Amortizing Garbled Circuits

Yan Huang[1], Jonathan Katz[1], Vladimir Kolesnikov[2], Ranjit Kumaresan[3], and Alex J. Malozemoff[1]

[1] University of Maryland
{yhuang,jkatz,amaloz}@cs.umd.edu
[2] Bell Labs
kolesnikov@research.bell-labs.com
[3] Technion
ranjit@cs.technion.ac.il

Abstract. We consider secure two-party computation in a *multiple-execution* setting, where two parties wish to securely evaluate the same circuit multiple times. We design efficient garbled-circuit-based two-party protocols secure against *malicious* adversaries. Recent works by Lindell (Crypto 2013) and Huang-Katz-Evans (Crypto 2013) have obtained optimal complexity for cut-and-choose performed over garbled circuits in the single execution setting. We show that it is possible to obtain much lower *amortized* overhead for cut-and-choose in the multiple-execution setting.

Our efficiency improvements result from a novel way to combine a recent technique of Lindell (Crypto 2013) with LEGO-based cut-and-choose techniques (TCC 2009, Eurocrypt 2013). In concrete terms, for 40-bit statistical security we obtain a $2\times$ improvement (per execution) in communication and computation for as few as 7 executions, and require only 8 garbled circuits (i.e., a $5\times$ improvement) per execution for as low as 3500 executions. Our results suggest the exciting possibility that secure two-party computation in the malicious setting can be less than an order of magnitude more expensive than in the semi-honest setting.

1 Introduction

Two-party secure computation (2PC) is a rapidly developing area of cryptography. While the basic approach for semi-honest security, *garbled circuits* (GC) [27], is extensively studied and is largely settled, security against malicious players has seen recent significant improvements. The classical technique for lifting the GC approach to work in the malicious setting is *cut-and-choose* (C&C), formalized and proven secure by Lindell and Pinkas [15]. Until recently, this approach required significant overhead: to guarantee probability of cheating $< 2^{-s}$, approximately $3s$ garbled circuits needed to be generated and sent. However, in Crypto 2013 two works reduced the number of garbled circuits required in cut-and-choose to $s + O(\log s)$ [9] and to s [14].

Our contribution. We further significantly reduce the replication factor for C&C-based protocols in the *multiple execution* setting, where the same function

(possibly with different inputs) is evaluated multiple times either in parallel or sequentially. To achieve this, we combine in a novel way the "fast C&C" technique of Lindell [14] with the "LEGO C&C" technique [6, 22].

Our setting and motivation. We consider the *multiple execution* setting, where two parties compute the same function on possibly different inputs either in parallel or sequentially. Here we argue that multiple evaluations of the same function is indeed a natural and frequently-occurring important scenario.

Today, 2PC is only beginning to enter practical deployment. However, we can reasonably speculate on likely future use cases. In the commercial setting, 2PC is natural in both business-to-business and business-to-customer interactions. For example, a bank customer could perform financial transactions (e.g., payments or transfers), a cell phone customer could perform private location-based queries, two businesses or government agencies might query their joint databases of customers, etc. In all of these scenarios, many of the securely evaluated functions are the same, only differing on their inputs. In fact, we conjecture that single-execution functions may be *less likely* to be used in commercial settings. This is because, as a rule-of-thumb of security, externally-accessible interfaces need to be clean and standardized. Allowing a small number of predetermined customer actions allows for more manageable overall security.

Additionally, many complex protocols from the research literature include multiple executions of the same function evaluated on different inputs. For example, Gordon et al. [8] propose sublinear 2PC based on oblivious RAM (ORAM). In their protocol, each ORAM step is executed by evaluating the same function using 2PC. Another frequently used subroutine is oblivious PRF, used, e.g., in the previously mentioned sublinear 2PC work [8] as well as in private database searches [4, 12]. A recent such work [23] traverses the database search tree by evaluating the same match function at each tree node. Finally, any two universal circuits (of the same size) are implementing the same function.

1.1 Preliminaries

Let s denote the statistical security parameter; namely, an adversary can succeed in cheating with probability up to 2^{-s}. Let n denote the computational security parameter. We let t denote the total number of times the parties wish to evaluate a given circuit, and let $\rho = \rho(s, t)$ represent the number of circuits, per evaluation, that need to be generated to achieve an error probability of 2^{-s}. Before discussing our specific technical contribution, we recall the main ideas of our building blocks.

Fast cut-and-choose using cheating punishment [14]. Cut-and-choose (C&C) protocols for GCs work by letting circuit constructor P_1 generate and send a number of GCs to the evaluator P_2, who then chooses a subset of circuits to open and check for correctness. If the checks pass, the remaining circuits are evaluated as in Yao's protocol [27], and the final output is obtained by taking majority over the individual outputs. In concrete terms, prior works [15, 25] required at least 125 circuits to be sent by P_1 to guarantee security 2^{-40}. Lindell's

improved technique [14] achieves 2^{-s} security while requiring P_1 to send only s circuits (i.e., 40 circuits for 2^{-40} security).

Lindell's protocol (which we call the "fast C&C" protocol) has two phases. In the first phase, P_1 with input x and P_2 with input y run a modified C&C which ensures that P_2 obtains a proof of cheating ϕ if it receives two inconsistent output values in any two evaluation circuits. Now, if all evaluation circuits produce the same output z, P_2 locally stores z as its output. Both parties *always* continue to the second *cheating-punishment* phase. In it, P_1 and P_2 securely evaluate a *smaller* circuit C', which takes as inputs P_1's input x and P_2's proof ϕ. (P_2 inputs random values if he does not have ϕ.) P_1 proves in zero-knowledge the consistency of its input x between the two phases. C' outputs x to P_2 if ϕ is a valid proof of cheating; otherwise P_2 receives nothing. The efficiency improvement is due to the fact that cheating is *punished* if there is any inconsistency in outputs.

LEGO cut-and-choose [6, 22]. These works take a different approach by implementing a two-stage C&C at the *gate* level. The evaluation circuit is then constructed from the unopened garbled gates. In the first stage, P_1 sends multiple garbled gates and P_2 performs a standard C&C with replication factor $\rho(s) = O(s/\log|C|)$. P_2 aborts if any opened gate is garbled incorrectly. In the next stage, P_2 partitions the $\rho(s)|C|$ garbled gates into *buckets* such that each bucket contains $O(\rho(s))$ garbled gates. This two-stage C&C ensures that, except with probability 2^{-s}, each bucket contains a *majority* of correctly constructed garbled gates.

To connect gates with one another, Nielsen and Orlandi [22] use homomorphic Pedersen commitments. The resulting computational efficiency is relatively poor as they perform several expensive public-key operations *per gate*. This is addressed in the miniLEGO work [6], where the authors (among other things) construct homomorphic commitments from oblivious transfer (OT), whose cost can be amortized by OT extension [10]. However, the overall efficiency of this construction is still lacking in concrete terms due to large constants inside the big-O notation. In particular, the communication efficiency is adversely affected by the use of asymptotically constant-rate codes that are concretely inefficient.

1.2 Overview of Our Approach

Our main idea for the multiple execution setting is to run two-stage LEGO C&C at the *circuit* level, and then use fast C&C in the second stage (thereby requiring only a single correctly constructed circuit from each bucket). In particular, now the size of C' used in each execution depends only on the input and output lengths of C, and is no longer proportional to $|C|$. In this section, we focus only on the cut-and-choose aspect of the protocol; namely, on preventing P_1's cheating by submitting incorrect garbled circuits. More detailed protocol descriptions for both the parallel and sequential settings can be found in Section 2 and Section 3.

In the first-stage cut-and-choose, P_1 constructs and sends to P_2 a total of ρt GCs. Next, P_2 requests that P_1 open a random $\rho t/2$-sized subset of the garbled circuits. If P_2 discovers that any opened garbled circuit is incorrectly constructed,

# of Executions	Replication	Replication for Fast C&C	
	parallel/sequential	*parallel*	*sequential*
2	32	40	41
4	24	40	42
7	20	40	42
20	16	40	44
100	12	40	46
3500	8	40	51

Table 1. The number of garbled circuits required *per execution* in order to guarantee a security loss of $< 2^{-40}$. For comparison, the last two columns show the number of circuits required by the fast C&C protocol [14] in the parallel and sequential settings. Note that when using the fast C&C protocol for sequential executions we need to increase the replication factor from s to $s + \log t$.

it aborts. Otherwise, P_2 proceeds to the second stage cut-and-choose, where it randomly assigns unopened circuits to t buckets such that each bucket contains $\rho/2$ circuits. Now, as in the fast C&C protocol [14], each of the t evaluations are executed in two phases. In the first phase of the kth execution, party P_2 evaluates the $\rho/2$ evaluation circuits contained in the kth bucket. The circuits are designed such that if P_2 obtains different outputs from evaluating circuits in the kth bucket, then it obtains a proof of cheating ϕ_k. Next, both parties continue to the cheating-punishment phase, where P_1 and P_2 securely evaluate a smaller circuit that outputs P_1's input x_k if P_2 provides a valid proof ϕ_k.

Clearly, P_1 succeeds in cheating only if (1) it constructed $m \geq \rho/2$ bad circuits, (2) none of these m bad circuits were caught in the first cut-and-choose stage (i.e., $m \leq \rho t/2$), and (3) in the second stage, there exists a bucket that contains all bad circuits. It is easy to see that the probability with which m bad circuits escape detection in the first stage cut-and-choose is $\binom{\rho t - m}{\rho t/2} / \binom{\rho t}{\rho t/2}$. Conditioned on this event happening, the probability that a particular bucket contains all bad circuits is $\binom{m}{\rho/2} / \binom{\rho t/2}{\rho/2}$. Applying the union bound, we conclude that the probability that P_1 succeeds in cheating is bounded by

$$t \binom{\rho t - m}{\rho t/2} \binom{m}{\rho/2} \Big/ \binom{\rho t}{\rho t/2} \binom{\rho t/2}{\rho/2}.$$

For any given t and s, the smallest ρ, hinging on the maximal probability of P_1's successful attack, can be determined by enumerating over all possible values of m (i.e., $\{\rho/2, \rho/2 + 1, \ldots, \rho t/2\}$).

As an example, for $t = 20$ in a parallel execution setting with $s = 40$, using our protocol the circuit generator needs to construct $16 \cdot t = 320$ garbled circuits, whereas using a naïve application of Lindell's protocol [14] requires $40 \cdot t = 800$ garbled circuits.

Parallel vs. sequential executions. As will be evident, it is important to distinguish between the settings where multiple evaluations are carried out in

parallel (e.g., when all inputs are available at the start of the protocol) and where these evaluations are carried out sequentially (e.g., when not all inputs are available as they, for example, depend on the outputs of previous executions). Below, we provide an overview of the main challenges of each setting, and an outline of our solutions.

Parallel executions. Under the DDH assumption, we apply our C&C technique in the parallel execution setting by modifying Lindell's protocol [14] as follows. We construct a generalized *cut-and-choose oblivious transfer* (C&C OT) functionality that supports *multi-stage* cut-and-choose. We call this functionality $\mathcal{F}_{\text{mcot}}$. Asymptotically, we can realize $\mathcal{F}_{\text{mcot}}$ using general secure computation, since the circuit for $\mathcal{F}_{\text{mcot}}$ depends only on the length of P_2's input and is otherwise independent of the circuit. However, such a realization is extremely inefficient in practice (the size of the circuit for realizing $\mathcal{F}_{\text{mcot}}$ needs to accept inputs of length at least $n\rho t\ell$, where n is the computational security parameter and ℓ is the input length). Instead, we show an efficient realization that is only a factor $\rho t^2/s$ less efficient (per execution) than the modified C&C OT realization of Lindell [14]. We elaborate more on this, and other important details, in Section 2.

Sequential executions. To prevent a malicious evaluator from choosing its inputs based on the garbled circuit, GC-based 2PC protocols perform OT *before* the constructor sends its GCs to the evaluator (i.e., before the cut-and-choose phase). This forces the parties, and in particular the evaluator, to "commit" to their inputs before performing the cut-and-choose. This, however, does not work in the sequential setting, where the parties may not know all their inputs at the beginning of the protocol. Standard solutions used in previous works [1, 7, 20] include assuming the garbled-circuit construction is adaptively secure or using adaptively-secure garbling [3] explicitly, assuming the programmable random-oracle model. Another issue is that since now we perform OTs for each execution separately, we can no longer use C&C OT or its variants; instead we rely on the "XOR-tree" approach of Lindell and Pinkas [15] to avoid selective failure attacks. We elaborate more on this, and other details, in Section 3.

Our solution for the sequential setting readily carries over to the parallel setting. In particular, adapting our protocol from the sequential to the parallel setting may address situations where the cost incurred by the use of $\mathcal{F}_{\text{mcot}}$ outweighs the cost of using both the XOR-tree approach and adaptively secure garbled circuits.

1.3 Related Work

Lindell and Pinkas [15] gave the first[4] rigorous 2PC protocol based on cut-and-choose. For $s = 40$, their protocol required at least $17s = 680$ garbled circuits. Subsequent work by the same authors [16] reduced the number of circuits to 128. This was later improved by shelat and Shen [25] to 125 using a more precise analysis of the C&C approach. In Crypto 2013, two works [9, 14] proposed

[4] C&C mechanisms were previously employed in works by Pinkas [24] and Malkhi et al. [18] but these approaches were later shown to be flawed [13, 19].

(among other things) dramatic improvements to the number of garbled circuits that need to be sent. In more detail, for achieving statistical security 2^{-s}, Huang et al.'s protocol [9] requires $2s + O(\log s)$ circuits, where each party generates half of them, and Lindell's protocol [14] requires exactly s circuits.

While all of the above works perform cut-and-choose over circuits, applying cut-and-choose at the gate-level has also been considered [5, 6, 21, 22]. As discussed above, this approach naturally extends to the multiple execution setting, and furthermore is not inherently limited to considering settings where the same function is evaluated multiple times. Nielsen et al. [21] indeed show concrete efficiency improvements using gate-level cut-and-choose techniques. However, the number of rounds grows linearly with the depth of the evaluated circuit.

Finally, in independent and concurrent work, Lindell and Riva [17] also investigate the multiple execution setting, and obtain performance improvements similar to ours. An interesting difference between our works is that while we always let the evaluator pick half the circuits to check, they show that varying the number of check circuits can lead to an additional performance improvement.

2 The Parallel Execution Setting

Consider a setting where two parties wish to securely evaluate the same function multiple times in parallel. Let f denote the function of interest, and let t denote the number of times the parties wish to evaluate f. Let P_1's (resp., P_2's) input in the kth execution be x_k (resp., y_k), and let $x = (x_1, \ldots, x_t)$ and $y = (y_1, \ldots, y_t)$. We define $f^{(t)}(x, y) = (f(x_1, y_1), \ldots, f(x_t, y_t))$.

We adapt Lindell's protocol [14] to support our cut-and-choose technique in the parallel execution setting. The main difficulty is the design and construction of a generalization of cut-and-choose oblivious transfer [16] which we use to avoid the "selective failure attack" where a malicious P_1 constructs invalid keys for P_2's input wires to try and deduce P_2's inputs based on if P_2 aborts execution or not. We discuss this more in Section 2.1. We note that the naïve idea of using the XOR-tree approach [15] in our setting does not appear to work without using adaptively secure garbled circuits. Specifically, it is no longer clear how P_1, without any knowledge of which circuits will end up as evaluation circuits, can batch P_2's input keys together in a way that lets P_2 learn different sets of input keys corresponding to different evaluation circuits and yet within each evaluation bucket guaranteeing that P_2 can learn only input keys corresponding to the same set of inputs.

We give details of our protocol construction for the parallel executions setting in Section 2.2.

2.1 Generalizing Cut-and-Choose Oblivious Transfer

Cut-and-choose oblivious transfer (C&C OT) [16] is an extension of standard one-out-of-two oblivious transfer (OT). The sender inputs L pairs of strings, and the receiver inputs L selection bits to select one string out of each pair of

Inputs:

- P_1 inputs ℓ vectors $\boldsymbol{x}_i$, each containing s pairs of values $x_0^{i,j}, x_1^{i,j} \in \{0,1\}^{n \times n}$, $i \in [\ell]$, $j \in [s]$. In addition, P_1 inputs s "check values" $\chi_1, \ldots, \chi_s \in (\{0,1\}^n)^s$.
- P_2 inputs $\sigma_1, \ldots, \sigma_\ell \in \{0,1\}$ and a set of indices $J \subseteq [s]$.

Outputs: P_1 receives no output. P_2 receives the following:

- For every $i \in [\ell]$ and $j \in J$, P_2 receives $(x_0^{i,j}, x_1^{i,j})$.
- For every $i \in [\ell]$, P_2 receives $\langle x_{\sigma_i}^{i,1}, \ldots, x_{\sigma_i}^{i,s} \rangle$.
- For every $k \notin J$, P_2 receives χ_k.

In other words, P_2 receives $\{\chi_j\}_{j \in [s] \setminus J}$ and $\{\{x_{\sigma_i}^{i,j}\}_{j \in [s] \setminus J}, \{(x_0^{i,j}, x_1^{i,j})\}_{j \in J}\}_{i \in [\ell]}$.

Fig. 1. Modified batch single-choice cut-and-choose OT functionality $\mathcal{F}_{\text{ccot}}$ [14].

sender strings. The receiver also inputs a set J of size $L/2$ that consists of indices where it wants *both* the sender's inputs to be revealed. Note that for indices not contained in J, only those sender inputs that correspond to the receiver's selection bits are revealed. In applications to secure computation, and in particular when transferring input keys corresponding to a particular input wire across all evaluation circuits, one needs *single-choice* cut-and-choose oblivious transfer, where the receiver is restricted to inputting the *same* selection bit in all the $L/2$ instances where it receives exactly one out of two sender strings. Furthermore, when transferring input keys for multiple input wires, it is crucial that the subset J input by the receiver is the same across each instance of single-choice C&C OT executed for all input wires. This variant, called *batch single-choice* C&C OT, can be realized from the decisional Diffie-Hellman problem [16].

Lindell [14] presented a variant of batch single-choice C&C OT [16] in order to address settings where the check set J input by the receiver may be of arbitrary size. We denote this variant by $\mathcal{F}_{\text{ccot}}$; see Figure 1 for the formal description. In this variant, in addition to obtaining one of the two sender inputs for pairs whose indices are not in J, the receiver also obtains a "check value" for each index not in J. These check values are used to confirm whether or not a circuit is an evaluation circuit.

For our purposes, we introduce a new variant of $\mathcal{F}_{\text{ccot}}$, which we call batch single-choice *multi-stage* C&C OT. We denote this primitive by $\mathcal{F}_{\text{mcot}}$ and present its formal description in Figure 2. At a high level, our variant differs from $\mathcal{F}_{\text{ccot}}$ in that receiver P_2 can now input multiple sets $J_1, \ldots, J_t$ (where J is now implicitly defined as $[\rho t] \setminus \cup_{k \in [t]} J_k$) and make independent selections for each of $J_1, \ldots, J_t$. Unlike in Lindell's scheme [14], we only need to consider sets $J_1, \ldots, J_t$ whose sizes are pre-specified in order to provide the desired security guarantees. However, as in the $\mathcal{F}_{\text{ccot}}$ functionality, $\mathcal{F}_{\text{mcot}}$ (1) does not require sets $J_1, \ldots, J_t$ to be of a particular size, and (2) delivers "check values" for indices contained in each of $J_1, \ldots, J_t$. These check values are used to confirm whether a circuit is an evaluation circuit in the kth bucket for some $k \in [t]$.

Inputs:

- P_1 inputs ℓ vectors $\boldsymbol{x}_i$, each containing ρt^2 pairs $x_0^{i,j}, x_1^{i,j} \in \{0,1\}^n$. In addition, P_1 inputs ρt^2 "check values" $\chi_1^1, \ldots, \chi_{\rho t}^1; \ldots; \chi_1^t, \ldots, \chi_{\rho t}^t \in \{0,1\}^n$.
- P_2 inputs $\boldsymbol{\sigma}_1 = (\sigma_{1,1}, \ldots, \sigma_{1,\ell}), \ldots, \boldsymbol{\sigma}_t = (\sigma_{t,1}, \ldots, \sigma_{t,\ell}) \in \{0,1\}^\ell$ and sets $J_1, \ldots, J_t$ that are pairwise non-intersecting subsets of $[\rho t]$.

Outputs: Party P_1 receives no output. Party P_2 receives the following:

- For every $k \in [t]$ and for every $j \in J_k$, party P_2 receives χ_j^k.
- Let $J = [\rho t] \setminus \cup_{k \in [t]} J_k$. For every $i \in [\ell]$ and $j \in [\rho t]$:
  - If $j \in J$, then P_2 receives $(x_0^{i,j}, x_1^{i,j})$.
  - Otherwise, if there exists a (unique) $k \in [t]$ such that $j \in J_k$, then P_2 receives $x_{\sigma_{k,i}}^{i,j}$.

In other words, P_2 receives sets $\{\chi_j^1\}_{j \in J_1}, \ldots, \{\chi_j^t\}_{j \in J_t}$ and $\{\{x_{\sigma_{1,i}}^{i,j}\}_{j \in J_1}, \ldots, \{x_{\sigma_{t,i}}^{i,j}\}_{j \in J_t}, \{(x_0^{i,j}, x_1^{i,j})\}_{j \in J}\}_{i \in [\ell]}$.

Fig. 2. Batch single-choice multi-stage cut-and-choose OT functionality $\mathcal{F}_{\text{mcot}}$.

Designing the $\mathcal{F}_{\text{mcot}}$ functionality. As in $\mathcal{F}_{\text{ccot}}$, the sender P_1 inputs ℓ vectors $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_\ell$ each of length ρt, where each element in the vector is a pair of values (corresponding to the 0-key and the 1-key of a given garbled wire). In addition, P_1 inputs ρt^2 "check values". Receiver P_2 inputs t vectors $\boldsymbol{\sigma}_1, \ldots, \boldsymbol{\sigma}_t$ each of length ℓ and pairwise non-intersecting sets $J_1, \ldots, J_t$. Upon receiving these inputs from P_1 and P_2, the functionality computes $J = [\rho t] \setminus \cup_{k \in [t]} I_k$, and delivers, for each $j \in J$, the jth element (i.e., both values in the jth pair) in each of the ℓ vectors. Next, for every $k \in [t]$ and for each $j \in J_k$, the functionality delivers to P_2 the $\sigma_{k,i}$ value in the jth pair of vector $\boldsymbol{x}_i$ for every $i \in [\ell]$ along with the check value χ_j^k.

Realizing $\mathcal{F}_{\text{mcot}}$ in the $\mathcal{F}_{\text{ccot}}$-hybrid model. We now proceed to construct a protocol for $\mathcal{F}_{\text{mcot}}$. Our goal is to provide an information-theoretic reduction from $\mathcal{F}_{\text{mcot}}$ to $\mathcal{F}_{\text{ccot}}$. We first consider a naïve approach which serves as a warm-up to our final construction and provides intuition behind our definition of $\mathcal{F}_{\text{mcot}}$.

The naïve approach. We propose the following natural approach to realizing $\mathcal{F}_{\text{mcot}}$ from $\mathcal{F}_{\text{ccot}}$: P_1 first performs a t-out-of-t additive secret sharing of all input keys corresponding to P_2's inputs. In addition, P_1 chooses ρt^2 check values. Next, P_1 and P_2 interact with the $\mathcal{F}_{\text{ccot}}$ functionality t times in parallel. In the kth interaction, P_1 provides the kth additive share of its input plus ρt check values $\chi_1^k, \ldots, \chi_{\rho t}^k$ (i.e., a check value for each circuit that could potentially be an evaluation circuit in the kth execution), while P_2 provides its inputs for the kth execution along with a set $[\rho t] \setminus J_k$, where J_k indicates the indices of the evaluation circuits to be used in the kth execution. Let $J = [\rho t] \setminus \cup_{k \in [t]} J_k$. At the end of the interaction, P_2 obtains (1) all t additive shares of input keys, and therefore all input keys, for circuits GC_j with $j \in J$, and (2) all t additive

shares of input keys that *correspond to its actual input* in the kth execution, and therefore its input keys, along with check values for circuits GC_j with $j \notin J$.

Note, in particular, that for the check circuits, P_2 does not obtain the check values, and for the evaluation circuits, P_2 does not obtain both input keys. Thus, the above protocol seems to successfully fulfill our requirements from the $\mathcal{F}_{\text{mcot}}$ functionality. However, note that there is no mechanism in place to enforce that P_2 supplies non-intersecting sets $J_1, \ldots, J_k$. In the following we show that this prevents the above protocol from realizing $\mathcal{F}_{\text{mcot}}$.

Suppose $t = 2$. A malicious P_2 may input overlapping sets J_1, J_2 to $\mathcal{F}_{\text{ccot}}$. The consequence of this is that P_2 now possesses check values χ_j^1 and χ_j^2 for $j \in J_1 \cap J_2$. Clearly, the functionality $\mathcal{F}_{\text{mcot}}$ does not allow this. On the other hand, recall that the input keys are all additively shared, and as a result P_2 does not possess input keys corresponding to its input in circuit GC_j unless its input in both executions are identical. At the surface, there does not seem to be any attack due to this malicious strategy. Sure, P_2 can now equivocate on assigning GC_j to either the first evaluation bucket or the second evaluation. However, as observed earlier, it either has no corresponding keys, or it is going to evaluate both circuits on the same input, say y (in which case it seems immaterial whether j is revealed as part of J_1 or J_2). Unfortunately, we show that the above strategy for malicious P_2 is not simulatable. In particular, at the end of the interaction with $\mathcal{F}_{\text{ccot}}$, the simulator successfully extracts P_2's input in the first and second execution, but is now unable to decide on how to fake the garbled circuit GC_j. On the one hand, if $j \in J_1$, then the fake garbled circuit has to output $z_1 = f(x_1, y)$. On the other hand, if $j \in J_2$, then the fake garbled circuit has to output $z_2 = f(x_2, y)$. Therefore, the simulator has to choose on how to fake GC_j in the dark. Note that a simulation strategy for this specific case that decides to fake GC_j to output z_1 with probability $1/2$, and to output z_2 with probability $1/2$, does indeed succeed with probability $1/2$. However, this strategy does not extend well to the case when t is large.

The discussion above motivates our definition of $\mathcal{F}_{\text{mcot}}$; in particular, it reinforces why $\mathcal{F}_{\text{mcot}}$ must deliver at most one check value per circuit. In the following, we explain how to modify the naïve construction to enforce this.

Our approach. The high level idea behind our protocol is to let P_1 perform independent additive sharings of both the input values as well as the check values. Then P_1 and P_2 query the $\mathcal{F}_{\text{ccot}}$ functionality t times to transfer the values as required by $\mathcal{F}_{\text{mcot}}$. We detail this below, explaining it in the context of our secure computation protocol.

Let $(x_0^{i,j}, x_1^{i,j})$ be the input keys corresponding to P_2's ith input wire in GC_j. First, P_1 performs a t-out-of-t additive secret sharing of all input values corresponding to P_2's inputs; i.e., for each $i \in [\ell], j \in [\rho t]$, P_1 secret shares $x_0^{i,j}$ (resp., $x_1^{i,j}$) into $\{x_0^{i,j,k}\}_{k\in[t]}$ (resp., $\{x_1^{i,j,k}\}_{k\in[t]}$). P_1 then chooses ρt^2 check values $\{\chi_1^k, \ldots, \chi_{\rho t}^k\}_{k\in[t]}$. It then performs a $(2\ell(t-1)+1)$-out-of-$(2\ell(t-1)+1)$ additive sharing of each value χ_j^k to obtain shares denoted $\widetilde{\chi}_j^k$, $\{\chi_{0,k}^{i,j,k'}, \chi_{1,k}^{i,j,k'}\}_{k'\in[t]\setminus\{k\}, i\in[\ell]}$. Then, instead of creating inputs to $\mathcal{F}_{\text{ccot}}$ using $x_c^{i,j,k}$

shares alone, P_1 instead creates a "share block" $X_c^{i,j,k} = (x_c^{i,j,k}, \chi_{c,1}^{i,j,k}, \ldots, \chi_{c,t}^{i,j,k})$. That is, a share block $X_c^{i,j,k}$ contains, in addition to a share of the input key, a share of all check values corresponding to circuit GC_j.

Next, P_1 and P_2 run t instances of $\mathcal{F}_{\text{ccot}}$ in parallel. In the kth interaction, in addition to the ρt check value shares $\widetilde{\chi}_1^k, \ldots, \widetilde{\chi}_{\rho t}^k$, P_1 provides its kth share block while P_2 provides its inputs for the kth execution along with a set $[\rho t] \setminus J_k$, where J_k indicates the indices of the evaluation circuits to be used in the kth execution. Let $J = [\rho t] \setminus \cup_{k \in [t]} J_k$. At the end of the interaction, P_2 obtains (1) all t share blocks of input keys, and therefore all input keys, for circuits GC_j with $j \in J$, and (2) all t share blocks of input keys that *correspond to its actual input* in the kth execution, and therefore its input keys, along with a check value $\widetilde{\chi}_j^k$ for circuits GC_j with $j \in J_k$.

Note, in particular, that for each check circuit GC_j, P_2 does not obtain the check value χ_j^k for any k, because it always misses the check value share $\widetilde{\chi}_j^k$. For each evaluation circuit GC_j with $j \in J_k$, P_2 does not obtain both input keys, and more importantly can obtain at most one check value (which is χ_j^k). This is because share blocks contain shares of input keys as well as shares of check values. For an evaluation circuit, party P_2 always misses a share block, and consequently shares of all values $\chi_j^{k'}$ with $k' \neq k$. Furthermore, if P_2 wants to ensure it receives χ_j^k, then it should never input $J_{k''}$ such that $k'' \neq k$ and yet $j \in J_{k''}$. This is because for $j \in J_{k''}$, P_2 is guaranteed to miss a share block that contains an additive share of χ_j^k. Note that the above observations suffice to deal with a malicious P_2 that inputs overlapping sets since in this case P_2 fails to obtain any check values corresponding to indices in the intersection.

The formal description of the protocol in the $\mathcal{F}_{\text{ccot}}$-hybrid model can be found in Figure 3. We prove the following in the full version.

Theorem 1. *There exists a protocol perfectly realizing $\mathcal{F}_{\text{mcot}}$ in the $\mathcal{F}_{\text{ccot}}$-hybrid model.*

2.2 Using $\mathcal{F}_{\text{mcot}}$ in the Parallel Execution Setting

The input vectors $\boldsymbol{x}_i$, for $i \in [\ell]$, contain the key pairs associated with the ith input wire for P_2 in each of the ρt circuits. The vector $\boldsymbol{\sigma}_k$ corresponds to the inputs used by P_2 in the kth execution. An honest P_2 chooses sets $J_1, \ldots, J_t$ such that they are pairwise non-intersecting and each set is of size exactly $\rho/2$. The main observation is that, for a given execution $k \in [t]$, P_2 obtains check values χ_j^k from $\mathcal{F}_{\text{mcot}}$ only for $j \in J_k$. Therefore, once the parties complete the interaction with $\mathcal{F}_{\text{mcot}}$ and P_1 sends all the garbled circuits, we let P_1 determine the evaluation circuits in each bucket based on whether P_2 sends the corresponding check values. At this point, P_1 checks that each bucket of evaluation circuits is well-defined and that these buckets are of equal size, i.e., $\rho/2$. If not, P_1 aborts. To overcome technical difficulties, we also require P_2 to provide "check values" for the check circuits as well. A check value for check circuit GC_j, denoted χ_j, may simply be the set of all input keys (i.e., both the 0-key and the 1-key) on all wires in circuit GC_j.

Inputs:

- P_1 inputs ℓ vectors of pairs $\boldsymbol{x}_i = \langle (x_0^{i,1}, x_1^{i,1}), \ldots, (x_0^{i,\rho t}, x_1^{i,\rho t}) \rangle$ for $i \in [\ell]$. In addition, P_1 inputs ρt^2 "check values" $(\chi_1^1, \ldots, \chi_{\rho t}^1), \ldots, (\chi_1^t, \ldots, \chi_{\rho t}^t)$. All values are in $\{0,1\}^n$.
- P_2 inputs $\boldsymbol{\sigma}_1 = (\sigma_{1,1}, \ldots, \sigma_{1,\ell}), \ldots, \boldsymbol{\sigma}_t = (\sigma_{t,1}, \ldots, \sigma_{t,\ell}) \in \{0,1\}^\ell$ and sets $J_1, \ldots, J_t$.

Protocol:

- For all $i \in [\ell]$, P_1 performs a t-out-of-t additive secret sharing of $\boldsymbol{x}_i$ to obtain shares $\boldsymbol{x}_{i,1}, \ldots, \boldsymbol{x}_{i,t}$. For $k \in [t]$, let $\boldsymbol{x}_{i,k} = \langle (x_0^{i,1,k}, x_1^{i,1,k}), \ldots, (x_0^{i,\rho t,k}, x_1^{i,\rho t,k}) \rangle$. Let $X_0^{i,j,k} = (x_0^{i,j,k}, \chi_{0,1}^{i,j,k}, \ldots, \chi_{0,t}^{i,j,k})$ and $X_1^{i,j,k} = (x_1^{i,j,k}, \chi_{1,1}^{i,j,k}, \ldots, \chi_{1,t}^{i,j,k})$, where $\chi_{0,1}^{i,j,k}, \ldots, \chi_{0,t}^{i,j,k}$ and $\chi_{1,1}^{i,j,k}, \ldots, \chi_{1,t}^{i,j,k}$ are random independent values in $\{0,1\}^n$. Let $\boldsymbol{X}_{i,k} = \langle (X_0^{i,1,k}, X_1^{i,1,k}), \ldots, (X_0^{i,\rho t,k}, X_1^{i,\rho t,k}) \rangle$.
- For all $k \in [t]$ and $j \in [\rho t]$, set $\tilde{\chi}_j^k = \chi_j^k \oplus \bigoplus_{k' \in [t] \setminus \{k\}, i \in [\ell]} (\chi_{0,k}^{i,j,k'} \oplus \chi_{1,k}^{i,j,k'})$.
- P_1 and P_2 run t instances of $\mathcal{F}_{\text{ccot}}$ in parallel as follows. In the kth instance:
  - P_1 inputs ℓ vectors of pairs $\boldsymbol{X}_{i,k}$ of length ρt for $i \in [\ell]$ and ρt "check values" $\widetilde{\chi}_1^k, \ldots, \widetilde{\chi}_{\rho t}^k$. P_2 inputs $\sigma_{k,1}, \ldots, \sigma_{k,\ell} \in \{0,1\}$ and the set $[\rho t] \setminus J_k$.
  - P_2 receives $\{\widetilde{\chi}_j^k\}_{j \in J_k}$ and $\{\{X_{\sigma_{k,i}}^{i,j,k}\}_{j \in J_k} \cup \{(X_0^{i,j,k}, X_1^{i,j,k})\}_{j \in [\rho t] \setminus J_k}\}_{i \in [\ell]}$.
- For all $k \in [t]$ and $j \in J_k$, P_2 reconstructs $\chi_j^k = \tilde{\chi}_j^k \oplus \bigoplus_{k' \in [t] \setminus \{k\}, i \in [\ell]} (\chi_{0,k}^{i,j,k'} \oplus \chi_{1,k}^{i,j,k'})$.
- Let $J = [\rho t] \setminus \cup_{k \in [t]} J_k$. For all $i \in [\ell]$ and $j \in [\rho t]$, P_2 does the following:
  - If $j \in J$: set $x_0^{i,j} = \bigoplus_{k \in [t]} x_0^{i,j,k}$, and $x_1^{i,j} = \bigoplus_{k \in [t]} x_1^{i,j,k}$.
  - If there exists (unique) $k \in [t]$ such that $j \in J_k$: set $x_{\sigma_{k,i}}^{i,j} = \bigoplus_{k \in [t]} x_{\sigma_{k,i}}^{i,j,k}$.
- P_2 outputs sets $\{\chi_j^1\}_{j \in J_1}, \ldots, \{\chi_j^t\}_{j \in J_t}$ and $\{\{(x_0^{i,j}, x_1^{i,j})\}_{j \in J}, \{x_{\sigma_{1,i}}^{i,j}\}_{j \in J_1}, \ldots, \{x_{\sigma_{t,i}}^{i,j}\}_{j \in J_t}\}_{i \in [\ell]}$.

Fig. 3. Realizing $\mathcal{F}_{\text{mcot}}$ in the $\mathcal{F}_{\text{ccot}}$-hybrid model.

Applying the cheating-punishment technique. Inspired by Lindell's protocol [14], we use the knowledge of two different garbled values for a single output wire as a "proof" that P_2 received inconsistent outputs in a given execution. P_2 can use this proof to obtain P_1's input in a cheating-punishment phase. This cheating-punishment phase is implemented via a secure computation protocol, and thus it is important that the second phase functionality has a small circuit. We employ several optimizations proposed by Lindell [14] to keep the size of this circuit small. One important difference in our setting is that, unlike in Lindell's protocol [14], we cannot have, for a given output wire w, the same output keys b_w^0, b_w^1 across all garbled circuits. This is because in our setting garbled circuits are assigned to different evaluation buckets, and the circuits in each bucket can be evaluated with different input values, and thus can produce different outputs. Thus (even an honest) P_2 could potentially learn, say, output key b_w^0 in one execution and output key b_w^1 in another. We address this by simply removing the

requirement that the set of output keys across different garbled circuits are the same. Thus, the circuit for the cheating-punishment phase for the kth execution must now take as input from P_1 *all* of the output keys in *all* of the evaluation circuits in the kth bucket, and from P_2 a pair of output keys that serve as proof of cheating. Somewhat surprisingly, we show that the size of the circuit (measured as the number of non-XOR gates) for the cheating-punishment phase is essentially the same as the circuit in Lindell's protocol [14].[5]

Another detail we wish to point out is that in our protocol we need to run separate cheating-punishment phases for each execution. This is a restriction imposed by the way in which P_1 proves consistency of its inputs [14, 16]. However, we can run all of the t cheating-punishment phases *in parallel*. For this reason we use the universally composable variant of Lindell and Pinkas's protocol [16] (which is essentially obtained by replacing oblivious transfers and zero-knowledge subprotocols with their universally composable variants) to implement each cheating-punishment phase.

Other details. We now describe other important details of our protocol.

- *Input consistency across multiple executions.* It is important to guarantee that P_1 provides consistent inputs across all circuits in the kth execution. Fortunately, existing mechanisms [14, 16] for ensuring input consistency in the single execution setting can be readily extended to the multiple execution setting as well.
- *Encoded translation tables for garbled circuits.* As in Lindell's protocol [14], we modify the output translation tables used in the garbled circuits. Specifically, for keys k_i^0, k_i^1 on output wire i, we create an *encoded* output table $[h(k_i^0), h(k_i^1)]$, where h is some one-way function. We require that the output keys (or more precisely, the output of h applied to the output keys) corresponding to 0 and 1 are distinct. This encoding gives us the following two properties: (1) P_2 after evaluating a garbled circuit can use the encoded translation tables to determine whether the output is 0 or 1, and (2) the encoded translation table does not reveal the other output key (since this is equivalent to inverting the one-way function) to P_2.
- *Optimizing the cheating-punishment circuit.* We can apply similar techniques as shown by Lindell [14] to optimize the size of the cheating-punishment circuit to contain only ℓ non-XOR gates. We leave the details to the full version.

Formal description. We proceed to the formal description of our protocol.

Inputs: P_1 has input $x = (x_1, \ldots, x_t)$, where $x_k \in \{0,1\}^\ell$, and P_2 has input $y = (y_1, \ldots, y_t)$, where $y_k \in \{0,1\}^\ell$.

[5] Of course, the cost of realizing our cheating-punishment phase is more than the corresponding cost in Lindell's protocol [14], mainly due to P_1's input being larger (but only by a factor of $\rho/2$).

Auxiliary Inputs: A statistical security parameter s, a computational security parameter n, the description of a circuit C where $C(x,y) = f(x,y)$, the number of evaluations t of the function f, and $(\mathbb{G}, q, g)$ where $\mathbb{G}$ is a cyclic group with generator g and prime order q, where q is of length n. Let $\mathsf{Ext} : \mathbb{G} \to \{0,1\}^n$ be a function mapping group elements to bitstrings. In the following, $\rho = \rho(s,t)$ is the replication factor defined as being the smallest $u \in \mathbb{N}$ such that for all $m \in \{u/2, \ldots, ut/2\}$ it holds that $t \cdot \binom{ut-m}{ut/2}\binom{m}{u/2} / \binom{ut}{ut/2}\binom{ut/2}{u/2} \leq 2^{-s}$. If no such u exists or if $\rho \geq s$, then parties abort this protocol, and instead run the fast C&C protocol [14] for the function $f^{(t)}$.

Outputs: P_2 receives $f^{(t)}(x,y)$ and P_1 receives no output. Let ℓ' denote the length of the output of $f(x,y)$.

Protocol:

1. **Input key choice and circuit preparation:**
   - P_1 chooses random values $a_1^0, a_1^1, \ldots, a_\ell^0, a_\ell^1 \in_R \mathbb{Z}_q$, $r_1, \ldots, r_{\rho t} \in_R \mathbb{Z}_q$ and $(b_{1,1}^0, b_{1,1}^1, \ldots, b_{1,\ell'}^0, b_{1,\ell'}^1), \ldots, (b_{\rho t,1}^0, b_{\rho t,1}^1, \ldots, b_{\rho t,\ell'}^0, b_{\rho t,\ell'}^1) \in_R \{0,1\}^{n\ell'}$ such that for every $c_1, c_2 \in \{0,1\}, j_1, j_2 \in [\rho t], i_1, i_2 \in [\ell']$ it holds that $b_{j_1,i_1}^{c_1} = b_{j_2,i_2}^{c_2}$ iff $i_1 = i_2$ and $j_1 = j_2$ and $c_1 = c_2$.
   - Let $w_1, \ldots, w_\ell$ denote the input wires corresponding to P_1's input, let $w_{i,j}$ denote the ith input wire in the jth garbled circuit, and let $k_{i,j}^b$ denote the key associated with bit b on wire $w_{i,j}$. P_1 sets $k_{i,j}^b$ as follows:
   $$k_{i,j}^0 = \mathsf{Ext}(g^{a_i^0 \cdot r_j}) \quad \text{and} \quad k_{i,j}^1 = \mathsf{Ext}(g^{a_i^1 \cdot r_j}).$$
   - Let $w'_1, \ldots, w'_{\ell'}$ denote the output wires. The keys for wire w'_i in the jth garbled circuit are set to $b_{j,i}^0$ and $b_{j,i}^1$.
   - P_1 constructs ρt independent garblings, $GC_1, \ldots, GC_{\rho t}$, of circuit C, using random keys except for wires $w_1, \ldots, w_\ell$ and $w'_1, \ldots, w'_m$, where the keys are set as above.
2. **Oblivious transfers:** P_1 and P_2 run $\mathcal{F}_{\text{mcot}}$ as follows:
   - For $i \in [\ell]$, let $\boldsymbol{z}_i$ denote a vector containing the ρt pairs of keys associated with P_2's ith input bit in all the garbled circuits. P_1 inputs $\boldsymbol{z}_1, \ldots, \boldsymbol{z}_\ell$, as well as random values $\chi_1^1, \ldots, \chi_{\rho t}^1; \ldots; \chi_1^t, \ldots, \chi_{\rho t}^t$.
   - P_2 inputs random sets $J_1, \ldots, J_t$ which are pairwise non-intersecting subsets of $[\rho t]$ such that for all $k \in [t]$ it holds that $|J_k| = \rho/2$. Let $J = [\rho t] \setminus \cup_{k \in [t]} J_k$. P_2 also inputs bits $(\sigma_{1,1}, \ldots, \sigma_{1,\ell}), \ldots, (\sigma_{t,1}, \ldots, \sigma_{t,\ell}) \in \{0,1\}^\ell$, where $\sigma_{k,i} = y_{k,i}$ for every $i \in [\ell]$ and $k \in [t]$.
   - For $j \in J$, P_2 receives both input keys associated with its input wires in garbled circuit GC_j, and for each $k \in [t]$ and $j \in J_k$, P_2 receives the keys associated with its input y_k on its input wires in garbled circuit GC_j. Also, for every $k \in [t]$ and $j \in J_k$, P_2 receives χ_j^k.
3. **Send circuits and commitments:** P_1 sends P_2 the garbled circuits $GC_1, \ldots, GC_{\rho t}$, the "seed" for the randomness extractor Ext, the following commitment to the garbled values associated with P_1's input wires:
   $$\{(i, 0, g^{a_i^0}), (i, 1, g^{a_i^1})\}_{i \in [\ell]} \quad \text{and} \quad \{(j, g^{r_j})\}_{j=1}^{\rho t}$$
   and the encoded output translation tables:
   $$\{[(h(b_{j,1}^0), h(b_{j,1}^1)), \ldots, (h(b_{j,\ell'}^0), h(b_{j,\ell'}^1))]\}_{j \in [\rho t]}.$$
   If $h(b_{j,i}^0) = h(b_{j,i}^1)$ for any $1 \leq i \leq \ell', 1 \leq j \leq \rho t$, then P_2 aborts.

4. **Send cut-and-choose challenge:** P_2 sends P_1 the sets $J, J_1, \ldots, J_t$ along with values $\{\chi_j^1\}_{j\in J_1}, \ldots, \{\chi_j^t\}_{j\in J_t}$, and all the keys associated with its input wires in all circuits GC_j for $j \in J$. If the values received by P_1 are (1) incorrect, or (2) the sets $J_1, \ldots, J_t$ are not pairwise non-intersecting, or (3) the input keys associated with P_2's input wires in circuits GC_j are revealed incorrectly, or (4) there exists some $k \in [t]$ such that $|J_k| \neq \rho/2$, then it outputs $\perp$ and aborts. Circuits GC_j for $j \in J$ are called *check circuits* and circuits GC_j for $j \in J_k$ are called *evaluation circuits* in the *kth bucket*.
5. **Send garbled input values in the evaluation circuits:** For each $k \in [t]$: P_1 sends the input keys associated with input x_k for the evaluation circuits in the kth bucket: For each $j \in J_k$ and every wire $i \in [\ell]$, P_1 sends the value $k'_{i,j} = g^{a_i^{x_{k,i}} \cdot r_j}$ and P_2 sets $k_{i,j} = \mathsf{Ext}(k'_{i,j})$.
6. **Circuit evaluation:** For each $k \in [t]$, P_2 does the following:
   - For each $j \in J_k$ and every wire $i \in [\ell']$, P_2 computes $b'_{j,i}$ by evaluating GC_j. If P_2 receives exactly *one* valid output value per output wire, then let z_k denote this output. In this case, it chooses random values $b_0^k, b_1^k \in_R \{0,1\}^n$. If P_2 receives *two* valid outputs on any output wire then it sets $b_0^k = b'_{j_1,i}$ and $b_1^k = b'_{j_2,i}$, where $j_1, j_2 \in J_k$ denote the conflicting circuit indices. If P_2 receives *no* valid output values on any output wire, then P_2 aborts.
7. **Run secure computation to detect cheating:** For each $k \in [t]$, P_1 and P_2 do the following *in parallel*:
   P_1 defines a circuit with the values $\{b_{j,1}^0, b_{j,1}^1, \ldots, b_{j,\ell'}^0, b_{j,\ell'}^1\}_{j\in J_k}$ hardcoded. The circuit computes the following function:
   - P_1 inputs $x_k \in \{0,1\}^\ell$ and has no output.
   - P_2 inputs a pair of values b_0^k, b_1^k.
   - If there exists values $i \in [\ell']$ and $j_1, j_2 \in J_k$ such that $b_0^k = b_{j_1,i}^0$ and $b_1^k = b_{j_2,i}^1$, then P_2's output is x_k; otherwise it receives no output.

   P_1 and P_2 run the UC-secure protocol of Lindell and Pinkas [16] on this circuit (except for the proof of P_1's input values), as follows:
   - P_1 inputs x_k; P_2 inputs b_0^k and b_1^k as computed in Step 6.
   - The garbled circuits constructed by P_1 use the same a_i^0, a_i^1 values as were chosen in Step 1, and the parties use $3(s + \log t)$ copies of the circuit for the cut-and-choose.

   If this computation results in an abort, then both parties halt.
8. **Check circuits for computing $f^{(t)}(x, y)$:**
   - For $j \in J$, P_1 sends r_j to P_2, and P_2 checks that these values are consistent with the pairs $\{(j, g^{r_j})\}_{j\in J}$ received in Step 3. If not, P_2 aborts.
   - For every $j \in J$, P_2 uses the $g^{a_i^0}, g^{a_i^1}$ values received in Step 3 and the r_j values received above to compute the keys for P_1's input wires as $k_{i,j}^0 = \mathsf{Ext}(g^{a_i^0 \cdot r_j}), k_{i,j}^1 = \mathsf{Ext}(g^{a_i^1 \cdot r_j})$. In addition, P_2 uses the keys obtained from $\mathcal{F}_{\text{mcot}}$ in Step 2 for its own input wires. P_2 verifies that GC_j is a correct garbling of C. If there exists a circuit for which this does not hold, then P_2 aborts.
9. **Verify consistency of P_1's input:** For each $k \in [t]$: Let $\widehat{J}_k$ be the set of check circuits used in the 2PC computation in Step 7 for the kth bucket, let $\widehat{r}_{j,k}$ be the value used in that computation, and 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 each $k \in [t]$, P_1 and P_2 do the following *in parallel*:

– For every input wire $i \in [\ell']$, P_1 proves a zero-knowledge proof-of-knowledge that there exist some $\sigma_{k,i} \in \{0,1\}$ such that for every $j \in J_k$ and every $j' \notin \hat{J}_k$, it holds that $k'_{i,j} = g^{a_i^{\sigma_{k,i}} \cdot r_j}$ and $\hat{k}_{i,j} = g^{a_i^{\sigma_{k,i}} \cdot \hat{r}_{j',k}}$. If any of the t proofs fail, then P_2 aborts.

10. **Output evaluation:** For each $k \in [t]$, P_2 does the following:
    – If P_2 received no inconsistent outputs in Step 6, then it uses the encoded translation tables to decode the outputs it received, and sets z_k to that value. If P_2 received inconsistent output, then let x_k be the output that P_2 received from the circuit in Step 7. Let $z_k = f(x_k, y_k)$ be the output in this case.

    P_2 outputs $z = (z_1, \ldots, z_t)$ and terminates.

We prove the following theorem in the full version.

Theorem 2. *Let s (resp., n) be the statistical (resp., computational) security parameter. If the decisional Diffie-Hellman assumption holds in $(\mathbb{G}, g, q)$, h is a one-way function, and the underlying circuit garbling procedure is secure, then for all $t = \mathsf{poly}(n)$, the protocol described above securely computes $f^{(t)}$ in the presence of a malicious adversary with error at most $2^{-s} + \mu(n)$ for some negligible function $\mu(\cdot)$.*

3 The Sequential Execution Setting

We now consider the setting where the parties securely evaluate the same function f multiple times sequentially. Let t denote the number of times the parties wish to evaluate f. Let P_1's (resp., P_2's) input in the kth execution be denoted by x_k (resp., y_k). Let $f^{[t]}$ denote the reactive functionality that computes f a total of t times sequentially.

The main difference between this setting and the parallel setting discussed in Section 2 is that in the sequential setting the parties may not know their inputs to all executions at the start of the protocol. In particular, inputs may depend on outputs from previous executions. Thus, the parallel execution protocol does not immediately carry over to the sequential setting. To see why, observe for instance that $\mathcal{F}_{\text{mcot}}$ requires P_2 to submit all of its inputs at once[6]. This is not possible since in the sequential setting we cannot assume that P_2 has all its inputs at the beginning of the protocol. Instead, we take a different route; namely, we use the "XOR-tree" approach [15, 26] to protect against the so-called "selective failure attack" [13, 19, 25]. (In the parallel execution setting, this attack was implicitly avoided due to the use of $\mathcal{F}_{\text{mcot}}$.) In this approach, the circuit C to be evaluated is first modified into an equivalent circuit C_{XT} (to include an "XOR-tree" for

[6] Standard oblivious transfer precomputation/"correction" techniques [2] still apply to $\mathcal{F}_{\text{mcot}}$ as well; however, it is not clear how to "correct" $\mathcal{F}_{\text{mcot}}$ correlations in a way suitable for the sequential setting.

P_2's inputs). Then, P_1 sends commitments to input keys corresponding to P_2's input wires in C_{XT}. The corresponding decommitments are revealed to P_2 via a standard one-out-of-two oblivious transfer. In order to prevent P_2 from using different inputs across evaluation circuits within the same bucket, P_1 batches together the decommitments corresponding to a particular input wire across all evaluation circuits in a given bucket. Note that herein lies an opportunity for a malicious P_1 to force P_2 to abort the protocol depending on its input. (This can be done for instance by sending incorrect decommitments for say only the 0-key on a particular wire.) However, the modified circuit C_{XT} is such that the success of any such selective OT attack is statistically independent of P_2's actual input value. Therefore, if an honest P_2 receives an invalid decommitment and is unable to decrypt the evaluation circuit, then it simply aborts knowing that its privacy is not compromised. Finally, we note that since we use one-out-of-two oblivious transfer (as opposed to $\mathcal{F}_{\mathrm{mcot}}$), we can leverage oblivious transfer extension techniques [10, 11, 21] to obtain better efficiency.

We stress that the oblivious transfer step happens *after* P_1 sends all the GCs to P_2. This is because P_2's inputs to all t executions are not available at the beginning of the protocol. Further, P_2's inputs may depend on previous outputs, which can be obtained only by decrypting evaluation circuits, i.e., after the evaluation bucket for the current execution is fully determined. Note that our cut-and-choose technique guarantees that there is at least one good evaluation circuit in every bucket under the assumption that P_1 has already committed to all its (good and bad) garbled circuits before the check sets and the evaluation sets are determined. Unfortunately, the above ordering of the oblivious transfer step and the garbled circuit sending step now allows a malicious P_2 to choose its input as a function of the garbled circuits it receives. To counter this, we need to use *adaptively secure garbling schemes* [3] instead of standard garbled circuits; adaptively secure garbling schemes can be constructed efficiently in the programmable random oracle model [3]. Note that we do not need the use of adaptively secure garbling schemes for implementing the cheating-punishment phase. Indeed, all the inputs for that subprotocol are known before the phase begins, and therefore, the oblivious transfer step can be carried out before P_1 sends its garbled circuits for that phase.

Due to lack of space, we leave both the formal description and the proof of the following theorem to the full version.

Theorem 3. *Let s (resp., n) be the statistical (resp., computational) security parameter. If the decisional Diffie-Hellman assumption holds in $(\mathbb{G}, g, q)$, h is a one-way function, and the circuit is garbled using an adaptively secure garbling scheme, then for all polynomial values of t, the protocol described above securely computes $f^{[t]}$ in the presence of a malicious adversary with error at most $2^{-s} + \mu(n)$ for some negligible function $\mu(\cdot)$.*

Acknowledgments

Work of Yan Huang and Jonathan Katz supported in part by NSF award #1111599. Work of Vladimir Kolesnikov supported in part by the Intelligence Advanced Research Project Activity (IARPA) via Department of Interior National Business Center (DoI/NBC) contract Number D11PC20194. Work of Ranjit Kumaresan supported by funding from the European Community's Seventh Framework Programme (FP7/2007–2013) under grant agreement number 259426. Work of Alex J. Malozemoff conducted with Government support through the National Defense Science and Engineering Graduate (NDSEG) Fellowship, 32 CFG 168a, awarded by DoD, Air Force Office of Scientific Research. The U.S. Government is authorized to reproduce and distribute reprints for Governmental purposes notwithstanding any copyright annotation thereon. Disclaimer: The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of IARPA, DoI/NBC, or the U.S. Government.

References

1. Applebaum, B., Ishai, Y., Kushilevitz, E., Waters, B.: Encoding functions with constant online rate or how to compress garbled circuits keys. In: Canetti, R., Garay, J.A. (eds.) CRYPTO 2013, Part II. LNCS, vol. 8043, pp. 166–184. Springer (Aug 2013)
2. Beaver, D.: Precomputing oblivious transfer. In: Coppersmith, D. (ed.) CRYPTO'95. LNCS, vol. 963, pp. 97–109. Springer (Aug 1995)
3. Bellare, M., Hoang, V.T., Rogaway, P.: Adaptively secure garbling with applications to one-time programs and secure outsourcing. In: Wang, X., Sako, K. (eds.) ASIACRYPT 2012. LNCS, vol. 7658, pp. 134–153. Springer (Dec 2012)
4. Cash, D., Jarecki, S., Jutla, C.S., Krawczyk, H., Rosu, M.C., Steiner, M.: Highly-scalable searchable symmetric encryption with support for boolean queries. In: Canetti, R., Garay, J.A. (eds.) CRYPTO 2013, Part I. LNCS, vol. 8042, pp. 353–373. Springer (Aug 2013)
5. Damgård, I., Orlandi, C.: Multiparty computation for dishonest majority: From passive to active security at low cost. In: Rabin, T. (ed.) CRYPTO 2010. LNCS, vol. 6223, pp. 558–576. Springer (Aug 2010)
6. Frederiksen, T.K., Jakobsen, T.P., Nielsen, J.B., Nordholt, P.S., Orlandi, C.: MiniLEGO: Efficient secure two-party computation from general assumptions. In: Johansson, T., Nguyen, P.Q. (eds.) EUROCRYPT 2013. LNCS, vol. 7881, pp. 537–556. Springer (May 2013)
7. Gennaro, R., Gentry, C., Parno, B.: Non-interactive verifiable computing: Outsourcing computation to untrusted workers. In: Rabin, T. (ed.) CRYPTO 2010. LNCS, vol. 6223, pp. 465–482. Springer (Aug 2010)
8. Gordon, S.D., Katz, J., Kolesnikov, V., Krell, F., Malkin, T., Raykova, M., Vahlis, Y.: Secure two-party computation in sublinear (amortized) time. In: Yu, T., Danezis, G., Gligor, V.D. (eds.) ACM CCS 12. pp. 513–524. ACM Press (Oct 2012)
9. Huang, Y., Katz, J., Evans, D.: Efficient secure two-party computation using symmetric cut-and-choose. In: Canetti, R., Garay, J.A. (eds.) CRYPTO 2013, Part II. LNCS, vol. 8043, pp. 18–35. Springer (Aug 2013)

10. Ishai, Y., Kilian, J., Nissim, K., Petrank, E.: Extending oblivious transfers efficiently. In: Boneh, D. (ed.) CRYPTO 2003. LNCS, vol. 2729, pp. 145–161. Springer (Aug 2003)
11. Ishai, Y., Prabhakaran, M., Sahai, A.: Founding cryptography on oblivious transfer - efficiently. In: Wagner, D. (ed.) CRYPTO 2008. LNCS, vol. 5157, pp. 572–591. Springer (Aug 2008)
12. Jarecki, S., Jutla, C.S., Krawczyk, H., Rosu, M.C., Steiner, M.: Outsourced symmetric private information retrieval. In: Sadeghi, A.R., Gligor, V.D., Yung, M. (eds.) ACM CCS 13. pp. 875–888. ACM Press (Nov 2013)
13. Kiraz, M., Schoenmakers, B.: A protocol issue for the malicious case of Yao's garbled-circuit construction. In: 27th Symposium on Information Theory in the Benelux. pp. 283–290 (Jun 2006)
14. Lindell, Y.: Fast cut-and-choose based protocols for malicious and covert adversaries. In: Canetti, R., Garay, J.A. (eds.) CRYPTO 2013, Part II. LNCS, vol. 8043, pp. 1–17. Springer (Aug 2013)
15. Lindell, Y., Pinkas, B.: An efficient protocol for secure two-party computation in the presence of malicious adversaries. In: Naor, M. (ed.) EUROCRYPT 2007. LNCS, vol. 4515, pp. 52–78. Springer (May 2007)
16. Lindell, Y., Pinkas, B.: Secure two-party computation via cut-and-choose oblivious transfer. In: Ishai, Y. (ed.) TCC 2011. LNCS, vol. 6597, pp. 329–346. Springer (Mar 2011)
17. Lindell, Y., Riva, B.: Cut-and-choose secure computation in the online/offline and batch settings. In: Garay, J., Gennaro, R. (eds.) CRYPTO 2014. LNCS, Springer, Santa Barbara, CA, USA (2014)
18. Malkhi, D., Nisan, N., Pinkas, B., Sella, Y.: Fairplay — a secure two-party computation system. In: Blaze, M. (ed.) 13th USENIX Security Symposium. USENIX Association (Aug 2004)
19. Mohassel, P., Franklin, M.: Efficiency tradeoffs for malicious two-party computation. In: Yung, M., Dodis, Y., Kiayias, A., Malkin, T. (eds.) PKC 2006. LNCS, vol. 3958, pp. 458–473. Springer (Apr 2006)
20. Mohassel, P., Riva, B.: Garbled circuits checking garbled circuits: More efficient and secure two-party computation. In: Canetti, R., Garay, J.A. (eds.) CRYPTO 2013, Part II. LNCS, vol. 8043, pp. 36–53. Springer (Aug 2013)
21. Nielsen, J.B., Nordholt, P.S., Orlandi, C., Burra, S.S.: A new approach to practical active-secure two-party computation. In: Safavi-Naini, R., Canetti, R. (eds.) CRYPTO 2012. LNCS, vol. 7417, pp. 681–700. Springer (Aug 2012)
22. Nielsen, J.B., Orlandi, C.: LEGO for two-party secure computation. In: Reingold, O. (ed.) TCC 2009. LNCS, vol. 5444, pp. 368–386. Springer (Mar 2009)
23. Pappas, V., Vo, B., Krell, F., Choi, S.G., Kolesnikov, V., Bellovin, S., Keromytis, A., Malkin, T.: Blind seer: A scalable private DBMS. In: 2014 IEEE Symposium on Security and Privacy. IEEE Computer Society Press (May 2014)
24. Pinkas, B.: Fair secure two-party computation. In: Biham, E. (ed.) EUROCRYPT 2003. LNCS, vol. 2656, pp. 87–105. Springer (May 2003)
25. shelat, a., Shen, C.H.: Two-output secure computation with malicious adversaries. In: Paterson, K.G. (ed.) EUROCRYPT 2011. LNCS, vol. 6632, pp. 386–405. Springer (May 2011)
26. Woodruff, D.P.: Revisiting the efficiency of malicious two-party computation. In: Naor, M. (ed.) EUROCRYPT 2007. LNCS, vol. 4515, pp. 79–96. Springer (May 2007)
27. Yao, A.C.C.: How to generate and exchange secrets (extended abstract). In: 27th FOCS. pp. 162–167. IEEE Computer Society Press (Oct 1986)