Black-Box Reusable NISC with Random Oracles

Yuval Ishai[1], Dakshita Khurana[2], Amit Sahai[3], and Akshayaram Srinivasan[4]

[1] Technion
[2] UIUC
[3] UCLA
[4] Tata Institute of Fundamental Research

Abstract. We revisit the problem of *reusable* non-interactive secure computation (NISC). A standard NISC protocol for a sender-receiver functionality f enables the receiver to encrypt its input x such that any sender, on input y, can send back a message revealing only $f(x, y)$. Security should hold even when either party can be malicious. A *reusable* NISC protocol has the additional feature that the receiver's message can be safely reused for computing multiple outputs $f(x, y_i)$. Here security should hold even when a malicious sender can learn partial information about the honest receiver's outputs in each session.
We present the first reusable NISC protocol for general functions f that only makes a *black-box* use of any two-message oblivious transfer protocol, along with a random oracle. All previous reusable NISC protocols either made a non-black-box use of cryptographic primitives (Cachin et al., ICALP 2002) or alternatively required a stronger arithmetic variant of oblivious transfer and were restricted to f in NC^1 or similar classes (Chase et al., Crypto 2019). Our result is obtained via a general compiler from standard NISC to reusable NISC that makes use of special type of honest-majority protocols for secure multiparty computation.
Finally, we extend the above main result to reusable *two-sided* NISC, in which two parties can encrypt their inputs in the first round and then reveal different functions of their inputs in multiple sessions. This extension either requires an additional (black-box) use of additively homomorphic commitment or alternatively requires the parties to maintain a state between sessions.

1 Introduction

Consider the following minimal setting for secure computation. There are two parties, a *sender* and a *receiver*, and two rounds of interaction. In the first round, the receiver encrypts its input x and sends the resulting message π_1 to the sender. In the second round, the sender uses the message π_1 and its input y to compute a message π_2. Based on π_2 and its secret randomness, the receiver should compute the output $f(x, y)$, for some predetermined f, but should not learn additional information about the sender's input y.

When the parties are semi-honest, the problem is relatively easy to solve by using garbled circuits [Yao86], under the (minimal) assumption that a two-message oblivious transfer (OT) protocol exists. When security needs to hold against malicious parties, the problem becomes more challenging, and is referred to as *non-interactive secure computation* (NISC) [IKO$^+$11].

NISC is a powerful general-purpose tool for computing on encrypted data. For instance, NISC enables users (acting as receivers) to safely post their encrypted sensitive data on the internet, such that any other user (acting as a sender) can perform a secure computation with them, say to determine whether their profiles match, by sending a single message. However, despite a significant amount of research, all of the existing solutions to NISC are unsatisfactory in either of the following ways:

– *Non-black-box use of cryptography.* The most natural approach for protecting NISC protocols against malicious parties is by using non-interactive zero-knowledge (NIZK) proofs for enforcing honest behavior [CCKM00, HK07, ASH$^+$20]. However, this NIZK-based approach is typically quite impractical, resulting in orders of magnitude of slowdown compared to the semi-honest baseline. A good explanation for this is the fact that such NISC protocols make a *non-black-box* use of the underlying cryptographic primitives, requiring their explicit representation rather than just making an oracle use of their input-output relation.
– *Limited reusability.* Motivated by the inefficiency of non-black-box protocols, several works obtained practical NISC protocols that make a black-box use of cryptographic primitives, typically only a two-message OT protocol and a pseudorandom generator[5] [IKO$^+$11, AMPR14, MR17, DILO22]. In fact, when cast in the "OT-hybrid" model, where the parties can make parallel calls to an ideal OT oracle, these protocols are secure against a *computationally unbounded* malicious sender. Furthermore, they can efficiently achieve full information-theoretic security for functions f in NC^1 and similar classes. A subtle but important vulnerability of these protocols is that they are not fully *reusable*: If the same receiver message is used in multiple sessions to generate malformed sender messages, supposedly for computing $f(x, y_1), f(x, y_2), \ldots$, then even a partial leakage of the receiver's outputs can lead to a total break of security. For example, in the case of zero-knowledge proofs, if the sender can learn whether the receiver accepts several malformed proofs of true statements, it can make the receiver accept a false statement. This limitation was shown in [CDI$^+$19] to be inherent for NISC in the OT-hybrid model with a computationally unbounded sender: There are explicit functions f for which no such NISC protocol can be reusable.
– *Restricted functionality.* To circumvent the above impossibility, Chase et al. [CDI$^+$19] (with subsequent efficiency improvement in [DIO21]) suggested the use of an arithmetic variant of OT, called oblivious linear evaluation (OLE), instead of standard OT. Their main positive result is an information-theoretic reusable NISC protocol for *arithmetic branching programs* in the OLE-hybrid model, efficiently capturing f in NC^1 and similar classes. Beyond the limitation on f, the reusable flavor of OLE required by the protocol of

[5] Since a pseudorandom generator can be constructed from an OT protocol in a black-box way, OT alone suffices.

Chase et al. [CDI$^+$19] is only known from the DCR assumption [Pai99] (or alternatively requires preprocessing) and is considerably more expensive to realize than OT. While it was shown in [CDI$^+$19] how to bootstrap from branching program to circuits, this step requires a non-black-box use of a pseudorandom generator.

The above state of affairs suggests the following open question:

Is there a general-purpose *reusable* NISC protocol that only makes a *black-box* use of a two-message OT protocol?

Shouldn't this be impossible? Recall that the impossibility result from [CDI$^+$19] rules out protocols that make parallel calls to an ideal OT oracle and achieve reusable security against a computationally unbounded sender. Then how can we hope to achieve the goals above? Our key idea for bypassing this impossibility result is to make black-box use of the *next-message functions* and *receiver output function* of a two-message OT protocol. Note that this is different than making black-box use of an ideal OT functionality because it allows for "explaining" the message produced by one of the next-message functions of the OT protocol by revealing the inputs and randomness used to produce that message. Nevertheless, except for the fact that we have to settle for computational security against a malicious sender, this still allows our protocol to make use of any off-the-shelf two-message OT protocol when instantiating our approach.

1.1 Our Contribution

Our main result is an affirmative answer to the above question in the *random oracle model.* This gives the first reusable NISC protocol for general functions f that only makes a black-box use of cryptography. In fact, we show the following more general result.

Theorem 1 (Reusable NISC from NISC, Informal). *There is a reusable NISC protocol for f in the random oracle model that makes a black-box use of any (non-reusable) NISC protocol for a related f'.*

The theorem is proved via a compiler from standard NISC to reusable NISC that makes use of special type of honest-majority MPC protocols. Note that standard NISC can be constructed from any two-message malicious OT in a black-box way [IKO$^+$11]. Since two-message malicious OT can obtained in a black-box way from two-message semi-honest OT in the random oracle model [IKSS22a], we can base our protocol on semi-honest OT. While in this work we focus on feasibility and do not attempt to optimize concrete efficiency, an optimized variant of our construction is likely to yield reusable NISC protocols with good concrete efficiency.

Finally, we extend the above main result to a reusable *two-sided* variant of NISC, in which two parties can encrypt their inputs (x, y) and then reveal different functions f_i of their inputs in multiple sessions.[6] This extension makes

[6] In fact, our result applies to a more general notion of two-sided NISC that strictly generalizes both the above notion and standard (one-sided) NISC.

a black-box use of a “reusable commit-and-prove” primitive which requires the commitments to the secret input to be reusable across different sessions with the verifier. We show how to construct such a primitive by making a black-box use of an additively homomorphic commitment scheme. Alternatively, we can construct this primitive unconditionally in the random oracle model if the parties can maintain an updatable state between sessions.

Theorem 2 (Reusable two-sided NISC from NISC, Informal). *Assume black-box access to a (non-reusable) one-sided NISC protocol and a non-interactive reusable commit-and-prove protocol. Then, there exists a reusable (two-sided) NISC protocol in the random oracle model.*

2 Technical Overview

In this section, we give a high-level overview of the key ideas behind our construction of a black-box reusable NISC protocol in the random oracle model. Later, we explain the additional challenges in extending these ideas to the two-sided setting and discuss our approaches to overcome them.

Reusable NISC Protocol. Recall that a non-interactive secure computation (NISC) protocol for a two-party functionality f is a two-message protocol between a receiver and a sender that delivers the output of f to the receiver. A NISC protocol is said to be reusable if the message from the honest receiver is fixed once and for all and the adversarial sender can execute multiple sessions with the honest receiver. In each such session, the adversarial sender generates a new second round message in the protocol and can learn the output computed by the honest receiver.[7] It can then adaptively decide to continue with the next session or stop the execution. We require the view of the adversarial sender, together with the output of the honest receiver, to be simulatable in an ideal world where the parties only have access to a trusted functionality that implements f.

Impossibility in the OT-hybrid Model. Before we explain our solution, let us first recall the intuition, already discussed in [IKO^{+}11], for why reusable security is challenging for “natural” NISC protocols. Let’s consider an honest receiver who has generated the first round message by making several calls to the OT oracle by acting as the OT receiver. We concentrate on one such call where the receiver’s choice bit is b. A malicious sender who tries to break the security of the protocol can make a guess b' for this bit and give two sender messages (m_0, m_1) such that $m_{b'}$ is correctly generated as per the protocol specification

[7] In the actual definition, we consider a more general situation where the adversary can learn some partial information about the output, such as whether the receiver aborts. This makes reusable security nontrivial even for functionalities such as OLE, where the receiver’s output reveals its input. However, for the sake of this overview, we make the simplifying assumption that the entire receiver output is given to the adversary.

but $m_{1-b'}$ is malformed. It provides these two messages as the sender input to the OT oracle. Now, if the guess b' was correct, the honest receiver does not notice this and continues to compute the output. On the other hand, if the guess was incorrect, then the receiver obtains the malformed sender's message. For natural NISC protocols, this makes the receiver abort. Thus, depending on whether the receiver aborts or not, the sender learns the value of the receiver's choice bit b in this OT execution. This is not a major problem in the single-use NISC setting, as there are standard ways to secret-share the receiver's input so that the receiver's abort event is uncorrelated with its actual input. However, this has a devastating effect in the case of reusable security. Specifically, for each one of the OT executions with the receiver, the sender can learn its choice bit one-at-a-time by mounting the above attack across different sessions. Once the sender does this, there is no hope of protecting the privacy of the receiver's input. Chase et al. [CDI$^+$19] extended this argument to arbitrary protocols, showing that information-theoretic reusable NISC in the OT-hybrid model is impossible. This applies even to simple functionalities, such as the OLE[8] functionality, for which efficient information-theoretic protocols in the OT-hybrid model exist in the non-reusable NISC setting.

Main Goals. Somewhat surprisingly, Chase et al. showed that this impossibility result can be circumvented if we replace OT-hybrid with the OLE-hybrid model. Specifically, they proved that even after many sessions with an honest receiver, a malicious sender cannot obtain any advantage over an ideal execution. Intuitively, unlike the case of OT, each receiver's input to the OLE oracle can only be guessed with negligible probability. This allows the receiver to detect every cheating attempt of the sender with overwhelming probability, thereby preventing the sender from gaining significant information about the OLE inputs. Chase et al. built on this idea and gave a construction of a reusable NISC in the OLE-hybrid model. This positive result showed that if we implement the OLE functionality using a two-message OLE protocol with reusable receiver security[9] in either the CRS/RO model, then we have a reusable NISC protocol with same kind of setup. Unfortunately, such a two-message OLE protocol [CDI$^+$19] is only known from the DCR assumption [Pai99] and makes heavy use of expensive public-key cryptography. Furthermore, Chase et al.'s construction for computing circuits (in contrast to the information-theoretic construction for branching programs) made non-black-box use of a PRG. Given the above state of the art, the two main goals of our work are:

[8] OLE is the arithmetic analogue of OT which takes in a field element x from the receiver, and two field elements (a, b) from the sender and outputs $ax + b$ to the receiver.

[9] In reusable receiver security game, we fix the first round message from the honest receiver and the corrupted sender could generate multiple second round messages. We require the joint distribution of the view of the sender and the receiver's output in each of the sender executions to be indistinguishable to an ideal world where the parties have access to the ideal OLE functionality.

1. Explore new approaches to bypass the impossibility in the OT-hybrid model without resorting to the more expensive OLE primitive.
2. Obtain reusable NISC for *circuits* while only making a black-box use of cryptography.

Our Approach: Making Black-Box use of Two-Message OT. The key approach we take to bypass this impossibility result is to settle for *computational security* against a malicious sender, while only making a black-box use of a *two-message OT protocol*. Before we explain the technical ideas in our construction, let us first explain how black-box use of a two-message OT is different from treating the OT functionality as an oracle (as it is done in the OT-hybrid model). In the OT-hybrid model, the receiver and the sender have access to an OT functionality. The OT functionality takes a choice bit b from the receiver and two messages (m_0, m_1) from the sender and provides m_b to the receiver. The only interface that this model provides is to receive the private inputs from the parties and give outputs. In particular, there is no way to "connect" the inputs that the parties provide to this oracle with the other components in the protocol. On the other hand, in the black-box two-message OT setting[10], we model the oblivious transfer using the cryptographic algorithms that implement this functionality. Specifically, we model a two-message OT protocol as a tuple of algorithms $(\mathsf{OT}_1, \mathsf{OT}_2, \mathsf{out}_{\mathsf{OT})})$. OT_1 is run by the receiver and takes the receiver's choice bit b and outputs the first round message otm_1. OT_2 is run by the sender and takes the receiver's message otm_1, the sender's private input (m_0, m_1) and outputs the second round message otm_2. $\mathsf{out}_{\mathsf{OT}}$ is run by the receiver and takes otm_2 and the receiver's private random tape and outputs m_b. Note that the interface that is provided by these oracles is to take inputs and randomness from the parties and provide *the protocol messages* that they need to send to the other parties. We model these messages as handles and importantly, these handles can be "opened" to the other party. Specifically, the parties can send the input and randomness used in generating these handles to the other party which can then check if this handle was generated correctly by querying the oracles. In other words, one can treat these handles as commitments to the sender and the receiver inputs to the OT functionality. As a result, we can use these commitments as a "link" between the inputs provided by the parties to the OT oracle and the rest of the protocol. In particular, this opens up new avenues to prove that the messages given to these handles are well-formed and hence, do not give rise to an input-dependent abort. Such a mechanism was impossible to achieve in the OT-hybrid model.

Challenges. Can we use this observation to upgrade any NISC protocol in the black-box OT model to have security in the reusable setting? Unfortunately, this does not seem to be the case and let us explain why. Almost all known black-box constructions of NISC use a two-message OT protocol to implement a mechanism called as the watchlists [IPS08]. Roughly, the watchlist mechanism is

[10] We restrict ourselves to the case of a two-message OT protocol as this gives a two-message NISC protocol.

a sophisticated cut-and-choose technique that delivers the input and randomness used by one of the parties in a subset of the executions *privately* to the other party. Each party then checks if the other party behaved honestly in the set of watched executions and if any deviation is detected, the party aborts. If the set of watched executions are chosen *randomly* and *privately*, then this check ensures that a majority of the unwatched executions are emulated honestly. Once this is ensured, all these works have developed clever approaches to robustly combine the outputs from the rest of the executions to compute the output of the functionality. For this to succeed, it is important that watched executions are hidden from the corrupted party before it generates its protocol message. This is typically done by implementing some version of a k-out-of-m OT functionality where one party choose a random subset of size k as part of its watchlist and the functionality delivers the input and randomness of the other party corresponding to each execution in this set. This k-out-of-m OT functionality is implemented via a black-box access to a 1-out-of-2 OT. Specifically, the receiver chooses a random subset of size k and computes an encoding of this set. Each bit of the encoding is used as the choice bit in an execution of an 1-out-of-2 OT protocol. Regrettably, this technique makes these constructions to again suffer from the same problem as the one described earlier. In particular, we observe that the sender can mount a similar selective failure attack (as in the OT-hybrid model) to learn encoding of the random subset sampled by the receiver one bit at a time. Once the sender learns this encoding, it can easily break the privacy of the receiver's input and cheat in all other executions that are not watched.

At a high-level what this attack shows is that we cannot hope to achieve reusable security by relying on any mechanism that hides a part of the receiver's randomness via an 1-out-of-2 OT. All such mechanisms are bound to be broken in the reusable setting as a malicious sender can learn this secret randomness bit-by-bit. In other words, we need a technique where the randomness used in generating the set of watched executions to come solely from the sender's side. This is a bit counter-intuitive as it seems to give the sender the power to fix this secret randomness to any value. Once the sender knows this value it can trivially cheat in the other unopened executions and break the security of the protocol.

Random Oracles to the Rescue. We overcome this conundrum by using random oracles to sample the set of watched executions. Specifically, we pass the sender's message through a random oracle and this gives a subset of the executions to be opened. The correlation-intractability of the random oracle guarantees that the sender does not have the power to fix this set of opened executions to any value of its choice. Importantly, we can ensure that this property holds even in the reusable setting as we can treat the output to every (new) query made to the random oracle as an independently chosen random subset. This idea of using random oracles to sample the set of watched executions is due to Ishai et al. [IKSS22a]. However, their motivation was to remove the use of malicious-secure OTs from the watchlist mechanism whereas our motivation is to obtain a construction in the reusable setting. Coincidentally, the random oracle paradigm used in their work lends itself nicely to solve the above mentioned issue in the reusable setting. This leads to a natural question of whether this idea alone is sufficient to achieve reusability. Unfortunately, this does not seem to be the case and specifically, the protocol from [IKSS22a] is not reusable.

Overview of [IKSS22a]. Before we see why the protocol from [IKSS22a] is not reusable, let us first give a high-level overview of this protocol. The protocol is based on the IPS compiler [IPS08] which makes use of three main ingredients. The first, called the *outer protocol*, is a 2-round, 2-client (namely, the receiver and the sender), m-server MPC protocol for computing the function f. The outer protocol should be secure against malicious adversaries that corrupt either one of the clients and $t = \Omega(m)$ servers, and has the following interaction pattern. In the first round, the clients send a message to each one of the servers using their private inputs. The servers perform some local computation on these messages and send the result of this computation to the receiver in the second round. The receiver then decodes these messages to learn the output of f. The second ingredient, called the *inner protocol*, is a semi-honest secure 2-party protocol for computing the next message function of the servers in the outer protocol. The third ingredient is the *watchlist mechanism* that is implemented using a random oracle. Let now explain how the compiled protocol works.

The sender and the receiver in the compiled protocol generate the first round messages to be sent to each of the servers in the outer protocol. They then start running m executions of the inner protocol where the i-th execution is computing the next message function of the i-th server. The private inputs that the clients use in the i-th inner protocol execution corresponds to the messages that they send to the i-th server in the outer protocol. The output of the inner protocol corresponds to the second round messages sent by the servers in the outer protocol and the receiver decodes these messages to learn the output of the functionality. To ensure that a majority of the inner protocol executions are performed correctly, the watchlist mechanism is used. Specifically, the parties after generating their respective messages to each of the m executions pass these messages to a random oracle that outputs a set K. The parties send their private inputs and randomness for each inner protocol execution in the set K. This is verified by the other party. This ensures that a malicious adversary cannot cheat in a large fraction of the inner protocol executions as otherwise the set K that is output by the random oracle will have a non-empty intersection with the cheating executions. Hence, we can now rely on the security of the outer protocol against a small fraction of the server corruptions to show that the compiled protocol is secure against malicious adversaries.

Key Challenge. To make the above construction reusable secure, we need each of the components used in the compiler to be secure in the reusable setting. As discussed earlier, the watchlist mechanism implemented by the random oracle paradigm is serendipitously suitable for the reusable setting. The inner protocol which is only required to be semi-honest secure is also trivially secure in the reusable setting. The key challenge we face is to make the outer protocol secure in the reusable setting.

2.1 Constructing a Reusable Outer Protocol

Let us first specify the security properties that a reusable outer protocol needs to satisfy.

Security Property. Consider an adversary that corrupts the sender client and a subset of the servers. The honest receiver generates the first round messages to the servers (using its private input) and this message is fixed. The adversary is now allowed to interact with the honest receiver and the servers in many sessions. In each session, the adversary generates a fresh first round sender message to the servers. The honest servers use the fixed receiver's message and the fresh sender message to compute the second round message in the protocol. The adversary sends an arbitrary second round message from the corrupted servers. It obtains the output computed by the honest receiver in this session and adaptively decides whether to continue with one more session or abort. We require the view of the adversary to be simulatable in the ideal world where the parties have access to the ideal functionality.

The Case of Constant-Degree Polynomials and Branching Programs. Before explaining our construction of a reusable outer protocol for computing general circuits, let us first start with a simple case of computing constant degree polynomials. Later, we explain how to extend this construction to securely evaluate branching programs.

Let $(p_1, \ldots, p_\ell)$ be a set of constant-degree polynomials. For the sake of this overview, let us assume that all these polynomials have degree 3. The work of Ishai et al. [IKSS22a] noted that it is not necessary for the outer protocol to satisfy security against stronger malicious adversaries but it is sufficient to start with an outer protocol that is secure against weaker pairwise verifiable adversaries. Pairwise verifiable adversaries are constrained to generate the first round message on behalf of the corrupted clients such that the messages sent to the honest servers pass a pairwise consistency check. Our first observation is that this also extends to the case of reusable security. Specifically, it is sufficient to construct an outer protocol that is reusable secure against pairwise verifiable senders.

Let us first explain the construction of the outer protocol for computing degree-3 polynomials given in [IKSS22a]. In the first round, the clients generate a secret sharing of their private inputs using a 3-multiplicative, pairwise verifiable secret sharing scheme[11] and send the shares to the servers. The servers then locally compute the degree-3 polynomials on these shares to compute the shares of the outputs. This step relies on the fact that the shares are 3-multiplicative. The servers then send the output shares to the receiver.[12] We note that this is protocol is already secure in the reusable setting. This is because the first round

[11] The standard Shamir secret sharing using bivariate polynomials satisfies this property.

[12] We note that the servers have to additionally re-randomize these shares but we ignore this step to keep the exposition simple.

message from the receiver to the servers consists of a secret sharing of its private input and this secret sharing can be reused across multiple sessions.

To construct a reusable protocol for securely evaluating branching programs, we make use of randomized encodings [IK00, AIK04]. It is known from these works that branching programs admit a statistically secure degree-3 randomized encoding. Thus, the task of constructing a reusable outer protocol for the case of branching programs reduces to constructing a reusable outer protocol for computing degree-3 polynomials. However, to generate the randomized encoding we need to additionally secret share the randomness used in computing it. A standard way to do this is for the clients to sample randomness r_1 and r_2 respectively and send the shares in the first round. The servers locally compute the shares of $r_1 + r_2$ and use them to generate the randomized encoding. However, since the first round message from the receiver is fixed once and for all, it means that we need to reuse the receiver's share of the randomness across multiple sessions. Will this affect security? Fortunately, this does not affect the security as the shares of the output are revealed to the receiver and not to the sender. This means that we can fix r_1 to be the all zeroes string and the sender can be tasked with generating a fresh sharing of the randomness in each session to generate the randomized encoding.

Extending to Circuits. All known constructions of randomized encodings for circuits require a PRG [Yao86, IK00, AIK04]. Naïvely incorporating the PRG computation inside the functionality would require non-black-box use of the PRG. Hence, previous NISC protocols for circuits needed to introduce clever mechanisms to ensure that the overall protocol is making black-box use of a PRG. An additional property we need from the outer protocol is that the servers cannot perform any cryptographic operations. This is because the server computations in the IPS compiler are emulated using the inner protocol and if the server computes any cryptographic operations, then functionality that is computed by the inner protocol requires the code of this operation. Therefore, constructions of the outer protocols given in [IPS08, IKSS21, IKSS22a, IKSS22b] required the PRG computations to be done by the clients and the result of these computations to be secret-shared between the servers. Once this is done, the servers can perform a constant degree computation on these shares along with the shares of the input and the randomness to compute a secret sharing of the randomized encoding. Of course, the clients could cheat and send shares of incorrect PRG computations. While there are mechanisms to mitigate this in the single-use setting, unfortunately, this creates serious issues in the reusable setting.

Specifically, consider a malicious adversary that corrupts the sender client and a subset of the servers. The malicious sender client cannot be forced to evaluate the PRGs correctly and hence, could send incorrect sharing of the PRG computations. At a high-level, this means that some entries in the garbled gate table are incorrectly computed. This could force an abort if these particular entries are decrypted in the garbled circuit evaluation. Hence, we need to make sure that the abort event is uncorrelated with the receiver's input. In the single-use setting this was mitigated using a specific garbled circuit construction due to Beaver et al. [BMR90]. In this construction, the value that is carried by each wire is masked with a random bit and thus, we only decrypt the garbled gate entries corresponding to these masked values. This random masking makes it is possible to argue that the abort event is uncorrelated with the receiver's private input. Unfortunately, in the reusable setting, these masks need to be reused as the receiver's first round message is fixed across sessions and hence, this offers no security. Thus, we need a brand new approach to prevent such input-dependent aborts in the reusable setting.

Our Approach: Weakening the Outer Protocol. This problem seems incredibly hard to solve as there are no black-box mechanisms which can force a malicious client to secret share the correct PRG evaluations. In hindsight, this was also the main reason for why the work of Chase et al. [CDI$^+$19] could not provide a black-box construction for the case of circuits. Instead of dealing with this problem at the outer protocol level, we design new mechanisms to deal with this problem in the protocol compiler. (These mechanisms will only apply to our random oracle based compiler, and do not apply to the "plain" OLE-hybrid model considered in [CDI$^+$19].) Specifically, we consider an outer protocol that is only secure against adversaries that compute the PRGs correctly. We call such adversaries as verifiable adversaries. Next, we give the details about our new protocol compiler that uses this weaker outer protocol to construct a reusable NISC.

2.2 A New Protocol Compiler

Our goal is to design a protocol compiler that starts with an outer protocol satisfying reusable security against verifiable adversaries and transforms it into a two-message reusable NISC protocol. In this technical overview, we will only concentrate on proving the reusable security against a malicious sender. Security against malicious receivers follows via standard techniques.

Let us assume that only the sender client needs to compute the PRG evaluations and secret-share them in the outer protocol (in fact, our construction will satisfy this property). Of course, we cannot force the sender client to open all the shares of the PRG computations as this would completely ruin the security of the randomized encoding. Our goal is to design a black-box mechanism that forces the sender to generate correct sharing of the PRG computations without compromising on the randomized encoding security.

We overcome this by adding one more layer of cut-and-choose. Specifically, instead of emulating one execution of the outer protocol (which consists of m servers), we emulate n (for $n = O(\lambda)$) such executions (each containing m servers). In total, we perform $n \cdot m$ executions of the inner protocol. Recall that each message sent by the client to a server in the outer protocol consists of two parts: the share of the client's private input and, if the client was the sender, it additionally consists of the share of the PRG evaluation. In each of the n executions of the outer protocol, we fix the client's shares of the private input to be the same. The sender generates independent PRG evaluations for every execution and generates the shares of these evaluations. If all the emulations are done correctly, then each execution of the outer protocol would be computing a randomized encoding of the function on the same private inputs but using independently chosen random strings. We need to make sure the following two conditions hold: (i) the shares of the private input that the parties use in each execution of the outer protocol are the same, and (ii) the PRG computations and their sharing are performed correctly. Instead of requiring these two conditions to hold exactly, we relax the requirement and ensure that they hold for a large fraction. Specifically, we will make sure that for a large fraction of the servers, the first round messages sent by the malicious sender are the same across all executions and for a large fraction of the executions, the PRG computations and their shares are generated correctly by the sender. We now explain why these two relaxations are sufficient to argue the security of the compiled protocol. The first relaxation does not create any problems we can rely on the security of the outer protocol to additionally corrupt these inconsistent servers (which comprise of a small fraction) and ensure that these inconsistencies do not affect the output obtained by the honest receiver. The second relaxation is a bit more subtle. Note that if the PRG computations are correct, then the receiver's output consists of the evaluation of a properly generated garbled circuit using the same private inputs but using an arbitrary randomness. It follows from the perfect correctness of the garbled circuit evaluation that all these evaluations provide the output of f applied on the private inputs of the clients. Thus, a majority of these values are going to be the same (corresponding to the correct output) and hence, we can correct the errors caused due to incorrect PRG evaluations by computing the majority function locally on all the n outputs.

These two relaxations are ensured via two applications of the random oracle based cut-and-choose paradigm. Specifically, we ask the sender to pass its second round message (corresponding to each one of the $m \cdot n$ executions of the inner protocol) to two random oracles. The first random oracle outputs a subset L_1 of the servers $[m]$ and the second random oracle outputs a subset L_2 of the executions $[n]$. For each server in the set L_1, the sender client opens up the private input and randomness used in generating the inner protocol messages for this server in each of the n executions. The honest receiver checks if these messages are correctly generated and if the share of the private input used in each one of the n executions are identical. For each execution in the set L_2, the sender client opens up the shares of the PRG computations sent to all the servers. The receiver checks if the shares correspond to a correct PRG evaluation. The first check ensures that for a majority of the servers, the malicious sender client is using the same share of the private input and these servers are emulated honestly. The second check ensures that except for a small fraction of the executions, the sender client emulates a verifiable adversary and we can rely on the security of the outer protocol against this weaker class to argue the security of the overall protocol. A pictorial representation of the protocol appears in Figure 2.

Additional Requirement from the Outer Protocol. An astute reader who is familiar with the IPS compiler might have noticed the following major challenge in achieving reusable security. An adversarial sender could potentially cheat in a small number of server emulations, such that this number is small enough to escape the watchlist mechanism with non-negligible probability. To be more concrete, assume that the server only cheats in a single execution. Then, the probability that this execution is a part of the watchlist (that is generated using the random oracle) is roughly $k/m = O(1)$. Though the number of such cheating sessions are small and are not sufficient to break the privacy of the outer protocol, they could decide if the honest receiver outputs $\perp$ or obtains the correct output. Hence, in the simulation, it is important to compute the same output that an honest receiver obtains in these cheating server emulations. To achieve this, we corrupt those cheating servers and learn the share that an honest receiver sent to these cheating servers. We then use this share to compute the output that an honest receiver would have obtained by decrypting the cheating sender message. This is possible if the inner protocol satisfied a special property called output equivocation [IKSS22b]. It was recently shown in [IKSS22b] that any NISC protocol with security against malicious senders satisfies output equivocation.

The above simulation technique does not create an issue in the single-use setting. In particular, we can corrupt the servers corresponding to these cheating executions in the outer protocol and obtain the private share sent by the honest receiver and continue with the simulation. However, this causes a serious problem in the reusable setting. Specifically, in each one of the reuse sessions, the adversarial sender client could cheat in a different set of the server executions and cumulatively learn all the private shares of the honest receiver. If this happens, the malicious sender can learn the private input of the receiver in its entirety.

To deal with this issue, we require the outer protocol to satisfy a stronger property called as error correction [IKSS22a]. Informally, this property requires that the output of the receiver's decoding function depends only on the messages sent from the honest servers and is independent of the messages sent by the corrupt servers. If this property holds, then in each reuse session, we can replace the output of the inner protocol in those cheating executions with a default value and apply the receiver's decoding function on these outputs. It follows from the error correction property that the output of the honest receiver remains the same after we perform this replacement. This helps in proving that an adversarial sender cannot break receiver privacy by cheating in a different set of executions in each reuse session. We use similar techniques as in [IKSS22a] to add this extra error correction property. We note that this property was added to the outer protocol in [IKSS22a] to construct a protocol compiler that only makes use of a semi-honest secure inner protocol. However, in our work, we rely on the error correction property to obtain security in the reusable setting.

2.3 Extension to the Two-Sided Setting

Let us first state the requirements from a two-sided NISC protocol.

Two-Sided Reusable NISC. We say that a NISC protocol is two-sided if the communication channel is bi-directional and the output of f is delivered to both the parties at the end. In a bit more detail, we model f as (f_0, f_1). For each $\beta \in \{0,1\}$, f_β takes in offline inputs x_0^{off} from P_0, x_1^{off} from P_1, a common public online input $x_{\mathsf{pub}}^{\mathsf{on}}$, and an online private input $x_{1-\beta}^{\mathsf{on}}$ from $P_{1-\beta}$ and delivers $f_\beta((x_0^{\mathsf{off}}, x_1^{\mathsf{off}}), x_{\mathsf{pub}}^{\mathsf{on}}, x_{1-\beta}^{\mathsf{on}})$ to P_β. The first round message of the protocol depends only on the offline private inputs and the second round message is generated depending on the online inputs. A two-sided NISC protocol is said to be reusable if an adversary corrupts either one of the parties and fixes the first round of interaction once and for all. It then interacts with the other party in multiple sessions. In every session, the honest party generates a second round message using (adaptively chosen) online private inputs and the adversary generates an arbitrary second round message. The adversary learns the output computed by the honest party in this session and adaptively decides whether to continue with one more session or stop. We require the view of the adversary in the real world to be simulatable in an ideal world with access to a trusted functionality that does the following. The parties send their offline inputs to the functionality in the beginning and interact with the functionality in multiple sessions. In every session, the parties send their online inputs to the functionality and it computes the output of f and delivers the result. We note that this way of modelling the two-sided functionality strictly generalizes the one-sided NISC setting. It also generalizes the prior works on reusable two-round secure computation [BGMM20, BL20, AJJM20, BJKL21, AJJM21, BGSZ22] where the parties commit to their private inputs in the first round and can compute a sequence of functions f_i on the committed inputs by sending different second round messages. We also note that a stricter model where the functionality takes in private online inputs from both the parties (instead of just one as in our case) is impossible to achieve against rushing adversaries.

Additional Challenges. In the two-sided setting, we face some additional challenges. Specifically, we cannot hope to run two instances of the one-sided protocol in the opposite directions to get a two-sided variant. This is because an adversarial client could use two different offline private inputs when acting as the sender and the receiver and learn two different outputs. This will break the security of the two-sided NISC protocol. Therefore, we need an additional mechanism to ensure that the malicious parties are forced to use the same input in those two executions.

Problem with the Standard Approach. A standard approach to do this is to give a zero-knowledge proof that the adversary is using the same input in both the sessions, one where it is acting as the sender and the other where it is acting as the receiver. However, as we are interested in giving a black-box construction, we must be careful with the exact zero-knowledge proof that is used.

A black-box way to prove that the adversary used the same offline private input in both the executions is to commit to these two inputs and then prove that the committed values are equal using a black-box commit-and-prove protocol.

Such a non-interactive black-box commit-and-prove protocol can be constructed in the random oracle model based on the "MPC-in-the-head" approach of Ishai et al. [IKOS07]. In this approach, the prover generates a secret sharing of the committed values and runs an MPC protocol in its head that reconstructs these two values from the shares and checks equality. It generates the view of each party in the MPC protocol and commits to the view of these virtual parties. The prover then passes these commitments through a random oracle to obtain a set of executions to be opened. The verifier checks if the opened views are consistent and if yes, accepts the proof if the output of the MPC protocol is 1. However, this approach does not directly translate to the reusable setting. This is because the commitments to the offline private input when the honest party acts as the receiver are generated in the first round. In particular, the honest party generates a secret sharing of this private input in the commit-and-prove protocol once and commits to these shares in the first round. For every new second round message in the protocol, we need to generate a fresh secret sharing of the sender offline inputs and prove that these shares correspond to the same value that was used in the receiver side. This means that for such reuse session, we need to generate a fresh proof of consistency and this could imply opening a different subset of the shares of the commitment generated in the first round. After a certain number of reuse sessions, we could open all the shares and this affects the privacy of the honest receiver's input.

A Reusable Black-Box Commit-and-Prove. To deal with this issue, we need a reusable variant of the commit-and-prove protocol. In this variant, the commitments to the secret values are generated once and fixed across multiple sessions. These fixed set of commitments allow a prover to prove in zero-knowledge that these secret values satisfy potentially different predicates in each session. The standard commit-and-prove protocols may not satisfy this reusability property. In this work, we give a construction of a reusable commit-and-prove protocol using additively homomorphic commitments. Specifically, we generate a commitment to the secret values using these homomorphic commitments. For each proof, we use the homomorphism property to generate a fresh secret sharing of the committed values. That is, we generate commitments to randomness and use the additive homomorphism to generate a linear secret sharing of the committed values using the committed randomness. Using these fresh sharings, we can now run an MPC protocol in the head to show that the reconstruction of the newly generated shares satisfy the predicate of interest. Specifically, for each reuse session, we generate a fresh set of secret shares and the problem mentioned above does not arise. In the full version, we give a construction such a commit-and-prove in the random oracle model (without any additional assumptions) in a weaker setting where the prover and verifier maintain a state that is updated at the end of every proof execution. This construction is based on proactive MPC protocols which allow an adversary to corrupt a different subset of the parties across different time epochs.

Organization. In Section 3, we give the formal definitions of reusable NISC and reusable two-sided NISC. In Section 4, we give the definition of reusable verifiable client-server protocol and we give the construction in full version. In Section 5, we give the construction of our black-box reusable NISC protocol. In Section 6, we give the definition of a reusable commit-and-prove protocol and give the construction of such a protocol in the full version. In Section 7, we state the main theorem regarding construction of our reusable two-sided NISC protocol and give the construction and the proof of security in the full version.

3 Definitions

In this section, we give the definition of reusable NISC and its two-sided version.

3.1 Reusable NISC Protocol

Let f be a two-party functionality between a receiver and a sender. Let x be the private input of the receiver and y be the private input of the sender. A NISC protocol[13] $(\Pi_1, \Pi_2, \mathsf{out}_\Pi)$ between the receiver and the sender is a two-message, malicious-secure protocol that securely computes the ideal functionality f and delivers the output to the receiver. Specifically, in this protocol, Π_1 is run by the receiver using its private input x to generate the first round message. Π_2 is run by the sender on its private input y and the receiver's first round message to compute the second round message in the protocol. out_Π is run by the receiver on the sender's message and its private random tape to compute the output of f. The security is modelled using the standard real-ideal paradigm. For completeness, we provide this definition in the full version .

A reusable NISC protocol is one where the first round message from the receiver is fixed once and for all and the sender can send multiple second round messages (potentially using different inputs). The receiver computes the output of f on its fixed input and the fresh sender input for each execution. For security, we require this protocol to satisfy standard security against malicious receivers and *reusable* security against malicious senders. In the reusable security game, the adversarial sender is allowed to generate an a priori unbounded polynomial number of second round messages (in an adaptive manner). We now give the formal definition of a reusable NISC protocol.

Definition 1 (Reusable NISC Protocol). *A NISC protocol $(\Pi_1, \Pi_2, \mathsf{out}_\Pi)$ for computing a two-party function f is a reusable NISC protocol if it satisfies standard security against malicious receivers and the following reusable sender security. For any PPT adversary $\mathcal{A}$ that corrupts the sender, there exists a PPT simulator $\mathsf{Sim}_{\Pi,S}$ such that for all non-uniform PPT (stateful) environments $\mathcal{Z}$, the following two distributions are computationally indistinguishable:*

[13] As our main results are in the random oracle model, we can avoid an explicit setup phase that samples the CRS uniformly and instead use the random oracle's output on some default input as the CRS.

- ***Real Execution.*** *The environment $\mathcal{Z}$ provides the private input x to the honest receiver and auxiliary input z to $\mathcal{A}$. The honest receiver generates the first round message in the protocol using x and this message is delivered to $\mathcal{A}$. Repeat the following until $\mathcal{A}$ outputs a special command* STOP*:*
  1. *$\mathcal{Z}$ provides an input y to the adversary and $\mathcal{A}$ generates an arbitrary second round message in the protocol.*
  2. *The honest receiver computes the output of the protocol using out_{Π} on the adversarial sender message and its private random tape.*
  3. *This output is forwarded to $\mathcal{Z}$ which sends some auxiliary information to $\mathcal{A}$.*
  4. *$\mathcal{A}$ either outputs* STOP *or continues to the next iteration.*

  We call each iteration where the adversary generates a second round message as a session. The output of the real execution corresponds to the output of the honest party in each session and the output of $\mathcal{A}$ at the end of all sessions.
- ***Ideal Execution.*** *This corresponds to the ideal world interaction where $\mathsf{Sim}_{\Pi,S}$ and the honest receiver have access a trusted functionality that implements f. The environment $\mathcal{Z}$ delivers the private input x to the honest receiver and auxiliary input z to $\mathsf{Sim}_{\Pi,S}$. The receiver forwards x to the ideal functionality. $\mathsf{Sim}_{\Pi,S}$ can interact with the ideal functionality in an a priori unbounded polynomial number of sessions. In each session,*
  1. *$\mathcal{Z}$ sends a private input y to $\mathsf{Sim}_{\Pi,S}$. $\mathsf{Sim}_{\Pi,S}$ sends an arbitrary input to the ideal functionality or a special instruction to the ideal functionality to deliver $\perp$ to the honest receiver.*
  2. *The trusted functionality returns the output to the receiver depending on $\mathsf{Sim}_{\Pi,S}$'s instruction and this is forwarded to $\mathcal{Z}$.*
  3. *$\mathcal{Z}$ sends some auxiliary information to $\mathsf{Sim}_{\Pi,S}$.*
  4. *$\mathsf{Sim}_{\Pi,S}$ decides whether to continue with one more session or stop.*

  The output of the ideal execution corresponds to the output of the honest party in each session and the output of $\mathsf{Sim}_{\Pi,S}$ at the end of all sessions.

3.2 Reusable Two-Sided NISC

A two-sided NISC protocol for computing a function $f = (f_0, f_1)$ is two-round protocol between P_0 and P_1 such that P_0 gets the output of f_0 and P_1 gets the output of f_1. For each $\beta \in \{0,1\}$, f_β takes in $(x_0^{\mathsf{off}}, x_1^{\mathsf{off}})$ which are the offline inputs of the parties, a common public online input $x_{\mathsf{pub}}^{\mathsf{on}}$, and a private online input $x_{1-\beta}^{\mathsf{on}}$ and delivers the output to P_β.

A two-sided NISC protocol is given by a tuple of algorithms $(\Pi_1, \Pi_2, \mathsf{out}_\Pi)$. Π_1 takes the index $\beta \in \{0,1\}$ of the party, its offline private input x_β^{off} and produces the first round message sent by P_β which is given by $\pi_1^{(\beta)}$. Π_2 takes the index $\beta \in \{0,1\}$ of the party, the public online input $x_{\mathsf{pub}}^{\mathsf{on}}$, the online private input x_β^{on}, the first round message generated by the other party $\pi_1^{(1-\beta)}$ and produces the second round message $\pi_2^{(\beta)}$ of P_β. out_Π takes in the index $\beta \in \{0,1\}$ of the party, its private random tape, and the second round message $\pi_2^{(1-\beta)}$ generated by $P_{1-\beta}$ and produces the output of f_β applied on $((x_0^{\mathsf{off}}, x_1^{\mathsf{off}}), x_{\mathsf{pub}}^{\mathsf{on}}, x_{1-\beta}^{\mathsf{on}})$. As in the one-sided setting, the security is modelled using the standard real-ideal security paradigm.

We say that a two-sided NISC is reusable if the parties can fix the first round message once and for all and send fresh second round message that depends only on the online private input. The parties use out_Π to learn the output of the function computed on their fixed offline private inputs and the new online inputs. We require this protocol to satisfy the following security property.

Definition 2 (Reusable Two-Sided NISC Protocol). *A two-sided NISC protocol $(\Pi_1, \Pi_2, \mathsf{out}_\Pi)$ is a reusable NISC protocol for computing $f = (f_0, f_1)$ if for any PPT adversary $\mathcal{A}$ that corrupts $P_{1-\beta}$ for some $\beta \in \{0,1\}$, there exists a PPT simulator Sim_Π such that for all non-uniform PPT (stateful) environments $\mathcal{Z}$, the following two distributions are computationally indistinguishable:*

- ***Real Execution.*** *For each $b \in \{0,1\}$, the environment delivers the private offline input x_b^{off} to P_b and provides auxiliary input z to $\mathcal{A}$. P_β uses this to generate the first round message in the protocol. The adversary $\mathcal{A}$ receives this first round message and sends the first round message on behalf of corrupt $P_{1-\beta}$. Repeat the following until $\mathcal{A}$ outputs a special command* STOP*:*
  1. *The environment $\mathcal{Z}$ provides an online input $(x_{\mathsf{pub}}^{\mathsf{on}}, x_b^{\mathsf{on}})$ to P_b for each $b \in \{0,1\}$.*
  2. *P_β generates the second round message using the online inputs and this is delivered to $\mathcal{A}$. P_β then receives the second round message sent by $\mathcal{A}$.*
  3. *The honest P_β computes the output of the protocol using out_Π on the adversarial second round message and its private random tape.*
  4. *The output computed by the receiver is delivered to $\mathcal{Z}$ who sends some auxiliary information to $\mathcal{A}$.*
  5. *$\mathcal{A}$ either outputs* STOP *or continues to the next iteration.*

  We call each iteration described above as a session. The output of the real execution corresponds to the output of honest P_β in each session and the output of $\mathcal{A}$ at the end of all sessions.
- ***Ideal Execution.*** *This corresponds to the ideal world interaction where Sim_Π (corrupting $P_{1-\beta}$) and the honest P_β have access a trusted functionality that implements f. For each $b \in \{0,1\}$, the environment delivers the private offline input x_b^{off} to P_b and auxiliary input z to Sim_Π. P_β sends this to the ideal functionality. Sim_Π sends an arbitrary offline input on behalf of $P_{1-\beta}$. Sim_Π interacts with the ideal functionality in an a priori unbounded polynomial number of sessions. In each session,*
  1. *The environment delivers an online input $(x_{\mathsf{pub}}^{\mathsf{on}}, x_b^{\mathsf{on}})$ to P_b for each $b \in \{0,1\}$. P_β forwards this to the ideal functionality.*
  2. *The ideal functionality computes $f_{1-\beta}$ on the fixed offline inputs and the new online input and delivers this output to Sim_Π.*
  3. *Sim_Π can send a special instruction to the ideal functionality to deliver $\perp$ to the honest receiver or sends an online input $(\overline{x}_{\mathsf{pub}}^{\mathsf{on}}, x_{1-\beta}^{\mathsf{on}})$. If $x_{\mathsf{pub}}^{\mathsf{on}} \neq \overline{x}_{\mathsf{pub}}^{\mathsf{on}}$, then the trusted functionality delivers $\perp$ to the receiver. Else, the trusted functionality returns either the output of f_β or $\perp$ to the honest receiver depending on the instruction from Sim_Π.*

4. The output delivered to the receiver is forwarded to $\mathcal{Z}$. $\mathcal{Z}$ sends some auxiliary information Sim_{Π}.

5. Sim_{Π} *then decides to continue with one more execution or stop.*

The output of the ideal execution corresponds to the output of honest P_β in each session and the output of Sim_{Π} *at the end of all sessions.*

4 Reusable Verifiable Client-Server Protocol

In this section, we define and construct a reusable verifiable client-server protocol. This protocol will be used as the main building block in the subsequent sections to construct a black-box reusable (two-sided) NISC. We require this protocol to satisfy the following properties.

- **Reusability:** This property requires that the first round message sent by the receiver to be reusable. To be more precise, the receiver sends a single first round message (depending on its private input) to each of the servers and this message is fixed once and for all. The sender can generate multiple (a priori unbounded polynomial number of) first round messages for different choices of its private input. The servers use the fixed first round message from the receiver and the fresh first round message from the sender to compute a second round message in the protocol. The receiver uses this second round message to compute the output of the functionality on its fixed private input and the (fresh) sender input.
- **Error Correction:** Consider an adversary that corrupts the sender and certain number of servers. This property requires that the output of the receiver's decoding algorithm to remain the same for any choice of second round message sent by the corrupted servers. In other words, the output computed by the receiver is uniquely determined by the messages sent by the honest servers. This property also implies that we can substitute the second round message sent by the adversarial servers with some default values without affecting the receiver's output.
- **Security against Verifiable Adversaries.** As noted in [IKSS22a], there are barriers in obtaining the error correction property against standard malicious adversaries. Hence, [IKSS22a] defined a weaker class of adversaries called *pairwise verifiable adversaries.* Pairwise verifiable adversaries generate the first round message on behalf of the adversarial client to the honest servers such that it passes some pairwise consistency check. They constructed a protocol that had this error correction property against this weaker class. However, we are unable to construct a protocol that satisfies both reusability as well as error correction against pairwise verifiable adversaries. Hence, we further weaken the pairwise verifiable adversaries to *verifiable adversaries* which generate the first round message in the protocol in a much more restricted way. Specifically, if the adversary corrupts a sender client then there is a predicate P' such that the first round messages sent to *all* the honest

servers by the adversary satisfy this predicate.[14]. In other words, there is some *global predicate* P' (instead of pairwise local predicate) that the adversarial sender messages must satisfy. On the other hand, if the adversary corrupts the receiver client then the first round messages sent by the receiver should satisfy some pairwise consistency check w.r.t. to a predicate P (this property is identical to the pairwise verifiable case). It is clear that verifiability restricts the adversarial power even more than pairwise verifiability.

4.1 Definition

We start by describing the syntax of a reusable verifiable client-server protocol.

Syntax. A reusable verifiable client-server protocol between two clients, the receiver R and a sender S and a set of m servers is given by a tuple of algorithms $(\mathsf{Share}_R, \mathsf{ShareInp}_S, \mathsf{ShareRand}_S, \mathsf{Eval}, \mathsf{Dec})$ with the following syntax.[15]

- Share_R takes the private input x of the receiver and outputs the first round message $\{\mathsf{msg}_{R,\mathsf{inp},i}\}_{i\in[m]}$ to be sent to each of the m servers. Recall that this algorithm is only run once and the messages sent to the servers are reused across different iterations with the sender.
- $\mathsf{ShareInp}_S$ takes the private input y of the sender and generates $\{\mathsf{msg}_{S,\mathsf{inp},i}\}_{i\in[m]}$. $\mathsf{ShareRand}_S$ takes a uniform random string from the sender and generates $\{\mathsf{msg}_{S,\mathsf{rand},i}\}_{i\in[m]}$. The first round message from the sender to the i-th server consists of $\{\mathsf{msg}_{S,\mathsf{inp},i}, \mathsf{msg}_{S,\mathsf{rand},i}\}$. We could have included $\mathsf{msg}_{S,\mathsf{rand},i}$ as part of $\mathsf{msg}_{S,\mathsf{inp},i}$ instead of computing it as an output of $\mathsf{ShareRand}_S$. However, we choose to split it into two separate algorithms as this presentation is more suitable to be used in our reusable (two-sided) NISC constructions. Looking ahead, we would require the first part of the sender message $\{\mathsf{msg}_{S,\mathsf{inp},i}\}_{i\in[m]}$ to satisfy local consistency check and the second part $\{\mathsf{msg}_{S,\mathsf{rand},i}\}_{i\in[m]}$ to satisfy global consistency check (see Footnote 14).
- The Eval algorithm takes in the identity i of the server, the first round messages sent by the clients to this server and outputs the second round message $\mathsf{msg}_{2,i}$ to be sent to the receiver.
- The Dec algorithm takes in $\{\mathsf{msg}_{2,i}\}_{i\in[m]}$ and computes the output.

Verifiable Adversary. Before stating the security properties, we start with the definition of a verifiable adversary. A verifiable adversary $\mathcal{A}$ corrupts either one of the clients and a set T of the servers. If the adversary corrupts a client $k \in \{R, S\}$, then $\{\mathsf{msg}_{k,\mathsf{inp},i}\}_{i\in[m]\setminus T}$ satisfies a pairwise consistency predicate P. If $k = S$, then we additionally require $\{\mathsf{msg}_{k,\mathsf{rand},i}\}_{i\in[m]\setminus T}$ to satisfy a global consistency

[14] We are little imprecise here and this global predicate acts only on a part of the sender's message and not on the whole message. To be more specific, the sender's message consists of two parts. We want the first part to satisfy local consistency and the second part to satisfy global consistency.

[15] We implicitly assume that all the algorithms take in the unary encoding of the security parameter 1^λ as part of their inputs.

predicate P'. We note that only the randomness part $\{\mathsf{msg}_{S,\mathsf{rand},i}\}_{i\in[m]\setminus T}$ is required to satisfy the global consistency predicate and it is sufficient for the input part $\{\mathsf{msg}_{S,\mathsf{inp},i}\}_{i\in[m]\setminus T}$ to only satisfy a pairwise consistency check. This property will again be crucially used in the construction of a reusable (two-sided) NISC protocol.

Definition 3 (Pairwise vs Global Predicate). *Let P be a pairwise predicate that takes a client index $k \in \{R, S\}$, two server indices $i, j \in [m]$, the first round message $(\mathsf{msg}_{k,\mathsf{inp},i}, \mathsf{msg}_{k,\mathsf{inp},j})$ sent by the client k to the servers i and j and outputs 1/0. Let P' be a global predicate that takes a set $H \subseteq [m]$, and the second part of the first round message $\{\mathsf{msg}_{S,\mathsf{rand},i}\}_{i\in H}$ sent by the sender S to the servers in H and outputs 1/0.*

Definition 4 (Verifiable Adversary). *An adversary $\mathcal{A}$ corrupting the client k and the set T of the servers is said to be verifiable w.r.t. the pairwise predicate P and global predicate P' if it satisfies the following:*

- *If $k \in \{R, S\}$, then for any two honest servers $i, j \in [m]\setminus T$, $P(k, i, j, \mathsf{msg}_{k,\mathsf{inp},i}, \mathsf{msg}_{k,\mathsf{inp},j}) = 1$ where $\mathsf{msg}_{k,\mathsf{inp},i}$ and $\mathsf{msg}_{k,\mathsf{inp},j}$ are generated by $\mathcal{A}$ in the protocol execution.*
- *If $k = S$, then the output of the predicate $P'([m]\setminus T, \{\mathsf{msg}_{S,\mathsf{rand},i}\}_{i\in[m]\setminus T}) = 1$ where $\{\mathsf{msg}_{S,\mathsf{rand},i}\}_{i\in[m]\setminus T}$ is generated by $\mathcal{A}$ in the protocol execution.*

Security Definition. We are now ready to state the security properties that a reusable verifiable client-server protocol needs to satisfy.

Definition 5 (Reusable Verifiable Client-Server Protocol). *Let f be a two-party functionality. A protocol $\Phi = (\mathsf{Share}_R, \mathsf{ShareInp}_S, \mathsf{ShareRand}_S, \mathsf{Eval}, \mathsf{Dec})$ is a reusable verifiable client-server protocol for computing f against t server corruptions if there exists a pairwise predicate P and a global predicate P' such that:*

1. ***Error Correction:*** *Informally, this requires that the output of* Dec *to be uniquely determined by the messages sent by the honest servers. Formally, for any verifiable adversary $\mathcal{A}$ (see Definition 4) w.r.t. P and P' corrupting the sender client S and a subset T (where $|T| \le t$) of the servers and for any two sets of second round messages $\{\mathsf{msg}_{2,j}\}_{j\in T}$ and $\{\overline{\mathsf{msg}}_{2,j}\}_{j\in T}$, we have:*

$$\mathsf{Dec}(\{\mathsf{msg}_{2,j}\}_{j\notin T}, \{\mathsf{msg}_{2,j}\}_{j\in T}) = \mathsf{Dec}(\{\mathsf{msg}_{2,j}\}_{j\notin T}, \{\overline{\mathsf{msg}}_{2,j}\}_{j\in T})$$

   where $\{\mathsf{msg}_{2,j}\}_{j\notin T}$ consists of the second round messages generated by the honest servers (i.e., $[m] \setminus T$) in the interaction with $\mathcal{A}$. In other words, the output of Dec *remains the same for any choice of second round messages sent by the corrupted servers.*
   Furthermore, consider a setting where the verifiable adversary $\mathcal{A}$ generates multiple first round sender messages that all have the same $\{\mathsf{msg}_{S,\mathsf{inp},j}\}_{j\notin T}$ but potentially different $\{\mathsf{msg}_{S,\mathsf{rand},j}\}_{j\notin T}$. Consider the second round messages generated by the servers for each of these sender messages. For each

set of these second round server messages (corresponding to each new sender message), we require the output of Dec *to be the same. In other words, if a verifiable adversary generates multiple sender messages using the same* $\{\mathsf{msg}_{S,\mathsf{inp},i}\}_{i\notin T}$*, then the output of* Dec *remains the same.*

2. ***Security against Verifiable Receivers:*** *For any (PPT) verifiable adversary $\mathcal{A}$ (see Definition 4) w.r.t. P and P' corrupting the receiver client and (adaptively) corrupting a set T of upto t servers, there exists an (PPT) ideal world simulator* $\mathsf{Sim}_{\Phi,R}$ *such that for any choice of private input y of the honest sender client, the following two distributions are computationally indistinguishable:*
   - ***Real Execution.*** *The verifiable adversary $\mathcal{A}$ interacts with the honest parties (the honest sender client and set of uncorrupted servers) in the protocol. The output of the real execution consists of the output of the verifiable adversary $\mathcal{A}$.*
   - ***Ideal Execution.*** *This corresponds to the ideal world interaction where* $\mathsf{Sim}_{\Phi,R}$ *and the honest sender client have access to the trusted party implementing f. The honest sender client sends its input y to f and* $\mathsf{Sim}_{\Phi,R}$ *sends an arbitrary input. The trusted functionality returns the output of f to* $\mathsf{Sim}_{\Phi,R}$*. The output of the ideal execution corresponds to the output of* $\mathsf{Sim}_{\Phi,R}$*.*
3. ***Reusable Security against Verifiable Senders:*** *For any (PPT) verifiable adversary $\mathcal{A}$ (see Definition 4) w.r.t. P and P' corrupting the sender client and a set of servers defined as below, there exists an ideal world (PPT) simulator* $\mathsf{Sim}_{\Phi,S}$ *such that for all non-uniform PPT (stateful) environments $\mathcal{Z}$, the following two distributions are computationally indistinguishable:*
   - ***Real Execution.*** *$\mathcal{Z}$ delivers the private input x to the honest receiver and auxiliary input z to $\mathcal{A}$. The receiver uses this private input to generate the first round message in the protocol. The adversary $\mathcal{A}$ corrupts a set T_1 of the servers and gets the first round messages sent by the honest receiver to T_1. Repeat the following until adversary $\mathcal{A}$ outputs a special command* STOP*:*
     (a) $\mathcal{Z}$ delivers the private input y to $\mathcal{A}$. $\mathcal{A}$ adaptively corrupts a set T of the servers and sends the first round message to the servers $[m] \setminus (T \cup T_1)$. Note that adversary does not receive the first round messages sent by the honest receiver to the servers indexed by T. Further, this set T could be different across each execution but we require that $|T \cup T_1| \leq t$. We additionally require the adversary to be verifiable w.r.t. to the predicates P and P' where the set of corrupted servers is given by $T \cup T_1$.
     (b) For each server in $[m] \setminus (T \cup T_1)$, we run Eval *on the first round message sent by the honest receiver and the first round message sent by the adversary in the previous step. The adversary sends arbitrary second round messages from the corrupted servers given by $T \cup T_1$.*
     (c) We run Dec *on the second round messages sent by the servers (both honest and the corrupt) and send this output to $\mathcal{Z}$.*
     (d) $\mathcal{Z}$ sends some auxiliary information to $\mathcal{A}$.

(e) $\mathcal{A}$ outputs the special symbol STOP *or decides to continue to the next iteration.*

We call each iteration described above as a session. The output of the real execution corresponds to the output of the receiver in each session and the output of $\mathcal{A}$ at the end of all the executions.

- ***Ideal Execution.*** *This corresponds to the ideal world interaction where $\mathsf{Sim}_{\Phi,S}$ and the honest receiver client have access to the trusted party that implements f. The environment delivers an input x to the receiver and auxiliary input z to Sim_{Π}. The receiver sends this to f. $\mathsf{Sim}_{\Phi,S}$ interacts with the ideal functionality in an a priori unbounded polynomial number of sessions. In each session,*
  (a) $\mathcal{Z}$ sends the private input y to $\mathsf{Sim}_{\Phi,S}$. $\mathsf{Sim}_{\Phi,S}$ sends an arbitrary input to the ideal functionality.
  (b) The trusted functionality returns the output delivered to the receiver to $\mathcal{Z}$.
  (c) $\mathcal{Z}$ sends some auxiliary information to $\mathsf{Sim}_{\Phi,S}$.
  (d) $\mathsf{Sim}_{\Phi,S}$ decides whether to continue with one more execution or stop.

  The output of the ideal execution corresponds to the output of the receiver in each session and the output of $\mathsf{Sim}_{\Phi,S}$ at the end of all executions.

We give the construction of reusable outer protocol in the full version.

5 Black-Box Reusable NISC

In this section, we give a construction of a black-box resusable NISC protocol. Specifically, we give a black-box transformation from a (non-reusable) NISC protocol to a reusable NISC protocol in the random oracle model. The main theorem we will prove in this section is:

Theorem 3. *Assume black-box access to a (non-reusable) NISC protocol. Then, there exists a reusable NISC protocol in the random oracle model.*

5.1 Construction

We first define a weaker variant of reusable security. In this variant, the resuable security needs to hold only against a weaker class of adversarial senders called as explainable senders [HIK^{+}11]. Intuitively, an explainable sender is required to give an explanation on how it generates the second round message in the protocol. This explanation consists of its private input and the random tape. If this explanation is invalid, we replace the output of the honest receiver with $\perp$. We give the formal definition of this variant below.

Definition 6 (Reusable Security against Explainable Senders). *This requirement is the same as the one given in Definition 2 except that in the real execution, the malicious adversary that corrupts the sender has to output an explanation of how it generated the second round message in each iteration. This* *explanation comprises of its input y and a random tape r that it used to generate the second round message. If this explanation is valid, we run the receiver's output decoding algorithm on the adversarial sender message and provide the output to the adversary. If the explanation is invalid, we replace the output of the receiver in that particular iteration with $\perp$.*

We observe that any (non-reusable) NISC protocol satisfies reusable security against explainable senders. This follows directly from the perfect correctness of evaluation algorithm and indistinguishability-based security of the receiver's message against semi-malicious senders (which is implied by security against malicious senders).

Proposition 1. *Any NISC protocol satisfies standard security against malicious receivers and reusable security against explainable senders.*

We are now ready to describe our construction.

Building Blocks. The construction uses the following building blocks:

1. A reusable verifiable client-server protocol $(\mathsf{Share}_R, \mathsf{ShareInp}_S, \mathsf{ShareRand}_S, \mathsf{Eval}, \mathsf{Dec})$ w.r.t. pairwise predicate P and global predicate P' for computing f against $t = 4\lambda$ server corruptions (see Definition 5). Let $m = 20\lambda + 1$ be the number of servers in this protocol (which follows the bounds on the pairwise verifiable 3-multiplicative, t-error-correctable secret sharing). Our construction given in the full version ensures that Eval algorithm does not compute any cryptographic operations.
2. A NISC protocol $(\Pi_{i,1}, \Pi_{i,2}, \mathsf{out}_{\Pi_i})$ for computing $\mathsf{Eval}(i,\cdot,\cdot)$ (i.e., the computation done by the i-th server) for each $i \in [m]$. As we are working in the random oracle model, the CRS can be sampled as the output of the random oracle on some default value. From observation 1, we infer that this protocol satisfies reusable security against explainable senders and standard security against malicious receivers.
3. A straight-line extractable non-interactive commitment $(\mathsf{Com}, \mathsf{Open})$ in the random oracle model (see [Pas03]). We require this commitment to be computationally hiding and statistically binding.
4. Let $n = 4\lambda$. Two hash functions $H_1 : \{0,1\}^* \to (\{0,1\}^{k_m})^m$ and $H_2 : \{0,1\}^* \to (\{0,1\}^{k_n})^n$ that are modelled as random oracles. Here, k_m and k_n are the number of random bits to needed to toss a biased coin that outputs 1 with probability $p_m = \frac{\lambda}{2m}$ and $p_n = \frac{\lambda}{2n}$ respectively. We model the output of hash functions H_1 and H_2 as subsets of $[m]$ and $[n]$ respectively where each element of the set is included independently with probability p_m and p_n respectively.

Description of Protocol. The formal description of the protocol is given in Figure 1. A pictorial representation of our construction is given in Figure 2.

Proof of Security. We defer the proof of security to the full version.

- **Round-1:** The receiver on private input x does the following:
  1. It computes $(\mathsf{msg}_{R,\mathsf{inp},1}, \ldots, \mathsf{msg}_{R,\mathsf{inp},m}) \leftarrow \mathsf{Share}_R(x)$.
  2. For each $i \in [m]$ and $j \in [n]$,
     (a) It samples a uniform random tape $r_{i,j}$ to be used in the protocol Π_i.
     (b) It computes $\pi_{i,j,1} := \Pi_{i,1}(\mathsf{msg}_{R,\mathsf{inp},i}; r_{i,j})$, $a_i \leftarrow \mathsf{Com}(\mathsf{msg}_{R,\mathsf{inp},i})$ and $b_{i,j} \leftarrow \mathsf{Com}(r_{i,j})$.
  3. It computes $K_1 = H_1(\mathsf{tag}_R, \{\pi_{i,j,1}, a_i, b_{i,j}\}_{i\in[m],j\in[n]})$ where $\mathsf{tag}_R \leftarrow \{0,1\}^\lambda$ and interprets K_1 as a subset of $[m]$.
  4. It sends $(\{\pi_{i,j,1}, a_i, b_{i,j}\}_{i\in[m],j\in[n]}, \mathsf{tag}_R, \{\mathsf{Open}(a_i), \mathsf{Open}(b_{i,j})\}_{i\in K_1,j\in[n]})$ as the first round message.
- **Round-2:** The sender on private input y does the following:
  1. **Check Phase:**
     (a) It recomputes K_1 as in step-3 of round-1 and checks if the openings are valid.
     (b) For each $i \in K_1$ and for each $j \in [n]$, it checks if $\pi_{i,j,1} = \Pi_{i,1}(\mathsf{msg}_{R,\mathsf{inp},i}; r_{i,j})$.
     (c) For each $i, i' \in K_1$, it checks if $\mathsf{msg}_{R,\mathsf{inp},i}$ and $\mathsf{msg}_{R,\mathsf{inp},i'}$ pass the pairwise consistency check P.
  2. If any of the above checks fail, it aborts.
  3. Else, it computes $(\mathsf{msg}_{S,\mathsf{inp},1}, \ldots, \mathsf{msg}_{S,\mathsf{inp},m}) \leftarrow \mathsf{ShareInp}_S(x)$.
  4. For each $j \in [n]$, it independently runs $\mathsf{ShareRand}_S$ to obtain $(\mathsf{msg}_{S,\mathsf{rand},1,j}, \ldots, \mathsf{msg}_{S,\mathsf{rand},m,j})$.
  5. For each $i \in [m]$ and $j \in [n]$,
     (a) It samples a uniform random tape $s_{i,j}$ to be used in the protocol Π_i.
     (b) It computes $\pi_{i,j,2} := \Pi_{i,2}(\pi_{i,j,1}, (\mathsf{msg}_{S,\mathsf{inp},i}, \mathsf{msg}_{S,\mathsf{rand},i,j}); s_{i,j})$ and $\mathsf{com}_{i,j} \leftarrow \mathsf{Com}(\pi_{i,j,2})$.
     (c) It computes $c_i \leftarrow \mathsf{Com}(\mathsf{msg}_{S,\mathsf{inp},i})$, $d_{i,j} \leftarrow \mathsf{Com}(s_{i,j})$, and $e_{i,j} \leftarrow \mathsf{Com}(\mathsf{msg}_{S,\mathsf{rand},i,j})$.
  6. It computes $L_1 = H_1(\mathsf{tag}_S, \{\mathsf{com}_{i,j}, c_i, d_{i,j}, e_{i,j}\}_{i\in[m],j\in[n]})$ and $L_2 = H_2(\mathsf{tag}_S\{\mathsf{com}_{i,j}, c_i, d_{i,j}, e_{i,j}\}_{i\in[m],j\in[n]})$ where $\mathsf{tag}_S \leftarrow \{0,1\}^\lambda$ and interprets L_1 as a subset of $[m]$ and L_2 as a subset of $[n]$.
  7. It sends
     (a) $\{\mathsf{com}_{i,j}, c_i, d_{i,j}, e_{i,j}\}_{i\in[m],j\in[n]}$, tag_S.
     (b) $\{\mathsf{Open}(\mathsf{com}_{i,j}), \mathsf{Open}(c_i), \mathsf{Open}(d_{i,j}), \mathsf{Open}(e_{i,j})\}_{i\in L_1,j\in[n]}$.
     (c) $\{\mathsf{Open}(e_{i,j})\}_{i\in[m],j\in L_2}$ and $\{\mathsf{Open}(\mathsf{com}_{i,j})\}_{i\in[m],j\notin L_2}$.
- **Output Computation:** The receiver does the following:
  1. **Check Phase:**
     (a) It recomputes L_1 and L_2 as in Step-6 of round-2 and checks if all the openings are valid.
     (b) Using the openings to the commitments $\mathsf{com}_{i,j}$, c_i, $d_{i,j}$ and $e_{i,j}$ given by the sender,
        i. It checks if for each $i \in L_1$ and $j \in [n]$ that $\pi_{i,j,2} = \Pi_{i,2}(\pi_{i,j,1}, (\mathsf{msg}_{S,\mathsf{inp},i}, \mathsf{msg}_{S,\mathsf{rand},i,j}); s_{i,j})$.
        ii. For each $i, i' \in L_1$, it checks if $\mathsf{msg}_{S,\mathsf{inp},i}$ and $\mathsf{msg}_{S,\mathsf{inp},i'}$ pass the pairwise consistency check P.
        iii. For each $j \in L_2$, it checks if $\{\mathsf{msg}_{S,\mathsf{rand},i,j}\}_{i\in[m]}$ pass the global predicate check P'.
  2. If any of the above checks fail, it aborts.
  3. Else, for each $j \in [n] \setminus L_2$,
     (a) It computes $\mathsf{msg}_{2,i,j} \leftarrow \mathsf{out}_{\Pi_i}(\pi_{i,j,2}, r_{i,j})$ for each $i \in [m]$.
     (b) It computes $\alpha_j = \mathsf{Dec}(\{\mathsf{msg}_{2,i,j}\}_{i\in[m]})$.
  4. o/p $\mathsf{Majority}(\{\alpha_j\}_{j\in[n]\setminus L_2})$.

Fig. 1: Construction of Reusable Black-Box NISC Protocol

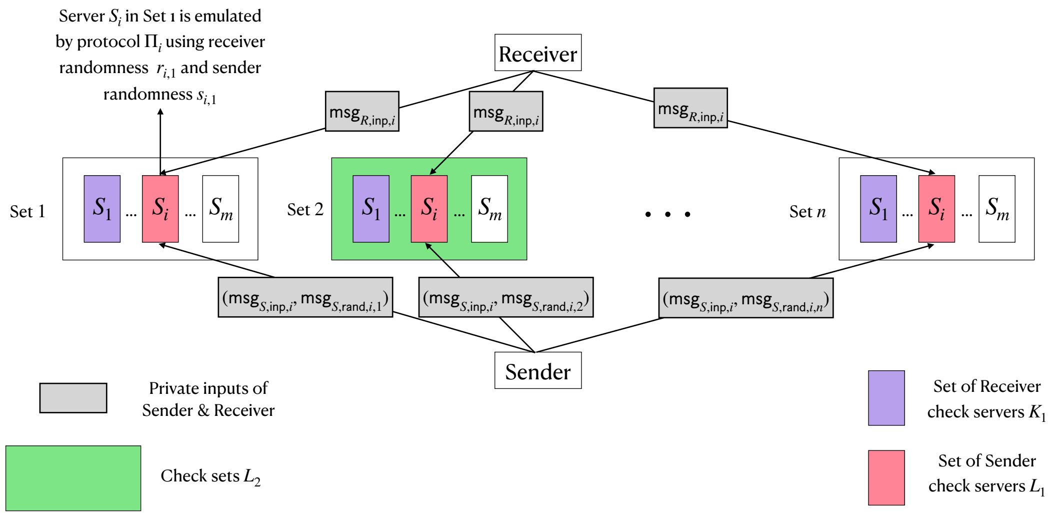
Server S_i in Set 1 is emulated by protocol Π_i using receiver randomness $r_{i,1}$ and sender randomness $s_{i,1}$
Receiver
$\mathsf{msg}_{R,\mathsf{inp},i}$
$\mathsf{msg}_{R,\mathsf{inp},i}$
$\mathsf{msg}_{R,\mathsf{inp},i}$
Set 1
S_1 ... S_i ... S_m
Set 2
S_1 ... S_i ... S_m
. . .
Set n
S_1 ... S_i ... S_m
$(\mathsf{msg}_{S,\mathsf{inp},i}, \mathsf{msg}_{S,\mathsf{rand},i,1})$
$(\mathsf{msg}_{S,\mathsf{inp},i}, \mathsf{msg}_{S,\mathsf{rand},i,2})$
$(\mathsf{msg}_{S,\mathsf{inp},i}, \mathsf{msg}_{S,\mathsf{rand},i,n})$
Sender
Private inputs of Sender & Receiver
Set of Receiver check servers K_1
Check sets L_2
Set of Sender check servers L_1

Fig. 2: Pictorial Representation of the Protocol. For each $i \in K_1$, the receiver opens $(\mathsf{msg}_{R,\mathsf{inp},i}, \{r_{i,j}\}_{j\in[n]})$. Similarly, for each $i \in L_1$, the sender opens $(\mathsf{msg}_{S,\mathsf{inp},i}, \{\mathsf{msg}_{S,\mathsf{rand},i,j}, s_{i,j}\}_{j\in[n]})$. For each $j \in L_2$, the sender opens $(\mathsf{msg}_{S,\mathsf{rand},1,j}, \ldots, \mathsf{msg}_{S,\mathsf{rand},m,j})$.

6 Non-Interactive Reusable Commit-and-Prove

In this section, we define and construct a non-interactive reusable commit-and-prove protocol. This protocol will be used as a key building block in the next section to construct a two-sided reusable NISC protocol.

6.1 Definition

Syntax. A non-interactive reusable commit-and-prove protocol is given by a tuple of algorithms $(\mathsf{Com}, \mathsf{Open}, \mathsf{Extract}, \mathsf{Prove}, \mathsf{Verify})$ with the following syntax.[16]

- Com : It takes a message x as input and outputs a commitment com to this message. We require this commitment to be computationally hiding and statistically binding.
- Open : It comprises of the openings to the commitments.
- $\mathsf{Extract}$: It takes as input a commitment com and outputs the message inside this commitment.
- Prove : It takes as input a sequence of commitments $(\mathsf{com}_1, \ldots, \mathsf{com}_n)$, a function f and their openings $(\mathsf{Open}(\mathsf{com}_1), \ldots, \mathsf{Open}(\mathsf{com}_n))$ as input and outputs a proof π.

[16] We implicitly assume that all these algorithms have access to a random oracle and hence, do not include an explicit setup phase. We also assume that all the algorithms take 1^λ as an additional input.

– Verify : It takes a sequence of commitments $(\mathsf{com}_1, \ldots, \mathsf{com}_n)$, a function f and a proof π as input and outputs 1/0 indicating whether the proof is accepting or rejecting.

We now state the properties that such a commit-and-prove protocol must satisfy.

Definition 7 (Non-Interactive Reusable Commit-and-Prove). *A tuple of algorithms* $(\mathsf{Com}, \mathsf{Open},$ $\mathsf{Extract}, \mathsf{Prove}, \mathsf{Verify})$ *is said to be a non-interactive reusable commit-and-prove protocol if it satisfies the following properties:*

– $(\mathsf{Com}, \mathsf{Open})$ *is a computationally hiding and statistical binding commitment scheme.* $\mathsf{Extract}$ *is a straight-line extractor for the commitment scheme.*
– ***Completeness.*** *We require that:*

$$\Pr[\mathsf{Verify}(X, \mathsf{Prove}(X, \mathsf{Open}(\mathsf{com}_1), \ldots, \mathsf{Open}(\mathsf{com}_n))) = 1] = 1$$

where $(\mathsf{com}_1, \ldots, \mathsf{com}_n)$ *be a sequence of commitments to the messages* $(x_1, \ldots, x_n)$, f *be a function such that* $f(x_1, \ldots, x_n) = 1$ *and* $X = (\mathsf{com}_1, \ldots, \mathsf{com}_n, f)$.
– ***Soundness.*** *Let* P^* *be a non-uniform PPT prover. We require the probability that* P^* *wins the following soundness game to be negligible.*
  - $(\mathsf{com}_1, \ldots, \mathsf{com}_n, f, \pi) \leftarrow P^*(1^\lambda)$.
  - *Let* $(x_1, \ldots, x_n)$ *be the output of* $\mathsf{Extract}$ *on inputs* $\mathsf{com}_1, \ldots, \mathsf{com}_n$ *respectively.*
  - *If* $f(x_1, \ldots, x_n) = 0$ *and* $\mathsf{Verify}(\mathsf{com}_1, \ldots, \mathsf{com}_n, f, \pi) = 1$*, then the prover wins this game.*
– ***Resuable Zero-Knowledge.*** *There exists a PPT simulator* Sim *such that for every non-uniform PPT verifier* V^**, we have:*

$$\mathsf{Real}(V^*) \approx_c \mathsf{Ideal}(\mathsf{Sim}, V^*)$$

where Real *and* Ideal *experiments are described in Figure 3.*

We defer the construction and proof of security to the full version.

7 Black-Box Reusable Two-Sided NISC

In this section, we give a construction of a black-box reusable two-sided NISC. The main theorem we show here is:

Theorem 4. *Assume black-box access to:*

1. *A (non-reusable) one-sided NISC protocol.*
2. *A non-interactive reusable commit-and-prove protocol satisfying Definition 7.*

Then, there exists a reusable (two-sided) NISC protocol in the random oracle model.

We give the proof of this theorem in the full version.

```
$\mathsf{Real}(V^*)$

1. $(x_1, \ldots, x_\ell, n) \leftarrow V^*(1^\lambda)$.
2. $\mathsf{com}_i \leftarrow \mathsf{Com}(x_i)$ for all $i \in [\ell]$.
3. Set $\mathsf{com}_i = \perp$ for all $i \in [\ell+1, n]$ and $\pi = \perp$.
4. Run until $V^*$ outputs a special symbol STOP :
   (a) $(n', x_{\ell+1}, \ldots, x_{n'}, f) \leftarrow V^*(\mathsf{com}_1, \ldots, \mathsf{com}_n, \pi)$.
   (b) Update the value of $n$ with $n'$.
   (c) Compute $\mathsf{com}_i \leftarrow \mathsf{Com}(x_i)$ for all $i \in [\ell + 1, n]$.
   (d) Set $X = (\mathsf{com}_1, \ldots, \mathsf{com}_n, f)$ and $w = (\mathsf{Open}(\mathsf{com}_1), \ldots, \mathsf{Open}(\mathsf{com}_n))$.
   (e) If $f(x_1, \ldots, x_n) = 1$, compute $\pi \leftarrow \mathsf{Prove}(X, w)$.
5. Output the final view of $V^*$.
```

```
$\mathsf{Ideal}(\mathsf{Sim}, V^*)$

1. $(x_1, \ldots, x_\ell, n) \leftarrow V^*(1^\lambda)$.
2. $\mathsf{com}_i \leftarrow \mathsf{Com}(x_i)$ for all $i \in [\ell]$.
3. Set $\mathsf{com}_i = \perp$ for all $i \in [\ell+1, n]$ and $\pi = \perp$.
4. Run until $V^*$ outputs a special symbol STOP :
   (a) $(n', x_{\ell+1}, \ldots, x_{n'}, f) \leftarrow V^*(\mathsf{com}_1, \ldots, \mathsf{com}_n, \pi)$.
   (b) Update the value of $n$ with $n'$.
   (c) Compute $\mathsf{com}_i \leftarrow \mathsf{Com}(x_i)$ for all $i \in [\ell + 1, n]$.
   (d) Set $X = (\mathsf{com}_1, \ldots, \mathsf{com}_n, f)$ and $w = (\mathsf{Open}(\mathsf{com}_1), \ldots, \mathsf{Open}(\mathsf{com}_n))$.
   (e) If $f(x_1, \ldots, x_n) = 1$, compute $\pi \leftarrow \mathsf{Sim}(1^\lambda, X)$.
5. Output the final view of $V^*$.
```

Fig. 3: Descriptions of Real and Ideal experiments.

Acknowledgments. Y. Ishai was supported in part by ERC Project NTSC (742754), BSF grant 2018393, ISF grant 2774/20, and a Google Faculty Research Award. D. Khurana was supported in part by DARPA SIEVE award and a gift from Visa Research. A. Sahai was supported in part from a Simons Investigator Award, DARPA SIEVE award, NTT Research, NSF Frontier Award 1413955, BSF grant 2012378, a Xerox Faculty Research Award, a Google Faculty Research Award, and an Okawa Foundation Research Grant. This material is based upon work supported by the Defense Advanced Research Projects Agency through Award HR00112020024. A. Srinivasan was supported in part by a SERB startup grant and Google India Research Award.

References

AIK04. Benny Applebaum, Yuval Ishai, and Eyal Kushilevitz. Cryptography in NC^0. In *45th FOCS*, pages 166–175. IEEE Computer Society Press, October 2004.

AJJM20. Prabhanjan Ananth, Abhishek Jain, Zhengzhong Jin, and Giulio Malavolta. Multi-key fully-homomorphic encryption in the plain model. In Rafael Pass and Krzysztof Pietrzak, editors, *TCC 2020, Part I*, volume 12550 of *LNCS*, pages 28–57. Springer, Heidelberg, November 2020.

AJJM21. Prabhanjan Ananth, Abhishek Jain, Zhengzhong Jin, and Giulio Malavolta. Unbounded multi-party computation from learning with errors. In Anne

Canteaut and François-Xavier Standaert, editors, *EUROCRYPT 2021, Part II*, volume 12697 of *LNCS*, pages 754–781. Springer, Heidelberg, October 2021.

AMPR14. Arash Afshar, Payman Mohassel, Benny Pinkas, and Ben Riva. Non-interactive secure computation based on cut-and-choose. In Phong Q. Nguyen and Elisabeth Oswald, editors, *EUROCRYPT 2014*, volume 8441 of *LNCS*, pages 387–404. Springer, Heidelberg, May 2014.

ASH$^+$20. Jackson Abascal, Mohammad Hossein Faghihi Sereshgi, Carmit Hazay, Yuval Ishai, and Muthuramakrishnan Venkitasubramaniam. Is the classical GMW paradigm practical? the case of non-interactive actively secure 2pc. In Jay Ligatti, Xinming Ou, Jonathan Katz, and Giovanni Vigna, editors, *CCS*, pages 1591–1605. ACM, 2020.

BGMM20. James Bartusek, Sanjam Garg, Daniel Masny, and Pratyay Mukherjee. Reusable two-round MPC from DDH. In Rafael Pass and Krzysztof Pietrzak, editors, *TCC 2020, Part II*, volume 12551 of *LNCS*, pages 320–348. Springer, Heidelberg, November 2020.

BGSZ22. James Bartusek, Sanjam Garg, Akshayaram Srinivasan, and Yinuo Zhang. Reusable two-round MPC from LPN. In Goichiro Hanaoka, Junji Shikata, and Yohei Watanabe, editors, *PKC*, volume 13177 of *Lecture Notes in Computer Science*, pages 165–193. Springer, 2022.

BJKL21. Fabrice Benhamouda, Aayush Jain, Ilan Komargodski, and Huijia Lin. Multiparty reusable non-interactive secure computation from LWE. In Anne Canteaut and François-Xavier Standaert, editors, *EUROCRYPT 2021, Part II*, volume 12697 of *LNCS*, pages 724–753. Springer, Heidelberg, October 2021.

BL20. Fabrice Benhamouda and Huijia Lin. Mr NISC: Multiparty reusable non-interactive secure computation. In Rafael Pass and Krzysztof Pietrzak, editors, *TCC 2020, Part II*, volume 12551 of *LNCS*, pages 349–378. Springer, Heidelberg, November 2020.

BMR90. Donald Beaver, Silvio Micali, and Phillip Rogaway. The round complexity of secure protocols (extended abstract). In *22nd ACM STOC*, pages 503–513. ACM Press, May 1990.

CCKM00. Christian Cachin, Jan Camenisch, Joe Kilian, and Joy Müller. One-round secure computation and secure autonomous mobile agents. In Ugo Montanari, José D. P. Rolim, and Emo Welzl, editors, *ICALP*, volume 1853 of *Lecture Notes in Computer Science*, pages 512–523. Springer, 2000.

CDI$^+$19. Melissa Chase, Yevgeniy Dodis, Yuval Ishai, Daniel Kraschewski, Tianren Liu, Rafail Ostrovsky, and Vinod Vaikuntanathan. Reusable non-interactive secure computation. In Alexandra Boldyreva and Daniele Micciancio, editors, *CRYPTO 2019, Part III*, volume 11694 of *LNCS*, pages 462–488. Springer, Heidelberg, August 2019.

DILO22. Samuel Dittmer, Yuval Ishai, Steve Lu, and Rafail Ostrovsky. Authenticated garbling from simple correlations. *IACR Cryptol. ePrint Arch.*, page 836, 2022.

DIO21. Samuel Dittmer, Yuval Ishai, and Rafail Ostrovsky. Line-point zero knowledge and its applications. In Stefano Tessaro, editor, *ITC 2021*, volume 199 of *LIPIcs*, pages 5:1–5:24. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2021.

HIK$^+$11. Iftach Haitner, Yuval Ishai, Eyal Kushilevitz, Yehuda Lindell, and Erez Petrank. Black-box constructions of protocols for secure computation. *SIAM J. Comput.*, 40(2):225–266, 2011.

HK07. Omer Horvitz and Jonathan Katz. Universally-composable two-party computation in two rounds. In Alfred Menezes, editor, *CRYPTO 2007*, volume 4622 of *LNCS*, pages 111–129. Springer, Heidelberg, August 2007.

IK00. Yuval Ishai and Eyal Kushilevitz. Randomizing polynomials: A new representation with applications to round-efficient secure computation. In *41st FOCS*, pages 294–304. IEEE Computer Society Press, November 2000.

IKO$^+$11. Yuval Ishai, Eyal Kushilevitz, Rafail Ostrovsky, Manoj Prabhakaran, and Amit Sahai. Efficient non-interactive secure computation. In Kenneth G. Paterson, editor, *EUROCRYPT 2011*, volume 6632 of *LNCS*, pages 406–425. Springer, Heidelberg, May 2011.

IKOS07. Yuval Ishai, Eyal Kushilevitz, Rafail Ostrovsky, and Amit Sahai. Zero-knowledge from secure multiparty computation. In David S. Johnson and Uriel Feige, editors, *39th ACM STOC*, pages 21–30. ACM Press, June 2007.

IKSS21. Yuval Ishai, Dakshita Khurana, Amit Sahai, and Akshayaram Srinivasan. On the round complexity of black-box secure MPC. In Tal Malkin and Chris Peikert, editors, *CRYPTO 2021, Part II*, volume 12826 of *LNCS*, pages 214–243, Virtual Event, August 2021. Springer, Heidelberg.

IKSS22a. Yuval Ishai, Dakshita Khurana, Amit Sahai, and Akshayaram Srinivasan. Round-optimal black-box protocol compilers. In Orr Dunkelman and Stefan Dziembowski, editors, *EUROCRYPT 2022, Part I*, volume 13275 of *LNCS*, pages 210–240. Springer, Heidelberg, May / June 2022.

IKSS22b. Yuval Ishai, Dakshita Khurana, Amit Sahai, and Akshayaram Srinivasan. Round-Optimal Black-Box Secure Computation from Two-Round Malicious Ot. In *TCC*, 2022.

IPS08. Yuval Ishai, Manoj Prabhakaran, and Amit Sahai. Founding cryptography on oblivious transfer - efficiently. In David Wagner, editor, *CRYPTO 2008*, volume 5157 of *LNCS*, pages 572–591. Springer, Heidelberg, August 2008.

MR17. Payman Mohassel and Mike Rosulek. Non-interactive secure 2PC in the offline/online and batch settings. In Jean-Sébastien Coron and Jesper Buus Nielsen, editors, *EUROCRYPT 2017, Part III*, volume 10212 of *LNCS*, pages 425–455. Springer, Heidelberg, April / May 2017.

Pai99. Pascal Paillier. Public-key cryptosystems based on composite degree residuosity classes. In Jacques Stern, editor, *EUROCRYPT'99*, volume 1592 of *LNCS*, pages 223–238. Springer, Heidelberg, May 1999.

Pas03. Rafael Pass. On deniability in the common reference string and random oracle model. In Dan Boneh, editor, *CRYPTO 2003*, volume 2729 of *LNCS*, pages 316–337. Springer, Heidelberg, August 2003.

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