Non-interactive Secure 2PC in the Offline/Online and Batch Settings

Payman Mohassel[1](✉) and Mike Rosulek[2]

[1] Visa Research, Palo Alto, USA
pmohasse@visa.com
[2] Oregon State University, Corvallis, USA
rosulekm@eecs.oregonstate.edu

Abstract. In cut-and-choose protocols for two-party secure computation (2PC) the main overhead is the number of garbled circuits that must be sent. Recent work (Lindell and Riva; Huang et al. Crypto 2014) has shown that in a batched setting, when the parties plan to evaluate the same function N times, the number of garbled circuits per execution can be reduced by a $O(\log N)$ factor compared to the single-execution setting. This improvement is significant in practice: an order of magnitude for N as low as one thousand. Besides the number of garbled circuits, communication round trips are another significant performance bottleneck. Afshar et al. (Eurocrypt 2014) proposed an efficient cut-and-choose 2PC that is round-optimal (one message from each party), but in the single-execution setting.

In this work we present new malicious-secure 2PC protocols that are round-optimal and also take advantage of batching to reduce cost. Our contributions include:

- A 2-message protocol for batch secure computation (N instances of the same function). The number of garbled circuits is reduced by a $O(\log N)$ factor over the single-execution case. However, other aspects of the protocol that depend on the input/output size of the function do not benefit from the same $O(\log N)$-factor savings.
- A 2-message protocol for batch secure computation, in the random oracle model. All aspects of this protocol benefit from the $O(\log N)$-factor improvement, except for small terms that do not depend on the function being evaluated.
- A protocol in the offline/online setting. After an offline preprocessing phase that depends only on the function f and N, the parties can securely evaluate f, N times (not necessarily all at once). Our protocol's online phase is only 2 messages, and the total online communication is only $\ell + O(\kappa)$ bits, where ℓ is the input length of f and κ is a computational security parameter. This is only $O(\kappa)$ bits more than the information-theoretic lower bound for malicious 2PC.

M. Rosulek—Partially supported by NSF awards 1149647 and 1617197.

J.-S. Coron and J.B. Nielsen (Eds.): EUROCRYPT 2017, Part III, LNCS 10212, pp. 425–455, 2017.
DOI: 10.1007/978-3-319-56617-7_15

1 Introduction

Secure two-party computation (2PC) allows two parties to compute a function of their inputs without revealing any other information. Yao's garbled circuit protocol [39] provides an efficient general-purpose 2PC in presence of semi-honest adversaries and has been the subject of various optimization [22,23,34,41]. The most common approach for obtaining security against malicious adversaries is the cut-and-choose paradigm wherein multiple circuits are garbled and a subset of them are opened to check for correctness, while the remaining circuits are evaluated to obtain the final output. A large body of work has focused on making cut-and-choose 2PC more efficient by (i) reducing the number of garbled circuits [15,24–26,36], (ii) minimizing rounds of interaction [1,9], and (iii) optimizing techniques for checking consistency of inputs to the computation [25,27–29,36,37].

Until recently, all protocols for cut-and-choose 2PC required at least 3λ garbled circuits in order to ensure the majority output is correct with probability $1-2^{-\lambda}$. Lindell [24] proposed a new technique for recovering from cheating that only relied on evaluation of one correct garbled circuit, hence reducing the number of garbled circuits to λ. The recent independent work of Lindell and Riva [26], and Huang et al. [15], building on ideas from earlier work of [11,31], showed how to further reduce the number of circuits to $\lambda/O(\log N)$ per execution, when performing N instances of 2PC for the same function. This leads to significant reduction in amortized communication and computation. For example for $N = 1024$, only 4 garbled circuits per execution are sufficient to achieve cheating probability of less than 2^{-40}. However, the proposed constructions require at least 4 rounds of interaction between the parties, rendering round complexity the main bottleneck when communicating over the internet as demonstrated in the recent implementation of [27].

Previous Two-Round 2PC and Shortcomings. A **non-interactive secure computation (NISC)** protocol for general computation can be constructed from Yao's garbled circuit, non-interactive zero-knowledge proofs (NIZK), and fully-secure one-round oblivious transfer (OT): P_1, who is the evaluator of the circuit, sends the first message of the OT protocol. P_2, who is the circuit constructor, returns a garbled circuit, the second message of the OT protocol, and a NIZK proof that its message is correct. (See, for example, [7,14] for such protocols.) Unfortunately, the NIZK proof in this case requires a *non black-box* use of cryptographic primitives (namely, it must prove the correctness of each encryption in each gate of the circuit).

Efficient NISC protocols that do not require such non black-box constructions are presented in [17] based on the MPC-in-the-head technique of [18]. The complexity of the NISC protocol of [17] is $|C| \cdot \mathit{poly}(\log(|C|), \log(\lambda)) + \mathit{depth}(C) \cdot \mathit{poly}(\log(|C|), \lambda)$ invocations of a Pseudo-Random Generator (PRG), where C is a boolean circuit that computes the function of interest. (Another protocol presented in that work uses only $O(|C|)$ PRG invocations, but is based on a relaxed security notion.) Although the protocols in [17] are very efficient asymptotically, their practicality is unclear and left as an open question in [17]. For instance, the protocols combine several techniques that are very efficient asymptotically, such as scalable MPC and using expanders in a non black-box way, each of which contributes large constant factors to the concrete complexity.

Afshar et al. [1], proposed a cut-and-choose 2PC with only two rounds of interaction, with concrete efficiency comparable to the state-of-the-art single-execution cut-and-choose 2PC. It is not clear how to adapt their solution to the batched execution setting to achieve better amortized efficiency. In particular, in batched cut-and-choose protocols, the sender generates and sends many garbled circuits. The receiver chooses a random subset of these circuits to check, and randomly arranges the remaining circuits into *buckets.* The kth bucket contains the circuits that will be evaluated in the kth execution. A main step for turning such a protocol into a NISC is a non-interactive mechanism for the "cut-and-choose" step and the bucket assignment. While in the single-execution setting this can be easily done using one OT per circuit [1], the task is more challenging when assigning many circuits to N buckets.

However, a bigger challenge is that the sender has no way of knowing *a priori* to which execution (i.e., which bucket) the ith circuit will be assigned. We must design a mechanism whereby the receiver can learn garbled inputs of the ith circuit that encode the input to kth execution, *if and only if* circuit i is assigned to the kth execution. Furthermore, in a typical cut-and-choose protocol, different mechanisms must be designed for checking consistency of the sender's and the receiver's inputs. For example, the sender must convince the receiver that all circuits in a particular bucket are evaluated with the same input, even though the sender does not know in advance the association between circuits and inputs (and other sibling circuits). Similarly, cheating-recovery enables the receiver to learn the sender's input if two valid circuits return different outputs in the same bucket. However, existing techniques implicitly assume the sender knows all circuits assigned to the same bucket, for example, by using the same wire labels on output wires of those circuits.

To further highlight the difficulty, consider a simple solution where for each garbled circuit GC_i, the sender prepares its garbled inputs and the input-consistency gadgets for all N possible bucket assignments and all inputs x_k, $k \in [N]$. Then, for each circuit parties perform a 1-out-of-N OT where the receiver's input is the index k such that GC_i is assigned to bucket k, and the sender's inputs are the N input garblings/gadgets for GC_i. First, note that this is prohibitively expensive as it needs to be repeated for each circuit and incurs a multiplicative factor of $N^2\lambda/\log N$ on input-related gadgets/commitments (compared to the expected $N\lambda/logN$ or $N\lambda$). Second, this still does not address how to route receiver's garbled input, and more importantly, how to incorporate cheating-recovery techniques since the existing solutions also depend on the choice of sibling circuits that are assigned to the same bucket.

Our Results. As discussed above, with current techniques, one either obtains a two-round cut-and-choose 2PC that requires λ circuits per execution or a multiple-round 2PC that requires $O(\lambda/\log N)$ circuits per execution. *The main* *question motivating this work is whether we can obtain the best of both worlds while maintaining concrete efficiency.* Our results are several protocols that achieve different combinations of features (summarized in Table 1):

- We propose the first cut-and-choose 2PC with two rounds of interaction that only requires $O(N\lambda/\log N)$ garbled circuits to evaluate a function N times in a single batch. The protocol is both asymptotically and concretely efficient and can be instantiated in the standard model and using only symmetric-key operations in the OT-hybrid model.
- In the above protocol, the number of garbled circuits is reduced by a factor $O(\log N)$ compared to the single-execution setting. This is the only part of the protocol whose cost depends on the size of the circuit for f. However, several mechanisms in the protocol depend on the input/output length of f, and these mechanisms scale as $O(N\lambda)$ instead of $O(N\lambda/\log N)$.

  We therefore describe a two-round protocol for batched 2PC *in the random oracle model*, in which all aspects of the protocol benefit from batching. That is, apart from protocol features that do not depend on f at all, the entire protocol scales with $O(\kappa N/\log N)$ rather than $O(\kappa N)$. Unfortunately, the number of garbled circuits now depends on the (larger) computational security parameter κ rather than the statistical security parameter λ as before. This is due to technical reasons (see Sect. 6.2).
- In the offline-online setting, parties perform dedicated offline preprocessing that depends only on the function f and number of times N they would like to evaluate it. Then, when inputs are known, the parties can engage in an online phase to securely obtain the output. The online phases need not be performed in a single batch—they can happen asynchronously.

  We describe a 2PC protocol in this offline-online setting. As in other offline-online protocols [15,26,35], the total costs are reduced by a $O(\log N)$ factor (and the number of circuits is dependent on the statistical security parameter λ). Unlike previous protocols, our online phase consist of only 2 rounds. The total online communication can be reduced to only $|x| + |y| + O(\kappa)$ bits, where x is the sender's input, y is the receiver's input, and κ is a computational security parameter. We note that $|x| + |y|$ bits of communication are required for malicious-secure 2PC,[1] so our protocol has nearly optimal online communication complexity.

Our Techniques. Our main NISC construction takes advantage of a two-round protocol for obliviously mapping garbled circuits and their associated input/output gadgets to many buckets while hiding from the garbler the bucket assignment and consequently what inputs a circuits would be evaluated on. As a result, we need to extend and adapt all existing techniques for obtaining garbled inputs, performing input consistency checks and cheating-recovery to this new setting.

[1] Each party must send a message at least as long as his/her input, otherwise it is information-theoretically impossible for the simulator to extract a corrupt party's input.

Table 1. Asymptotic efficiency of our protocols. $n_{\mathsf{in}}, n_{\mathsf{out}}$ are number of input/output wires. $n_{\mathsf{both}} = n_{\mathsf{in}} + n_{\mathsf{out}}$. Rounds are listed as offline + online. κ is the computational security parameter, and λ is the statistical security parameter.

	NISC	RO-NISC	Online-offline
Rounds	$0 + 2$	$0 + 2$	$2 + 2$
# GC	$O(N\lambda/\log N)$	$O(N\kappa/\log N)$	$O(N\lambda/\log N)$
# plain commit	$O(n_{\mathsf{in}}N\lambda/\log N)$	$O(n_{\mathsf{in}}N\kappa/\log N)$	$O(n_{\mathsf{in}}N\lambda/\log N)$
# hom commit	$O(n_{\mathsf{out}}N\lambda/\log N)$	$O(n_{\mathsf{out}}N\kappa/\log N)$	$O(n_{\mathsf{out}}N\lambda/\log N)$
OSN OTs	$O(n_{\mathsf{both}}N\lambda)$	-	-
Other OTs	$O(n_{\mathsf{in}}N)$	$O(n_{\mathsf{in}}N)$	$O(n_{\mathsf{in}}N)$

Another main ingredient of our constructions is a homomorphic commitment scheme with homomorphic properties on the decommitment strings. Such a primitive can be efficiently instantiated using both symmetric-key and public-key primitives, trading-off communication for computation. We show how such a commitment scheme combined with an oblivious switching network protocol [30] allows a sender to obliviously open linear relations between various committed values without *a priori* knowledge of the choice of committed values. See Sect. 4.1 for a detailed overview of the techniques used in our main protocol.

2 Preliminaries

2.1 Garbled Circuits

Garbled Circuits were first introduced by Yao [40]. A garbling scheme consists of a garbling algorithm that takes a random seed σ and a function f and generates a garbled circuit F and a decoding table dec; the encoding algorithm takes input x and the seed σ and generates garbled input $\widehat{x}$; the evaluation algorithm takes $\widehat{x}$ and F as input and returns the garbled output $\widehat{z}$; and finally, a decoding algorithm that takes the decoding table dec and $\widehat{z}$ and returns $f(x)$. We require the garbling scheme to satisfy the standard security properties formalized in [6]. Our construction uses the garbling scheme in a black-box way and hence can incorporate all recent optimizations proposed in the literature. In the offline-online setting, the scheme needs to adaptively secure in the sense of [5].

2.2 Commitments

A standard commitment scheme Com allow a party to commit to a message m, by computing $C = \mathsf{Com}(m; d)$ using a decommitment d. To open a commitment $C = \mathsf{Com}(m; d)$, the committer reveals (m, d). The verifier recomputes the commitment and accepts if it obtains the same C, and rejects otherwise. We require standard standalone security properties of a commitment scheme:

- *Hiding:* For any a, b, the distributions $\mathsf{Com}(a; d_a)$ and $\mathsf{Com}(a; d_b)$, induced by random choice of d_a, d_b, are indistinguishable.
- *Binding:* It is computationally infeasible to compute $m \neq m', d, d'$ such that $\mathsf{Com}(m; d) = \mathsf{Com}(m'; d')$.

Homomorphic Commitments. In a homomorphic commitment scheme HCom, we further require the scheme to be homomorphic with respect to an operation on the message space denoted by $\oplus$. In particular given two commitments $C_a = \mathsf{HCom}(a, d_a)$ and $C_b = \mathsf{HCom}(b, d_b)$, the committer can open $a \oplus b$ (revealing nothing beyond $a \oplus b$) by giving $d_a \oplus d_b$.

Note that here we have assumed that the homomorphic operation also operates on the decommitment values. This is indeed the case for most instantiations of homomorphic commitments, as we discuss in Sect. 5.2. The security properties are extended for homomorphic commitments as follows:

- *Hiding:* For a set of values $v_1, \ldots, v_n$ and a set $S \subseteq [n]$, define $v(S) = \oplus_{i \in S} v_i$. Then, informally, the hiding property is that commitments to $v_1, \ldots, v_n$ and openings of $v(S_1), \ldots, v(S_k)$ reveal no more than the $v(S_1), \ldots, v(S_k)$ values. More formally, for all $\boldsymbol{v} = (v_1, \ldots, v_n), \boldsymbol{v'} = (v'_1, \ldots, v'_n)$, and sets $S_1, \ldots, S_k$ where $v(S_j) = v'(S_j)$ for each j, the following distributions are indistinguishable:

$$(\mathsf{Com}(v_1; d_1), \ldots, \mathsf{Com}(v_n; d_n); d(S_1), \ldots, d(S_k)),$$
$$\text{and } (\mathsf{Com}(v'_1; d_1), \ldots, \mathsf{Com}(v'_n; d_n); d(S_1), \ldots, d(S_k))$$

- *Binding:* Intuitively, it should be hard to decommit to inconsistent values. More formally, it should be hard to generate commitments $C_1, \ldots, C_n$ and values $\{(S_j, d_j, m_j)\}_j$ such that d_j is a valid decommitment of $\bigoplus_{i \in S_j} C_i$ to the value m_j, and yet there is no solution (in the x_i's) to the system of equations defined by equations: $\left\{ \bigoplus_{i \in S_j} x_i = m_j \right\}_j$.

2.3 Probe-Resistant Input Encoding

In garbled-circuit-based 2PC, the receiver uses oblivious transfers to pick up his garbled inputs. A standard problem is that a malicious sender can give incorrect wire labels in these OTs. Furthermore, if the sender gives an incorrect value for only one of the pair of wire labels, then the receiver picks up incorrect values (and presumably aborts), *based on his private input.* Hence, a malicious sender causes the receiver to abort, depending on the receiver's private input. This cannot be simulated in the ideal world, so it is indeed an attack.

A standard way to deal with this is the idea of a probe-resistant matrix:

Definition 1 [25,37]**.** *A boolean matrix $M \in \{0,1\}^{n \times n'}$ is λ-**probe resistant** if for all $R \subseteq [n]$, the Hamming weight of $\bigoplus_{i \in R} M_i$ is at least λ, where M_i denotes the ith row of M.*

The idea is for Bob, with input y to choose a random encoding $\tilde{y}$ such that $M\tilde{y} = y$. Then the parties will evaluate the function $\tilde{f}(x, \tilde{y}) = f(x, M\tilde{y}) = f(x, y)$. The matrix M can be public, so the computation $M\tilde{y}$ uses only XOR operations (free in a typical garbling scheme [23]).

Suppose the parties perform n' OTs. In each OT the sender provides two items, and the receiver uses the bits of $\tilde{y}$ to select one. The items can be either *good* or *bad*, and the receiver will abort if it receives any *bad* item. If for any single OT, both inputs are *bad*, then the receiver will always abort. However, if every OT has at least one *good* item, then the receiver will abort based on $\tilde{y}$.

Lemma 2 [25,37]**.** *Suppose M is λ-probe-resistant, and fix a set of sender's inputs to the OTs as described above. Let $P(y)$ denote the probability that the receiver aborts (i.e., sees a bad item) when it chooses a random $\tilde{y}$ such that $M\tilde{y} = y$, and uses $\tilde{y}$ as the choice bits in the OTs. Then for all y, y', we have $|P(y) - P(y')| = O(2^{-\lambda})$.*

Hence, the abort probability is *nearly independent* of the receiver's input, when using this probe-resistant technique.

2.4 Secure Computation and the NISC Model

We consider security in the *universal composability* framework of Canetti [8]. We refer the reader to that work for detailed security definitions. Roughly speaking, the definition considers a *real interaction* and an *ideal one.*

In the *real* interaction, parties interact in the protocol. Their inputs are chosen by an *environment*, and their outputs are given to the environment. An adversary who attacks the protocol takes control of one of the parties and causes it to arbitrarily deviate from the protocol. The adversary may also communicate arbitrarily with the environment before/during/after the protocol interaction.

In the *ideal* interaction, parties simply forward their inputs to a trusted party called a *functionality.* They receive output from the functionality which they forward to the environment.

A protocol **UC-securely realizes** an ideal functionality if, for all adversaries attacking the real world, there exists an adversary in the ideal world (called a simulator) such that for all environments, the view of the environment is indistinguishable between the real and ideal interactions.

NISC. Ishai et al. [17] defined a special model of secure computation called *non-interactive secure computation (NISC).* A protocol is NISC if it consists of a single message from one party to the other, possibly with some (static, parallel) calls to some ideal functionality (typically an oblivious transfer functionality).

One can think of replacing the calls to an ideal oblivious transfer functionality with a two-round secure OT protocol (like that of [33]). Then the NISC protocol becomes a two-message protocol: in the first message the OT receiver sends the first OT protocol message. In the second message, the OT sender sends the OT response along with the single NISC protocol message.

2.5 Correlation Robust

One of our techniques requires a correlation-robust hash function. This property was defined in Ishai et al. [16].

Definition 3 [16]**.** *A function $H : \{0,1\}^\kappa \to \{0,1\}^n$ is* ***correlation robust*** *if $F(s,x) = H(x \oplus s)$ is a weak PRF (with s as the seed). In other words, the distribution of: $\Big(x_1, \ldots, x_m; H(x_1 \oplus s), \ldots, H(x_m \oplus s)\Big)$ is pseudorandom, for random choice of x_i's and s.*

2.6 Compressed Garbled Inputs

Applebaum et al. [2] described a technique for randomized encodings with low online complexity. In the language of garbled circuits, this corresponds to a way to compress garbled inputs in the online phase of a protocol, at the expense of more data in an offline phase. We abstract their primitive as a **garbled input compression** scheme, as follows.

Let $\boldsymbol{e} = (e_{1,0}, e_{1,1}, \ldots, e_{n,0}, e_{n,1})$ be a set of wire labels (i.e., $e_{j,b}$ is the wire label encoding value b on wire j). In a traditional protocol, the garbled encoding of a string x is $(e_{1,x_1}, \ldots, e_{n,x_n})$, which is sent in the online phase of the protocol. Using the approach of [2], we can do the following to reduce the online cost:

- In an offline phase, the garbler runs $\mathsf{Compress}(\boldsymbol{e}) \to (sk, \widehat{e})$, and sends $\widehat{e}$ to the evaluator.
- In the online phase, when garbled encoding of x is needed, the garbler runs $\mathsf{Online}(sk, x) \to \widehat{x}$ and sends $\widehat{x}$ to the evaluator.
- The evaluator runs $\mathsf{Decompress}(\widehat{e}, x, \widehat{x})$, which returns the garbled encoding $(e_{1,x_1}, \ldots, e_{n,x_n})$.

The security of the compression scheme is that $(\widehat{e}, \widehat{x}, x)$ can be simulated given only the garbled encoding $(e_{1,x_1}, \ldots, e_{n,x_n})$. In other words, the compressed encoding reveals no more than the expected garbled encoding.

In a traditional garbling scheme, the size of the garbled encoding is $n\kappa$. Applebaum et al. [2] give constructions where the online communication $\widehat{x}$ has size only $n + O(\kappa)$. These constructions are proven secure under a variety of assumptions (DDH, LWE, RSA). We refer the reader to their paper for details.

3 Switching Networks

3.1 Definitions

A **switching network** is a circuit of gates that we call **switches**, whose behavior is described below. The network as a whole has n *primary* inputs (strings, or more generally, elements from some group) and p *programming* inputs (bits). All wires in the network have no branching. Each **switch** has two inputs and two outputs. A switch is parameterized by an index $j \in [p]$. The behavior of an individual switch is that when its primary input wires have values (X, Y) and the jth programming input to the circuit is 0, then the outputs are (X, Y); otherwise (the jth programming input is 1) the outputs are (Y, X).

Note that many switches can be tied to the same programming input. When $\mathcal{S}$ is a switching network and π is a programming string, we let $\mathcal{S}^{\pi}(X_1, \ldots, X_n)$ denote the output of the switching network when the primary inputs are $X_1, \ldots, X_n$ and its programming input is π.

3.2 Oblivious Switching Network Protocol

In the full version, we describe the **oblivious switching network (OSN) protocol** of [30]. The idea is that the parties agree on a switching network $\mathcal{S}$. The sender has inputs $(X_1, \ldots, X_n)$ and $(Z_1, \ldots, Z_m)$. The receiver has input π, and learns $\mathcal{S}^{\pi}(X_1, \ldots, X_n) \oplus (Z_1, \ldots, Z_m)$. The sender learns nothing.

The cost of the protocol is essentially a 1-out-of-2 OT (for values on the switching network's wires) for each switch in the network. All of the OTs can be performed in parallel, and hence the protocol can be realized as a NISC protocol in the OT-hybrid model.

This protocol will be used as a subroutine in our main NISC functionality. Yet we do *not* abstract the OSN protocol in terms of an ideal functionality. This is because the protocol does not ensure that a malicious sender acts consistently with the switching network. However, this turns out to be non-problematic in our larger NISC protocol. We simply abstract out the properties of this subprotocol as follows:

Observation 4. *When the sender is honest and the receiver is corrupt, the simulator can extract the corrupt receiver's programming string π. When the underlying OTs are performed in parallel, the simulator extracts π before simulating any outputs from these OTs.*

Observation 5. *When the sender is honest, the receiver's view can be simulated given only π and the output $\mathcal{S}^{\pi}(X_1, \ldots, X_n) \oplus (Z_1, \ldots, Z_m)$.*

While we described the OSN protocol for the $\oplus$ operation, we note that it is easy to replace $\oplus$ for any group operations. In particular, we also use the protocol in scenarios where $\oplus$ represent homomorphic operations on message domain and/or decommitment domain of a homomorphic commitment.

4 Batched NISC

In this section we describe a protocol for securely evaluating many instances of the same function f in a single batch. The ideal functionality we achieve is described in Fig. 1.

We let N denote the number of instances of 2PC being executed, $\widehat{N}$ the number of garbled circuits computed and B the number of garbled circuits assigned to each execution/bucket. For a full treatment of these parameters, we refer the

Parameters: A function f and number N of instances.

Behavior: On input $(y_1, \dots, y_N)$ from the receiver, internally record these values and send (input) to the sender. Later, on input $(x_1, \dots, x_N)$ from the sender, do the following. If $x_i = \perp$ for any i, then give output $\perp$ to the receiver. Otherwise compute $z_i = f(x_i, y_i)$ for $i \in [N]$ and give $(z_1, \dots, z_N)$ to the receiver.

Fig. 1. Ideal functionality for batch 2PC

reader to [26]. For our purposes, we will assume that the parameters satisfy the following combinatorial property: The adversary generates $\widehat{N}$ items, some *good*, some *bad*. The items are randomly assigned into N buckets of B items each. The remaining $\widehat{N} - NB$ items are *opened*. Then the probability that all opened items are *good* while there exists a bucket with *all bad* items is at most $2^{-\lambda}$. Here λ is a statistical security parameter (often $\lambda = 40$). Asymptotically, $\widehat{N} = O(\lambda N / \log N)$ and $B = O(\lambda / \log N)$.

Regarding our conventions for notation: we use i to index a garbled circuit, j to index a wire in the circuit computing f, k to index a bucket (an evaluation of f, or the special "check bucket" defined below), and l to index a position within a bucket. We let $\mathsf{SendInpWires}, \mathsf{RecvInpWires}, \mathsf{OutWires}$ denote the set of wire indices corresponding to inputs of Alice, inputs of Bob, and outputs of f, respectively.

4.1 Overview of Techniques

Bucket-Coupling via Switching Networks. Recall that the receiver must choose randomly which circuits are checked, and which circuits are mapped to each bucket. For simplicity, let us say that checked circuits are assigned to "bucket #0." Recall that the cut-and-choose statistical bounds require the receiver to choose a random assignment of circuits into buckets. Suppose the cut-and-choose parameters call for N buckets, B circuits per bucket, and $\widehat{N} > NB$ total circuits (with $\widehat{N} - NB$ circuits being checked). Think of this process as first randomly permuting the $\widehat{N}$ circuits, assigning the first $\widehat{N} - NB$ circuits to bucket #0, assigning the next B circuits to bucket #1, and so on. More formally, we can define public functions bkt and pos so that, *after randomly permuting* the circuits, the ith circuit will be the $\mathsf{pos}(i)$'th circuit placed in bucket $\mathsf{bkt}(i)$.

A main building block in our NISC protocol is one we call **bucket coupling**, which is a non-interactive way to bind information related to garbled circuits to information related to a particular bucket, under a bucketing-assignment chosen by the receiver. Suppose the parties use the OSN subprotocol of Sect. 3, on a universal switching network $\mathcal{S}$, where the sender's input is $(A_1, \dots, A_{\widehat{N}}), (B_1, \dots, B_{\widehat{N}})$, and the receiver's input is the programming string for a random permutation π. Then the receiver will learn $A_{\pi(i)} \oplus B_i$.

Interpret π as the receiver's random permutation of circuits when assigning circuits to buckets as described above. Then we can interchangeably use B_v and

$B_{\mathsf{bkt}(v),\mathsf{pos}(v)}$, since there is a one-to-one correspondence between these ways of indexing. We have the following generic functionality:

Bucket coupling: The sender has an item A_i for each circuit i, and an item $B_{k,l}$ for each position l in the kth bucket. The receiver holds a bucketing assignment π. The receiver learns $A_i \oplus B_{k,l}$ if and only if π assigns circuit i to position l of bucket k.

We can perform many such couplings, *all with respect to the same permutation* π. Simply imagine a switching network that is a disjoint union of many universal switching networks, but where corresponding switches are programmed by the same programming bit (this is enforced in the OSN protocol).

Of course, our OSN protocol does not guarantee consistent behavior by the sender. Furthermore, the sender might not even use the expected inputs to the OSN protocol. However, we argue that these shortcomings do not lead to problems in our larger NISC protocol. Intuitively, the worst the sender can do is to cause inconsistent outputs for the receiver in a way that depends on the receiver's choice of bucket-assignments π. But π is chosen independently of his input to the *NISC protocol!* Hence the simulator can exactly simulate the abort probability of the honest receiver, by sampling a uniform π just as the honest receiver does.

Basic Cut-and-Choose. The sender Alice generates $\widehat{N}$ garblings $\{F_i\}_i$ of f (along with some other associated data, described below). Let σ_i denote the seed used to generate all the randomness for the ith circuit. The parties can perform a coupling whereby **Bob learns σ_i if and only if circuit i is assigned to bucket 0** (in the notation above, $A_i = \sigma_i$ and $B_{0,l} = 0^\kappa$ and $B_{k,l}$ random for $k \neq 0$). Then every circuit mapped to bucket 0 (i.e., every check circuit) can be verified by Bob.

Delivering the Receiver's Garbled Input. Let $\mathsf{RecvInpWires}$ denote the set of input wires corresponding to Bob's input to f. Let $\mathsf{in}_{i,j,b}$ denote the input wire label on the jth wire of the ith circuit, encoding logical bit b. When circuit i is mapped to bucket k, we must let Bob obtain his garbled input value $\mathsf{in}_{i,j,b}$, where b is the jth bit of Bob's input for the kth execution. Recall that the association between circuits (i) and executions (k) is not known to Alice.

Alice commits to each input wire label as follows, and sends the commitments to Bob:

$$C^{\mathsf{in}}_{i,j,b} \leftarrow \mathsf{Com}(\mathsf{in}_{i,j,b}; d^{\mathsf{in}}_{i,j,b})$$

The randomness for these commitments is derived from σ_i, so that the commitments can be checked by Bob if circuit i is assigned to be a check-circuit.

Then, for each execution $k \in [N]$ and each $j \in \mathsf{RecvInpWires}$, Alice chooses random *input tokens* $\mathsf{tok}_{k,j,0}$ and $\mathsf{tok}_{k,j,1}$. The parties use an instance of OT so that Bob picks up the correct $\mathsf{tok}_{k,j,b}$, where b is Bob's input value on wire j in the kth evaluation of f.

Let PRF be a PRF. Then for each $b \in \{0,1\}, j \in \mathsf{RecvInpWires}$ the parties perform a coupling in which **Bob learns $d^{\mathsf{in}}_{i,j,b} \oplus \mathsf{PRF}(\mathsf{tok}_{k,j,b};\ l)$ if and only if circuit i is assigned to position l of bucket k.** If Bob has input bit b on the jth wire in the kth evaluation of f, then he holds $\mathsf{tok}_{k,j,b}$ and can decrypt the corresponding $d^{\mathsf{in}}_{i,j,b}$ and use it to decommit to the appropriate input wire label for the ith garbled circuit. If he does not have input bit b, then these outputs of the coupling subprocess look independently pseudorandom by the guarantee of the PRF.

If Alice sends inconsistent values into the coupling, then Bob may not receive the decommitment values $d^{\mathsf{in}}_{i,j,b}$ he expects. If this happens, then Bob aborts. Because this abort event would then depend on Bob's private input, we have Bob encode his input in a λ-probe-resistant encoding, following the discussion in Sect. 2.3. This standard technique makes Bob's abort probability independent of his private input.

Enforcing Consistency of Sender's Inputs. We must ensure that Alice uses the same input for all of the circuits mapped to a particular bucket $k \neq 0$, despite Alice not knowing which circuits will be assigned to that bucket. This must furthermore be done without leaking Alice's input to Bob in the process.

We use an approach similar to [27] based on a XOR-homomorphic commitment scheme. But here the sender does not know *a priori* which committed values' XOR it needs to open. Hence, we need a mechanism for letting the receiver obliviously learn the decommitment strings for XOR of the appropriate committed values.

For each circuit i, we have Alice choose a random string s_i and commit individually to all of her input wire labels, permuted according to s_i. More precisely, she computes commitments:

$$C^{\mathsf{in}}_{i,j,0} \leftarrow \mathsf{Com}(\mathsf{in}_{i,j,s_{i,j}}; d^{\mathsf{in}}_{i,j,0})$$
$$C^{\mathsf{in}}_{i,j,1} \leftarrow \mathsf{Com}(\mathsf{in}_{i,j,\overline{s_{i,j}}}; d^{\mathsf{in}}_{i,j,1})$$

Here $s_{i,j}$ denotes the jth bit of s_i. Hence $C^{\mathsf{in}}_{i,j,b}$ is a commitment to the input wire label representing *truth value* $b \oplus s_{i,j}$.

Alice also commits to s_i under a homomorphic commitment scheme $C^s_i \leftarrow \mathsf{HCom}(s_i; d^s_i)$. As before, the randomness used in all of these commitments is derived from σ_i so the commitments can be checked in the cut-and-choose.

For each bucket k, Alice gives a homomorphic commitment to x_k, her input in that execution—$C^x_k \leftarrow \mathsf{HCom}(x_k; d^x_k)$. The parties perform a coupling so that **Bob learns $d^s_i \oplus d^x_k$ iff circuit i is assigned to bucket k.** The result is a decommitment value that Bob can use to learn $s_i \oplus x_k$. The soundness of the commitment scheme ensures that Bob knows values $o_i = s_i \oplus x_k$ for a *consistent* x_k. Given that the commitments to Alice's input wires $(C^{\mathsf{in}}_{i,j,b})$ are arranged/permuted using s_i (a property enforced with high probability by the cut-and-choose), the commitments indexed by o_i correspond to the garbled inputs that encode the logical value x_k. Hence, to ensure that Alice uses consistent inputs within each bucket, Bob expects Alice to open the commitments indexed by o_i.

Routing the Sender's Inputs. We must let Bob obtain garbled inputs encoding Alice's inputs to the ith garbled circuit. As above, when circuit i is mapped to bucket k, it suffices to let Bob learn the decommitment to $C^{\mathsf{in}}_{i,j,o_{i,j}}$ where $o_i = s_i \oplus x_k$. The challenge is to accomplish this without Alice knowing *a priori* which circuit i will be assigned to which bucket k, and hence which input x_k needs to be garbled. We propose a novel and efficient technique for this step that, for each input wire, only requires one symmetric-key operation and the routing of one string of length κ through the switching network.

For each wire $j \in \mathsf{SendInpWires}$, Alice chooses random Δ_j. As a matter of notation, when b is a bit, we let $b\Delta_j$ denote the value [if $b = 0$ then 0^κ else Δ_j].

For each circuit i and wire $j \in \mathsf{SendInpWires}$, Alice chooses random $r_{i,j}$ and sends an encryption $e_{i,j,b} = H(r_{i,j} \oplus b\Delta_j) \oplus d^{\mathsf{in}}_{i,j,b}$ to Bob. Here H is a correlation-robust hash function (Sect. 2.5).

For each wire $j \in \mathsf{SendInpWires}$ the parties perform a coupling in which **Bob learns** $(r_{i,j} \oplus s_{i,j}\Delta_j) \oplus x_{k,j}\Delta_j$ **if and only if circuit** i **is assigned to bucket** k**.** Simplifying, we see that Bob learns:

$$K_{i,j} = (r_{i,j} \oplus s_{i,j}\Delta_j) \oplus x_{k,j}\Delta_j = r_{i,j} \oplus (s_{i,j} \oplus x_{k,j})\Delta_j = r_{i,j} \oplus o_{i,j}\Delta_j$$

Indeed, this is the key that Bob can use to decrypt $e_{i,j,o_{i,j}}$ to obtain $d^{\mathsf{in}}_{i,j,o_{i,j}}$. He can then use this value to decommit to the wire label encoding truth value $x_{k,j}$, as desired. Bob will abort if he is unable to decommit to the expected wire labels in this way. Here, the abort probability depends only on Alice's behavior, and is not influenced by Bob's input in any way.

Note that the decommitment values for the "other" wire labels are masked by a term of the form $H(K_{i,j} \oplus \Delta_j)$, where Δ_j is unknown to Bob. Even though the same Δ_j is used for many such ciphertexts, the correlation-robustness of H ensures that these masks look random to Bob.

Cheating Recovery. Lindell [24] introduced a *cheating recovery* technique, where if the receiver detects the sender cheating, the receiver is able to learn the sender's input (and hence evaluate the function in the clear). This technique is crucial in reducing the number of garbled circuits, since now only a *single* circuit in a bucket needs to be correctly generated. Our protocol also adapts this technique, but in a non-interactive setting. The approach here is similar to that used in [1], but it is describe more generally in terms of any homomorphic commitment scheme and of course adapted to the batch setting.

For each output bit j and each bucket k, Alice generates $\mathsf{w}_{k,j,0}$ at random and sets $\mathsf{w}_{k,j,1} = x_k - \mathsf{w}_{k,j,0}$. The main idea is two-fold:

- We will arrange so that if Bob evaluates *any* circuit in bucket k and obtains output b on wire j, then Bob will learn $\mathsf{w}_{k,j,b}$.

– Then, if Bob evaluates two circuits in the same bucket that disagree on their output—say, they disagree on output bit j—then Bob can recover Alice's input $x_k = \mathsf{w}_{k,j,0} + \mathsf{w}_{k,j,1}$.

For technical reasons, we must introduce *pre-output* and *post-output* wire labels for each garbled circuit. When evaluating a garbled circuit, the evaluator obtains *pre-output* wire labels. We denote by $d^{\mathsf{out}}_{i,j,b}$ the pre-output wire label for wire j of circuit i encoding truth value b. We use this notation since the pre-output wire labels are used as decommitment values.

Alice chooses random *post-output* wire labels, $\{\mathsf{out}_{i,j,b}\}$ and generates a homomorphic commitment to them using the pre-output labels as the randomness:

$$C^{\mathsf{out}}_{i,j,b} \leftarrow \mathsf{HCom}(\mathsf{out}_{i,j,b}; d^{\mathsf{out}}_{i,j,b})$$

The technical reason for having both pre- and post-output labels is so that there is a homomorphic commitment that is bound to each output wire of each circuit, that *can be checked* in the cut-and-choose. Indeed, these commitments can be checked in the cut-and-choose, since they use the circuit's [pre-]output wire labels as their randomness.

Separately, for each bucket $k \neq 0$, Alice generates and sends homomorphic commitments:

$$C^{\mathsf{w}}_{k,j,b} \leftarrow \mathsf{HCom}(\mathsf{w}_{k,j,b}; d^{\mathsf{w}}_{k,j,b})$$

She sends a homomorphic opening to the linear expression $\mathsf{w}_{k,j,0} + \mathsf{w}_{k,j,1} - x_k$, to prove that this expression is all-zeroes (*i.e.*, to prove that $\mathsf{w}_{k,j,0} + \mathsf{w}_{k,j,1} = x_k$).

Then, for each $j \in$ *outpwires* and $b \in \{0,1\}$ the parties do a coupling in which **Bob learns $d^{\mathsf{out}}_{i,j,b} \oplus d^{\mathsf{w}}_{k,j,b}$ when circuit i is assigned to bucket k.** Bob can use the result to decommit to the value of $\mathsf{out}_{i,j,b} \oplus \mathsf{w}_{k,j,b}$.

Putting things together, Bob evaluates a circuit i assigned to bucket k. He learns the corresponding pre-output wire labels $d^{\mathsf{out}}_{i,j,b}$, which he uses to decommit to the post-output wire labels $\mathsf{out}_{i,j,b}$. Since he has learned $\mathsf{out}_{i,j,b} \oplus \mathsf{w}_{k,j,b}$ from the coupling, he can therefore compute $\mathsf{w}_{k,j,b}$ (a *bucket-specific* value, whereas $\mathsf{out}_{i,j,b}$ was a *circuit-specific* value). If any two circuits disagree in their output, he can recover the sender's input x_k as described above and compute the correct output. Otherwise, since at least one circuit in the bucket is guaranteed (by the cut-and-choose bounds) to be generated honestly, Bob can uniquely identify the correct output.

4.2 Detailed Protocol Description

We present our complete protocol in Fig. 2. We refer the reader to the full version for the proof of the following Theorem.

Theorem 6. *The protocol in Fig. 2 is a UC-secure realization of the functionality in Fig. 1.*

Parameters: A function f and number N of instances. $\widehat{N}$ denotes the number of garbled circuits, chosen according to the discussion in the text. λ is the statistical security parameter.

Inputs: Alice has inputs $(x_1, \ldots, x_N)$ and Bob has inputs $(y_1, \ldots, y_N)$.

1. Bob chooses a random permutation π, and uses it as input to all coupling subprotocols below (i.e., all couplings are performed in parallel and bound to the same π). The parties agree on a λ-probe resistant matrix M, and Bob encodes each y_k as $\tilde{y}_k$ where $M\tilde{y}_k = y_k$.
2. For each circuit $i \in [\widehat{N}]$: Alice chooses a PRF seed σ_i and uses it to derive *all randomness used in this step* of the protocol:
   Alice generates a garbling of the function $\tilde{f}(x, \tilde{y}) = f(x, M\tilde{y})$; let F_i denote the garbled circuit, and let $\mathsf{in}_{i,j,b}$ (resp. $d^{\mathsf{out}}_{i,j,b}$) denote the input (resp. output) wire label encoding truth value b on wire j of circuit i. She sends each F_i to Bob.
   Alice chooses random "post-output" keys $\{\mathsf{out}_{i,j,b}\}_{j \in \mathsf{OutWires}, b \in \{0,1\}}$. She generates and sends the following commitments (where d^{in} and d^s values are derived randomly from σ_i):
   $$\begin{aligned} C^{\mathsf{in}}_{i,j,b} &\leftarrow \mathsf{Com}(\mathsf{in}_{i,j,b \oplus s_{i,j}}; d^{\mathsf{in}}_{i,j,b \oplus s_{i,j}}) && \text{for } j \in \mathsf{SendInpWires}, b \in \{0,1\} \\ C^{\mathsf{in}}_{i,j,b} &\leftarrow \mathsf{Com}(\mathsf{in}_{i,j,b}; d^{\mathsf{in}}_{i,j,b}) && \text{for } b \in \{0,1\}, j \in \mathsf{RecvInpWires} \\ C^{\mathsf{out}}_{i,j,b} &\leftarrow \mathsf{HCom}(\mathsf{out}_{i,j,b}; d^{\mathsf{out}}_{i,j,b}) && \text{for } b \in \{0,1\}, j \in \mathsf{OutWires} \\ C^s_i &\leftarrow \mathsf{HCom}(s_i; d^s_i) \end{aligned}$$
3. The parties perform a coupling with input for Alice $\{\sigma_i\}_i$, all-zeroes masks for bucket #0, and random masks for other buckets. Bob learns σ_i if circuit i is mapped to bucket 0. For such i, Bob checks that F_i and corresponding commitments from the previous step are generated using randomness derived from σ_i, and aborts if this is not the case.
4. For $j \in \mathsf{SendInpWires}$, Alice chooses a random Δ_j. For $j \in \mathsf{SendInpWires}, i \in [\widehat{N}]$, Alice chooses a random $r_{i,j}$. Alice generates and sends input-encryptions:
   $$e_{i,j,b} = H(r_{i,j} \oplus b\Delta_j) \oplus d^{\mathsf{in}}_{i,j,b}$$
5. For $k \in [N], j \in \mathsf{OutWires}$, Alice chooses random $\mathsf{w}_{k,j,0}$ and sets $\mathsf{w}_{k,j,1} = x_k \oplus \mathsf{w}_{k,j,0}$ (recall x_k is her input to the kth execution). Alice generates and sends commitments:
   $$\begin{aligned} C^{\mathsf{w}}_{k,j,b} &\leftarrow \mathsf{HCom}(\mathsf{w}_{k,j,b}; d^{\mathsf{w}}_{k,j,b}) && \text{for } k \in [N], j \in \mathsf{OutWires}, b \in \{0,1\} \\ C^x_k &\leftarrow \mathsf{HCom}(x_k; d^x_k) && \text{for } k \in [N] \end{aligned}$$
   Alice also gives homomorphic decommitments:
   $$d^{\mathsf{w}}_{k,j,0} \oplus d^{\mathsf{w}}_{k,j,1} \oplus d^x_k \qquad \text{for } k \in [N], j \in \mathsf{OutWires}$$
   Bob aborts if these values do not decommit $C^{\mathsf{w}}_{k,j,0} \oplus C^{\mathsf{w}}_{k,j,1} \oplus C^x_k$ to the all-zeroes string.

(protocol description continues...)

Fig. 2. Batch NISC protocol

6. For $k \in [N], j \in \mathsf{RecvInpWires}$, Alice chooses random $\mathsf{tok}_{k,j,0}, \mathsf{tok}_{k,j,1}$. Parties engage in an instance of OT with inputs $(\mathsf{tok}_{k,j,0}, \mathsf{tok}_{k,j,1})$ for Alice and $\tilde{y}_{k,j}$ (i.e., jth bit of $\tilde{y}_k$) for Bob. Bob gets input $\mathsf{tok}_{k,j,\tilde{y}_{k,j}}$.
7. For $k \in [N], j \in \mathsf{RecvInpWires}, b \in \{0,1\}$, the parties perform a coupling with inputs $\{d^{\mathsf{in}}_{i,j,b}\}_i, \{\mathsf{PRF}(\mathsf{tok}_{k,j,b}; l)\}_{k,l}$ for Alice. Bob learns $\beta_{i,j,b} = d^{\mathsf{in}}_{i,j,b} \oplus \mathsf{PRF}(\mathsf{tok}_{k,j,b}; l)$ when circuit i is assigned to position l of bucket k. Bob aborts if $\beta_{i,j,\tilde{y}_{i,j}} \oplus \mathsf{PRF}(\mathsf{tok}_{k,j,\tilde{y}_{i,j}}; l)$ is not a valid decommitment of $C^{\mathsf{in}}_{i,j,\tilde{y}_{i,j}}$. Otherwise, Bob sets $\mathsf{in}^*_{i,j}$ to be the result of the decommitment.
8. The parties perform a coupling with input $\{d^s_i\}_i, \{d^x_k\}_k$ for Alice. Bob learns $d^s_i \oplus d^x_k$ when circuit i is assigned to bucket k, and aborts if this is not a valid opening of $C^s_i \oplus C^x_k$. Otherwise, Bob sets o_i to be the result of this decommitment.
9. For $k \in [N], j \in \mathsf{SendInpWires}$ the parties perform a coupling with input $\{r_{i,j} \oplus s_{i,j}\Delta_j\}_i, \{x_{k,j}\Delta_j\}_k$ for Alice. Bob learns $K_{i,j} = (r_{i,j} \oplus s_{i,j}\Delta_j) \oplus x_{k,j}\Delta_j$ when circuit i is assigned to bucket k.

   For $i \in [\widehat{N}], j \in \mathsf{SendInpWires}$, Bob aborts if $e_{i,j,o_{i,j}} \oplus H(K_{i,j})$ is not a valid decommitment to $C^{\mathsf{in}}_{i,j,o_{i,j}}$. Otherwise, Bob sets $\mathsf{in}^*_{i,j}$ to be the result of this decommitment.
10. For $j \in \mathsf{OutWires}, b \in \{0,1\}$ the parties perform a coupling with input $\{d^{\mathsf{out}}_{i,j,b}\}_i, \{d^{\mathsf{w}}_{k,j,b}\}_k$ for Alice. Bob gets $d^{\mathsf{out}}_{i,j,b} \oplus d^{\mathsf{w}}_{k,j,b}$ if circuit i is assigned to bucket k. Bob aborts if this value is not a valid decommitment to $C^{\mathsf{out}}_{i,j,b} \oplus C^{\mathsf{w}}_{k,j,b}$. Otherwise, Bob sets $\delta_{i,j,b}$ to be the result of the decommitment.
11. For $i \in [\widehat{N}]$, where circuit i has not been mapped to bucket #0: Bob evaluates garbled circuit F_i with input wire labels $\{\mathsf{in}^*_{i,j}\}_{j \in \mathsf{SendInpWires} \cup \mathsf{RecvInpWires}}$. The result is plain output z_i and corresponding pre-output wire labels $\{d^{\mathsf{out}}_{i,j,z_{i,j}}\}$. If for some j, $d^{\mathsf{out}}_{i,j,z_{i,j}}$ is not a valid decommitment of $C^{\mathsf{out}}_{i,j,z_{i,j}}$ then Bob changes $z_i = \perp$. Otherwise, Bob opens the commitments to obtain $\mathsf{out}_{i,j,z_{i,j}}$ values.
12. For each bucket $k \neq 0$: If $z_i = \perp$ for all i assigned to this bucket, then abort. If there are $z_i \neq z_{i'}$, neither of them $\perp$, in this bucket, then let j be some position for which $z_{i,j} \neq z_{i',j}$. Bob computes

   $$\tilde{x}_k = (\mathsf{out}_{i,j,z_{i,j}} \oplus \delta_{i,j,z_{i,j}}) \oplus (\mathsf{out}_{i',j,z_{i',j}} \oplus \delta_{i',j,z_{i',j}})$$

   and sets $z^*_k = f(\tilde{x}_k, \tilde{y}_k)$. Otherwise, let z^*_k be the unique value such that $z_i \in \{\perp, z^*_k\}$ for all i in this bucket.
13. Bob outputs $z^*_1, \ldots, z^*_N$.

Fig. 2. (*Continued*)

5 Protocol Efficiency and Choice of Commitments

We review the efficiency of our construction. First, we note that besides the calls to an ideal OT (in the main protocol and also in the OSN subprotocol), the protocol consists of a monolothic message from Alice to Bob (containing garbled circuits, commitments, etc.). All instances of OT are performed in parallel. Hence, ours is a NISC protocol in the sense of [17]. Concretely, the OT can be instantiated with a two-round protocol such as that of [33], making our protocol also a two-round protocol (Bob sends the first OT message, Alice sends the second OT message along with her monolothic NISC protocol message.)

5.1 Effect of Oblivious Switching Network

From Table 1 we see that the parts of the protocol that involve the oblivious switching network (OSN) scale with $N\lambda$, whereas everything else scales with $N\lambda/\log N$ (or independent of λ altogether). The $\log N$ term in the denominator is a result of savings by batching the cut-and-choose step. In particular, the number of garbled circuits (which is the main communication overhead in general), as well as their associated commitments, benefits from batching.

However, information related to the various commitments is sent as input into the OSN. The OSN incurs a $\log \widehat{N}$ overhead which "cancels out" the benefits of batching, for these values. We elaborate on this fact:

We instantiate the OSN with a Waksman network [38], which is a universal switching network (i.e., it can be programmed to realize any permutation). Each "bucket coupling" step requires a permutation on $\widehat{N}$ items, leading to a Waksman network with $O(\widehat{N} \log \widehat{N}) = O(N\lambda)$ switches.

Note that only decommitment and similar values are processed via the OSN subprotocol (bucket coupling steps). The garbled circuits and their associated commitments are not.

5.2 Instantiating Homomorphic Commitments

Pedersen Commitment. Let g be the generator for a prime order group G where the discrete-log problem is hard, and let $h = g^x$ for a random secret x. In our setting g, h can either be chosen by the receiver and sent along with its first OT message, or it can be part of a CRS.

In Pedersen commitments [32], to commit to a message m, we let $\mathsf{Com}(m; r) = g^m h^r$ for a random r. The decommitment string is (m, r). The scheme is statistically hiding and computationally binding. It is also homomorphic (with respect to addition over $\mathbb{Z}_p$) on the message space and the decommitment. In particular, given $\mathsf{Com}(m; r)$ and $\mathsf{Com}(m'; r')$, we can decommit to $m + m'$ by sending $(m+m', r+r')$ to the receiver who can check whether $\mathsf{Com}(m; r) \cdot \mathsf{Com}(m'; r') = g^{m+m'} h^{r+r'}$.

Regarding their suitability for our scheme: Clearly Pedersen commitments have optimal communication overhead (commitment length is equal to the message length). However, they require exponentiations in a DH group. In practice these operations are much slower than symmetric-key primitives like hash functions or block ciphers, which would be preferred. Pedersen commitments are homomorphic over the group $(\mathbb{Z}_p, +)$. For many of the commitments in our scheme (in particular, the $\mathsf{out}_{i,j,b}$ and $\mathsf{w}_{k,j,b}$ values) the choice of group is not crucial, but we actually require the commitments to x_k and s_i to be combined with respect to bitwise XOR. Later in this section we discuss techniques for combining Pedersen commitments with other kinds of homomorphic commitments.

OT-Based Homomorphic Commitments. We discuss a paradigm for homomorphic commitments based on simple OTs.

Starting Point. Our starting point is an XOR-homomorphic commitment of Lindell and Riva [27], that is further based on a technique of Kilian [20] for proving equality of committed values (*i.e.*, proving that the XOR of two commitments is zero). The Lindell-Riva commitment has an interactive opening phase, but we will show how to make it non-interactive.

Let Com be a regular commitment. To generate a homomorphic commitment to message m, the sender secret shares $m_0 \oplus m_1 = m$ and generates plain commitments $\mathsf{Com}(m_0)$ and $\mathsf{Com}(m_1)$.

Suppose commitments to m and m' exist (*i.e.*, there are plain commitments to m_0, m_1, m'_0, m'_1). To open $m \oplus m'$ the parties do the following:

- Preamble: the sender gives $\Delta = m \oplus m'$ (the claimed xor of the two commitments) and $\delta = m_0 \oplus m'_0$
- Challenge: receiver chooses random $b \leftarrow \{0, 1\}$
- Response: sender opens $\mathsf{Com}(m_b)$ and $\mathsf{Com}(m'_b)$. Receiver checks: $m_b \oplus m'_b \overset{?}{=} \delta \oplus b\Delta$

This scheme has soundness 1/2, but can be repeated in parallel λ times to achieve soundness $2^{-\lambda}$.

If we settle for the Fiat-Shamir technique to generate the challenge bits, the above scheme can easily become non-interactive. Similarly, in the offline-online variant of our construction where the commitments and preambles can all be sent in the offline phase, the online phase will be non-interactive (challenge and response). But for our main construction in the standard model, we need to make the above scheme non-interactive.

Making It Non-interactive. In our NISC application, we already assume access to an ideal oblivious transfer functionality. Then the above approach can be modified to both do away with the standalone commitments and to make a non-interactive decommitment phase.

The idea is to replace commitments and a public challenge with an instance of OT. To commit to m, the commitment phase proceeds as follows:

- The receiver chooses a random string $b = b_1 \cdots b_\lambda$ and uses the bits of b as choice bits to λ instances of OT.
- The sender chooses λ pairs $(m_{1,0}, m_{1,1}), \ldots, (m_{\lambda,0}, m_{\lambda,1})$ so that $m_{i,0} \oplus m_{i,1} = m$. The sender uses these pairs as inputs to the instances of OT. Hence, the receiver picks up m_{i,b_i}.

We note that when committing to many values as is the case in our constructions, the *same OTs are used for all commitments.* That is, the same challenge bits b are used for all commitments.

Suppose two such commitments have been made in this way, to m and to m'. Then to decommit to $\Delta = m \oplus m'$ the sender can simply send Δ and $\delta = (\delta_1, \ldots, \delta_\lambda) = (m_{1,0} \oplus m'_{1,0}, \ldots, m_{\lambda,0} \oplus m'_{\lambda,0})$. The receiver can check the soundness equations:

$$m_{i,b_i} \oplus m'_{i,b_i} \overset{?}{=} \delta_i \oplus b_i\Delta$$

Note that the same b_i challenges are shared for all commitments, so the receiver will indeed have m_{i,b_i} and m'_{i,b_i} for a consistent b_i. Since the sender's view is independent of the receiver's challenge b, soundness follows from the same reasoning as above.

In this way, the decommitment string for a commitment to m is $(m, m_{1,0}, \ldots, m_{\lambda,0})$. Furthermore, to decommit to $m \oplus m'$, the decommitment value is the XOR of the individual decommitment values. In other words, the scheme satisfies the homomorphic-opening property described in Sect. 2.2. Finally, note that since we use the same challenge bits for all commitments, it easy to prove multiple XOR relations involving the same committed value.

Code-Based Homomorphic Commitments. A recent series of works [10,12] construct homomorphic commitments from an oblivious-transfer-based setup.

Looking abstractly at our presentation of the Lindell-Riva commitment above, their construction takes the payload m and generates $(m_{1,0}, m_{1,1}, \ldots, m_{\lambda,0}, m_{\lambda,1})$, where $(m_{1,0} \oplus m_{1,1}, \ldots, m_{\lambda,0} \oplus m_{\lambda,1})$ is an encoding of m. In this case, the encoding is a λ-repetition encoding.

The idea behind [12] is to choose an encoding with better rate. Namely, the sender generates $(m_{1,0}, m_{1,1}, \ldots, m_{n,0}, m_{n,1})$, where $(m_{1,0} \oplus m_{1,1}, \ldots, m_{n,0} \oplus m_{n,1})$ encodes m in some error-correcting code. Here the total length of the encoding may be much smaller than $2\lambda|m|$ as in the Lindell-Riva scheme. The binding property of the construction is related to the minimum distance of this code. We refer the reader to [12] for details about the construction and how to choose an appropriate error-correcting code. Instead, we point out some facts that are relevant to our use of homomorphic commitments:

- When the error-correcting code is *linear*, then the commitments are additively homomorphic. Following our pattern, the decommitment value for a commitment is the vector $(m, m_{1,0}, \ldots, m_{n,0})$. These decommitment values are indeed homomorphic in the sense we require.
- The *rate* of a commitment scheme is the length of the commitment's payload divided by the communication cost of the commitment. For example, the Lindell-Riva scheme has rate $O(1/\lambda)$. By a suitable choice of error-correcting codes, the *rate* of the scheme in [12] can be made constant, or even $1 + o(1)$. Concretely, to commit to 128 bits requires the committer to send only 262 bits when using an appropriate BCH code, leading to a rate 0.49.

Unfortunately, unlike the Lindell-Riva construction, the scheme of [12] requires some additional interaction in the setup phase. In particular, there must be some mechanism to ensure that the sender is indeed using valid codewords. The sender can violate binding, for instance, by choosing a non-codeword that is "halfway between" two valid codewords. In [12], after the parties have performed the OTs of the setup phase, the receiver challenges the sender to open some random combination of values to ensure that they are consistent with valid codewords.

Removing this interaction turns out to be problematic. In our offline/online application the extra interaction is in the offline phase, and so not a problem.

In our offline/online application we therefore use this highly efficient commitments. However, we cannot afford the extra round of interaction in our batch NISC application. Hence, our options are: (1) use less efficient homomorphic commitment schemes like the Lindell-Riva one; (2) remove the round of interaction using the Fiat-Shamir heuristic, since the receiver's challenge is random.

5.3 Reducing Cost of Homomorphic Commitments

We described our main protocol without specifying exactly which homomorphic commitment to use. Based on the previous discussion, we have several options, none of them ideal:

– Pedersen commitments, which are rate 1, but require public-key operations and are homomorphic only with respect to addition in $\mathbb{Z}_p$.
– Lindell-Riva-style commitments based on OTs, which have rate $O(1/\lambda)$ and are homomorphic with respect to XOR.
– FJNT [12] commitments, which have constant rate and are homomorphic with respect to XOR, but require some interaction in the initialization step (unless one is satisfied with the Fiat-Shamir heuristic).

The only "off-the-shelf" choice that is compatible with our construction is the Lindell-Riva-style commitments, which are the least efficient in terms of communication.

We therefore describe two methods to significantly improve the efficiency related to homomorphic commitments in our construction.

Linking Short-to-Long Commitments. The protocol performs homomorphic decommitments that combine x_k and $\mathsf{w}_{k,j,b}$ values—hence, these values must have the same length ($|\mathsf{SendInpWires}|$). There is an $\mathsf{w}_{k,j,b}$ value for each bucket $k \in [N]$ and each circuit output wire $j \in \mathsf{OutWires}$. Accounting for the total communication cost for these commitments in the Lindell-Riva-style scheme, we get $O(\lambda N |\mathsf{OutWires}| \cdot |\mathsf{SendInpWires}|)$. For circuits with relatively long inputs/outputs, the cost $|\mathsf{OutWires}| \cdot |\mathsf{SendInpWires}|$ is undesirable.

We propose a technique for reducing this cost when $|\mathsf{SendInpWires}|$ is long (longer than a computational security parameter κ). Recall that the purpose of the $\mathsf{w}_{k,j,b}$ values is that if the receiver learns both $\mathsf{w}_{k,j,0}$ and $\mathsf{w}_{k,j,1}$, then he can combine them to learn x_k. We modify the construction so that the sender gives a homomorphic commitment to a random ("short") w_k for each bucket k, where

$$\mathsf{w}_{k,j,0} \oplus \mathsf{w}_{k,j,1} = \mathsf{w}_k \qquad (\forall j \in \mathsf{OutWires})$$

(i.e., we have replaced x_k with w_k in the above expression). The $\mathsf{w}_k, \mathsf{w}_{k,j,b}$ values have length κ, so the total cost of these commitments to $\mathsf{w}_{k,j,b}$ is $O(\kappa\lambda N|\mathsf{OutWires}|)$.

Now we must modify the protocol so that if the receiver ever learns w_k, then he (non-interactively) can recover x_k, where w_k is "short" and x_k is "long." Recall that to commit to x_k and w_k in the Lindell-Riva scheme, the sender needs to generate λ independent additive sharings: $\{x_{k,0,i}, x_{k,1,i}\}_{i\in[\lambda]}$ and $\{\mathsf{w}_{k,0,i}, \mathsf{w}_{k,1,i}\}_{i\in[\lambda]}$ and using them as sender's inputs to the challenge OTs. To "link" w_k to x_k, we simply have the sender also send ciphertexts $\{\mathsf{Enc}(\mathsf{w}_{k,b,i}; x_{k,b,i})\}_{i\in[\lambda],b\in\{0,1\}}$, i.e. encrypting each additive share of x_k using the corresponding share of w_k as the key. Note that Enc can be a symmetric-key encryption scheme and therefore relatively fast.

In the Lindell-Riva scheme, the receiver learns one share from each pair of shares. He learns either $(\mathsf{w}_{k,0,i}, x_{k,0,i})$ or $(\mathsf{w}_{k,b,i}, x_{k,b,i})$. Hence, the receiver can check half of the ciphertexts sent by the sender, and abort if any are not correct/consistent. If the receiver doesn't abort, this guarantees that with high probability the majority of these linking encryptions are correct. To bound the probability of error to $2^{-\lambda}$, we must increase the number of parallel repetitions to $\sim 3\lambda$.

Now if the receiver learns w_k at some later time, it can solve for both shares $\mathsf{w}_{k,0,i}, \mathsf{w}_{k,1,i}$ for every i, and use them to decrypt the shares $x_{k,0,i}, x_{k,1,i}$. The receiver thus recovers x_k as the *majority value* among all $x_{k,0,i} \oplus x_{k,1,i}$.

If the receiver never cheats, then the value of x_k remains hidden by the semantic security of the Enc-encryptions.

Replacing with Pedersen Commitments. We can reduce the $\kappa\lambda$ term in the communication complexity to κ by using Pedersen commitments for the output wires. In particular, $O(|\mathsf{OutWires}|\widehat{N})$ Pedersen commitments are sufficient for committing to the w_k, $\mathsf{w}_{k,j,b}$, and $\mathsf{out}_{i,j,b}$ values. However, we also need to "link" w_k to x_k. To do so, we can use the input consistency check technique used in [1] that uses an El Gamal encryption of x_k, and algebraically links the Pedersen commitments to the output wires with the Elgamal encryption of the input. We refer the reader to [1] for details of this approach.

We note that the use of Pedersen Commitments provides a trade-off between communication and computation as the computation cost will likely increase due the public-key operations required by the scheme, but we save on the communication requires for cheating-recovery.

Reducing Communication Using the Seed Technique. In the full version of the paper, we show how to further reduce communication of our protocol by incorporating the seed technique of [1,13] wherein only the garbled circuits that are evaluated are communicated in full.

6 Optimizations for the Offline-Online Setting and Random Oracle Model

Using the random oracle model, we can remove or improve several sources of inefficiency in our construction. To introduce these improvements, we first describe a 2PC protocol in the offline-online setting, which may be of independent interest.

6.1 Offline-Online Protocol

The Setting. In this setting, the parties know that they will securely evaluate some function f, N times (perhaps not altogether in a single batch). In an *offline phase* they perform some pre-processing that depends only on f and N. Then, when it comes time to securely evaluate an instance of f, they perform an *online phase* that is as inexpensive as possible, and depends on their inputs to this evaluation of f.

We will describe how to modify our NISC protocol to obtain an offline-online 2PC protocol where:

- The offline phase is constant-round.
- Each online phase is two rounds, consisting of a length-$|y|$ message from the receiver Bob followed by a message of length $(|x| + |y|)\kappa$ (or $|x| + |y| + O(\kappa)$, after further optimization) from Alice.
- The total cost of N secure evaluations of f is $O(N/\log N)$ times that of a single secure evaluation. In particular, batching improves *all aspects* of the protocol by a $\log N$ factor (unlike in the NISC protocol where the cost associated with circuit inputs/outputs did not have a $\log N$-factor saving).

Removing the Switching Network. Recall that in our NISC protocol the costs associated with *garbled circuits* scale as $O(N/\log N)$, while the costs associated with circuit inputs/outputs scales as $O(N)$. The reason is that decommitment information related to inputs/outputs is sent through the oblivious switching network (OSN). The switching network has $\log \widehat{N}$ depth, and incurs a $\log \widehat{N}$ factor overhead that cancels out the $\log N$ savings incurred by the batch cut-and-choose.

The main reason for the oblivious switching network protocol was to non-interactively choose an assignment of circuits to buckets. We showed how to perform this task using a two-round OSN protocol in the standard model. However, the assignment of circuits to buckets can be done in the offline phase, as it does not depend on the parties' inputs to f.

Let π denote the receiver's assignment of circuits to buckets. In the non-interactive setting, it was necessary to hide π from the sender—the sender cannot know in advance which circuits will be checked in the cut-and-choose. However, in principle π does not need to be completely secret; it merely suffices for it to be chosen *after* the sender commits to the garbled circuits.

When we allow more interaction in the offline phase, we can do away with the oblivious switching network (and its $\log \widehat{N}$ overhead on garbled inputs/outputs) altogether. The main changes to remove the OSN subprotocols are as follows:

- The receiver chooses a random assignment π and commits to it.
- For the coupling subprotocols involving σ_i, we instead have the sender commit individually to each σ_i. The sender also sends all of the garbled circuits and various commitments, just as in the NISC protocol.
- After the σ_i's are committed, the receiver opens the commitment of π.

- The sender opens the commitments to σ_i for i assigned to be checked. This allows the receiver to learn the σ_i's while avoiding the bucket-coupling subprotocol involving these values. For all other couplings, the sender simply sends whatever the receiver's output would have been in the NISC protocol. This is possible since the sender knows π.

In this way, we remove all invocations of the switching network, and their associated $O(\log \widehat{N})$ overhead.

To argue that the protocol is still secure, we need to modify the simulator for the NISC protocol. When the sender is corrupt, the simulator extracts the σ_i values, but does not use any special capabilities for the other couplings—it merely runs these couplings honestly and uses only their output. In this offline/online modification, the simulator can still extract the σ_i's from the commitments. Then it can receive the other values (formerly obtained via the couplings) directly from the sender. To simulate a corrupt receiver, the simulator need only extract the commitment to π in the first step, similar to how the NISC simulator extracts π as its first operation. However, in this setting the inputs to the function are chosen *after* the receiver has seen the garbled circuits. Hence, we require a garbling scheme that has **adaptive security** [5].

Note that in this setting we can apply the optimization of Goyal et al. [13]: the sender can initially send only a *hash* of each garbled circuit. For circuits that are assigned to be checked, it is not necessary to send the entire garbled circuit – the receiver can simply recompute the circuit from the seed and compare to the hash. Only circuits that are actually evaluated must be sent. This optimization reduces concrete cost by a significant constant factor.

Optimizing Sender's Garbled Input. First, we can do away with the encryptions $e_{i,j,b}$ (step 4) and the associated coupling (step 9). These were needed only to route the sender's inputs to the correct buckets without a priori knowledge of the bucketing assignment. Instead, the sender (after learning the bucketing assignment) can simply directly send the decommitments to the correct commitments to her garbled input.

Besides this optimization, we observe that the NISC protocol uses the sender's input x_k in several places. We briefly describe ways to move the bulk of these operations to the offline phase.

Offline commitments to sender's input: In the NISC protocol the sender gives a homomorphic commitment to x_k. For each circuit i assigned to bucket k, the receiver learns the decommitment to $s_i \oplus x_k$, and in the online phase will expect the sender to open commitments indexed by $s_i \oplus x_k$, since these will be the commitments to wire labels holding truth value x_k. Furthermore, the sender chooses bucket-wide values $\mathsf{w}_{k,j,b}$ so that $x_k = \mathsf{w}_{k,j,0} \oplus \mathsf{w}_{k,j,1}$. The idea is that, if the receiver obtains conflicting outputs within a bucket, he can learn x_k.

To reduce the online dependence on x_k, we make the following change. Instead of giving a homomorphic commitment to x_k, the sender uses a random value μ_k. Since μ_k is unrelated to her input x_k, all of the commitments and homomorphic openings can be done in the offline phase. In other words, the homomorphic commitments are arranged so that the receiver learns $\mu_k \oplus s_i$, and so that the receiver learns μ_k if he obtains conflicting outputs in the bucket. Then in the online phase, the sender simply gives $x_k \oplus \mu_k$ *in the clear.* The receiver will expect the sender to open commitments indexed by $(x_k \oplus \mu_k) \oplus (\mu_k \oplus s_i) = x_k \oplus s_i$. If cheating is detected, the receiver learns μ_k and thus obtains $x_k = (x_k \oplus \mu_k) \oplus \mu_k$.

Packaging together sender's garbled inputs: Suppose there are B circuits assigned to each bucket. The receiver will be expecting the sender to decommit to B values for each input bit ($j \in \mathsf{SendInpWires}$). This leads to $O(B|x|\kappa)$ communication from the sender in each execution.

But since the sender knows the bucket-assignment in the offline phase, she can "package" the corresponding decommitment values together in the following way. For each of her input wires $j \in \mathsf{SendInpWires}$ and value $b \in \{0, 1\}$ the sender can choose a bucket-specific token $\mathsf{tok}_{k,j,b}$. Then, in she can encrypt all of the B different openings that will be necessary in the event that she has truth value b on wire j in the online phase. She can generate these ciphertexts in the offline phase and send them (in a random order with respect to the b-values). Then in the online phase, she need only send a single $\mathsf{tok}_{k,j,b}$ value for each bit of her input, at a cost of only $|x|\kappa$ per execution.

Optimizing Receiver's Garbled Input. In the online phase, the parties must perform the OTs for the receiver's inputs. As in the NISC protocol these OTs are already on bucket-wide "tokens" and not B sets of wire labels per input wire.

Note that the number of OTs per execution is $|\tilde{y}_k|$, where $\tilde{y}_k$ is the λ-probe-resistant encoding of the receiver's true input y_k. Indeed, $\tilde{y}_k$ is longer than y_k by a significant constant factor in practice. However, we can reduce the online cost to $|y_k|$ by using an optimization proposed by Lindell and Riva [27] in their offline/online protocol, which we describe below:

Recall that M is the λ-probe-resistant matrix, and the parties are evaluating the function $\tilde{f}(x, \tilde{y}) = f(x, M\tilde{y})$. We instead ask the parties to evaluate the function $g(x, \tilde{r}, m) = f(x, m \oplus M\tilde{r})$. Note that $\tilde{r}$ is the length of a λ-probe-resistant-encoded input, while m has the same length as y. The idea is for Bob to choose an encoding $\tilde{r}$ of a *random* r, in the offline phase. The parties can perform OTs for $\tilde{r}$ in the offline phase. Then in the online phase, Bob announces $m = r \oplus y$ in the clear. Alice must then decommit to the input wire labels corresponding to m (in the protocol description we refer to these input wires of g as $\mathsf{PubInpWires}$). As above, the decommitments for all B circuits in this bucket can be "packaged" together with encryptions sent in the offline phase. Therefore, the online cost attributed to the receiver's input is the receiver sending m and the sender sending $|m|$ encryption keys (where $|m| = |y|$).

Futher Compressing the Online Phase. Using the optimizations listed above, each online phase consists only of a length-$|y|$ message from Bob and a reply from Alice of length $(|x| + |y|)\kappa$. However, we point out that the message from Alice can be shortened *even further* using a technique of Applebaum et al. [2] that we summarize in Sect. 2.6. As a result, the *total communication* in the online phase is $|x| + |y| + O(\kappa)$ bits—only $O(\kappa)$ bits less than the information-theoretic minimum for secure computation.

Protocol Description. The detailed protocol description is given in Fig. 4. For simplicity this description does not include the technique of Applebaum et al. [2] for compressing garbled inputs in the online phase. This optimization can be applied in a black-box manner to our protocol.

Theorem 7. *The protocol in Fig. 4 is a UC-secure realization of the functionality in Fig. 3. The online phase is 2 rounds, and requires a length-$|y|$ message from the receiver and length-$(|x| + O(\kappa))$ message from the sender.*

Parameters: A function f and number N of instances.

Behavior: On input `setup` from the sender, give output `setup` to the receiver. Then do the following N times: wait for input x from the sender and y from the receiver. Then give output $f(x, y)$ to the receiver.

Fig. 3. Ideal functionality for offline/online 2PC

6.2 NISC, Optimized for Random Oracle Model

In the offline/online protocol we just described, the receiver first commits to π, receives garbled circuits and commitments, then opens π. Suppose we remove the commitment to π from the protocol. In other words, suppose the offline phase begins with the sender giving the garbled circuits and associated commitments, and then the receiver sends a random π in the clear.

This modified offline phase is then *public-coin* for the verifier. The only messages sent by the verifier are the random π and a random challenge for the FJNT homomorphic commitment scheme setup (not explicitly shown in the protocol description). We can therefore apply the **Fiat-Shamir** technique to make the protocol non-interactive again, in the programmable random oracle model.[2] In doing so we obtain a batch-NISC protocol that is considerably more efficient than our standard-model protocol. In particular:

- The RO protocol makes no use of the switching network, so avoids the associated overhead on garbled inputs/outputs.
- The RO protocol can be instantiated with the lightweight homomorphic commitments of [12].
- The RO protocol avoids communication for garbled circuits that are assigned to be checked.
- Unlike in the NISC setting, the offline/online protocol can take advantage of efficient OT extension techniques [3,4,16,19,21] which greatly reduce the cost of the (many) OTs in the protocol, but require interaction. This property is of course shared by all 2PC protocols that allow for more than 2 rounds.

[2] When considering a corrupt receiver, instead of extracting π from the commitment, the simulator can simply choose π upfront and then program the random oracle to output π on the appropriate query.

Parameters: A function f and number N of instances. $\widehat{N}$ denotes the number of garbled circuits, chosen according to the discussion in the text. λ is the statistical security parameter.

Offline phase:

1. Bob chooses a random permutation π, and commits to it.
2. For each circuit $i \in [\widehat{N}]$: Alice chooses a PRF seed σ_i and uses it to derive *all randomness used in this step* of the protocol:
   Alice generates a garbling of the function $\tilde{f}(x, \tilde{r}, m) = f(x, m \oplus M\tilde{r})$; let F_i denote the garbled circuit, and let $\mathsf{in}_{i,j,b}$ (resp. $d^{\mathsf{out}}_{i,j,b}$) denote the input (resp. output) wire label encoding truth value b on wire j of circuit i. She computes $h_i = H(F_i)$ where H is a CRHF, and sends h_i to Bob.
   Alice chooses random "post-output" keys $\{\mathsf{out}_{i,j,b}\}_{j \in \mathsf{OutWires}, b \in \{0,1\}}$. She generates and sends the following commitments (where d^{in} and d^s values are derived randomly from σ_i):
   $$\begin{aligned} C^{\mathsf{in}}_{i,j,b} &\leftarrow \mathsf{Com}(\mathsf{in}_{i,j,b \oplus s_{i,j}}; d^{\mathsf{in}}_{i,j,b \oplus s_{i,j}}) && \text{for } j \in \mathsf{SendInpWires}, b \in \{0,1\} \\ C^{\mathsf{in}}_{i,j,b} &\leftarrow \mathsf{Com}(\mathsf{in}_{i,j,b}; d^{\mathsf{in}}_{i,j,b}) && \text{for } b \in \{0,1\}, j \in \mathsf{RecvInpWires} \\ C^{\mathsf{out}}_{i,j,b} &\leftarrow \mathsf{HCom}(\mathsf{out}_{i,j,b}; d^{\mathsf{out}}_{i,j,b}) && \text{for } b \in \{0,1\}, j \in \mathsf{OutWires} \\ C^s_i &\leftarrow \mathsf{HCom}(s_i; d^s_i) && \end{aligned}$$
3. For each $i \in [\widehat{N}]$, Alice commits to each σ_i.
4. Bob opens the commitment to π.
5. For all i assigned to be checked by π, Alice opens the commitment to σ_i. Bob checks that h_i and corresponding commitments from the previous step are generated using randomness derived from σ_i, and aborts if this is not the case.
   For all i *not* assigned to be checked, Alice sends F_i; Bob aborts if $h_i \neq H(F_i)$.
6. For $k \in [N]$, Alice chooses a random μ_k. For $k \in [N], j \in \mathsf{OutWires}$, Alice chooses random $\mathsf{w}_{k,j,0}$ and sets $\mathsf{w}_{k,j,1} = \mu_k \oplus \mathsf{w}_{k,j,0}$. Alice generates and sends commitments:
   $$\begin{aligned} C^{\mathsf{w}}_{k,j,b} &\leftarrow \mathsf{HCom}(\mathsf{w}_{k,j,b}; d^{\mathsf{w}}_{k,j,b}) && \text{for } k \in [N], j \in \mathsf{OutWires}, b \in \{0,1\} \\ C^{\mu}_k &\leftarrow \mathsf{HCom}(\mu_k; d^x_k) && \text{for } k \in [N] \end{aligned}$$
   Alice also gives homomorphic decommitments:
   $$d^{\mathsf{w}}_{k,j,0} \oplus d^{\mathsf{w}}_{k,j,1} \oplus d^{\mu}_k \qquad \text{for } k \in [N], j \in \mathsf{OutWires}$$
   Bob aborts if these values do not decommit $C^{\mathsf{w}}_{k,j,0} \oplus C^{\mathsf{w}}_{k,j,1} \oplus C^{\mu}_k$ to the all-zeroes string.
7. For $j \in \mathsf{OutWires}, b \in \{0,1\}$ and all circuits i assigned to bucket k, Alice sends $d^{\mathsf{out}}_{i,j,b} \oplus d^{\mathsf{w}}_{k,j,b}$. Bob aborts if this is not a valid decommitment to $C^{\mathsf{out}}_{i,j,b} \oplus C^{\mathsf{w}}_{k,j,b}$; otherwise he sets $\delta_{i,j,b}$ to be the result of this decommitment.
8. For all circuits i assigned to bucket k, Alice sends $d^{\mu}_k \oplus d^s_i$. Bob aborts if this is not a valid decommitment to $C^{\mu}_k \oplus C^s_i$; otherwise he sets o_i to be the result of this decommitment.

(protocol description continues...)

Fig. 4. Online-offline protocol

9. For all $k \in [N]$, Bob chooses a random λ-probe-resistant encoding $\tilde{r}_k$. For $j \in \mathsf{RecvInpWires}$, the parties engage in an instance of OT with inputs $(\{d^{\mathsf{in}}_{i,j,0}\}_i, \{d^{\mathsf{in}}_{i,j,b}\}_i)$ for Alice and input $\tilde{r}_{k,j}$ for Bob. Here the index i ranges over circuits assigned to bucket k.
   Hence Bob learns input wire labels $\{d^{\mathsf{in}}_{i,j,\tilde{r}_{k,j}}\}_i$. He aborts if these are not valid decommitments to $\{C^{\mathsf{in}}_{i,j,\tilde{r}_{k,j}}\}_i$. Otherwise he sets $\mathsf{in}^*_{i,j}$ to be the corresponding decommitted values.
10. For $k \in [N], j \in \mathsf{SendInpWires}, b \in \{0,1\}$, Alice chooses a random token $\mathsf{tok}_{k,j,b}$, generates and sends an encryption:

    $$e_{k,j,b} = \mathsf{Enc}\Big(\mathsf{tok}_{k,j,b}; \{d^{\mathsf{in}}_{i,j,\mu_{k,j}}\}_i\Big)$$

    Here the index i ranges over circuits assigned to bucket k. These are decommitments to wire labels indexed by μ_k, hence wire labels having truth value $\mu_k \oplus s_i$.
    Similarly, for $k \in [N], j \in \mathsf{PubInpWires}, b \in \{0,1\}$, Alice chooses a random token $\mathsf{tok}_{k,j,b}$, generates and sends an encryption:

    $$e_{k,j,b} = \mathsf{Enc}\Big(\mathsf{tok}_{k,j,b}; \{d^{\mathsf{in}}_{i,j,b}\}_i\Big)$$

11. For $k \in [N]$, Alice generates compressed garbled encodings of the tokens for her input wires and public input wires:

    $$(sk_k, \widehat{e}_k) \leftarrow \mathsf{Compress}(\{\mathsf{tok}_{k,j,b} \mid j \in \mathsf{SendInpWires} \cup \mathsf{PubInpWires}; b \in \{0,1\}\})$$

    She sends $\widehat{e}_k$ to Bob.

(protocol description continues...)

Fig. 4. (*Continued*)

Unfortunately, in this protocol we must use the *computational* security parameter κ (e.g., 128), and not the statistical security parameter λ (e.g., 40) to determine the bucket sizes. In the other protocols, the sender is committed to her choice of garbled circuits before the cut-and-choose challenge and bucketing assignment are chosen. Hence, cheating in the cut-and-choose phase is a one-time opportunity. In this Fiat-Shamir protocol, the sender can generate many candidate first protocol messages, until it finds one whose hash is favorable (i.e., it allows her to cheat undetected). Since this step involves no interaction, she has as many opportunities to try to find an advantageous first protocol message as her computation allows. Hence the probability of undetected cheating in the cut-and-choose step must be bound by the computational security parameter.

We note that the garbled-input-compressing technique of Applebaum et al. [2] is not useful in NISC since it increases total cost to improve online cost. In the NISC setting, there is no distinction between offline and online, so their technique simply increases the cost.

Online phase: For the kth time the online phase is invoked, Alice has input x_k and Bob has input y_k.

1. Bob computes $m_k = y_k \oplus M\tilde{r}_k$ and sends it to Alice.
2. Alice computes $\gamma_k = x_k \oplus \mu_k$. She computes online compressed garbled encoding $\widehat{v}_k \leftarrow \mathsf{Online}(sk_k, m_k \| \gamma_k)$, and sends both γ_k and $\widehat{v}_k$ to Bob.
3. Bob decompresses the garbled encodings:
$$\{\mathsf{tok}_{k,j,m_{k,j}} \mid j \in \mathsf{PubInpWires}\} \cup \{\mathsf{tok}_{k,j,\gamma_{k,j}} \mid j \in \mathsf{SendInpWires}\} \\ \leftarrow \mathsf{Decompress}(\widehat{e}_k, m_k \| \gamma_k, \widehat{v}_k)$$
4. Bob decrypts the corresponding ciphertexts as follows:
$$\{d^{\mathsf{in}}_{i,j,\gamma_{k,j}}\}_i = \mathsf{Dec}\Big(\mathsf{tok}_{k,j,\gamma_{k,j}}; e_{k,j,\gamma_{k,j}}\Big) \qquad \text{for } j \in \mathsf{SendInpWires}$$
$$\{d^{\mathsf{in}}_{i,j,m_{k,j}}\}_i = \mathsf{Dec}\Big(\mathsf{tok}_{k,j,m_{k,j}}; e_{k,j,m_{k,j}}\Big) \qquad \text{for } j \in \mathsf{PubInpWires}$$
Bob aborts if the $d^{\mathsf{in}}_{i,j,b}$ values are not valid decommitments of the corresponding $C^{\mathsf{in}}_{i,j,b}$ commitments. Otherwise, Bob sets $\mathsf{in}^*_{i,j}$ to be the result of decommitment. Now, for all circuits i in this bucket, Bob has a complete garbled input (with wire labels for $\mathsf{RecvInpWires}$ obtained in step 9 of the offline phase).
5. For each circuit i assigned to bucket k, Bob evaluates garbled circuit F_i with input wire labels $\{\mathsf{in}^*_{i,j}\}_j$. The result is plain output z_i and corresponding pre-output wire labels $\{d^{\mathsf{out}}_{i,j,z_{i,j}}\}$. If for some j, $d^{\mathsf{out}}_{i,j,z_{i,j}}$ is not a valid decommitment of $C^{\mathsf{out}}_{i,j,z_{i,j}}$ then Bob changes $z_i = \bot$. Otherwise, Bob opens the commitments to obtain $\mathsf{out}_{i,j,z_{i,j}}$ values.
6. If $z_i = \bot$ for all i assigned to this bucket, then abort. If there are $z_i \neq z_{i'}$, neither of them $\bot$, in this bucket, then let j be some position for which $z_{i,j} \neq z_{i',j}$. Bob computes
$$\tilde{x}_k = (\mathsf{out}_{i,j,z_{i,j}} \oplus \delta_{i,j,z_{i,j}}) \oplus (\mathsf{out}_{i',j,z_{i',j}} \oplus \delta_{i',j,z_{i',j}}) \oplus \gamma_k$$
and outputs $z^*_k = f(\tilde{x}_k, y_k)$. Otherwise, Bob outputs the unique value z^*_k such that $z_i \in \{\bot, z^*_k\}$ for all i in this bucket.

Fig. 4. (*Continued*)

Theorem 8. *There is a UC-secure batch NISC protocol in the programmable random oracle model, that evaluates N instances of f with total cost $N/O(\log N)$ times more than a single evaluation of f (plus some small additive terms that do not depend on f).*

References

1. Afshar, A., Mohassel, P., Pinkas, B., Riva, B.: Non-interactive secure computation based on cut-and-choose. In: Nguyen, P.Q., Oswald, E. (eds.) EUROCRYPT 2014. LNCS, vol. 8441, pp. 387–404. Springer, Heidelberg (2014). doi:10.1007/978-3-642-55220-5_22

2. 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. LNCS, vol. 8043, pp. 166–184. Springer, Heidelberg (2013). doi:10.1007/978-3-642-40084-1_10
3. Asharov, G., Lindell, Y., Schneider, T., Zohner, M.: More efficient oblivious transfer extensions with security for malicious adversaries. In: Oswald, E., Fischlin, M. (eds.) EUROCRYPT 2015. LNCS, vol. 9056, pp. 673–701. Springer, Heidelberg (2015). doi:10.1007/978-3-662-46800-5_26
4. Asharov, G., Lindell, Y., Schneider, T., Zohner, M.: More efficient oblivious transfer and extensions for faster secure computation. In: ACM CCS 2013, pp. 535–548. ACM Press, November 2013
5. 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, Heidelberg (2012). doi:10.1007/978-3-642-34961-4_10
6. Bellare, M., Hoang, V.T., Rogaway, P.: Foundations of garbled circuits. In: ACM CCS 2012, pp. 784–796. ACM Press, October 2012
7. Cachin, C., Camenisch, J., Kilian, J., Müller, J.: One-round secure computation and secure autonomous mobile agents. In: Montanari, U., Rolim, J.D.P., Welzl, E. (eds.) ICALP 2000. LNCS, vol. 1853, pp. 512–523. Springer, Heidelberg (2000). doi:10.1007/3-540-45022-X_43
8. Canetti, R.: Universally composable security: a new paradigm for cryptographic protocols. In: 42nd FOCS, pp. 136–145. IEEE Computer Society Press, October 2001
9. Canetti, R., Jain, A., Scafuro, A.: Practical UC security with a global random oracle. In: Proceedings of ACM CCS 2014, pp. 597–608. ACM (2014)
10. Cascudo, I., Damgård, I., David, B., Giacomelli, I., Nielsen, J.B., Trifiletti, R.: Additively homomorphic UC commitments with optimal amortized overhead. In: Katz, J. (ed.) PKC 2015. LNCS, vol. 9020, pp. 495–515. Springer, Heidelberg (2015). doi:10.1007/978-3-662-46447-2_22
11. 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, Heidelberg (2013). doi:10.1007/978-3-642-38348-9_32
12. Frederiksen, T.K., Jakobsen, T.P., Nielsen, J.B., Trifiletti, R.: On the complexity of additively homomorphic UC commitments. In: Kushilevitz, E., Malkin, T. (eds.) TCC 2016. LNCS, vol. 9562, pp. 542–565. Springer, Heidelberg (2016). doi:10.1007/978-3-662-49096-9_23
13. Goyal, V., Mohassel, P., Smith, A.: Efficient two party and multi party computation against covert adversaries. In: Smart, N. (ed.) EUROCRYPT 2008. LNCS, vol. 4965, pp. 289–306. Springer, Heidelberg (2008). doi:10.1007/978-3-540-78967-3_17
14. Horvitz, O., Katz, J.: Universally-composable two-party computation in two rounds. In: Menezes, A. (ed.) CRYPTO 2007. LNCS, vol. 4622, pp. 111–129. Springer, Heidelberg (2007). doi:10.1007/978-3-540-74143-5_7
15. Huang, Y., Katz, J., Kolesnikov, V., Kumaresan, R., Malozemoff, A.J.: Amortizing garbled circuits. In: Garay, J.A., Gennaro, R. (eds.) CRYPTO 2014. LNCS, vol. 8617, pp. 458–475. Springer, Heidelberg (2014). doi:10.1007/978-3-662-44381-1_26
16. 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, Heidelberg (2003). doi:10.1007/978-3-540-45146-4_9

17. Ishai, Y., Kushilevitz, E., Ostrovsky, R., Prabhakaran, M., Sahai, A.: Efficient non-interactive secure computation. In: Paterson, K.G. (ed.) EUROCRYPT 2011. LNCS, vol. 6632, pp. 406–425. Springer, Heidelberg (2011). doi:10.1007/978-3-642-20465-4_23
18. 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, Heidelberg (2008). doi:10.1007/978-3-540-85174-5_32
19. Keller, M., Orsini, E., Scholl, P.: Actively secure OT extension with optimal overhead. In: Gennaro, R., Robshaw, M. (eds.) CRYPTO 2015. LNCS, vol. 9215, pp. 724–741. Springer, Heidelberg (2015). doi:10.1007/978-3-662-47989-6_35
20. Kilian, J.: Founding cryptography on oblivious transfer. In: 20th ACM STOC, pp. 20–31. ACM Press, May 1988
21. Kolesnikov, V., Kumaresan, R.: Improved OT extension for transferring short secrets. In: Canetti, R., Garay, J.A. (eds.) CRYPTO 2013. LNCS, vol. 8043, pp. 54–70. Springer, Heidelberg (2013). doi:10.1007/978-3-642-40084-1_4
22. Kolesnikov, V., Mohassel, P., Rosulek, M.: FleXOR: flexible garbling for XOR gates that beats free-XOR. In: Garay, J.A., Gennaro, R. (eds.) CRYPTO 2014. LNCS, vol. 8617, pp. 440–457. Springer, Heidelberg (2014). doi:10.1007/978-3-662-44381-1_25
23. Kolesnikov, V., Schneider, T.: Improved garbled circuit: Free XOR gates and applications. In: Aceto, L., Damgård, I., Goldberg, L.A., Halldórsson, M.M., Ingólfsdóttir, A., Walukiewicz, I. (eds.) ICALP 2008. LNCS, vol. 5126, pp. 486–498. Springer, Heidelberg (2008). doi:10.1007/978-3-540-70583-3_40
24. Lindell, Y.: Fast cut-and-choose based protocols for malicious and covert adversaries. In: Canetti, R., Garay, J.A. (eds.) CRYPTO 2013. LNCS, vol. 8043, pp. 1–17. Springer, Heidelberg (2013). doi:10.1007/978-3-642-40084-1_1
25. 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, Heidelberg (2007). doi:10.1007/978-3-540-72540-4_4
26. Lindell, Y., Riva, B.: Cut-and-choose Yao-based secure computation in the online/offline and batch settings. In: Garay, J.A., Gennaro, R. (eds.) CRYPTO 2014. LNCS, vol. 8617, pp. 476–494. Springer, Heidelberg (2014). doi:10.1007/978-3-662-44381-1_27
27. Lindell, Y., Riva, B.: Blazing fast 2PC in the offline, online setting with security for malicious adversaries. In: Ray, I., Li, N., Kruegel, C. (eds.) ACM CCS 2015, pp. 579–590. ACM Press, October 2015
28. 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, Heidelberg (2006). doi:10.1007/11745853_30
29. 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. LNCS, vol. 8043, pp. 36–53. Springer, Heidelberg (2013). doi:10.1007/978-3-642-40084-1_3
30. Mohassel, P., Sadeghian, S.: How to hide circuits in MPC an efficient framework for private function evaluation. In: Johansson, T., Nguyen, P.Q. (eds.) EUROCRYPT 2013. LNCS, vol. 7881, pp. 557–574. Springer, Heidelberg (2013). doi:10.1007/978-3-642-38348-9_33
31. Nielsen, J.B., Orlandi, C.: LEGO for two-party secure computation. In: Reingold, O. (ed.) TCC 2009. LNCS, vol. 5444, pp. 368–386. Springer, Heidelberg (2009). doi:10.1007/978-3-642-00457-5_22

32. Pedersen, T.P.: Non-interactive and information-theoretic secure verifiable secret sharing. In: Feigenbaum, J. (ed.) CRYPTO 1991. LNCS, vol. 576, pp. 129–140. Springer, Heidelberg (1992). doi:10.1007/3-540-46766-1_9
33. Peikert, C., Vaikuntanathan, V., Waters, B.: A framework for efficient and composable oblivious transfer. In: Wagner, D. (ed.) CRYPTO 2008. LNCS, vol. 5157, pp. 554–571. Springer, Heidelberg (2008). doi:10.1007/978-3-540-85174-5_31
34. Pinkas, B., Schneider, T., Smart, N.P., Williams, S.C.: Secure two-party computation is practical. In: Matsui, M. (ed.) ASIACRYPT 2009. LNCS, vol. 5912, pp. 250–267. Springer, Heidelberg (2009). doi:10.1007/978-3-642-10366-7_15
35. Rindal, P., Rosulek, M.: Faster malicious 2-party secure computation with online/offline dual execution. In: Holz, T., Savage, S. (eds.) 25th USENIX Security Symposium, pp. 297–314. USENIX Association (2016)
36. 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, Heidelberg (2011). doi:10.1007/978-3-642-20465-4_22
37. Shelat, A., Shen, C.-H.: Fast two-party secure computation with minimal assumptions. In: ACM CCS 2013, pp. 523–534. ACM Press, November 2013
38. Waksman, A.: A permutation network. J. ACM (JACM) **15**(1), 159–163 (1968)
39. Yao, A.C.-C.: Protocols for secure computations (extended abstract). In: 23rd FOCS, pp. 160–164. IEEE Computer Society Press, November 1982
40. Yao, A.C.-C.: How to generate and exchange secrets (extended abstract). In: 27th FOCS, pp. 162–167. IEEE Computer Society Press, October 1986
41. Zahur, S., Rosulek, M., Evans, D.: Two halves make a whole. In: Oswald, E., Fischlin, M. (eds.) EUROCRYPT 2015. LNCS, vol. 9057, pp. 220–250. Springer, Heidelberg (2015). doi:10.1007/978-3-662-46803-6_8