits along with their corresponding input-wire labels.
2. **Oblivious transfer:** Each party uses the Naor-Pinkas OT protocol (cf. Section 2.1) to obtain its input-wire labels for the garbled circuits constructed by the other party. This is done in such a way that a party must use the *same* effective input across all circuits.
3. **"Cut-and-choose":** Each party sends the garbled circuits they constructed to the other party. Using coin tossing, parties choose half of each of their circuits for checking. Then:
   (a) For each of its check circuits, each party (1) sends all the input-wire labels for that circuit (to prove that the check circuit was constructed correctly) and (2) reveals all the values it used as the OT sender in step 2 (to prove that it used the correct input-wire labels in the OT execution corresponding to the check circuit).
   (b) For each of its remaining circuits (the evaluation circuits), each party sends the input-wire labels corresponding to its own input.
4. **Output determination:** Each party evaluates the garbled circuits they received from the other party, using the input-wire labels obtained in steps 2 and 3(b). For each output wire i of the circuit, each party decides on output $z_i \in \{0, 1\}$ iff at least one of the circuits they evaluated (that the other party constructed) gave output z_i *and* at least one of the circuits the other party evaluated (that they constructed) gave output z_i.

We defer the details of step 4, and for now just assume it can be done. We also assume that if a party successfully passes the cut-and-choose step, then for at least one of that party's evaluation circuits (1) the evaluation circuit is constructed correctly and (2) the correct input-wire labels were used in the corresponding OT; this assumption holds except with probability at most $\binom{\kappa}{\kappa/2}^{-1}$.

The main issue to address is to ensure that a malicious party uses the same (effective) input in step 2 (when it obtains input-wire labels for its own input from the other party using OT) and for *all* the input-wire labels it sends in step 3(b) (for the garbled circuits that it generated). We achieve this by noting that when an honest receiver obtains the input-wire labels for its ith input wire during the OT step, it sends a message h_i for which (1) it knows $\log_g h_i$ when its effective input (on the ith wire) is 0, and (2) it knows $\log_g(C/h_i)$ when its effective input (on the ith wire) is 1. We require the parties to use this same "template" for the input-wire labels corresponding to their own input in the garbled circuits they prepare. That is, for each garbled circuit and each input wire i corresponding to an input of the circuit generator, the input-wire label v_i^0 corresponding to 0 is chosen such that $\log_g v_i^0$ is known, and the input-wire label v_i^1 corresponding to 1 is chosen such that $\log_g(C/v_i^1)$ is known. Moreover, this property is verified to hold (for the check circuits) during the cut-and-choose step. When sending its ith input-wire label v_i in step 3(b), each party must then also prove[4] that it knows $\log_g(v_i/h_i)$. This is reminiscent of a similar technique used by Shelat and Shen [33] to enforce input consistency among input-wire labels sent by the circuit generator; here, we extend it to enforce consistency also to the input-wire labels *received* as a circuit evaluator.

Given this—and still assuming step 4 can be carried out—one can informally verify that the protocol is secure. Assume for concreteness that P_2 is honest. Privacy of P_2's input is easy to see. As for correctness, P_2 constructed all its garbled circuits correctly and sent input-wire labels for its own input y in all its evaluation circuits. In step 2, P_1 obtained input-wire labels for *its* own (effective) input x in all of P_2's evaluation circuits. So all of P_2's garbled circuits that were evaluated by P_1 yield output $z \overset{\text{def}}{=} f(x, y)$. In the other direction, with high probability at least one of P_1's evaluation circuits GC^* was constructed correctly, and moreover the correct input-wire labels (for P_2's input) were used in the corresponding OT; thus, P_2 obtained the correct input-wire labels for its input y in GC^*. Furthermore, from the previous paragraph we know that the input-wire labels for P_1's input in GC^* correspond to the same input x it used before. Thus, evaluation of GC^* by P_2 also yields $z = f(x, y)$, and thus z will be the final output of P_2 in the protocol.

The missing piece is to show how to implement step 4, and this is the most involved part of our protocol. The basic idea is for each party to choose a secret value s_i^b for each output wire i of the circuit and each possible value $b \in \{0, 1\}$ that wire can take. Each secret is then split into κ shares $w_{1,i}^b, \ldots, w_{\kappa,i}^b$ using a $(\kappa/2 + 1)$-out-of-κ secret-sharing scheme. Share $w_{j,i}^b$ is then used as the label corresponding to b on the ith output wire of the jth garbled circuit. The net result is that for each output wire i and bit b, the other party can reconstruct s_i^b if and only if it learns $\kappa/2 + 1$ of the shares corresponding to that wire and bit.

Note that $\kappa/2$ shares of *every* wire and bit will be revealed as part of the cut-and-choose phase. Assuming again that P_2 is honest, we thus have the following:

[4] Actually, as in [33], the party can simply reveal $\log_g(v_i/h_i)$.

- As noted earlier, all of the garbled circuits that P_2 constructed will evaluate to the same value $z = f(x, y)$. This means that P_1 only learns shares corresponding to the secrets $s_1^{z_1}, \ldots, s_n^{z_n}$, and learns nothing about the remaining secrets $s_1^{1-z_1}, \ldots, s_n^{1-z_n}$. This gives P_2 a way to "test" whether the circuits it constructed (that were evaluated by P_1) resulted in output z by checking which of each pair of secrets P_1 knows (e.g., using a secure equality test).
- In the opposite direction, as long as *one* of the garbled circuits constructed by P_1 (and evaluated by P_2) yields z, this will give P_2 one additional share of each of $\tilde{s}_1^{z_1}, \ldots, \tilde{s}_n^{z_n}$ (where we use $\tilde{s}$ here to denote that these secrets are chosen by P_1) and hence P_2 will be able to reconstruct each of those secrets. Note that it does not matter *which* garbled circuit evaluates to z, as any correctly constructed circuit that evaluates to z reveals the requisite share.

One point omitted from the above discussion is that now it must be possible to check during the cut-and-choose phase that correct shares were used when constructing the garbled circuits. For this reason, we use *verifiable* secret sharing (see Section 2.3). We defer to the next section additional technical details of the protocol needed for the proof of security.

4 Formal Specification of the Protocol

Fix a function $f : \{0,1\}^n \times \{0,1\}^n \to \{0,1\}^n$ that parties P_1 and P_2 wish to compute over their respective inputs $x, y \in \{0,1\}^n$. We assume both parties learn the output, but it is easy to modify the protocol so that only one party learns the output. The protocol proceeds as follows.

1. P_1 chooses $C \leftarrow \mathbb{G}$ and sends it to P_2. Symmetrically, P_2 chooses $\widetilde{C} \leftarrow \mathbb{G}$ and sends it to P_1.
2. P_1 generates $4n$ input-wire labels for each of κ garbled circuits in the following way. For $j = 1, \ldots, \kappa$, it chooses $a_{j,1}^0, a_{j,1}^1, \ldots, a_{j,n}^0, a_{j,n}^1 \leftarrow \mathbb{Z}_q$ and sets the first $2n$ input-wire labels of circuit j to be of the form $\{v_{j,i}^0 = g^{a_{j,i}^0}\}_{i=1}^n$ and $\{v_{j,i}^1 = \widetilde{C}/g^{a_{j,i}^1}\}_{i=1}^n$. It chooses the next $2n$ input-wire labels of circuit j uniformly as $v_{j,n+1}^0, v_{j,n+1}^1, \ldots, v_{j,2n}^0, v_{j,2n}^1 \leftarrow \{0,1\}^n$.
   Symmetrically,[5] P_2 generates $4n$ input-wire labels $\widetilde{v}_{j,1}^0, \widetilde{v}_{j,1}^1, \ldots, \widetilde{v}_{j,2n}^0, \widetilde{v}_{j,2n}^1$ for $j = 1, \ldots, \kappa$.
   Each party then uses Naor-Pinkas OT to obtain the input-wire labels corresponding to its own input in the circuits generated by the other party. I.e., for $i = 1, \ldots, n$ party P_1 chooses $k_i \leftarrow \mathbb{Z}_q$, generates $(h_i^0, h_i^1) = (g^{k_i}, \widetilde{C}/g^{k_i})$, and sends $h_i \overset{\text{def}}{=} h_i^{x_i}$ to P_2. Then P_2 generates κ independent responses as in the Naor-Pinkas protocol, using inputs $(\widetilde{v}_{j,n+i}^0, \widetilde{v}_{j,n+i}^1)$ in the jth such instance where, recall, $\widetilde{v}_{j,n+i}^b$ denotes the label corresponding to bit b on the $(n+i)$th input wire in the jth garbled circuit. P_1 recovers $\widetilde{v}_{1,n+i}^{x_i}, \ldots, \widetilde{v}_{\kappa,n+i}^{x_i}$. P_2 acts symmetrically to obtain $v_{1,n+i}^{y_i}, \ldots, v_{\kappa,n+i}^{y_i}$ for $i = 1, \ldots, n$.

[5] Recall that the first n input wires always denote the inputs of the party generating the circuit.

3. For $i \in \{1, \ldots, n\}$ and $b \in \{0,1\}$, party P_1 chooses $s_i^b \leftarrow \mathbb{Z}_q$ and generates a $(\kappa/2+1)$-out-of-κ secret sharing $(\mathsf{pub}_i^b, w_{1,i}^b, \ldots, w_{\kappa,i}^b) \leftarrow \mathsf{Share}(s_i^b)$. It uses $w_{j,i}^b$ as the label for bit b on the ith output wire in the jth circuit, i.e., for $j = 1, \ldots, \kappa$ it computes the garbled circuit
$$\mathsf{GC}_j = \mathsf{GenGC}\left(\{v_{j,i}^0, v_{j,i}^1\}_{i=1}^{2n}, \{w_{j,i}^0, w_{j,i}^1\}_{i=1}^{n}\right).$$
It sends $\{\mathsf{GC}_j\}_{j=1}^{\kappa}$ and $\{\mathsf{pub}_i^0, \mathsf{pub}_i^1\}_{i=1}^{n}$ to P_2.

   P_2 acts symmetrically to obtain $\tilde{s}_i^b$ and $(\widetilde{\mathsf{pub}}_i^b, \tilde{w}_{1,i}^b, \ldots, \tilde{w}_{\kappa,i}^b)$ and to generate $\widetilde{\mathsf{GC}}_j$; it sends $\{\widetilde{\mathsf{GC}}_j\}_{j=1}^{\kappa}$ and $\{\widetilde{\mathsf{pub}}_i^0, \widetilde{\mathsf{pub}}_i^1\}_{i=1}^{n}$ to P_1.
4. For $j = 1, \ldots, \kappa$ and $i = 1, \ldots, n$, party P_1 commits to the input-wire labels $v_{j,i}^0$ and $v_{j,i}^1$ corresponding to its own input, in random permuted order. Let $\mathsf{ComSet}_{j,i}$ denote the resulting pair of commitments. P_2 acts symmetrically.
5. The parties run secure coin-tossing protocols to generate strings $\mathcal{J}, \widetilde{\mathcal{J}} \in \{0,1\}^{\kappa}$ that are each uniform among strings containing exactly $\kappa/2$ ones.[6] These are interpreted in the natural way as subsets of $\{1, \ldots, \kappa\}$ of size $\kappa/2$. $\widetilde{J}$ is used to check the garbled circuits constructed by P_1. Specifically, for $j = 1, \ldots, \kappa$:
   (a) If $j \in \widetilde{J}$ the jth circuit is a *check circuit.* Then P_1 sends $\{a_{j,i}^0, a_{j,i}^1\}_{i=1}^{n}$, $\{v_{j,i}^0, v_{j,i}^1\}_{i=n+1}^{2n}$, $\{w_{j,i}^0, w_{j,i}^1\}_{i=1}^{n}$, and the randomness it used to generate GC_j. It also reveals the sender-randomness it used in all the OTs corresponding to the jth circuit, and opens both commitments in $\mathsf{ComSet}_{j,i}$ for $i = 1, \ldots, n$.

       P_2 sets $v_{j,i}^0 = g^{a_{j,i}^0}$ and $v_{j,i}^1 = \widetilde{C}/g^{a_{j,i}^1}$ for $i = 1, \ldots, n$. It re-generates the jth garbled circuit and verifies that it matches GC_j. It verifies that $\{v_{j,i}^0, v_{j,i}^1\}_{i=n+1}^{2n}$ were used in the OTs for the jth circuit, and that the commitments in $\mathsf{ComSet}_{j,i}$ open to $\{v_{j,i}^0, v_{j,i}^1\}$ in some order. Finally, it checks that $\mathsf{Vrfy}(\mathsf{pub}_i^b, j, w_{j,i}^b) = 1$ for $i = 1, \ldots, n$ and $b \in \{0,1\}$. It aborts if any of these fail.

   (b) If $j \notin \widetilde{J}$ the jth circuit is an *evaluation circuit.* In this case, P_1 sends $(v_{j,1}, \ldots, v_{j,n}) \overset{\text{def}}{=} (v_{j,1}^{x_1}, \ldots, v_{j,n}^{x_n})$ (i.e., the wire labels corresponding to P_1's input in the jth circuit) to P_2. It also opens the commitment in $\mathsf{ComSet}_{j,i}$ that corresponds to $v_{j,i}$. Finally, it sends $\log_g(v_{j,1}/h_1)$, $\ldots$, $\log_g(v_{j,n}/h_n)$. (Recall that $h_1, \ldots, h_n$ are the values used by P_1 when acting as receiver in the Naor-Pinkas OT protocol.)

       P_2 checks that one of the commitments in $\mathsf{ComSet}_{j,i}$ opens to $v_{j,i}$, and verifies the discrete logarithms sent by P_1. It aborts if any inconsistencies are found.

   Symmetrically, the parties use $\mathcal{J}$ to check the garbled circuits constructed by P_2.

[6] This can be implemented easily by using a standard coin-tossing protocol to generate polynomially many uniform bits, and then using those bits as the random coins for applying a Knuth shuffle to the string $0^{\kappa/2}1^{\kappa/2}$.

6. For each evaluation circuit j of P_2, party P_1 evaluates $\widetilde{\mathsf{GC}}_j$ using the input-wire labels it obtained in steps 2 and 5. By doing so, it learns n values $\widetilde{b}_{j,1} \| \widetilde{w}_{j,1}, \ldots, \widetilde{b}_{j,n} \| \widetilde{w}_{j,n}$.
   For $i = 1, \ldots, n$ and $b \in \{0, 1\}$, party P_1 tries to recover[7] $\widetilde{s}_i^b$. To do so, it finds an evaluation circuit j for which $\widetilde{b}_{j,i} = b$ and $\widetilde{w}_{j,i}$ is a valid share of $\widetilde{s}_i^b$ (i.e., $\mathsf{Vrfy}(\widetilde{\mathsf{pub}_i^{\widetilde{b}_{j,i}}}, j, \widetilde{w}_{j,i}) = 1$). If no such j exists, it chooses $t_i^b \leftarrow \mathbb{Z}_q$. Otherwise, it computes t_i^b by running Rec using $\widetilde{\mathsf{pub}_i^{\widetilde{b}_{j,i}}}$, the $\kappa/2$ shares $\{(k, \widetilde{w}_{k,i}^{\widetilde{b}_{j,i}})\}_{k \in \mathcal{J}}$ it learned in step 5, and the additional share $(j, \widetilde{w}_{j,i})$.
   P_2 acts symmetrically to compute $\widetilde{t}_i^0, \widetilde{t}_i^1$ for $i = 1, \ldots, n$.
7. For $i = 1, \ldots, n$, the parties do the following: Run a secure equality test, with P_1 using input $s_i^0 \| t_i^0$ and P_2 using input $\widetilde{t}_i^0 \| \widetilde{s}_i^0$. If the result is 1, each party sets $z_i = 0$ and goes to the next i. Otherwise, the parties run a second equality test with P_1 using input $s_i^1 \| t_i^1$ and P_2 using input $\widetilde{t}_i^1 \| \widetilde{s}_i^1$. If the result is 1, each party sets $z_i = 1$ and goes to the next i. If neither equality test succeeds for some i then cheating is detected and the parties abort.
   Assuming no abort has occurred, each party then outputs $z = z_1 \cdots z_n$.

4.1 Optimizations

For simplicity in our proof of security in the following section, we analyze the protocol as presented above. However, we observe that the following optimizations can be applied to the protocol (and the reader can verify that the proof can be suitably modified for each of these).

Naor-Pinkas OT. We assume a variant of Naor-Pinkas OT is used in which the receiver gives a witness-indistinguishable (WI) proof of knowledge that its message was computed correctly (see Section 2.1). This is used in our proof to extract the receiver's input. In fact, as shown in [27], such WI proofs are not necessary and extraction can be done using the random-oracle queries of the receiver. The same is true in our setting, though it complicates the presentation of the proof.

Secure coin tossing. In the (programmable) random-oracle model, very efficient coin tossing is possible since it is trivial to construct an equivocal and extractable commitment scheme.

Secure equality testing. In our proof, we assume a hybrid world in which the parties have access to an ideal functionality for equality testing; equivalently (relying on standard composition theorems [3]), we assume that the equality test is done using a fully secure protocols for this task.

In fact, using a fully secure equality test is overkill for our purposes. Instead, we can use a different approach that is very efficient in the random-oracle model. First, assume the VSS scheme has the stronger property of indistinguishability, i.e., given pub and $t - 1$ shares of a uniform secret $s \in \{0, 1\}^n$, it is hard to

[7] In an honest execution, only one of $\widetilde{s}_i^0$ or $\widetilde{s}_i^1$ will be recovered.

distinguish s from an independent uniform value $s' \in \{0,1\}^n$. (Any VSS scheme satisfying the unpredictability requirement from Section 2.3 can be modified to achieve this stronger notion in the random-oracle model by simply hashing the secret.) Then, rather than performing an equality test using values $s_i^0 \| t_i^0$ and $\widetilde{t}_i^0 \| \widetilde{s}_i^0$ (resp., $s_i^1 \| t_i^1$ and $\widetilde{t}_i^1 \| \widetilde{s}_i^1$) as before, the parties now carry out an equality test on values $s_i^0 \oplus t_i^0$ and $\widetilde{t}_i^0 \oplus \widetilde{s}_i^0$ (resp., $s_i^1 \oplus t_i^1$ and $\widetilde{t}_i^1 \oplus \widetilde{s}_i^1$). At this point, we observe that a full-fledged equality test is not needed since (1) the honest party's input to the equality test is either known to the malicious party or is (indistinguishable from) uniform, and (2) in either case, it is ok if the honest party's input to the equality test is leaked to the other party after equality is checked. Thus, it suffices to use a "cheap" equality test in which P_1 (resp., P_2) commits to, e.g., $s_i^0 \oplus t_i^0$ (resp., to $\widetilde{t}_i^0 \oplus \widetilde{s}_i^0$) using an extractable and equivocal commitment scheme (which is easily constructed in the random-oracle model), and then each party decommits and checks equality of the decommitted results in the clear.

Saving bandwidth. Following an observation in [9], we can modify the way we do cut-and-choose as follows: Parties construct their jth garbled circuit by choosing a random seed seed_j and using that seed to generate certain (pseudo)random choices they need for constructing that circuit. (In our case, this would mean using seed_j to generate $\{a_{j,i}^0, a_{j,i}^1\}_{i=1}^n$, $\{v_{j,i}^0, v_{j,i}^1\}_{i=n+1}^{2n}$, and the randomness used to generate GC_j.) Then, in step 3, the parties send the *hash* $\mathsf{hGC}_j = H(\mathsf{GC}_j)$ in place of GC_j. If j is a check circuit then seed_j is sent; the other party re-generates GC_j and verifies that $H(\mathsf{GC}_j) = \mathsf{hGC}_j$. If j is an evaluation circuit then GC_j is sent and the other party checks that $H(\mathsf{GC}_j) = \mathsf{hGC}_j$. Since $|\mathsf{seed}_j| + |\mathsf{hGC}_j| \ll |\mathsf{GC}_j|$, this has the effect of reducing the bandwidth in steps 3 and 5 (which dominate the bandwidth of the entire protocol) by roughly half.

Batch verification. We can use batch verification [2] when simultaneously verifying validity of shares in step 5(a) (assuming Feldman VSS is used) and the discrete logarithms in step 5(b).

Efficient garbled circuits. Our protocol is fully compatible with existing optimizations for garbled circuits such as garbled-row reduction [28] and the free-XOR technique [17].[8]

5 Proof of Security

Theorem 1. *Under the assumptions outlined in Section 2, and modeling H as a random oracle, the protocol in the previous section securely computes f in the presence of malicious adversaries.*

Since we are not in an asymptotic setting, technically speaking "secure" is not well-defined. In the proof below, all steps introduce a computational security

[8] We cannot apply the free-XOR optimization at first-level gates because of the way the circuit generator chooses the input-wire labels. However, the free-XOR method can be used at all lower levels of the circuit.

factor (which can be set as small as desired by setting the cryptographic security parameter large enough) except for one step which introduces a statistical security factor of $\binom{\kappa}{\kappa/2}^{-1} = 2^{-\kappa+O(\log\kappa)}$.

All our assumptions are standard, and can be based on the CDH assumption in $\mathbb{G}$. We remark that the only place the random oracle is used is for the Naor-Pinkas OT. It would be possible to remove the random oracle by switching, e.g., to the OT protocol of [31] (and modifying the rest of the protocol accordingly). Although this would impact the efficiency, the effect would be proportional to the input length and not the size of the circuit being computed.

Proof. We analyze the protocol in a hybrid world in which the parties have access to ideal functionalities for coin tossing and equality testing. Using standard composition theorems [3], this implies security when those sub-routines are instantiated using secure protocols for those tasks.

Since the protocol is symmetric, we assume without loss of generality that P_1 is malicious. Let y denote the input of P_2. We define a sequence of experiments, beginning with the real execution of the protocol between P_1 and P_2 (in the hybrid world discussed above) and ending with an ideal execution involving a simulator $\mathcal{S}$ playing the role of the first party and interacting with a trusted party computing f. We show that each experiment is indistinguishable from the one before it, taking into account both the view/output of the malicious party and the output of P_2.

Experiment 0. This is the real execution of the protocol (in the hybrid world discussed above) between P_1 and the honest P_2 holding input y.

Experiment 1. Here we change the way P_2 behaves when acting as OT sender in step 2 and when sending commitments in step 4. First of all, we now pick $\mathcal{J}$ at the outset of the experiment. This defines the check circuits and evaluation circuits for P_2. Next, in each instance i in which P_1 acts as OT receiver in step 2 and sends message h_i, we extract (using the WI proof of knowledge) either $\log_g h_i$ or $\log_g(\widetilde{C}/h_i)$. In the former case we set $x_i = 0$ and in the latter case we set $x_i = 1$. Then, when computing the κ responses for the ith OT, in each response that corresponds to an evaluation circuit j of P_2 we continue to use $\widetilde{v}_{j,n+i}^{x_i}$ but we replace $\widetilde{v}_{j,n+i}^{1-x_i}$ with the all-0 string. (Responses that correspond to check circuits of P_2 are treated exactly as before.)

In addition, for each evaluation circuit j of P_2 and $i = 1, \ldots, n$, we now set $\mathsf{ComSet}_{j,i} = \{\mathsf{Com}(\widetilde{v}_{j,i}^{y_i}), \mathsf{Com}(g)\}$, in random permuted order.

Indistinguishability of Experiments 0 and 1 follows easily from the security of Naor-Pinkas OT (based on the CDH assumption in the random-oracle model) and computational hiding of Com.

Experiment 2. Now we generate all the evaluation circuits of P_2 using the garbled-circuit simulator SimGC. In more detail: after extracting P_1's effective input x as in the previous experiment, we compute $z = f(x, y)$. In step 3, once the $\{\widetilde{w}_{j,i}^b\}$ have been determined we compute for every evaluation circuit j the simulated garbled circuit[9] $\widetilde{\mathsf{GC}}_j \leftarrow \mathsf{SimGC}\left(x, z, \{\widetilde{v}_{j,i}^{y_i}\}_{i=1}^{n}, \{\widetilde{v}_{j,n+i}^{x_i}\}_{i=1}^{n}, \{\widetilde{w}_{j,i}^{z_i}\}_{i=1}^{n}\right)$. The remainder of the experiment is exactly as in Experiment 1.

Indistinguishability of Experiments 1 and 2 follows from security of the garbled-circuit simulation algorithm as defined in Section 2.2.

Note that in Experiment 2, we no longer use $\{\widetilde{v}_{j,i}^{1-y_i}\}_{i=1}^{n}$, $\{\widetilde{v}_{j,n+i}^{1-x_i}\}_{i=1}^{n}$, or $\{\widetilde{w}_{j,i}^{1-z_i}\}_{i=1}^{n}$ for any evaluation circuit j of P_2.

Experiment 3. This is the same as the previous experiment, except that now when performing the ith pair of equality tests we proceed as follows: if $z_i = 1$, we return 0 to both parties in the first equality test; if $z_i = 0$, we return 0 to both parties in the second equality test (if run).

Indistinguishability of this experiment from Experiment 2 follows from secrecy of VSS. Specifically, for $i = 1, \ldots, n$ only $\widetilde{\mathsf{pub}}_i^{1-z_i}$ and $\kappa/2$ shares of the secret $\widetilde{s}_i^{1-z_i}$ are used throughout the entire experiment before the equality tests. Thus, the probability (in Experiment 2) that P_1 can make any of the equality tests corresponding to $1 - z_i$ return 1 is negligible.

Experiment 4. If P_1 successfully responds to the "challenge" $\widetilde{\mathcal{J}}$ chosen during the cut-and-choose step, we repeatedly rewind P_1 in an attempt to find a $\widetilde{\mathcal{J}}' \neq \widetilde{\mathcal{J}}$ for which P_1 also responds correctly.[10] If no such $\widetilde{\mathcal{J}}'$ is found, output fail. Otherwise, re-send the original challenge $\widetilde{\mathcal{J}}$ and continue as in the previous experiment.

The only difference between this experiment and the previous one occurs in case P_1 responds correctly to only a single challenge $\widetilde{\mathcal{J}}$ and that challenge happens to be the one chosen during the experiment. This can occur with probability at most $1/\binom{\kappa}{\kappa/2}$.

Experiment 5. We now change how we compute $\tilde{t}_i^{z_i}$ for all i. (Recall that $\tilde{t}_i^{z_i}$ represents P_2's guess for P_1's secret $s_i^{z_i}$.) Assuming P_1 answers two different challenges $\widetilde{\mathcal{J}}, \widetilde{\mathcal{J}}'$ correctly, there is some $j^* \in \{1, \ldots, \kappa\}$ such that j^* is an evaluation circuit with respect to $\widetilde{\mathcal{J}}$ but a check circuit with respect to $\widetilde{\mathcal{J}}'$. For any such j^*, we reconstruct $s_i^{z_i}$ using the share $w_{j^*,i}^{z_i}$ sent by P_1 when answering challenge $\widetilde{\mathcal{J}}'$, along with the $\kappa/2$ other shares of $s_i^{z_i}$ that were sent by P_1 when answering challenge $\widetilde{\mathcal{J}}$. We then set $\tilde{t}_i^{z_i} = s_i^{z_i}$ and use that value in the relevant equality test later.

We claim that this experiment is indistinguishable from the previous one; this is the crux of the proof. To prove this, we show that the shares $\{w_{j^*,i}^{z_i}\}_{i=1}^{n}$ computed in Experiment 5 are, except with negligible probability, the same shares that P_2 obtains by evaluating circuit GC_{j^*} in Experiment 4. Verifiability of the secret-sharing scheme then implies that, except with negligible probability,

[9] Recall that the first n input wires always denote the inputs of the party generating the circuit, so in this case correspond to input y.

[10] We use standard techniques in order to ensure that the experiment runs in expected polynomial time. Specifically, in parallel with rewinding P_1 and sending a random challenge $\widetilde{\mathcal{J}}' \neq \widetilde{\mathcal{J}}$ we also enumerate over all possible $\widetilde{\mathcal{J}}'$; we output fail if we find that $\widetilde{\mathcal{J}}$ is the only challenge to which P_1 responds correctly.

the same values $\{\widetilde{t}_i^{z_i}\}_{i=1}^n$ are computed in both experiments (namely, even if in Experiment 4 a valid share from an evaluation circuit other than j^* is used by P_2 to reconstruct some $s_i^{z_i}$).

Fix i. To see that the same share $w_{j^*,i}^{z_i}$ is computed in each experiment, observe that in Experiment 4 the share $w_{j^*,i}^{z_i}$ is computed by evaluating garbled circuit GC_{j^*} using input-wire labels for P_2's input that P_2 obtains from the OTs corresponding to circuit j^*, and input-wire labels for P_1's input that were sent by P_1 in step 5. Because P_1 responds correctly to challenge $\widetilde{\mathcal{J}}'$, in which j^* is a check circuit, we know that: (1) GC_{j^*} is correctly constructed; (2) the input-wire labels that P_2 obtained from the OTs are correct labels for GC_{j^*} that correspond to P_2's input y; (3) the input-wire labels for its own input that P_1 sends must be correct labels for GC_{j^*} (this follows from binding of the commitments in $\{\mathsf{ComSet}_{j^*,i}\}_{i=1}^n$) and moreover must correspond to the same effective input x defined by P_1's execution as OT receiver (otherwise we obtain a discrete logarithm of the random group element $\widetilde{C}$). Since GC_{j^*}, when evaluated on input-wire labels corresponding to x and y, yields the share $w_{j^*,i}^{z_i}$ on the ith output wire, we are done.

In Experiment 5 none of P_1's evaluation circuits need to be evaluated by P_2. Moreover, P_2 no longer needs to compute its output in any of the OTs in which it acts as receiver.

Experiment 6. In the previous experiment, when P_2 acts as OT receiver it sends $\widetilde{h}_i$ with either $\log_g \widetilde{h}_i$ or $\log_g(C/\widetilde{h}_i)$ known (depending on y_i). The input-wire labels $\{\widetilde{v}_{j,i}^{y_i}\}_{i=1}^n$ (when j is an evaluation circuit) are chosen in a similar way. In this experiment, for $i = 1, \ldots, n$ we choose $\widetilde{h}_i$ uniform with $\log_g \widetilde{h}_i$ known so that we are simply running the OT execution honestly using input 0. Similarly, choose $\widetilde{v}_{j,i}^{y_i}$ uniform with $\log_g \widetilde{v}_{j,i}^{y_i}$ known for every evaluation circuit j. (Note that this allows P_2 to reveal $\log_g(\widetilde{v}_{j,i}/\widetilde{h}_i)$ in step 5 for every evaluation circuit j.)

This experiment is distributed *identically* to the previous experiment, since g^k and C/g^k (where k is uniform in each case) have the same distribution. (P_2 also gives a WI proof of knowledge of either $\log_g \widetilde{h}_i$ or $\log_g(C/\widetilde{h}_i)$, but we assume a *perfect* WI proof is used.)

To conclude, we observe that Experiment 6 can equivalently be described in terms of an ideal-world execution in which the honest P_2 and a simulator $\mathcal{S}$ (playing the role of the first party, and running P_1 as a subroutine) interact with a trusted party computing f. Namely, $\mathcal{S}$ works as follows:

1. Choose $\mathcal{J}$ in advance; this defines the check circuits and the evaluation circuits for the simulated P_2. Choose $\widetilde{C} \leftarrow \mathbb{G}$ and send it to P_1. Receive in return $C \in \mathbb{G}$.
2. For each check circuit j, generate input-wire labels as in the real protocol. For each evaluation circuit j, choose $\widetilde{a}_{j,1}, \ldots, \widetilde{a}_{j,n} \leftarrow \mathbb{Z}_q$ and set $\widetilde{v}_{j,i} = g^{\widetilde{a}_{j,i}}$ for $i = 1, \ldots, n$. Also choose $\widetilde{v}_{j,n+i} \leftarrow \{0,1\}^n$ for $i = 1, \ldots, n$.
   When P_2 acts as OT receiver, run the OT protocol honestly using input 0.
   In each instance i in which P_2 acts as OT sender, extract from P_1 (by rewinding the WI proof of knowledge) either $\log_g h_i$ or $\log_g(\widetilde{C}/h_i)$. In the

former case set $x_i = 0$ and in the latter case set $x_i = 1$. Then, for check circuits send the final OT message exactly as in the real protocol, and for any evaluation circuit j send the final OT message using $\widetilde{v}_{j,n+i}$ as the x_i-input, and the 0-string as the $(1 - x_i)$-input.

3. Send x to the trusted party, and receive in return an output z.

   Generate $\{\widetilde{\mathsf{pub}}_i^b, \widetilde{w}_{j,i}^b\}$ as in the real protocol. Then for each evaluation circuit j, compute $\widetilde{\mathsf{GC}}_j \leftarrow \mathsf{SimGC}\left(x, z, \{\widetilde{v}_{j,i}\}_{i=1}^{2n}, \{\widetilde{w}_{j,i}^{z_i}\}_{i=1}^{n}\right)$; for each check circuit j, compute $\widetilde{\mathsf{GC}}_j$ as in the real protocol. Send $\{\widetilde{\mathsf{GC}}_j\}_{j=1}^{\kappa}$ and $\{\widetilde{\mathsf{pub}}_i^0, \widetilde{\mathsf{pub}}_i^1\}_{i=1}^{n}$ to P_1.

   Receive in return $\{\mathsf{GC}_j\}_{j=1}^{\kappa}$ and $\{\mathsf{pub}_i^0, \mathsf{pub}_i^1\}_{i=1}^{n}$ from P_1.
4. For each check circuit j, compute $\{\widetilde{\mathsf{ComSet}}_{j,i}\}_{i=1}^{n}$ as in the real protocol. For each evaluation circuit j, set $\widetilde{\mathsf{ComSet}}_{j,i} = \{\mathsf{Com}(\widetilde{v}_{j,i}), \mathsf{Com}(g)\}$ in random permuted order. Send all these pairs of commitments to P_1, and receive in return all the pairs of commitments from P_1.
5. Give P_1 the value $\mathcal{J}$ as the output of the appropriate coin-tossing protocol. Respond for all check circuits as in the real protocol. For each evaluation circuit j, send $\{\widetilde{v}_{j,i}\}_{i=1}^{n}$, open the appropriate commitment from $\{\widetilde{\mathsf{ComSet}}_{j,i}\}_{i=1}^{n}$, and send $\{\log_g(\widetilde{v}_{j,i}/\widetilde{h}_i)\}_{i=1}^{n}$, where $\widetilde{h}_i$ is the message sent by P_2 in the ith OT when P_2 is receiver.

   Choose $\widetilde{\mathcal{J}}$ at random as in the real protocol, and give it to P_1. If P_1 responds correctly, then repeatedly rewind to find $\widetilde{\mathcal{J}}' \neq \widetilde{\mathcal{J}}$ for which P_1 responds correctly. (If none is found, $\mathcal{S}$ aborts with output fail.) Rewind again and continue the interaction using $\widetilde{\mathcal{J}}$.
6. Let j^* be a circuit which is an evaluation circuit in $\widetilde{\mathcal{J}}$, but a check circuit in $\widetilde{\mathcal{J}}'$. For $i = 1, \ldots, n$, use the $\kappa/2$ shares of $s_i^{z_i}$ from P_1's check circuits (with respect to $\mathcal{J}$) plus the additional share of $s_i^{z_i}$ from circuit j^* (that was a check circuit with respect to $\widetilde{\mathcal{J}}'$) to reconstruct $s_i^{z_i}$. Set $\widetilde{t}_i^{z_i} = s_i^{z_i}$.
7. For $i = 1, \ldots, n$, do the following.
   - If $z_i = 0$, obtain P_1's input $s_i^0 \| t_i^0$ to the first equality test. If $s_i^0 \| t_i^0 = \widetilde{t}_i^0 \| \widetilde{s}_i^0$, return 1; else return 0. Return 0 to the second equality test (if run).
   - If $z_i = 1$, return 0 to the first equality test. Then obtain P_1's input $s_i^1 \| t_i^1$ to the second equality test. If $s_i^1 \| t_i^1 = \widetilde{t}_i^1 \| \widetilde{s}_i^1$, return 1; else return 0.

   If for some i both equality tests return 0, abort. If an abort occurred, send abort to the trusted party; otherwise, send $\mathsf{continue}$.

This completes the proof.

Acknowledgments

Yan Huang would like to thank Ivan Damgård and Jesper Nielsen for their helpful comments on this work.

References

1. Y. Aumann and Y. Lindell. Security against covert adversaries: Efficient protocols for realistic adversaries. *Journal of Cryptology*, 23(2):281–343, 2010.
2. M. Bellare, J. A. Garay, and T. Rabin. Fast batch verification for modular exponentiation and digital signatures. In *Advances in Cryptology — Eurocrypt '98*, volume 1403 of *LNCS*, pages 236–250. Springer, 1998.
3. R. Canetti. Security and composition of multiparty cryptographic protocols. *Journal of Cryptology*, 13(1):143–202, 2000.
4. I. Damgård, M. Keller, E. Larraia, C. Miles, and N. P. Smart. Implementing AES via an actively/covertly secure dishonest-majority MPC protocol. In *8th Intl. Conf. on Security and Cryptography for Networks (SCN)*, volume 7485 of *LNCS*, pages 241–263. Springer, 2012.
5. I. Damgård, V. Pastro, N. P. Smart, and S. Zakarias. Multiparty computation from somewhat homomorphic encryption. In *Advances in Cryptology — Crypto 2012*, volume 7417 of *LNCS*, pages 643–662. Springer, 2012.
6. P. Feldman. A practical scheme for non-interactive verifiable secret sharing. In *28th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 427–437. IEEE, 1987.
7. O. Goldreich. *Foundations of Cryptography, vol. 2: Basic Applications.* Cambridge University Press, Cambridge, UK, 2004.
8. O. Goldreich, S. Micali, and A. Wigderson. How to play any mental game, or a completeness theorem for protocols with honest majority. In *19th Annual ACM Symposium on Theory of Computing (STOC)*, pages 218–229. ACM Press, 1987.
9. V. Goyal, P. Mohassel, and A. Smith. Efficient two party and multi party computation against covert adversaries. In *Advances in Cryptology — Eurocrypt 2008*, volume 4965 of *LNCS*, pages 289–306. Springer, 2008.
10. W. Henecka, S. Kögl, A.-R. Sadeghi, T. Schneider, and I. Wehrenberg. TASTY: tool for automating secure two-party computations. In *17th ACM Conf. on Computer and Communications Security (CCS)*, pages 451–462. ACM Press, 2010.
11. Y. Huang, D. Evans, and J. Katz. Private set intersection: Are garbled circuits better than custom protocols? In *Network and Distributed System Security Symposium (NDSS).* The Internet Society, 2012.
12. Y. Huang, D. Evans, J. Katz, and L. Malka. Faster secure two-party computation using garbled circuits. In *20th USENIX Security Symposium.* USENIX Association, 2011.
13. Y. Huang, J. Katz, and D. Evans. Quid pro quo-tocols: Strengthening semi-honest protocols with dual execution. In *IEEE Symposium on Security & Privacy*, pages 239–254. IEEE, 2012.
14. Y. Ishai, M. Prabhakaran, and A. Sahai. Founding cryptography on oblivious transfer — efficiently. In *Advances in Cryptology — Crypto 2008*, volume 5157 of *LNCS*, pages 572–591. Springer, 2008.
15. S. Jarecki and V. Shmatikov. Efficient two-party secure computation on committed inputs. In *Advances in Cryptology — Eurocrypt 2007*, volume 4515 of *LNCS*, pages 97–114. Springer, 2007.
16. M. Kiraz and B. Schoenmakers. A protocol issue for the malicious case of yao's garbled-circuit construction. In *Proc. 27th Symp. on Information Theory in the Benelux*, pages 283–290, 2006.
17. V. Kolesnikov and T. Schneider. Improved garbled circuit: Free XOR gates and applications. In *35th Intl. Colloquium on Automata, Languages, and Programming (ICALP), Part II*, volume 5126 of *LNCS*, pages 486–498. Springer, 2008.

18. B. Kreuter, A. Shelat, and C. Shen. Billion-gate secure computation with malicious adversaries. In *21st USENIX Security Symposium.* USENIX Association, 2012.
19. Y. Lindell. Fast cut-and-choose based protocols for malicious and covert adversaries. Crypto 2013, to appear.
20. Y. Lindell, E. Oxman, and B. Pinkas. The IPS compiler: Optimizations, variants and concrete efficiency. In *Advances in Cryptology — Crypto 2011*, volume 6841 of *LNCS*, pages 259–276. Springer, 2011.
21. Y. Lindell and B. Pinkas. An efficient protocol for secure two-party computation in the presence of malicious adversaries. In *Advances in Cryptology — Eurocrypt 2007*, volume 4515 of *LNCS*, pages 52–78. Springer, 2007.
22. Y. Lindell and B. Pinkas. A proof of security of Yao's protocol for two-party computation. *Journal of Cryptology*, 22(2):161–188, 2009.
23. Y. Lindell and B. Pinkas. Secure two-party computation via cut-and-choose oblivious transfer. *Journal of Cryptology*, 25(4):680–722, 2012.
24. Y. Lindell, B. Pinkas, and N. Smart. Implementing two-party computation efficiently with security against malicious adversaries. In *6th Intl. Conf. on Security and Cryptography for Networks (SCN)*, volume 5229 of *LNCS*, pages 2–20. Springer, 2008.
25. D. Malkhi, N. Nisan, B. Pinkas, and Y. Sella. Fairplay — a secure two-party computation system. In *Proc. 13th USENIX Security Symposium*, pages 287–302. USENIX Association, 2004.
26. P. Mohassel and M. Franklin. Efficiency tradeoffs for malicious two-party computation. In *9th Intl. Conference on Theory and Practice of Public Key Cryptography (PKC)*, volume 3958 of *LNCS*, pages 458–473. Springer, 2006.
27. M. Naor and B. Pinkas. Efficient oblivious transfer protocols. In *12th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 448–457. ACM-SIAM, 2001.
28. M. Naor, B. Pinkas, and R. Sumner. Privacy preserving auctions and mechanism design. In *Proc. 1st ACM Conf. on Electronic Commerce*, pages 129–139. ACM, 1999.
29. J. B. Nielsen, P. S. Nordholt, C. Orlandi, and S. S. Burra. A new approach to practical active-secure two-party computation. In *Advances in Cryptology — Crypto 2012*, volume 7417 of *LNCS*, pages 681–700. Springer, 2012.
30. J. B. Nielsen and C. Orlandi. LEGO for two-party secure computation. In *6th Theory of Cryptography Conference — TCC 2009*, volume 5444 of *LNCS*, pages 368–386. Springer, 2009.
31. C. Peikert, V. Vaikuntanathan, and B. Waters. A framework for efficient and composable oblivious transfer. In *Advances in Cryptology — Crypto 2008*, volume 5157 of *LNCS*, pages 554–571. Springer, 2008.
32. B. Pinkas, T. Schneider, N. Smart, and S. Williams. Secure two-party computation is practical. In *Advances in Cryptology — Asiacrypt 2009*, volume 5912 of *LNCS*, pages 250–267. Springer, 2009.
33. A. Shelat and C.-H. Shen. Two-output secure computation with malicious adversaries. In *Advances in Cryptology — Eurocrypt 2011*, volume 6632 of *LNCS*, pages 386–405. Springer, 2011.
34. D. P. Woodruff. Revisiting the efficiency of malicious two-party computation. In *Advances in Cryptology — Eurocrypt 2007*, volume 4515 of *LNCS*, pages 79–96. Springer, 2007.
35. A. C.-C. Yao. How to generate and exchange secrets. In *27th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 162–167. IEEE, 1986.

A Experimental Results

We describe some preliminary experimentals indicating that our protocol significantly outperforms the recent work of [18].

We implemented our protocol in Java using all the optimizations of Section 4.1. We evaluated the protocol at the 80-bit security level, which means in particular that (1) each party generates 84 garbled circuits, 42 of which are checked; (2) the length of all wire labels is 80 bits; and (3) we use an order-q subgroup of $\mathbb{Z}_p^*$ where $|p| = 1024, |q| = 160$. We ran experiments over a LAN using two laptops with Intel Core i7 2.4GHz processors. Note that 80-bit security was also used in the experiments of [18].

In typical settings where the number of gates in the underlying circuit is much larger than the number of inputs/outputs, the dominant overall cost of the protocol is the generation, sending, and checking of the garbled circuits. When each side uses only a single core, our protocol evaluates circuits at the rate of 1.4 ms/gate. By comparison, the implementation of Kreuter et al. [18] evaluates circuits at the rate of about 8 ms/gate when a single thread is used.

When each side utilizes two cores, our protocol evaluates circuits at the rate of 0.8 ms/gate; by comparison, the two-threaded execution in [18] achieved a rate of roughly 4 ms/gate. We do not gain a factor of 2 in performance by leveraging a second core in part because the parties are sometimes idle, and in part because of inter-thread interference (e.g., due to cache contention and dependence on shared hardware and I/O).

Our measured performance gains relative to [18] exceed the expected factor of 3. This may be due to differences in hardware or implementation, or the complexity of managing multiple threads in the implementation of [18] regardless of how many cores are being used.

The number of public-key operations used in our protocol scales linearly with the lengths of the inputs and outputs, though we stress again that in typical scenarios the number of gates is much larger than the number of inputs/outputs and so the overall performance impact of these public-key operations is small. Nevertheless, we measured performance of this aspect of our protocol as well. When each side uses a single core, our protocol processes inputs at the rate of 0.7 s/bit (our experiments assume the lengths of the parties' inputs are the same). Output is computed at the rate of 0.1 s/bit.