Fast Cut-and-Choose Based Protocols for Malicious and Covert Adversaries[⋆]

Yehuda Lindell

Dept. of Computer Science
Bar-Ilan University, Israel
lindell@biu.ac.il

Abstract. In the setting of secure two-party computation, two parties wish to securely compute a joint function of their private inputs, while revealing only the output. One of the primary techniques for achieving efficient secure two-party computation is that of Yao's garbled circuits (FOCS 1986). In the semi-honest model, where just one garbled circuit is constructed and evaluated, Yao's protocol has proven itself to be very efficient. However, a malicious adversary who constructs the garbled circuit may construct a garbling of a different circuit computing a different function, and this cannot be detected (due to the garbling). In order to solve this problem, many circuits are sent and some of them are opened to check that they are correct while the others are evaluated. This methodology, called *cut-and-choose*, introduces significant overhead, both in computation and in communication, and is mainly due to the number of circuits that must be used in order to prevent cheating.

In this paper, we present a cut-and-choose protocol for secure computation based on garbled circuits, with security in the presence of malicious adversaries, that vastly improves on all previous protocols of this type. Concretely, for a cheating probability of at most 2^{-40}, the best previous works send between 125 and 128 circuits. In contrast, in our protocol 40 circuits alone suffice (with some additional overhead). Asymptotically, we achieve a cheating probability of 2^{-s} where s is the number of garbled circuits, in contrast to the previous best of $2^{-0.32s}$. We achieve this by introducing a new cut-and-choose methodology with the property that in order to cheat, *all* of the evaluated circuits must be incorrect, and not just the *majority* as in previous works.

1 Introduction

Background. Protocols for secure two-party computation enable a pair of parties P_1 and P_2 with private inputs x and y, respectively, to compute a function f of their inputs while preserving a number of security properties. The most central of these properties are *privacy* (meaning that the parties learn the output $f(x,y)$ but nothing else), *correctness* (meaning that the output received is indeed $f(x,y)$ and not something else), and

[⋆] This work was funded by the European Research Council under the European Union's Seventh Framework Programme (FP/2007-2013) / ERC Grant Agreement n. 239868.

independence of inputs (meaning that neither party can choose its input as a function of the other party's input). The standard way of formalizing these security properties is to compare the output of a real protocol execution to an "ideal execution" in which the parties send their inputs to an incorruptible trusted party who computes the output for the parties. Informally speaking, a protocol is then secure if no real adversary attacking the real protocol can do more harm than an ideal adversary (or simulator) who interacts in the ideal model [12,10,24,2,4,11]. An important parameter when considering this problem relates to the power of the adversary. Three important models are the *semi-honest model* (where the adversary follows the protocol specification exactly but tries to learn more than it should by inspecting the protocol transcript), the *malicious model* (where the adversary can follow any arbitrary polynomial-time strategy), and the *covert model* (where the adversary may behave maliciously but is guaranteed to be caught with probability ϵ if it does [1]).

Efficient secure computation and Yao's garbled circuits. The problem of efficient secure computation has recently gained much interest. There are now a wide variety of protocols, achieving great efficiency in a variety of settings. These include protocols that require exponentiations for every gate in the circuit [28,16] (these can be reasonable for small circuits but not large ones with tens or hundreds of thousands of gates), protocols that use the "cut and choose" technique on garbled circuits [20,21,29,25], and more [27,14,15,6,18,3,26,7]. The recent protocols of [26,7] have very fast online running time. However, for the case of Boolean circuits and when counting the entire running time (and not just the online time), the method of cut-and-choose on garbled circuits is still the most efficient way of achieving security in the presence of covert and malicious adversaries.

Protocols for cut-and-choose on garbled circuits [20,21,29,25] all work in the following way. Party P_1 constructs a large number of garbled circuits and sends them to party P_2. Party P_2 then chooses a subset of the circuits which are opened and checked. If all of these circuits are correct, then the remaining circuits are evaluated as in Yao's protocol [30], and P_2 takes the *majority* output value as the output. The cut-and-choose approach forces P_1 to garble the *correct* circuit, since otherwise it will be caught cheating. However, it is important to note that even if all of the opened circuits are correct, it is not guaranteed that all of the unopened circuits are correct. This is due to the fact that if there are only a small number of incorrect circuits, then with reasonable probability these may not be chosen to be opened. For this reason, it is critical that P_2 outputs the majority output, since the probability that a majority of unopened circuits are incorrect when all opened circuits are correct is exponentially small in the number of circuits. We stress that it is not possible for P_2 to abort in case it receives different outputs in different circuits, even though in such a case it knows that P_1 cheated, because this opens the door to the following attack. A malicious P_1 can construct a single incorrect circuit that computes the following function: if the first bit of P_2's input equals 0 then output random garbage; else compute the correct function. Now, if this circuit is not opened (which happens with probability 1/2) and if the first bit of P_2's input equals 0, then P_2 will receive a different output in this circuit and in the others. In contrast, if the first bit of P_2's input equals 1 then it always receives the same output in all circuits. Thus, if the protocol instructs P_2 to abort if it receives different outputs, then P_1 will learn the first bit of P_2's input (based on whether or not P_2 aborts). By having P_2 take the majority value as output, P_1 can only cheat if the majority of the unopened circuits are incorrect, while all the opened ones are correct. In [21] it was shown that when s circuits are sent and half of them are opened, the probability that P_1 can cheat is at most $2^{-0.311s}$. Thus, concretely, in order to obtain an error probability of 2^{-40}, it is necessary to set $s = 128$ and so use 128 circuits, which means that the approximate cost of achieving security in the presence of malicious adversaries is 128 times the cost of achieving security in the presence of semi-honest adversaries. In [29], it was shown that by opening and checking 60% of the circuits instead of 50%, then the error becomes $2^{-0.32s}$ which means that it suffices to send 125 circuits in order to obtain a concrete error of 2^{-40}. It was claimed in [29] that these parameters are "optimal for the cut-and-choose method" and that they establish "a close characterization of the limit of the cut-and-choose method". We show that these protocols are actually far from the "limit" of this method.

Our results. In this paper, we present a novel twist on the cut-and-choose strategy used in [20,21,29,25] that enables us to achieve an error of just 2^{-s} with s circuits (and some small additional overhead). Concretely, this means that just 40 circuits are needed for error 2^{-40}. Our protocol is therefore much more efficient than previous protocols (there is some small additional overhead but this is greatly outweighed by the savings in the garbled circuits themselves unless the circuit being computed is small). We stress that the bottleneck in protocols of this type is the computation and communication of the s garbled circuits. This has been demonstrated in implementations. In [9], the cost of the circuit communication and computation for secure AES computation is approximately 80% of the work. Likewise in [17, Table 7] regarding secure AES computation, the bandwidth due to the circuits was 83% of all bandwidth and the time was over 50% of the time. On large circuits, as in the edit distance, this is even more significant with the circuit generation and evaluation taking 99.999% of the time [17, Table 9]. Thus, the reduction of this portion of the computation to a third of the cost is of great significance.

We present a high-level outline of our new technique in Section 2. For now, we remark that the cut-and-choose technique on Yao's garbled circuits introduces a number of challenges. For example, since the parties evaluate numerous circuits, it is necessary to enforce that the parties use the same input in all circuit computations. In addition, a selective input attack whereby P_1 provides correct garbled inputs only for a subset of the possible inputs of P_2 must be prevented (since otherwise P_2 will abort if its input is not in the subset because it cannot compute any circuit in this case, and thus P_1 will learn something about P_2's input based on whether or not it aborts). There are a number of different solutions to these problems that have been presented in [20,21,29,25,9]. The full protocol that we present here is based on the protocol of [21]. However, these solutions are rather "modular" (although this is meant in an informal sense), and can also be applied to our new technique; this is discussed at the end of Section 2. Understanding which technique is best will require implementation since they introduce tradeoffs that are not easily comparable. We leave this for future work, and focus on the main point of this work which is that it is possible to achieve error 2^{-s} with just s circuits. In Section 3.1 we present an exact efficiency count of our protocol.

Covert adversaries. Although not always explicitly proven, the known protocols for cut-and-choose on garbled circuits achieve covert security where the deterrent probability ϵ that the adversary is caught cheating equals 1 minus the statistical error of the protocol. That is, the protocol of [21] yields covert security of $\epsilon = 1 - 2^{-0.311s}$ (actually, a little better), and the protocol of [29] yields covert security with $\epsilon = 1-2^{-0.32s}$. Our protocol achieves covert security with deterrent $\epsilon = 1 - 2^{-s+1}$ (i.e., the error is 2^{-s+1}) which is far more efficient than all previous work. Specifically, in order to obtain $\epsilon = 0.99$, the number of circuits needed in [21] is 24. In contrast, with our protocol, it suffices to use 8 circuits. Furthermore, with just 11 circuits, we achieve $\epsilon = 0.999$, which is a high deterrent.

2 The New Technique and Protocol Outline

The idea behind our new cut-and-choose strategy is to design a protocol with the property that the party who constructs the circuits (P_1) can cheat if and only if *all* of the checked circuits are correct and *all* of the evaluated circuits are incorrect. Recall that in previous protocols, if the circuit evaluator (P_2) aborts if the evaluated circuits don't all give the same output, then this can reveal information about P_2's input to P_1. This results in an absurd situation: P_2 knows that P_1 is cheating but cannot do anything about it. In our protocol, we run an additional *small* secure computation after the cut-and-choose phase so that if P_2 catches P_1 cheating (namely, if P_2 receives inconsistent outputs) then in the second secure computation it learns P_1's full input x. This enables P_2 to locally compute the correct output $f(x, y)$ once again. Thus, it is no longer necessary for P_2 to take the majority output. Details follow.

Phase 1 – first cut-and-choose:

– Parties P_1 (with input x) and P_2 (with input y) essentially run a protocol based on cut-and-choose of garbled circuits, that is secure for malicious adversaries (like [21] or [29]). P_1 constructs just s circuits (for error 2^{-s}) and the strategy for choosing check or evaluation circuits is such that each circuit is *independently* chosen as a check or evaluation circuit with probability 1/2 (unlike all previous protocols where a fixed number of circuits are checked).
– If all of the circuits successfully evaluated by P_2 give the same output z, then P_2 locally stores z. Otherwise, P_2 stores a "proof" that it received two inconsistent output values in two different circuits. Such a proof could be having a garbled value associated with 0 on an output wire in one circuit, and a garbled value associated with 1 on the same output wire in a different circuit. (This is a proof since if P_2 obtains a single consistent output then the garbled values it receives on an output wire in different circuits are all associated with the same bit.)

Phase 2 – secure evaluation of cheating: P_1 and P_2 run a protocol that is secure for malicious adversaries with error 2^{-s} (e.g., they use the protocol of [21,29] with approximately $3s$ circuits), in order to compute the following:

– P_1 inputs the same input x as in the computation of phase 1 (and *proves* this).
– P_2 inputs random values if it received a single output z in phase 1, and inputs the proof of inconsistent output values otherwise.
– If P_2's input is a valid proof of inconsistent output values, then P_2 receives P_1's input x; otherwise, it receives nothing.

If this secure computation terminates with abort, then the parties abort.

Phase 3 – output determination: If P_2 received a single output z in phase 1 then it outputs z and halts. Otherwise, if it received inconsistent outputs then it received x in phase 2. P_2 locally computes $z = f(x, y)$ and outputs it. We stress that P_2 does not provide any indication as to whether z was received from phase 1 or locally computed.

Security. The argument for the security of the protocol is as follows. Consider first the case that P_1 is corrupted and so may not construct the garbled circuits correctly. If all of the check circuits are correct and all of the evaluation circuits are incorrect, then P_2 may receive the same incorrect output in phase 1 and will therefore output it. However, this can only happen if each incorrect circuit is an evaluation circuit and each correct circuit is a check circuit. Since each circuit is an evaluation or check circuit with probability exactly 1/2 this happens with probability exactly 2^{-s}. Next, if all of the evaluation circuits (that yield valid output) are correct, then the correct output will be obtained by P_2. This leaves the case that there are two different evaluation circuits that give two different outputs. However, in such a case, P_2 will obtain the required "proof of cheating" and so will learn x in the 2nd phase, thereby enabling it to still output the correct value. Since P_1 cannot determine which case yielded output for P_2, this can be easily simulated.

Next consider the case that P_2 is corrupted. In this case, the only way that P_2 can cheat is if it can provide output in the second phase that enables it to receive x. However, since P_1 constructs the circuits correctly, P_2 will not obtain inconsistent outputs and so will not be able to provide such a "proof". (We remark that the number of circuits s sent is used for the case that P_1 is corrupted; for the case that P_2 is corrupted a single circuit would actually suffice. Thus, there is no need to justify the use of fewer circuits than in previous protocols for this corruption case.)

Implementing phase 2. The main challenge in designing the protocol is phase 2. As we have hinted, we will use the knowledge of two different garbled values for a single output wire as a "proof" that P_2 received inconsistent outputs. However, it is also necessary to make sure that P_1 uses the same input in phase 1 and in phase 2; otherwise it could use x or x', respectively, and then learn whether P_2 received output via phase 1 or 2. The important observation is that all known protocols already have a mechanism for ensuring that P_1 uses the same input in all computed circuits, and this mechanism can be used for the circuits in phase 1 and 2, since it does not depend on the circuits being computed being the same.

Another issue that arises is the efficiency of the computation in phase 2. In order to make the circuit for phase 2 small, it is necessary to construct all of the output wires in all the circuits of phase 1 so that they have the same garbled values on the output wires. This in turn makes it necessary to open and check the circuits only after phase 2 (since opening a circuit to check it reveals both garbled values on an output wire which means that this knowledge can no longer be a proof that P_1 cheated). Thus, the structure of the actual protocol is more complex than previous protocols; however, this relates only to its description and not efficiency.

We remark that we use the method of [21] in order to prove the consistency of P_1's input in the different circuits and between phase 1 and phase 2. However, we believe that the methods used in [29,25], for example, would also work, but have not proven this.

3 The Protocol

Preliminaries – modified batch single-choice cut-and-choose OT. The cut-and-choose OT primitive was introduced in [21]. Intuitively, a cut-and-choose OT is a series of 1-out-of-2 oblivious transfers with the special property that in some of the transfers the receiver obtains a single value (as in regular oblivious transfer), while in the others the receiver obtains both values. For cut-and-choose on Yao's garbled circuits, the functionality is used for the receiver to obtain all garbled input values in the circuits that it wishes to open and check, and to obtain only the garbled input values associated with its input on the circuits to be evaluated.

In [21], the functionality defined is such that the receiver obtains both values in exactly half of the transfers; this is because in [21] exactly half of the circuits are opened. In this work, we modify the functionality so that the receiver can choose at its own will in which transfers it receives just one value and in which it receives both. We do this since we want P_2 to check each circuit with probability exactly 1/2, independently of all other circuits. This yields an error of 2^{-s} instead of $\binom{s}{s/2}^{-1}$, which is smaller (this is especially significant in the setting of covert adversaries).

This modification introduces a problem since at a later stage in the protocol the receiver needs to prove to the sender for which transfers it received both values and for which it received only one. If it is known that the receiver obtains both values in exactly half of the transfers, or for any other known number, then the receiver can just send both values in these transfers (assuming that they are otherwise unknown, as is the case in the Yao circuit use of the functionality), and the sender knows that the receiver did *not* obtain both values in all others; this is what is done in [21]. However, here the receiver can obtain both values in an unknown number of transfers, as it desires. We therefore need to introduce a mechanism enabling the receiver to prove to the sender in which transfers it did not receive both values, in a way that it cannot cheat. We solve this by having the sender input s random "check" values, and having the receiver obtain such a value in every transfer for which it receives a single value only. Thus, at a later time, the receiver can send the appropriate check values, and this constitutes a proof that it did not receive both values in these transfers. See Figure 1 for the formal functionality definition.

FIGURE 1 (Modified Batch Single-Choice Cut-and-Choose OT $\mathcal{F}_{\text{ccot}}$)

Inputs:

- S inputs ℓ *vectors* of pairs $\boldsymbol{x}_i$ of length s, for $i = 1, \ldots, \ell$. (Every vector consists of s pairs; i.e., $\boldsymbol{x}_i = \langle (x_0^{i,1}, x_1^{i,1}), (x_0^{i,2}, x_1^{i,2}), \ldots, (x_0^{i,s}, x_1^{i,s}) \rangle$. There are ℓ such vectors.) In addition, S inputs s "check values" $\chi_1, \ldots, \chi_s$. All values are in $\{0,1\}^n$.
- R inputs $\sigma_1, \ldots \sigma_\ell \in \{0,1\}$ and a set of indices $\mathcal{J} \subseteq [s]$.

Output: The sender receives no output. The receiver obtains the following:

- For every $i = 1, \ldots, \ell$ and for every $j \in \mathcal{J}$, the receiver R obtains the jth pair in vector $\boldsymbol{x}_i$. (I.e., for every $i = 1, \ldots, \ell$ and every $j \in \mathcal{J}$, R obtains $(x_0^{i,j}, x_1^{i,j})$.)
- For every $i = 1, \ldots, \ell$, the receiver R obtains the σ_i value in every pair of the vector $\boldsymbol{x}_i$. (I.e., for every $i = 1, \ldots, \ell$, R obtains $\langle x_{\sigma_i}^{i,1}, x_{\sigma_i}^{i,2}, \ldots, x_{\sigma_i}^{i,s} \rangle$.)
- For every $k \notin \mathcal{J}$, the receiver R obtains χ_k.

A protocol for securely computing the $\mathcal{F}_{\text{ccot}}$ functionality, that is based on the protocol in [21], is provided in the full version of this paper [22]. The computational complexity of the protocol is as follows:

Operation	**Exact Cost**	**Approximate Cost**
Regular exponentiations	$1.5s\ell + 18.5s + 25$	$1.5s\ell$
Fixed-base exponentiations	$9s\ell + \ell + 2s + 1$	$9s\ell$
Bandwidth (group elements)	$5s\ell + \ell + 11s + 15$	$5s\ell$

Encoded translation tables. We modify the output translation tables typically used in Yao's garbled circuits as follows. Let k_i^0, k_i^1 be the garbled values on wire i, which is an output wire, and let H be a collision-resistant hash function. Then, the encoded output translation table for this wire is simply $\left[H(k_i^0), H(k_i^1)\right]$. We require that $k_i^0 \neq k_i^1$ and if this doesn't hold (which will be evident since then $H(k_i^0) = H(k_i^1)$), P_2 will automatically abort. Observe that given a garbled value k, it is possible to determine whether k is the 0 or 1 key (or possibly neither) by just computing $H(k)$ and seeing if it equals the first or second value in the pair, or neither.

However, given the encoded translation table, it is not feasible to find the actual garbled values, since this is equivalent to inverting the one-way function. This is needed in our protocol, as we will see below. We remark that both k_i^0, k_i^1 are revealed by the end of the protocol, and only need to remain secret until Step 7 has concluded (see the protocol below). Thus, they can be relatively short values.

PROTOCOL 2 (Computing $f(x,y)$)

Inputs: P_1 has input $x \in \{0,1\}^\ell$ and P_2 has input $y \in \{0,1\}^\ell$.

Auxiliary input: a statistical security parameter s, the description of a circuit C such that $C(x,y) = f(x,y)$, and $(\mathbb{G}, q, g)$ where $\mathbb{G}$ is a cyclic group with generator g and prime order q, and q is of length n. In addition, they hold a hash function H that is a suitable randomness extractor; see [8].
Specified output: Party P_2 receives $f(x,y)$ and party P_1 receives no output; denote the length of the output of $f(x,y)$ by m.

The protocol:

1. Input key choice and circuit preparation:
   (a) P_1 chooses random values $a_1^0, a_1^1, \ldots, a_\ell^0, a_\ell^1; r_1, \ldots, r_s \in_R \mathbb{Z}_q$ and $b_1^0, b_1^1, \ldots, b_m^0, b_m^1 \in_R \{0,1\}^n$.
   (b) Let $w_1, \ldots, w_\ell$ be the input wires corresponding to P_1's input in C, and denote by $w_{i,j}$ the instance of wire w_i in the jth garbled circuit, and by $k_{i,j}^b$ the key associated with bit b on wire $w_{i,j}$. P_1 sets the keys for its input wires to:
$$k_{i,j}^0 = H(g^{a_i^0 \cdot r_j}) \text{ and } k_{i,j}^1 = H(g^{a_i^1 \cdot r_j}).$$
   (c) Let $w'_1, \ldots, w'_m$ be the output wires in C. Then, the keys for wire w'_i in *all* garbled circuits are b_i^0 and b_i^1 (unlike all other wires in the circuit, the same values are used for the output wires in all circuits).
   (d) P_1 constructs s independent copies of a garbled circuit of C, denoted $GC_1, \ldots, GC_s$, using random keys except for wires $w_1, \ldots, w_\ell$ (P_1's input wires) and $w'_1, \ldots, w'_m$ (the output wires) which are as above.
2. Oblivious transfers: P_1 and P_2 run a modified batch single-choice cut-and-choose oblivious transfer, with parameters ℓ (the number of parallel executions) and s (the number of pairs in each execution):
   (a) P_1 defines vectors $\boldsymbol{z}_1, \ldots \boldsymbol{z}_\ell$ so that $\boldsymbol{z}_i$ contains the s pairs of random symmetric keys associated with P_2's ith input bit y_i in all garbled circuits $GC_1, \ldots, GC_s$. P_1 also chooses random values $\chi_1, \ldots, \chi_s \in_R \{0,1\}^n$. P_1 inputs these vectors and the $\chi_1, \ldots, \chi_s$ values.
   (b) P_2 chooses a random subset $\mathcal{J} \subset [s]$ where every $j \in \mathcal{J}$ with probability exactly 1/2, under the constraint that $\mathcal{J} \neq [s]$. P_2 inputs the set $\mathcal{J}$ and bits $\sigma_1, \ldots, \sigma_\ell \in \{0,1\}$, where $\sigma_i = y_i$ for every i.
   (c) P_2 receives all the keys associated with its input wires in all circuits GC_j for $j \in \mathcal{J}$, and receives the keys associated with its input y on its input wires in all other circuits.
   (d) P_2 receives χ_j for every $j \notin \mathcal{J}$.

PROTOCOL 3 (PROTOCOL 2 – continued)

3. Send circuits and commitments: P_1 sends P_2 the garbled circuits (i.e., the garbled gates). In addition, P_1 sends P_2 the "seed" for the randomness extractor H, and the following displayed values (which constitute a "commitment" to the garbled values associated with P_1's input wires):
$$\left\{(i,0,g^{a_i^0}),(i,1,g^{a_i^1})\right\}_{i=1}^{\ell} \quad \text{and} \quad \left\{(j,g^{r_j})\right\}_{j=1}^{s}$$
In addition, P_1 sends P_2 the encoded output translation tables, as follows:
$$\left[\left(H(b_1^0),H(b_1^1)\right),\ldots,\left(H(b_m^0),H(b_m^1)\right)\right].$$
If $H(b_i^0)=H(b_i^1)$ for any $1\le i\le m$, then P_2 aborts.
4. Send cut-and-choose challenge: P_2 sends P_1 the set $\mathcal{J}$ along with the values χ_j for every $j\notin\mathcal{J}$. If the values received by P_1 are incorrect, it outputs $\perp$ and aborts. Circuits GC_j for $j\in\mathcal{J}$ are called check-circuits, and for $j\notin\mathcal{J}$ are called evaluation-circuits.
5. P_1 sends its garbled input values in the evaluation-circuits: P_1 sends the keys associated with its inputs in the evaluation circuits: For every $j\notin\mathcal{J}$ and every wire $i=1,\ldots,\ell$, party P_1 sends the value $k'_{i,j}=g^{a_i^{x_i}\cdot r_j}$; P_2 sets $k_{i,j}=H(k'_{i,j})$.
6. Circuit evaluation: P_2 uses the keys associated with P_1's input obtained in Step 5 and the keys associated with its own input obtained in Step 2c to evaluate the circuits GC_j for every $j\notin\mathcal{J}$. If P_2 receives only one valid output value per output wire (i.e., one of b_i^0,b_i^1, verified against the encoded output translation tables) and it does not abort in the next step, then this will be its output. If P_2 receives two valid outputs on one output wire (i.e., both b_i^0 and b_i^1 for output wire w'_i) then it uses these in the next step. If there exists an output wire for which P_2 did not receive a valid value in *any* evaluation circuit (neither b_i^0 nor b_i^1), then P_2 aborts.
7. Run secure computation to detect cheating:
   (a) P_1 defines a circuit with the values $b_1^0,b_1^1,\ldots,b_m^0,b_m^1$ hardcoded. The circuit computes the following function:
      i. P_1's input is a string $x\in\{0,1\}^\ell$, and it has no output.
      ii. P_2's input is a pair of values b_0,b_1.
      iii. If there exists a value i $(1\le i\le m)$ such that $b_0=b_i^0$ *and* $b_1=b_i^1$, then P_2's output is x; otherwise it receives no output.
   (b) P_1 and P_2 run the protocol of [21] on this circuit (except for the proof of P_1's input values), as follows:
      i. P_1 inputs its input x; If P_2 received b_i^0,b_1^i for some $1\le i\le m$, then it inputs the pair b_i^0,b_i^1; otherwise it inputs garbage.
      ii. The garbled circuit constructed by P_1 uses the same a_i^0,a_i^1 values as above (i.e., the same triples $(i,0,g^{a_i^0}),(i,1,g^{a_i^1})$), but independent r_j values. In addition, regular translation tables are used, and not encoded translation tables. Finally, the parties use $3s$ copies of the circuit (and not s).
      iii. P_2 takes the majority output from the evaluation circuits, as in [21]. If any of the checked circuits are invalid, then P_2 aborts. We stress that this check includes the check that the circuit has the correct $b_1^0,b_1^1,\ldots,b_m^0,b_m^1$ values hardcoded; P_2 checks this relative to the encoded translation tables that it received.

PROTOCOL 2 – continued

7. Run secure computation to detect cheating (cont.): If this computation results in an abort, then both parties halt at this point and output $\perp$. (Note that in the protocol of [21] both parties must know the circuit. However, the oblivious transfers that determine P_2's input are run before the circuit is sent and checked. Thus, P_1 can send the $b_1^0, b_1^1, \ldots, b_m^0, b_m^1$ values to P_2 after the oblivious transfers are concluded; P_2 can check these values against the encoded translation tables and can then check that these are the values that are hardwired into the circuit.)
8. Check circuits for computing $f(x, y)$:
   (a) Send all input garbled values in check-circuits: For every *check-circuit* GC_j, party P_1 sends the value r_j to P_2, and P_2 checks that these are consistent with the pairs $\{(j, g^{r_j})\}_{j \in \mathcal{J}}$ received in Step 3. If not, P_2 aborts outputting $\perp$.
   (b) Correctness of check circuits: For every $j \in \mathcal{J}$, P_2 uses the $g^{a_i^0}, g^{a_i^1}$ values it received in Step 3, and the r_j values it received in Step 8a, to compute the values $k_{i,j}^0 = H(g^{a_i^0 \cdot r_j}), k_{i,j}^1 = H(g^{a_i^1 \cdot r_j})$ associated with P_1's input in GC_j. In addition it sets the garbled values associated with its own input in GC_j to be as obtained in the cut-and-choose OT. Given all the garbled values for all input wires in GC_j, party P_2 decrypts the circuit and verifies that it is a garbled version of C, using the encoded translation tables for the output values. If there exists a circuit for which this does not hold, then P_2 aborts and outputs $\perp$.
9. Verify consistency of P_1's input: Let $\hat{\mathcal{J}}$ be the set of check circuits in the computation in Step 7, and let $\hat{r}_j$ be the value used to generate the keys associated with P_1's input in the jth circuit, just like r_j in Step 1a (i.e., $H(g^{a_i^0 \cdot \hat{r}_j})$ is the 0-key on the ith input wire of P_1 in the jth garbled circuit used in Step 7). Let $\hat{k}_{i,j}$ be the analogous value of $k'_{i,j}$ in Step 5 received by P_2 in the computation in Step 7.
   For every input wire $i = 1, \ldots, \ell$, party P_1 proves a zero-knowledge proof of knowledge that there exists a $\sigma_i \in \{0, 1\}$ such that for every $j \notin \mathcal{J}$ and every $j' \notin \mathcal{J}'$, $k'_{i,j} = g^{a_i^{\sigma_i} \cdot r_j}$ AND $\hat{k}_{i,j} = g^{a_i^{\sigma_i} \cdot \hat{r}_j}$ (note that P_2 has g^{r_j} and $g^{\hat{r}_j}$ for every j, and $g^{a_i^0}, g^{a_i^1}$ for every i; thus this is just a Diffie-Hellman tuple proof). If any of the proofs fail, then P_2 aborts and outputs $\perp$.
10. Output evaluation: If P_2 received no inconsistent outputs from the evaluation circuits GC_i ($i \notin \mathcal{J}$), then it decodes the outputs it received using the encoded translation tables, and outputs the string received. If P_2 received inconsistent output, then let x be the output that P_2 received from the second computation in Step 7. Then, P_2 computes $f(x, y)$ and outputs it.

The circuit for step 7. A naive circuit for computing the function in Step 7 can be quite large. Specifically, to compare two n bit strings requires $2n$ XORs followed by $2n$ ORs; if the output is 0 then the strings are equal. This has to be repeated m times, once for every i, and then the results have to be ORed. Thus, there are $2mn + m$ non-XOR gates. Assuming n is of size 80 (e.g., which suffices for the output values) and m is of size 128, this requires $20,480$ non-XOR gates, which is very large. An alternative is therefore to compute the following garbled circuit:

1. For every $i = 1, \ldots, m$,
   (a) Compare $b_0 \| b_1$ to $b_i^0 \| b_i^1$ (where '$\|$' denotes concatenation) by XOR-ing bit-by-bit, and take the NOT of each bit. This is done as in a regular garbled circuit; by combining the NOT together with the XOR this has the same cost as a single XOR gate.
   (b) Compute the $2n$-wise AND of the bits from above. Instead of using $2n - 1$ Boolean AND gates, this can be achieved by encrypting the 1-key on the output wire under all n keys (together with redundancy so that the circuit evaluator can know if it received the correct value). Furthermore, this encryption can be a "one-time pad" and thus is just the XOR of all of the 1-keys on the input wires together with the 1-key on the output way. The 0-key for the output can be given in the clear, since it provides no additional information, but is not needed so can just not be given (note that P_2 knows exactly which case it is in). Note that the result of this operation is 1 if and only if $b_0 \| b_1 = b_i^0 \| b_i^1$ and so P_2 had both keys on the ith output wire.
2. Compute the OR of the m bits resulting from the above loop. Instead of using $m - 1$ Boolean OR gates, this can be achieved by simply setting the 1-key on all of the output wires from the n-wise ANDs above to be the 1-key on the output wire of the OR. This ensures that as soon as the 1-key is received from an n-wise AND, the 1-key is received from the OR, as required. (This reveals for which i the result of the n-wise AND was 1. However, this is fine here since P_2 knows exactly where equality should be obtained in any case.)
3. Compute the AND of the output from the previous step with all of the input bits of P_1. This requires ℓ Boolean AND gates.
4. The output wires include the output of the OR (so that P_2 can know if it received x or nothing), together with the output of all of the ANDs with the input bits of P_1.

The original and optimized circuits are depicted in the full version [22]. The number of non-XOR operations required to securely compute this circuit is just ℓ binary AND gates. Assuming $\ell = 128$ (e.g., as in the secure AES example), we have that there are only 128 non-XOR gates. When using 128 circuits as in our instantiation of Step 7 via [21], this comes to 16,384 garbled gates *overall*, which is significant but not too large. We stress that the size of this circuit is independent of the size of the circuit for the function f to be computed. Thus, this becomes less significant as the circuit becomes larger. On the other hand, for very small circuits or when the input size is large relative to the overall circuit size, our approach will not be competitive. To be exact, assume a garbled circuit approach that requires $3s$ circuits. If $3s|C| < s|C| + 3s \cdot \ell$ then our protocol will be slower (since the cost of our protocol is $s|C|$ for the main computation plus $3s\ell$ for the circuit of Step 7, in contrast to $3s|C|$ for the other protocol). This implies that our protocol will be faster as long as $|C| > \frac{3\ell}{2}$. Concretely, if $\ell = 128$ and $s = 40$, it follows that our protocol will be faster as long as $|C| > 192$. Thus, our protocol is much faster, except for the case of very small circuits.

Additional optimizations. Observe that although the above circuit is very small, P_2's input size is $2n$ and this is quite large. Since the input size has a significant effect on the cost of the protocol (especially when using cut-and-choose oblivious transfer), it would be desirable to reduce this. This can be achieved by first having P_2 input $b_0 \oplus b_1$ instead of $b_0 \| b_1$, reducing the input length to n (this is sound since if P_2 does not have both keys on any output wire then it cannot know their XOR). Furthermore, in order to obtain a cheating probability of 2^{-40} it suffices for the circuit to check only the first 40 bits of $b_0 \oplus b_1$. (Note that b_0^i and b_1^i have to be longer since $H(b_i^0), H(b_i^1)$ are published; nevertheless, only 40 bits need to be included in the circuit. When using this optimization, the length of b_0^i, b_1^i can be 128 bits and not 80, which is preferable.) Finally, by choosing all of the b_i^0, b_i^1 values so that they have the same fixed XOR (i.e., for some Δ it holds that for all i, $b_i^0 \oplus b_i^1 = \Delta$, as in the free XOR technique), the size of the circuit is further reduced. This significantly reduces the bandwidth; a diagram of this circuit appears in the full version [22].

Security. In the full version of this paper [22], we prove the following theorem:

Theorem 4. *Assume that the Decisional Diffie-Hellman assumption holds in $(\mathbb{G}, g, q)$, and that H is a collision-resistant hash function. Then, Protocol 2 securely computes f in the presence of malicious adversaries (with error $2^{-s} + \mu(n)$ where $\mu(\cdot)$ is some negligible function).*

3.1 A Detailed Efficiency Count and Comparison

In this section we provide an exact efficiency count of our protocol. This will enable an exact comparison of our protocol to previous and future works, as long as they also provide an exact efficiency count. We count exponentiations, symmetric encryptions and bandwidth. We let n denote the length of a symmetric encryption, and an arbitrary string of length of the security parameter (e.g., χ_j).

Step	Fixed-base exponent.	Regular exponent.	Symmetric Encryptions	Group elms sent	Symmetric comm
1	$2s\ell$	0	$4s\|C\|$		
2	$9s\ell$	$1.5s\ell$		$5s\ell$	
3	$\ell + s$	0		$2\ell + s$	$4ns\|C\|$
4					$\frac{s}{2} \cdot n$
5					nm
6			$\frac{s}{2} \cdot \|C\|$		
7	$9s\ell + 5040s$	$480s$	19.5ℓ	$21s\ell$	$12sn\ell$
8	$s/2 + s\ell$		$\frac{s}{2} \cdot 4\|C\|$		$\frac{s}{2} \cdot n$
9		$2s\ell + 18\ell$		10	$2s\ell n$
TOTAL	$\mathbf{21s\ell + 5040s}$	$\mathbf{3.5s\ell + 18\ell + 480s}$	$\mathbf{6.5s\|C\| + 19.5s\ell}$	$\mathbf{26s\ell}$	$\mathbf{4ns\|C\| + 14s\ell n}$

The number of symmetric encryptions is counted as follows: each circuit requires $4|C|$ symmetric encryptions to construct, $4|C|$ symmetric encryption to check, and $|C|$ encryptions to evaluate (we assume a single encryption per entry; if standard double-encryption is used then this should be doubled). Since approximately half of the circuits are check and half are evaluation, the garbling, checking and evaluation of the main garbled circuit accounts for approximately $s \cdot 4|C| + \frac{s}{2} \cdot 4|C| + \frac{s}{2} \cdot |C| = 6.5s|C|$ symmetric encryptions. The garbled circuit used in Step 7 has ℓ non-XOR gates and so the same analysis applies on this size. However, the number of circuits sent in this step is $3s$ and thus we obtain an additional $3 \times 6.5 \cdot s \cdot \ell = 19.5s\ell$.

The bandwidth count for Step 7 is computed based on the counts provided in [21], using $3s$ circuits. The cost of the exponentiations is based on the fact that in [21], if P_1 has input of length ℓ_1 and P_2 has input of length ℓ_2, and s' circuits are used, then there are $3.5s'\ell_1 + 10.5s'\ell_2$ fixed-base exponentiations and $s'\ell_2$ regular exponentiations. However, $0.5s'\ell_1$ of the fixed-base exponentiations are for the proof of consistency and these are counted in Step 9 instead. Now, in Step 7, P_1's input length is ℓ (it is the same x as for the entire protocol) and P_2's input is comprised of two garbled values for the output wires. Since these must remain secret for only a short amount of time, it is possible to take 80-bit values only and so P_2's input length is 160 bits (this is irrespective of P_2's input length to the function f). Taking $s' = 3s$ and plugging these lengths this into the above, we obtain the count appearing in the table.

The proof of consistency of P_1's input is carried out ℓ times (once for each bit of P_1's input) and over $s + 3s = 4s$ circuits (since there are s circuits for the main computation of C, plus another $3s$ circuits for the computation in Step 7). By the count in [21], this proof therefore costs $\frac{4s\ell}{2} + 18\ell$ exponentiations, and bandwidth of 10 group elements and another $8s\ell$ short strings (this can therefore be counted as $2s\ell n$.

A comparison to [21]. In order to to get a concrete understanding of the efficiency improvement, we will compare the cost to [21] for the AES circuit of size 6,800 gates [31], and input and output sizes of 128. Now, as we have mentioned, the overall cost of the protocol of [21] is $3.5s'\ell_1 + 10.5s\ell_2$ fixed-base exponentiations, $s'\ell_2$ regular exponentiations and $6.5s'|C|$ symmetric encryptions. In this case, $\ell_1 = \ell_2 = 128$, $s' = 125$ ($s' = 125$ was shown to suffice for 2^{-40} security in [17]), and so we have that the cost is 224,000 fixed-base exponentiations, 16,000 regular exponentiations, and $812.5|C|$ =5,525,000 symmetric encryptions. In contrast, taking $\ell = 128$ and $s = 40$ we obtain here 309,120 fixed-base exponentiations, $37,120$ regular exponentiations, and 1,874,800 symmetric encryptions. In addition, the bandwidth of [21] is approximately $112,000$ group elements and 3,400,000 symmetric ciphertexts. At the minimal cost of 220 bits per group element (e.g., using point compression) and 128 bits per ciphertext, we have that this would come to approximately 449,640,000 bits, or close to half a gigabyte (in practice, it would be significantly larger due to communication overheads). In contrast, the bandwidth of our protocol for this circuit would be 133,120 group elements and 1,159,680 ciphertexts. With the same parameters as above, this would be approximately 177,725,440 bits, which is under 40% of the cost of [21]. This is very significant since bandwidth is turning out to be the bottleneck in many cases.

Protocol	**Fixed-base exp.**	**Regular exp.**	**Symmetric encryptions**	**Bandwidth**
[21]	224,000	16,000	5,525,000	449,640,000
Here	309,120	37,120	1,874,800	177,725,440

Fig. 1. Comparison of protocols for secure computation of AES

We stress that in larger circuits the difference would be even more striking.

4 Variants – Universal Composability and Covert Adversaries

Universal composability [5]. As in [21], by instantiating the cut-and-choose oblivious transfer and the zero-knowledge proofs with variants that universally composable, the result is that Protocol 2 is universally composable.

Covert adversaries [1]. Observe that in the case that P_2 is corrupted, the protocol is fully secure irrespective of the value of s used. In contrast, when P_1 is corrupted, then the cheating probability is $2^{-s}+\mu(n)$. However, this cheating probability is *independent* of the input used by the P_2 (as shown in the proof of Theorem 4). Thus, Protocol 2 is suitable for the model of security in the presence of *covert adversaries*. Intuitively, since the adversary can cheat with probability only 2^{-s} and otherwise it is caught cheating, the protocol achieves covert security with deterrent $\epsilon = 1 - 2^{-s}$. However, on closer inspection, this is incorrect. Specifically, as we have discussed above, if P_2 catches P_1 cheating due to the fact that two different circuits yield two different outputs, then it is not allowed to reveal this fact to P_1. Thus, P_2 cannot declare that P_1 is a cheat in this case, as is required in the model of covert adversaries. However, if P_2 detects even a single bad circuit in the check phase, then it can declare that P_1 is cheating, and this happens with probability at least 1/2 (even if only a single circuit is bad). We can use this to show that for every s, Protocol 2 securely computes f in the presence of covert adversaries **with deterrent $\boldsymbol{\epsilon = 1 - 2^{-s+1}}$**. In actuality, we need to make a slight change to Protocol 2, in order to achieve this. See the full version of this paper for details [22].

As discussed in the introduction, this yields a huge efficiency improvement over previous results, especially for small values of s. For example, 100 circuits are needed to obtain $\epsilon = 0.99$ in [1], 24 circuits are needed to obtain $\epsilon = 0.99$ in [21], and here ***8 circuits alone*** suffice to obtain $\epsilon = 0.99$. Observe that when covert security is desired, the number of circuits sent in Step 7 needs to match the level of covert security. For example, in order to obtain $\epsilon = 0.99$, 8 circuits are used in our main protocol and 24 circuits are used in Step 7.

We remark that our protocol would be a little simpler if P_2 always asked to open exactly half the circuits (especially in the cut-and-choose oblivious transfer). In this case, the error would be $\binom{s}{s/2}^{-1}$ instead of 2^{-s}. In order to achieve an error of 2^{-40} this would require 44 circuits which is a 10% increase in complexity, and reason enough to use our strategy of opening each circuit independently with probability 1/2. However, when considering covert security, the difference is huge. For example, with $s = 8$ we have that $\binom{8}{4}^{-1} = 1/70$ whereas $2^{-8} = 1/256$. This is a very big difference.

Acknowledgements

We thank Benny Pinkas and Ben Riva for helpful discussions.

References

1. Y. Aumann and Y. Lindell. Security Against Covert Adversaries: Efficient Protocols for Realistic Adversaries. In the *Journal of Cryptology*, 23(2):281–343, 2010 (extended abstract at *TCC 2007*).
2. D. Beaver. Foundations of Secure Interactive Computing. In *CRYPTO'91*, Springer-Verlag (LNCS 576), pages 377–391, 1991.
3. R. Bendlin, I. Damgård, C. Orlandi and S. Zakarias. Semi-homomorphic Encryption and Multiparty Computation. In *EUROCRYPT 2011*, Springer (LNCS 6632), pages 169–188, 2011.
4. R. Canetti. Security and Composition of Multiparty Cryptographic Protocols. *Journal of Cryptology*, 13(1):143–202, 2000.
5. R. Canetti. Universally Composable Security: A New Paradigm for Cryptographic Protocols. In 42*nd FOCS*, pages 136–145, 2001. Full version available at `http://eprint.iacr.org/2000/067`.
6. I. Damgård and C. Orlandi. Multiparty Computation for Dishonest Majority: From Passive to Active Security at Low Cost. In *CRYPTO 2010*, Springer (LNCS 6223), pages 558–576, 2010.
7. I. Damgård, V. Pastro, N.P. Smart and S. Zakarias. Multiparty Computation from Somewhat Homomorphic Encryption. In *CRYPTO 2012*, Springer (LNCS 7417), pages 643–662, 2012.
8. Y. Dodis, R. Gennaro, J. Hastad, H. Krawczyk and T. Rabin. Randomness Extraction and Key Derivation Using the CBC, Cascade and HMAC Modes. In *CRYPTO 2004*, Springer (LNCS 3152), pages 494–510, 2004.
9. T.K. Frederiksen and J.B. Nielsen. Fast and Maliciously Secure Two-Party Computation Using the GPU. *Cryptology ePrint Archive: Report 2013/046*, 2013.
10. S. Goldwasser and L. Levin. Fair Computation of General Functions in Presence of Immoral Majority. In *CRYPTO'90,* Springer-Verlag (LNCS 537), pages 77–93, 1990.
11. O. Goldreich. *Foundations of Cryptography: Volume 2 – Basic Applications.* Cambridge University Press, 2004.
12. O. Goldreich, S. Micali and A. Wigderson. How to Play any Mental Game – A Completeness Theorem for Protocols with Honest Majority. In 19*th STOC,* pages 218–229, 1987.

13. C. Hazay and Y. Lindell. *Efficient Secure Two-Party Protocols: Techniques and Constructions.* Springer, November 2010.
14. Y. Ishai, M. Prabhakaran and A. Sahai. Founding Cryptography on Oblivious Transfer – Efficiently. In *CRYPTO 2008*, Springer (LNCS 5157), pages 572–591, 2008.
15. Y. Ishai, M. Prabhakaran and A. Sahai. Secure Arithmetic Computation with No Honest Majority. In *TCC 2009*, Springer (LNCS 5444), pages 294–314, 2009.
16. S. Jarecki and V. Shmatikov. Efficient Two-Party Secure Computation on Committed Inputs. In *EUROCRYPT 2007*, Springer (LNCS 4515), pages 97–114, 2007.
17. B. Kreuter, A.Shelat, and C. Shen. Billion-Gate Secure Computation with Malicious Adversaries. In the 21*st USENIX Security Symposium*, 2012.
18. Y. Lindell, E. Oxman and B. Pinkas. The IPS Compiler: Optimizations, Variants and Concrete Efficiency. In *CRYPTO 2011*, Springer (LNCS 6841), pages 259–276, 2011.
19. Y. Lindell and B. Pinkas. A Proof of Yao's Protocol for Secure Two-Party Computation. In the *Journal of Cryptology*, 22(2):161–188, 2009.
20. Y. Lindell and B. Pinkas. An Efficient Protocol for Secure Two-Party Computation in the Presence of Malicious Adversaries. In *EUROCRYPT 2007*, Springer (LNCS 4515), pages 52–78, 2007.
21. Y. Lindell and B. Pinkas. Secure Two-Party Computation via Cut-and-Choose Oblivious Transfer. In *TCC 2011*, Springer (LNCS 6597), pages 329–346, 2011.
22. Y. Lindell. Fast Cut-and-Choose Based Protocols for Malicious and Covert Adversaries. *Cryptology ePrint Archive: Report 2013/079*, 2013.
23. A.J. Menezes, P.C. van Oorschot and S.A. Vanstone. *Handbook of Applied Cryptography*, CRC Press, 2001.
24. S. Micali and P. Rogaway. Secure Computation. Unpublished manuscript, 1992. Preliminary version in *CRYPTO'91*, Springer-Verlag (LNCS 576), pages 392–404, 1991.
25. P. Mohassel and B. Riva. Garbled Circuits Checking Garbled Circuits: More Efficient and Secure Two-Party Computation. *Cryptology ePrint Archive*, Report 2013/051, 2013.
26. J.B. Nielsen, P.S. Nordholt, C. Orlandi and S.Sheshank Burra. A New Approach to Practical Active-Secure Two-Party Computation. In *CRYPTO 2012*, Springer (LNCS 7417), pages 681–700, 2012.
27. J.B. Nielsen and C. Orlandi. LEGO for Two-Party Secure Computation. In *TCC 2009*, Springer (LNCS 5444), pages 368–386, 2009.
28. B. Schoenmakers and P. Tuyls. Practical Two-Party Computation Based on the Conditional Gate. In *ASIACRYPT 2004*, Springer (LNCS 3329), pages 119–136, 2004.
29. A. Shelat, C.H. Shen. Two-Output Secure Computation with Malicious Adversaries. In *EUROCRYPT 2011*, Springer (LNCS 6632), pages 386–405, 2011.
30. A. Yao. How to Generate and Exchange Secrets. In 27*th FOCS*, pages 162–167, 1986. See [19] for details.
31. Bristol Cryptography Group. Circuits of Basic Functions Suitable For MPC and FHE. `http://www.cs.bris.ac.uk/Research/CryptographySecurity/MPC/`.