Black-Box Reusable NISC with Random Oracles

Yuval Ishai[*] Dakshita Khurana[†] Amit Sahai[‡] Akshayaram Srinivasan[§]

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].

[*]Technion.
[†]UIUC.
[‡]UCLA.
[§]Tata Institute of Fundamental Research.

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 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[1] [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 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 programs to circuits, this step requires *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?

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

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. Indeed, except for the fact that we have to settle for computational security against a malicious sender, our protocol achieves security while being able to make black-box 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 black-box use of cryptography. In fact, we show the following more general result.

Theorem 1.1 (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'.*[2]

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.[3] This extension makes 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 in the random oracle model by making 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 updatable state between sessions.

Theorem 1.2 (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

[2] We remark that given the description of f, f' can be obtained by a polynomial time deterministic procedure.

[3] 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.

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 then 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 is allowed to learn the output computed by the honest receiver.[4] 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 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[5] 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 (over a super-polynomially large finite field) can only be guessed with negligible probability. This allows

[4] 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.

[5] 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.

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[6] 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:

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 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 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[7], 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

[6] 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. In each session, the sender could learn partial information about the receiver's output (for instance, whether the receiver aborted or not). 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.

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

hence do not give rise to input-dependent abort. Such a mechanism was impossible to achieve in the OT-hybrid model, but now we can at least hope that this might be possible.

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 watchlists [IPS08]. Roughly, the watchlist mechanism is 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 a malicious sender can mount a similar selective failure attack (as in the OT-hybrid model) to learn encoding of the random subset of watched executions 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 the 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.[8] 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 key security property that a reusable outer protocol needs to satisfy.

[8]To ensure that the corrupt parties are using uniformly sampled randomness as their random tape, the work of Ishai et al. [IKSS22a] sampled this string as the output of the random oracle. For the purpose of this overview, we ignore this technical detail.

Key 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 these messages are 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.[9] 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[10] 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.[11] We note that this is protocol is already secure in the reusable setting. This is because the first round 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

[9] Again, in the formal definition, we only allow the sender to learn partial information about this output but we ignore this in this overview.

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

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

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 4.

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 $k/m = O(1)$ (where k is the number of servers in the watchlist). 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 crucially 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.[12] 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 as one can mount resetting attacks.

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

[12] Again, the formal definition allows the adversary to only learn partial information about the outputs.

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 Appendix B, 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.

3 Standard Cryptographic Definitions

Let λ denote the cryptographic security parameter. We assume that all cryptographic algorithms implicitly take 1^λ as input. A function $\mu(\cdot) : \mathbb{N} \to \mathbb{R}^+$ is said to be negligible if for any polynomial $\mathsf{poly}(\cdot)$, there exists λ_0 such that for all $\lambda > \lambda_0$, we have $\mu(\lambda) < \frac{1}{\mathsf{poly}(\lambda)}$. We will use $\mathsf{negl}(\cdot)$ to denote an unspecified negligible function and $\mathsf{poly}(\cdot)$ to denote an unspecified polynomial function.

We say that two distribution ensembles $\{X_\lambda\}_{\lambda\in\mathbb{N}}$ and $\{Y_\lambda\}_{\lambda\in\mathbb{N}}$ are computationally indistinguishable if for every non-uniform PPT distinguisher D there exists a negligible function $\mathsf{negl}(\cdot)$ such that $|\Pr[D(1^\lambda, X_\lambda) = 1]| - \Pr[D(1^\lambda, Y_\lambda) = 1]| \le \mathsf{negl}(\lambda)$. If the above guarantee holds for even unbounded distinguishers D, we say that the two ensembles are statistically indistinguishable. We use $\approx_c$ to denote computational indistinguishability and $\approx_s$ to denote statistical indistinguishability.

3.1 Extractable Commitments in ROM

In our constructions, we make use of non-interactive, straight-line extractable commitments $(\mathsf{Com}, \mathsf{Open})$ in the random oracle model. Namely, the commitments are computationally hiding, statistically binding, and straight-line extractable by observing the queries that the adversary makes to the random oracle. Such commitments were constructed in [Pas03].

3.2 Pairwise Verifiable Secret Sharing

We now recall the definition of pairwise verifiable secret sharing from [IKSS22a]. This subsection is taken verbatim from [IKSS22a].

Consider a linear t-out-of-m threshold secret sharing scheme where the secrets are over a finite field $\mathbb{F}$ and the shares are over another finite field $\mathbb{F}'$. We use $+$ and $\cdot$ to denote the addition and multiplication operations over both the fields.

Definition 3.1 (Pairwise Verifiable Predicate)**.** *A predicate P is a pairwise verifiable predicate if it takes a threshold t, two indices $j, k \in [m]$ and the purported j-th and k-th shares x_j and x_k and outputs 1/0. Further, if $P(t, j, k, (x_j, x_k)) = 1$, $P(t, j, k, (x'_j, x'_k)) = 1$ and $P(k.t, j, k, (x''_j, x''_k)) = 1$ (where $k \in \mathbb{N}$), then $P(t, j, k, (x_j + x'_j, x_k + x'_k)) = 1$ and $P((k+1)t, j, k, (x_j \cdot x''_j, x_k \cdot x''_k)) = 1$.*

In the main body, we also extend the definition of the pairwise verifiable predicate P to take in a vector of pair of shares and apply the above pairwise check for each pair.

Definition 3.2 (Pairwise Verifiable and Error Correctable Secret Sharing)**.** *A t-out-of-m threshold linear secret sharing scheme $(\mathsf{Share}_{(t,m)}, \mathsf{Rec}_{(t,m)})$ is said to be k-multiplicative and ℓ-error-correctable w.r.t. pairwise predicate P if:*

1. *k-**Multiplicative**: Given m shares of elements $x_1, \ldots, x_k$ arranged as a matrix M of k rows and m columns, the row vector obtained by computing the product of each column of M is a kt-out-of-m secret sharing of $x_1 \cdot x_2 \ldots \cdot x_k$.*

2. ***Pairwise Verifiable Error Correction**: Let T be a subset of $[m]$ of size at most ℓ. Let $(x_1, \ldots, x_m)$ be arbitrary elements such that for any threshold $t' \le kt$ and for any $j, k \in [m] \setminus T$, $P(t', j, k, x_j, x_k) = 1$. Then, for any $\{\overline{x}_i\}_{i\in T}$, $\mathsf{Rec}_{(t',m)}(\{x_i\}_{i\in T}, \{x_i\}_{i\notin T}) = \mathsf{Rec}_{(t',m)}(\{\overline{x}_i\}_{i\in T}, \{x_i\}_{i\notin T}) = x$. Furthermore, there exists an efficient procedure* $\mathsf{Extrapolate}$ *that on input t', $\{x_i\}_{i\notin T}$ outputs the unique $\{x'_i\}_{i\in T}$ such that $(\{x_i\}_{i\notin T}, \{x'_i\}_{i\in T}) \in \mathsf{supp}(\mathsf{Share}_{(t',m)}(x))$.*

The above definition of pairwise verifiable secret sharing is the same as the one given in [IKP10] except that we additionally need error correction property as well. We note that bivariate Shamir secret sharing is a t-out-of-m secret sharing scheme that is k-multiplicative and ℓ-error correctable as long as $m \ge kt + 2\ell + 1$. The pairwise predicate corresponds to equality checking of polynomial evaluations.

3.3 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 Appendix A.

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 (see Appendix A) 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 3.3 (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* $\mathcal{Z}$ *and for all non-uniform PPT distinguishers* D*, we have:*

$$\left|\Pr[\mathsf{Real}(1^\lambda) = 1] - \Pr[\mathsf{Ideal}(1^\lambda) = 1]\right| \leq \mathsf{negl}(\lambda)$$

where Real *and* Ideal *experiments are defined below.*

- ***Real Execution:*** Real. *The environment* $\mathcal{Z}$ *on input* 1^λ *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 D *on the output of the honest party in each session and the output of* $\mathcal{A}$ *at the end of all sessions.*

[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.

- ***Ideal Execution:*** Ideal. *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* $\mathcal{Z}$ *on input* 1^λ *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* $\mathsf{Sim}_{\Pi,S}$ *and* $\mathsf{Sim}_{\Pi,S}$ *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 D *on the output of the honest party in each session and the output of* $\mathsf{Sim}_{\Pi,S}$ *at the end of all sessions.*

3.4 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 3.4 (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* $\mathcal{Z}$ *and non-uniform PPT distinguishers* D*, we have:*

$$\left|\Pr[\mathsf{Real}(1^\lambda) = 1] - \Pr[\mathsf{Ideal}(1^\lambda) = 1]\right| \leq \mathsf{negl}(\lambda)$$

where Real *and* Ideal *experiments are defined below.*

- ***Real Execution:*** Real. *For each $b \in \{0,1\}$, the environment $\mathcal{Z}$ on input 1^λ delivers the private offline input x_b^{off} to P_b and auxiliary input 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 D on the output of honest P_β in each session and the output of $\mathcal{A}$ at the end of all sessions.

- ***Ideal Execution:*** Ideal. *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 $\mathcal{Z}$ on input 1^λ delivers the private offline input x_b^{off} to P_b and auxiliary input 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 D on 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. This second round message is sent to the receiver. 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.

Organization. This section is organized as follows. In Section 4.1, we give the syntax and the security properties of a reusable verifiable client-server protocol. In Section 4.2, we give a construction of such a protocol for computing branching programs. In Section 4.3, we show how to extend this construction to compute arbitrary circuits.

4.1 Definition

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

[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.

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 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 4.1 (Pairwise vs Global Predicate)**.** *We now define pairwise and global predicates.*

- *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 4.2 (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.*

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

- *If $k = S$, then the output of the predicate $P'([m]\backslash T, \{\mathsf{msg}_{S,\mathsf{rand},i}\}_{i\in[m]\backslash T}) = 1$ where $\{\mathsf{msg}_{S,\mathsf{rand},i}\}_{i\in[m]\backslash T}$ 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 4.3 (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 4.2) w.r.t. P and P' corrupting the sender client S and a subset T (where $|T| \leq t$) 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 4.2) 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 4.2) 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* $\mathcal{Z}$ *and non-uniform PPT distinguisher* D*, we have:*

$$\left|\Pr[\mathsf{Real}(1^\lambda) = 1] - \Pr[\mathsf{Ideal}(1^\lambda) = 1]\right| \leq \mathsf{negl}(\lambda)$$

where Real *and* Ideal *experiments are given below.*

- ***Real Execution:*** Real*.* $\mathcal{Z}$ *on input* 1^λ *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* 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 D *on the output of the receiver in each session and the output of* $\mathcal{A}$ *at the end of all the executions.*

- ***Ideal Execution:*** Ideal*. 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* $\mathcal{Z}$ *on input* 1^λ *delivers an input* x *to the receiver and auxiliary input* z *to* $\mathsf{Sim}_{\Pi,S}$*. 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* 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 D *on the output of the receiver in each session and the output of* $\mathsf{Sim}_{\Phi,S}$ *at the end of all executions.*

4.2 Protocol for Computing Branching Programs

In this section, we give a construction of a reusable verifiable client-server protocol for computing branching programs (more generally, the class of functions in $\mathcal{SREN}$). The construction is a simplification of the one given in [IKSS22a] (which only satisfies error correction but not reusability).

4.2.1 Construction

Building Block. The construction uses:

1. A t-out-of-m, 3-multiplicative, t-error-correctable secret sharing scheme $(\mathsf{Share}_{(t,m)}, \mathsf{Rec}_{(t,m)})$ w.r.t. pairwise predicate P (see Definition 3.2). Let the shares be elements of a finite field $\mathbb{F}$ where the size of the field is $\mathsf{poly}(m)$.

Notation. Let f be the functionality that is computed using a branching program. Let x be the private input of the receiver and let y be the private input of the sender. Let $\widehat{f}$ be the randomized encoding [IK00, AIK04] of f that takes (x, y) and a random tape r and outputs $\widehat{f}((x, y); r)$. [IK00, AIK04] showed that $\widehat{f}$ can be expressed as a sequence of degree-3 polynomials $p_1, \ldots, p_\ell$ in (x, y, r).[16]

Description of the Protocol. We give the formal description of the protocol in Figure 1. The protocol is same as a two-round client-server, semi-honest secure protocol for computing degree-3 polynomials except that we use the special secret sharing scheme that is given by $(\mathsf{Share}_{(t,m)}, \mathsf{Rec}_{(t,m)})$.

4.2.2 Proof of Security

The pairwise consistency predicate P is the same as that of the underlying secret sharing scheme. Let P' be the global predicate that on input $(H, \{\mathsf{msg}_{S,\mathsf{rand},i}\}_{i \in H})$ does the following:

1. It parses $\mathsf{msg}_{S,\mathsf{rand},i}$ as $(r_i, \{z_{j,i}\}_{j \in [\ell]})$.

2. It checks if for each $k, k' \in H$, $P(t, k, k', r_k, r_{k'}) = 1$. It also checks if for each $j \in [\ell]$, if $P'(3t, k, k', z_{j,k}, z_{j,k'}) = 1$.

3. It checks if $\mathsf{Rec}_{(3t,m)}(\{z_{j,i}\}_{i \in H}, \{\perp\}_{i \notin H}) = 0$ for each $j \in [\ell]$.

Error Correction. Let $\mathcal{A}$ be any verifiable adversary corrupting a subset T of the servers of size at most t. This implies that each of the input shares sent by $\mathcal{A}$ to every pair of honest servers pass the pairwise verifiability check. Since $\gamma_{i,j}$ for each $i \in [m]$ and $j \in [\ell]$ is computed as a degree-3 polynomial, it follows from Definition 3.1 that for each pair of honest servers u, v, $\gamma_{u,j}$ and $\gamma_{v,j}$ for every $j \in [\ell]$ pass the pairwise verifiability check. Thus, the fact that the output of Dec depends only on the messages sent by the honest servers follows directly from the pairwise verifiable error correction property of the underlying secret sharing scheme. To prove the furthermore part, notice that if the adversary is verifiable then in each of the sender messages, $\{z_{j,i}\}_{j \in [\ell], i \in H}$ reconstruct to 0. Hence, for each sender message, $\gamma_{i,j}$ correspond to a $3t$-out-of-m secret sharing of the outputs of $p_1, \ldots, p_m$ computed on the same x, y but with potentially a different r. It now follows from the perfect correctness of the decoding algorithm of the randomized encoding that for each one of the sender messages, the output of Dec remains the same.

[16] We note that the randomized encodings take inputs in $\{0, 1\}$. As noted in [IKSS22a], to convert them into functions that takes elements from a finite field $\mathbb{F}$ (of size $p = \mathsf{poly}(m)$), we take each field element a and compute $a^{p-1} \mod p$. This gives a 0/1 value and it can be computed by a branching program of length polynomial in p.

- $\mathsf{Share}_R(x)$: On input the receiver's private input x, it does the following:
  1. It generates $(x_1, \ldots, x_m) \leftarrow \mathsf{Share}_{(t,m)}(x)$.
  2. It sets $\mathsf{msg}_{R,\mathsf{inp},i} = x_i$ for each $i \in [m]$.
- $\mathsf{ShareInp}_S(y)$: On input the sender's private input y, it does the following:
  1. It generates $(y_1, \ldots, y_m) \leftarrow \mathsf{Share}_{(t,m)}(y)$.
  2. It sets $\mathsf{msg}_{S,\mathsf{inp},i} = y_i$.
- $\mathsf{ShareRand}(\cdot)$:
  1. It samples a uniform random tape r for computing the randomized encoding $\widehat{f}$.
  2. It generates $(r_1, \ldots, r_m) \leftarrow \mathsf{Share}_{(t,m)}(r)$.
  3. It generates $(z_{j,1}, \ldots, z_{j,m}) \leftarrow \mathsf{Share}_{(3t,m)}(0)$ for each $j \in [\ell]$.
  4. It sets $\mathsf{msg}_{S,\mathsf{rand},i} = (r_i, \{z_{j,i}\}_{j\in[\ell]})$.
- $\mathsf{Eval}(i, \mathsf{msg}_{R,\mathsf{inp},i}, (\mathsf{msg}_{S,\mathsf{inp},i}, \mathsf{msg}_{S,\mathsf{rand},i}))$: It does the following:
  1. It parses $\mathsf{msg}_{R,\mathsf{inp},i}$ as x_i, $\mathsf{msg}_{S,\mathsf{inp},i}$ as y_i, and $\mathsf{msg}_{S,\mathsf{rand},i}$ as $(r_i, \{z_{j,i}\}_{j\in[\ell]})$.
  2. For each $j \in [\ell]$, it computes $\gamma_{i,j} = p_j(x_i, y_i, r_i) + z_{j,i}$.
  3. It sets $\mathsf{msg}_{2,i} = \{\gamma_{i,j}\}_{j\in[\ell]}$.
- $\mathsf{Dec}(\{\mathsf{msg}_{2,i}\}_{i\in[m]})$: It does the following:
  1. It parses $\mathsf{msg}_{2,i}$ as $\{\gamma_{i,j}\}_{j\in[\ell]}$ for each $i \in [m]$.
  2. For each $j \in [\ell]$, it computes $\alpha_j \leftarrow \mathsf{Rec}_{(3t,m)}(\gamma_{1,j}, \ldots, \gamma_{m,j})$.
  3. It runs the decoding algorithm of $\widehat{f}$ on $(\alpha_1, \ldots, \alpha_\ell)$ and outputs the result of the decoding.

Figure 1: Reusable Verifiable Client-Server Protocol for Computing Branching Programs

Security against Verifiable Receivers. Let $\mathcal{A}$ be a verifiable adversary that corrupts the receiver client and (adaptively) corrupts a set of T servers for $|T| \leq t$. We now show that the protocol described in Figure 1 satisfies security against verifiable receivers (see Definition 4.3). We start with the description of $\mathsf{Sim}_{\Phi,R}$.

Description of $\mathsf{Sim}_{\Phi,R}$. $\mathsf{Sim}_{\Phi,R}$ uses a dummy private input for the honest sender client and a dummy random tape r and generates the first round messages sent to the corrupted servers and sends them to $\mathcal{A}$. On input the first round message sent by $\mathcal{A}$ to the honest servers on behalf of the corrupted receiver, $\mathsf{Sim}_{\Phi,R}$ reconstructs x using $\mathsf{Rec}_{(t,m)}$ as these messages are guaranteed to be pairwise verifiable. It sends x to the ideal functionality and receives the output $f(x, y)$. It uses this output to run the simulator for the randomized encoding and obtains the simulated outputs of $p_1, \ldots, p_\ell$. It runs the $\mathsf{Extrapolate}$ algorithm to compute the purported shares that a verifiable receiver client would have sent to the corrupted servers using the shares received by the honest servers. Using these input shares and the shares sent on behalf of the honest sender client, it computes $\gamma_{i,j}$ for each $i \in T$ and $j \in [\ell]$ using the honest evaluation algorithm. Conditioned on fixing these values of $\gamma_{i,j}$, it generates a uniform $3t$-out-of-m secret sharing of the output of p_j for each $j \in [\ell]$ and sends the shares of the honest servers to the adversary as the second round messages.

Proof of Indistinguishability. We show that the real execution and the ideal execution are indistinguishable using an hybrid argument. This argument is almost identical to the one given in [IKSS22a] and many parts are taken verbatim from [IKSS22a].

- $\underline{\mathsf{Hyb}_0}$: This corresponds to the output of the real execution.
- $\underline{\mathsf{Hyb}_1}$: In this hybrid, we do the following:
  - Based on the shares sent to the honest servers, we extract x.
  - We compute the output $p_1, \ldots, p_\ell$ using the honest sender input.
  - We compute the purported shares that the verifiable receiver clients sent to the corrupted servers based on the shares received by the honest servers (using the $\mathsf{Extrapolate}$ algorithm). Using these input shares, we compute $\gamma_{i,j}$ for each $i \in T$ and $j \in [\ell]$ using the honest evaluation algorithm.
  - Conditioned on the fixing the above computed values, we sample the second round message from the honest servers as fresh $3t$-out-of-m secret sharing of the output of $p_1, \ldots, p_\ell$.

  This hybrid is identical to the previous hybrid since the adversary $\mathcal{A}$ is verifiable and the honest sender client sends a $3t$-out-of-m secret sharing of 0 for each $j \in [\ell]$.
- $\underline{\mathsf{Hyb}_2 :}$ In this hybrid, we replace the first round message sent by honest sender client to the corrupted servers to be shares of default value. This hybrid is identically distributed to the previous one from the perfect privacy of the underlying secret sharing scheme.
- $\underline{\mathsf{Hyb}_3}$: In this hybrid, we compute the output $f(x, y)$ and use this as input to the simulator for randomized encoding simulator to compute the outputs of $p_1, \ldots, p_\ell$. It follows from the security of randomized encoding that this hybrid is statistically close to Hyb_2. This hybrid is identical to the output of the ideal execution.

Reusable Security against Verifiable Senders. Let $\mathcal{A}$ be a verifiable adversary that corrupts the sender client and the set of servers as described in Definition 4.3. We now show that the protocol described in Figure 1 satisfies reusable security against verifiable senders (given in Definition 4.3). We start with the description of $\mathsf{Sim}_{\Phi,S}$.

Description of $\mathsf{Sim}_{\Phi,S}$. $\mathcal{A}$ gives the description of the set T_1 and $\mathsf{Sim}_{\Phi,S}$ uses a dummy private input for the honest receiver client and generates the messages corresponding to T_1 and sends them to $\mathcal{A}$. For each execution initiated by the adversary,

- $\mathsf{Sim}_{\Phi,S}$ receives the first round message sent by $\mathcal{A}$ to the honest servers $[m] \setminus (T \cup T_1)$.
- $\mathsf{Sim}_{\Phi,S}$ reconstructs (y, r) using $\mathsf{Rec}_{(t,m)}$ as these messages are guaranteed to be pairwise verifiable.
- $\mathsf{Sim}_{\Phi,S}$ sends y to the ideal functionality and forwards whatever it receives from the environment to $\mathcal{A}$.

Proof of Indistinguishability. We show that the real execution and the ideal execution are statistically close using a hybrid argument. This is again similar to the argument given in [IKSS22a] and many parts of the proof given below are taken verbatim from [IKSS22a].

- $\underline{\mathsf{Hyb}_0}$: This corresponds to the output of the real execution.
- $\underline{\mathsf{Hyb}_1}$: In this hybrid, we do the following:
  - In each execution, based on the shares sent to the honest servers, we extract (y, r) using $\mathsf{Rec}_{(t,m)}$.

  This hybrid is identical to the previous hybrid since the adversary $\mathcal{A}$ is verifiable and we are not making any changes to the distribution of the messages sent by the honest party.
- $\underline{\mathsf{Hyb}_2}$: In this hybrid, for every execution, we replace the output shares from the corrupted servers in $T \cup T_1$ to be some arbitrary values. We run the Dec algorithm on these replaced shares and compute the output of the honest receiver. It follows from the error correction property and the fact that $\mathcal{A}$ is verifiable that Hyb_1 and Hyb_2 are identical.
- $\underline{\mathsf{Hyb}_3}$: In this hybrid, we replace the first round message sent by honest receiver client to the corrupted servers in T_1 to be shares of default value. This hybrid is identically distributed to the previous one from the perfect privacy of the underlying secret sharing scheme as $|T_1| \leq t$.
- $\underline{\mathsf{Hyb}_4}$: In this hybrid, instead of computing the receiver's output using the decoding procedure of the randomized encoding, we output $f(x, y)$. Note that since $\mathcal{A}$ is verifiable, $\{z_{j,i}\}_{j \in [\ell], i \notin (T \cup T_1)}$ is a secret sharing of 0. Hence, this hybrid is identically distributed to the previous hybrid from the perfect correctness of the decoding procedure of randomized encoding. Note that Hyb_4 is identically distributed to the ideal execution.

4.3 Protocol for Computing Circuits

In this subsection, we give a construction of a reusable verifiable client-server protocol for computing arbitrary circuits by making black-box use of a PRG.

4.3.1 Construction

Building Block. The construction uses:

1. A t-out-of-m, 3-multiplicative, t-error-correctable secret sharing scheme $(\mathsf{Share}_{(t,m)}, \mathsf{Rec}_{(t,m)})$ w.r.t. pairwise predicate P (see Definition 3.2). Let the shares be elements of a finite field $\mathbb{F}$ where the size of the field is $\mathsf{poly}(m)$.
2. A pseudorandom function $\mathsf{PRF}_k : \{0,1\}^\lambda \to \{0,1\}^\lambda$ where the secret key k is a uniform bit string of length λ.

Notation. Let f be the functionality that is computed using a circuit. Let x be the private input of the receiver and let y be the private input of the sender. Let $\widehat{f}$ be the garbled circuit of f that takes (x, y) and a random tape r and outputs $\widehat{f}((x, y); r)$. We note that $\widehat{f}$ uses a part of the random tape r as PRF keys and evaluates the PRF using these keys on certain input values (which depends only on the description of the function f). It then uses these PRF evaluations to compute the garbling $\widehat{f}$. We note that the garbled circuit construction is such that given these PRF evaluations on the appropriate inputs (which we will collectively denote by s), $\widehat{f}$ can be expressed as a sequence of degree-3 polynomials $p_1, \ldots, p_\ell$ in (x, y, r, s).

Description of the Protocol. We give the formal description of the reusable verifiable client-server protocol for computing circuits in Figure 2. This protocol is nearly identical to the one given in Figure 1 except for the following changes. In $\mathsf{ShareRand}$ protocol, we pre-compute the PRF evaluations (collectively denoted as s) used in generation of the garbled circuit and share these PRF evaluations using $\mathsf{Share}_{(t,m)}$. The Eval algorithm will use the shares of (x_i, y_i, r_i, s_i) to compute a $3t$-out-of-m sharing of the outputs of $p_1, \ldots, p_\ell$.

- $\mathsf{Share}_R(x)$: On input the receiver's private input x, it does the following:
  1. It generates $(x_1, \ldots, x_m) \leftarrow \mathsf{Share}_{(t,m)}(x)$.
  2. It sets $\mathsf{msg}_{R,\mathsf{inp},i} = x_i$ for each $i \in [m]$.
- $\mathsf{ShareInp}_S(y)$: On input the sender's private input y, it does the following:
  1. It generates $(y_1, \ldots, y_m) \leftarrow \mathsf{Share}_{(t,m)}(y)$.
  2. It sets $\mathsf{msg}_{S,\mathsf{inp},i} = y_i$.
- $\mathsf{ShareRand}(\cdot)$: It does the following:
  1. It samples a uniform random tape r for computing the randomized encoding $\widehat{f}$.
  2. It computes the PRF evaluations using the keys derived from r on inputs which depend on the description of the function f. Let us use s to denote the collection of all such PRF evaluations.
  3. It generates $(r_1, \ldots, r_m) \leftarrow \mathsf{Share}_{(t,m)}(r)$ and $(s_1, \ldots, s_m) \leftarrow \mathsf{Share}_{(t,m)}(s)$.
  4. It generates $(z_{j,1}, \ldots, z_{j,m}) \leftarrow \mathsf{Share}_{(3t,m)}(0)$ for each $j \in [\ell]$.
  5. It sets $\mathsf{msg}_{S,\mathsf{rand},i} = (r_i, s_i, \{z_{j,i}\}_{j \in [\ell]})$.
- $\mathsf{Eval}(i, \mathsf{msg}_{R,\mathsf{inp},i}, \mathsf{msg}_{S,\mathsf{inp},i}, \mathsf{msg}_{S,\mathsf{rand},i})$: It does the following:
  1. It parses $\mathsf{msg}_{R,\mathsf{inp},i}$ as x_i, $\mathsf{msg}_{S,\mathsf{inp},i}$ as y_i, and $\mathsf{msg}_{S,\mathsf{rand},i}$ as $(r_i, s_i, \{z_{j,i}\}_{j \in [\ell]})$.
  2. For each $j \in [\ell]$, it computes $\gamma_{i,j} = p_j(x_i, y_i, r_i, s_i) + z_{j,i}$.
  3. It sets $\mathsf{msg}_{2,i} = \{\gamma_{i,j}\}_{j \in [\ell]}$.
- $\mathsf{Dec}(\{\mathsf{msg}_{2,i}\}_{i \in [m]}, \{k_j\}_{j \in [\ell]})$: It does the following:
  1. It parses $\mathsf{msg}_{2,i} = \{\gamma_{i,j}\}_{j \in [\ell]}$ for each $i \in [m]$.
  2. For each $j \in [\ell]$, it computes $\alpha_j \leftarrow \mathsf{Rec}_{(3t,m)}(\gamma_{1,j}, \ldots, \gamma_{m,j})$.
  3. It runs the garbled circuit evaluation algorithm on $(\alpha_1, \ldots, \alpha_\ell)$ and outputs the result of the evaluation.

Figure 2: Reusable Verifiable Client-Server Protocol for Computing Circuits

Remark 4.4. *We note that the* Eval *algorithm described in Figure 2 is information-theoretic and does not use any cryptographic operations. This property will be crucially used in the construction of a black-box reusable (two-sided) NISC.*

4.3.2 Proof of Security

The pairwise consistency predicate P is the same as that of the underlying secret sharing scheme. Let P' be the global predicate that on input $(H, \{\mathsf{msg}_{S,\mathsf{rand},i}\}_{i\in H})$ does the following:

1. It parses $\mathsf{msg}_{S,\mathsf{rand},i}$ as $(r_i, s_i, \{z_{j,i}\}_{j\in[\ell]})$.
2. It checks if for each $k, k' \in H$, $P(t, k, k', r_k, r_{k'}) = 1$ and $P(t, k, k', s_k, s_{k'}) = 1$. It also checks if for each $j \in [\ell]$, if $P'(3t, k, k', z_{j,k}, z_{j,k'}) = 1$.
3. It checks if $\mathsf{Rec}_{(3t,m)}(\{z_{j,i}\}_{i\in H}, \{\perp\}_{i\notin H}) = 0$ for each $j \in [\ell]$.
4. It computes $r = \mathsf{Rec}_{(t,m)}(\{r_i\}_{i\in H}, , \{\perp\}_{i\notin H})$ and $s = \mathsf{Rec}_{(t,m)}(\{s_i\}_{i\in H}, , \{\perp\}_{i\notin H})$ and checks if s is correctly computed from r using the PRF evaluations on the appropriate inputs using the correct keys derived from r.

Error Correction. The error correction property follows via an identical argument to the previous case. Let $\mathcal{A}$ be any verifiable adversary corrupting a subset T of the servers of size at most t. This implies that each of the input shares sent by $\mathcal{A}$ to every pair of honest servers pass the pairwise verifiability check. Since $\gamma_{i,j}$ for each $i \in [m]$ and $j \in [\ell]$ is computed as a degree-3 polynomial, it follows from Definition 3.1 that for each pair of honest servers u, v, $\gamma_{u,j}$ and $\gamma_{v,j}$ for every $j \in [\ell]$ pass the pairwise verifiability check. Hence, it follows from the pairwise verifiable error correction property of secret sharing scheme that for any choice of corrupted server messages, the output of Dec remains the same. Also, since the messages pass the global consistency predicate P', we infer that s is correctly derived from r by computing the PRF in appropriate inputs and each $(z_{j,1}, \ldots, z_{j,m})$ is a $3t$-out-of-m secret sharing of 0. Hence, $\gamma_{i,j}$ correspond to a $3t$-out-of-m secret sharing of the outputs of $p_1, \ldots, p_m$ computed on the same x, y but with potentially a different r. It now follows from the perfect correctness of the garbled circuit evaluation that for each one of the sender messages, the output of Dec remains the same and this proves the furthermore part.

Security against Verifiable Receivers. Let $\mathcal{A}$ be a verifiable adversary that corrupts the receiver client and (adaptively) corrupts a set of T servers for $|T| \leq t$. The proof that the protocol in Figure 2 satisfies security against verifiable receivers (see Definition 4.3) follows identically to the case of branching programs. The only difference being that we rely on the computational security of the garbled circuits simulator in Hyb_3 rather than the statistical security in the case of branching programs.

Remark 4.5. *Consider a setting where in the interaction with a verifiable receiver, the honest sender runs* $\mathsf{ShareInp}(\cdot)$ *once on its private input y but generates k independent sharing of the randomness using* $\mathsf{ShareRand}$. *For each of these independent runs of* $\mathsf{ShareRand}$*, the servers run the* Eval *algorithm (on fixed input sharing of the sender) and fresh sharing generated by* $\mathsf{ShareRand}_S$ *and send the second round message to the receiver. The proof given above can be extended in a straightforward way to show that in this setting, the receiver learns no information beyond the output of the functionality.*

Reusable Security against Verifiable Senders. Let $\mathcal{A}$ be a verifiable adversary that corrupts the sender client and the set of servers as described in Definition 4.3. This proof is again identical to the one given in the case of branching programs except in the last hybrid (i.e., in Hyb_4). Here, we additionally use the fact that the adversarial sender messages satisfy the global predicate P' to ensure that the garbled circuit $\widehat{f}$ is correctly constructed. We can now use the perfect correctness of garbled circuit evaluation to show that Hyb_3 and Hyb_4 are identically distributed.

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 5.1. *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 5.2 (Reusable Security against Explainable Senders)**.** *This requirement is the same as the one given in Definition 3.3 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 correctness of the evaluation algorithm and indistinguishability-based security of the receiver's message against semi-malicious senders (which is implied by security against malicious senders).

Proposition 5.3. *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 4.3). 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 Section 4.3 ensures that Eval algorithm does not compute any cryptographic operations (see Remark 4.4)

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 5.3, 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 3. A pictorial representation of our construction is given in Figure 4.

5.2 Proof of Security

In this section, we show that the protocol given in Figure 3 is a reusable NISC protocol (see Definition 3.3). Specifically, in Section 5.2.1, we show that it satisfies standard security against malicious receivers and in Section 5.2.2, we show that it satisfies reusable security against corrupted senders.

5.2.1 Security against Malicious Receivers

Let $\mathcal{A}$ be a non-uniform PPT adversary that corrupts the receiver. For simplicity, we give our proofs in the standalone setting but all our proofs extend to the UC setting in a straightforward manner as our simulators and reductions are straight-line. We first give the description of the simulator Sim_R.

Description of Sim_R.

1. Sim_R answers all the random oracle queries of the adversary using uniformly chosen random values from the co-domain. It chooses a random $\mathsf{tag}_S \leftarrow \{0,1\}^\lambda$ and if the adversary makes any query to H_1 or H_2 that starts with tag_S before receiving the second-round message from Sim_R, it aborts.

2. It samples sets L_1 and L_2 as subsets of $[m]$ and $[n]$ respectively where each element is independently included in L_1 and L_2 with probability p_m and p_m respectively. If $|L_1| \geq 2p_m m$ or $|L_2| \geq 2p_n n$, then it aborts. It programs the random oracles H_1 and H_2 to produce the output L_1 and L_2 respectively when queried on the messages it generates on behalf of the honest sender.

- **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. It outputs $\mathsf{Majority}(\{\alpha_j\}_{j\in[n]\setminus L_2})$.

Figure 3: Construction of Reusable Black-Box NISC Protocol

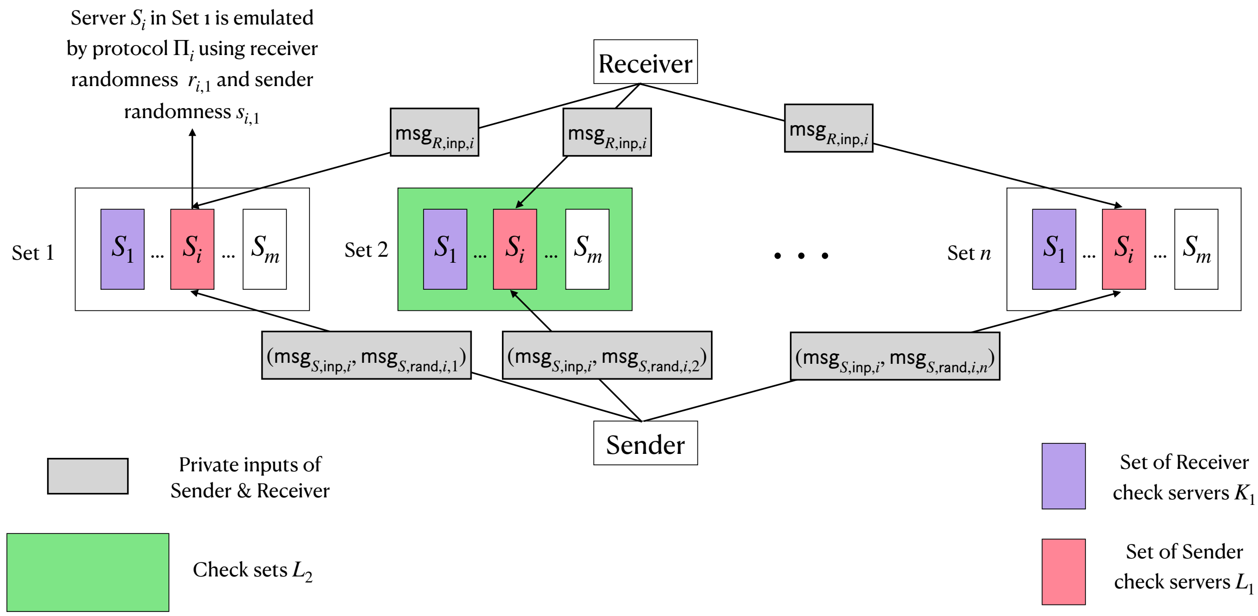

Figure 4: Pictorial Representation of the Protocol. Here, $\{\mathsf{msg}_{R,\mathsf{inp},i}\}_{i\in[m]}$ denotes the shares of the receiver's private input, $\{\mathsf{msg}_{S,\mathsf{inp},i}\}_{i\in[m]}$ denotes the shares of the sender's private input, and $\{\mathsf{msg}_{S,\mathsf{rand},i,j}\}_{i\in[m],j\in[n]}$ denotes the shares of the sender's randomness. 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 $(\mathsf{msg}_{S,\mathsf{rand},1,j}, \ldots, \mathsf{msg}_{S,\mathsf{rand},m,j})$.

3. **First Round Message from $\mathcal{A}$.** It receives the first round message from $\mathcal{A}$ and runs the extractor of the commitment scheme on $\{a_i\}_{i\in[m]}$ and $\{b_{i,j}\}_{i\in[m],j\in[n]}$ to obtain $\{\mathsf{msg}_{R,\mathsf{inp},i}\}_{i\in[m]}$ and $\{r_{i,j}\}_{i\in[m],j\in[n]}$ respectively.

4. **Check Phase.** It performs the same checks that an honest sender does and if any of these checks fail, it aborts. Else,

   (a) It initiates an empty set I_1. For each $i \in [m]$, it checks if for all $j \in [n]$, $\pi_{i,j,1} = \Pi_{i,1}(\mathsf{msg}_{R,\mathsf{inp},i}; r_{i,j})$. If there is some $j \in [n]$ for which this check fails, it adds i to I_1.

   (b) It then constructs an inconsistency graph G on m vertices and it adds an edge between (i, i') in G if $\mathsf{msg}_{R,\mathsf{inp},i}$ and $\mathsf{msg}_{R,\mathsf{inp},i'}$ do not pass the pairwise consistency check P. It runs a 2-approximation algorithm for the minimum vertex cover for this graph G and calls this set as I_2.

   (c) If $|I_1| \geq \lambda$ or $|I_2| \geq \lambda$, it aborts.

5. **Simulated Second Round Message.** To generate the second round message on behalf of the honest sender, Sim_R does the following:

   (a) It starts running the simulator $\mathsf{Sim}_{\Phi,R}$ by corrupting the receiver client and the set of servers indexed by $L_1 \cup I_1 \cup I_2$. It requests for $|[n] \setminus L_2|$ independent shares of randomness generated using $\mathsf{ShareRand}$ (see Remark 4.5). $\mathsf{Sim}_{\Phi,R}$ provides $\{\mathsf{msg}_{S,\mathsf{inp},i}, \mathsf{msg}_{S,\mathsf{rand},i,j}\}_{i\in L_1\cup I_1\cup I_2, j\notin L_2}$.

(b) Sim_Π sends $\{\mathsf{msg}_{R,\mathsf{inp},i}\}_{i \notin L_1 \cup I_1 \cup I_2}$ as the first round messages sent from the corrupt receiver to the honest servers. $\mathsf{Sim}_{\Phi,R}$ queries the ideal functionality on the malicious receiver input x which we forward it to our trusted party. On receiving $f(x,y)$ from the functionality, we forward it to $\mathsf{Sim}_{\Phi,R}$. $\mathsf{Sim}_{\Phi,R}$ returns $\{\mathsf{msg}_{2,i,j}\}_{i \notin L_1 \cup I_1 \cup I_2, j \notin L_2}$ as the second round messages from the honest servers.

(c) For each $i \notin (L_1 \cup I_1 \cup I_2)$ and $j \notin L_2$, it generates $\pi_{i,j,2}$ using the simulator $\mathsf{Sim}_{\Pi_i,R}$ on input $\pi_{i,j,1}$ as the receiver message and $\mathsf{msg}_{2,i,j}$ as the output of the computation. It generates $\{\mathsf{com}_{i,j}\}_{i \notin (L_1 \cup I_1 \cup I_2), j \notin L_2}$ using these values. For each $i \notin (L_1 \cup I_1 \cup I_2)$ and $j \in L_2$, it generates $\mathsf{com}_{i,j}$ as a commitment to a dummy value.

(d) For each $i \in I_1 \cup I_2 \cup L_1$ and $j \in [n]$, it samples a uniform random tape $s_{i,j}$ and computes $\pi_{i,j,2} := \Pi_{i,2}(\pi_{i,j,1}, (\mathsf{msg}_{S,\mathsf{inp},i}, \mathsf{msg}_{S,\mathsf{rand},i,j}); s_{i,j})$. It generates $\{\mathsf{com}_{i,j}\}_{i \in (L_1 \cup I_1 \cup I_2), j \in [n]}$ using these values.

(e) For each $i \notin L_1$ and each $j \in [n]$, it replaces the message inside c_i and $d_{i,j}$ with a dummy value.

(f) For each $i \notin L_1$ and each $j \in [n] \setminus L_2$, it replaces the message inside $e_{i,j}$ with a dummy value. For each $j \in [L_2]$, sample $\{\mathsf{msg}_{S,\mathsf{rand},i,j}\}_{i \in [m]}$ and use this to generate $\{e_{i,j}\}_{i \in [m]}$.

(g) It generates the rest of the second round messages as per the protocol description and sends it to the adversary.

6. Sim_R finally outputs whatever the adversary $\mathcal{A}$ outputs.

Proof of Indistinguishability. We show that the real world and the ideal world executions are computationally indistinguishable via a hybrid argument.

- $\underline{\mathsf{Hyb}_0}$: This corresponds to the real world execution of the protocol.

- $\underline{\mathsf{Hyb}_1}$: In this hybrid, we answer all the random oracle queries to H_1 and H_2 made by the adversary using uniformly chosen random values from the co-domain. Furthermore, we choose a random $\mathsf{tag}_S \leftarrow \{0,1\}^\lambda$ and if the adversary makes any query to H_1 or H_2 that starts with tag_S before the honest sender sends its message, we abort.

  Since tag_S is a uniformly chosen string of length λ, the probability that we abort is at most $q \cdot 2^{-\lambda}$ where q is the total number of queries made by the adversary. Therefore, Hyb_1 is statistically close to Hyb_0.

- $\underline{\mathsf{Hyb}_2}$: We make the following changes in this hybrid.

  1. On receiving the first round message from $\mathcal{A}$, for each $i \in [m]$ and $j \in [n]$, we run the extractor of the commitment scheme on a_i to extract $\mathsf{msg}_{R,\mathsf{inp},i,j}$ and on $b_{i,j}$ to extract $r_{i,j}$.
  2. We initiate empty sets I_1 and I_2.
  3. For each $i \in [m]$, we check if for all $j \in [n]$, $\pi_{i,j,1} = \Pi_{i,1}(\mathsf{msg}_{R,\mathsf{inp},i}; r_{i,j})$. If not, we add i to I_1.
  4. We construct an inconsistency graph G on m vertices where we add an edge between (i,i') in G if $\mathsf{msg}_{R,\mathsf{inp},i}$ and $\mathsf{msg}_{R,\mathsf{inp},i'}$ do not pass the pairwise consistency check P. We run a 2-approximation algorithm for the minimum vertex cover for this graph G and call this set as I_2.

5. If $|I_1| \geq \lambda$ or $|I_2| \geq \lambda$, we abort.

We prove in Lemma 5.4 that Hyb_2 and Hyb_1 are statistically close.

- $\underline{\mathsf{Hyb}_3}$: We make the following changes in this hybrid.
  1. We sample sets L_1 and L_2 as subsets of $[m]$ and $[n]$ respectively where each element is independently included in L_1 and L_2 with probability p_m and p_m respectively. We program the outputs of the random oracles H_1 and H_2 to output L_1 and L_2 when the honest sender makes its query.
  2. If $|L_1| \geq 2p_m m$ or $|L_2| \geq 2p_n n$, we abort.

  This hybrid is statistically close to Hyb_2 using standard Chernoff bounds.

- $\underline{\mathsf{Hyb}_4}$: We make the following changes in this hybrid.
  1. For each $j \in L_2$ and $i \in [m] \setminus (L_1 \cup I_1 \cup I_2)$, we change the message inside $\mathsf{com}_{i,j}$ to be a dummy value.
  2. For each $i \notin L_1$ and each $j \in [n]$, we replace the message inside c_i and $d_{i,j}$ with a dummy value.
  3. For each $i \notin L_1$ and each $j \in [n] \setminus L_2$, we replace the message inside $e_{i,j}$ with a dummy value.

  The computational indistinguishability between Hyb_3 and Hyb_4 directly reduces to the computational hiding property of Com.

- $\underline{\mathsf{Hyb}_5}$: We make the following changes in this hybrid.
  1. We compute the sets I_1 and I_2 as before.
  2. For each $i \notin (L_1 \cup I_1 \cup I_2)$ and $j \notin L_2$, we generate $\pi_{i,j,2}$ using the simulator $\mathsf{Sim}_{\Pi_i,R}$ on input $\pi_{i,j,1}$ and $\mathsf{msg}_{2,i,j}$ (which is computed using the honest sender shares). We generate commitments $\{\mathsf{com}_{i,j}\}_{i \notin (L_1 \cup I_1 \cup I_2), j \notin L_2}$ using the above generated values.

  In Lemma 5.5, we prove that Hyb_4 and Hyb_5 are computationally indistinguishable using the security of the NISC protocol against malicious receivers.

- $\underline{\mathsf{Hyb}_6}$: We make the following changes in this hybrid.
  1. We receive the first round message from the receiver and compute the sets I_1 and I_2 respectively.
  2. We run the simulator $\mathsf{Sim}_{\Phi,R}$ for the protocol Φ by corrupting the receiver client and the set of servers given by $L_1 \cup I_1 \cup I_2$. We request $|[n] \setminus L_2|$ number of instances of the random shares that are output by $\mathsf{ShareRand}_S$ for a fixed set of input shares of the sender (see Remark 4.5). $\mathsf{Sim}_{\Phi,R}$ returns $\{\mathsf{msg}_{S,\mathsf{inp},i}, \mathsf{msg}_{S,\mathsf{rand},i,j}\}_{i \in L_1 \cup I_1 \cup I_2, j \notin L_2}$.
  3. For each $j \in L_2$, we run independent instances of $\mathsf{ShareRand}_S$ to generate $\{\mathsf{msg}_{S,\mathsf{rand},i,j}\}_{i \in [m]}$.
  4. We return $\{\mathsf{msg}_{R,\mathsf{inp},i}\}_{i \notin (L_1 \cup I_1 \cup I_2)}$ as the first round message sent by the corrupt receiver to the honest senders. $\mathsf{Sim}_{\Phi,R}$ queries the ideal functionality on the malicious receiver input x which we forward it to our trusted party. On receiving $f(x, y)$ from the functionality, we forward it to $\mathsf{Sim}_{\Phi,R}$.

5. The simulator $\mathsf{Sim}_{\Phi,R}$ returns the second round messages $\{\mathsf{msg}_{2,i,j}\}_{i \notin (L_1 \cup I_1 \cup I_2), j \notin L_2}$ from the honest servers to the corrupt receiver. We give this as the output of the server emulations to $\mathsf{Sim}_{\Pi_i,R}$ to generate the second round messages $\pi_{i,j,2}$.

In Lemma 5.6, we show that Hyb_5 and Hyb_6 using the security of Φ against verifiable receivers. We note that Hyb_6 is identically distributed to the output of the ideal execution using Sim_R.

Lemma 5.4. $\mathsf{Hyb}_2 \approx_s \mathsf{Hyb}_1$.

Proof. We note that the only difference in Hyb_2 and Hyb_1 is that in Hyb_2 if $|I_1|$ or $|I_2|$ is greater than or equal to λ, we abort. We show that if this event happens, then we abort in Hyb_1 as well except with negligible probability.

- **Case-I: $|I_1| \geq \lambda$:** Consider any query that the adversary makes to the random oracle H_1 where $|I_1| \geq \lambda$. We show that the output K_1 sampled by the random oracle has the property that $K_1 \cap I_1 \neq \emptyset$ except with negligible probability. If this happens, then the honest sender client aborts in Hyb_2.We now argue that the probability that $|K_1 \cap I_1| = 0$ is $\exp(-\lambda)$.

$$\begin{aligned} \Pr[|K_1 \cap I_1| = 0] &= (1 - p_m)^{|I_1|} \\ &\leq (1 - p_m)^{\lambda} \\ &= (1 - \frac{\lambda}{2m})^{\lambda} \\ &\leq \exp(-\lambda) \end{aligned}$$

  as $m = O(\lambda)$.

- **Case-II: $|I_2| \geq \lambda$:** Consider any query that the adversary makes to the random oracle H_1 where $|I_2| \geq \lambda$. Since $|I_2|$ is the output of a 2-approximation algorithm for the minimum vertex cover, the size of the minimum vertex cover is at least $|I_2|/2 \geq \lambda/2$. By a standard connection between minimum vertex cover and maximum matching, we infer that there is a matching in G of size at least $\lambda/4$. If any edge in this matching is present in K_1, then the honest sender client in Hyb_2 will also abort. Let p be the probability that no edge of this matching is present in K_1.

$$\begin{aligned} p &\leq (1 - p_m^2)^{\lambda/4} \\ &= (1 - (\frac{\lambda}{2m})^2)^{\lambda/4} \\ &\leq \exp(-\lambda). \end{aligned}$$

  as $m = O(\lambda)$.

By a standard union bound, the probability that adversary makes any query to the random oracle oracle H_1 such that $|I_1|$ or $|I_2|$ is greater than or equal to λ but the output K_1 does not force an honest sender client to abort in Hyb_1 is at most $q \cdot \exp(-\lambda)$ where q is the total number of queries that the adversary makes. Since q is polynomial in λ, this proves that Hyb_2 and Hyb_1 are statistically close. $\square$

Lemma 5.5. *Assuming the security against malicious receivers of the NISC protocol Π_i for each $i \in [n]$, we have* $\mathsf{Hyb}_4 \approx_c \mathsf{Hyb}_5$.

Proof. Assume for the sake of contradiction that Hyb_4 and Hyb_5 are distinguishable with non-negligible advantage. We now give a reduction that breaks the security of one of $\{\Pi_i\}_{i\in[m]\setminus(L_1\cup I_1\cup I_2)}$.

Via a standard averaging argument, there exists two hybrids Hyb'_4 and Hyb'_5 and an $i \in [m] \setminus (L_1 \cup I_1 \cup I_2)$ and $j \notin L_2$ such that Hyb'_4 and Hyb'_5 are distinguishable with non-negligible advantage and these two hybrids only differ in how $\pi_{i,j,2}$ is generated.[17] Let us assume w.l.o.g. that in Hyb'_4, $\pi_{i,j,2}$ is generated using the honest sender algorithm whereas in Hyb'_5, it is generated using Sim_{Π_i}. We now give a reduction that contradicts the security of Π_i against malicious senders.

We start interacting with the challenger and provide $(\mathsf{msg}_{S,\mathsf{inp},i}, \mathsf{msg}_{S,\mathsf{rand},i,j})$ as the private sender input. On receiving the first round message from the adversarial receiver, we forward $\pi_{i,j,1}$ to the external challenger and receive $\pi_{i,j,2}$. We generate the rest of the second round messages in the protocol exactly as in Hyb'_4 and forward this to the adversary. We finally output whatever the adversary outputs.

We note that if $\pi_{i,j,2}$ is generated using the honest sender algorithm then the output of the above reduction is identically distributed to Hyb'_4. Else, if it was generated as the output of the simulator $\mathsf{Sim}_{\Pi_i,R}$, then the output of the reduction is identically distributed to Hyb'_5. Thus, if Hyb'_4 and Hyb'_5 are computationally distinguishable, then the above reduction breaks the security of Π_i against malicious receivers and this is a contradiction. □

Lemma 5.6. *Assuming the security of Φ against verifiable receiver, we have* $\mathsf{Hyb}_5 \approx_c \mathsf{Hyb}_6$.

Proof. Assume for the sake of contradiction that Hyb_5 and Hyb_6 are computationally distinguishable with non-negligible advantage. We will show that this contradicts the security of Φ against a verifiable receiver. To be a bit more precise, we will consider the modified security experiment as described in Remark 4.5 and show that if Hyb_5 and Hyb_6 are computationally distinguishable then this contradicts the security against verifiable receiver in the modified setting.

We start interacting with the challenger and provide y as the challenge sender input and request $|[n] \setminus L_2|$ number of independent shares generated using $\mathsf{ShareRand}$ (see Remark 4.5). We receive the first round message from the adversarial receiver and corrupt the set of servers indexed by $L_1 \cup I_1 \cup I_2$. We receive the sender messages $\{\mathsf{msg}_{S,\mathsf{inp},i}, \mathsf{msg}_{S,\mathsf{rand},i,j}\}_{i\in L_1\cup I_1\cup I_2, j\in[n]\setminus L_2}$. We send $\{\mathsf{msg}_{R,\mathsf{inp},i}\}_{i\notin L_1\cup I_1\cup I_2}$ extracted from the commitments as the first round message sent by the adversarial receiver to the honest servers. The challenger replies with $\{\mathsf{msg}_{2,i,j}\}_{i\notin L_1\cup I_1\cup I_2, j\in[n]\setminus L_2}$. We use this to generate the second round messages $\pi_{i,j,2}$ by giving them as inputs to $\mathsf{Sim}_{\Pi_i,R}$. We generate the rest of the second round sender message as before and finally output whatever the adversary outputs.

We note that if the messages generated by the challenger were using the honest receiver and server algorithms then the output of the above reduction is identically distributed to Hyb_5. This follows from the definition of I_1 which ensures that for each $i \notin I_1$, the value $\mathsf{msg}_{R,\mathsf{inp},i}$ that is extracted by Sim_{Π_i} is the same as the one obtained from the extractable commitment. Else, if the messages are generated by the challenger using the simulated algorithm then the output of the reduction is identically distributed to Hyb_6. Further, the total number of corrupted servers $|L_1 \cup I_1 \cup I_2| \leq 3\lambda$ is less than the threshold t and the reduction emulates a verifiable adversary (which follows from the definition of I_2). Since we assumed that Hyb_5 and Hyb_6 are computationally distinguishable with non-negligible advantage, the above reduction breaks the security of the protocol Φ against verifiable receivers and this is a contradiction. □

[17] Recall that in the previous hybrid, we have stopped generating $\pi_{i,j,2}$ for each $i \in [m] \setminus (L_1 \cup I_1 \cup I_2)$ and $j \in L_2$.

5.2.2 Reusable Security against Malicious Senders

Let $\mathcal{A}$ be a non-uniform PPT adversary that corrupts the sender client. We first give the description of the simulator Sim_S.

Description of Sim_S.

1. In each session, Sim_S initializes $\mathcal{A}$ with the private input provided by $\mathcal{Z}$. Sim_S answers all the random oracle queries to H_1 and H_2 by the adversary using uniformly chosen random values from the co-domain. Furthermore, it chooses a random $\mathsf{tag}_R \leftarrow \{0,1\}^\lambda$ and if the adversary makes any query to H_1 or H_2 that starts with tag_R before it receives the first round message, it aborts.

2. It samples a set K_1 as a subset of $[m]$ where each element is independently included in K_1 with probability p_m. It programs the output of the random oracle H_1 to output K_1 when queried on the messages it generates on behalf of the honest receiver. If $|K_1| \geq 2p_m m$, it aborts.

3. **Simulated First Round Message from the Receiver.**
   (a) It starts running the simulator $\mathsf{Sim}_{\Phi,S}$ by corrupting the sender client and sets K_1 to be the initial set of server corruptions (which is given by T_1 in the definition). $\mathsf{Sim}_{\Phi,S}$ returns $\{\mathsf{msg}_{R,\mathsf{inp},i}\}_{i \in K_1}$ and it uses this to generate the messages $\pi_{i,j,1}$ for each $i \in K_1$ and $j \in [n]$ as described in the protocol.
   (b) For each $i \notin K_1$ and each $j \in [n]$, it replaces the messages inside the commitments a_i and $b_{i,j}$ to be dummy values and generates $\pi_{i,j,1}$ using $\mathsf{Sim}_{\Pi_i,S}$. It generates the rest of components in the first round message as per the protocol specification and sends this message to $\mathcal{A}$.

4. **Second Round Message from $\mathcal{A}$ and Output Computation.**
   (a) On receiving the second round message from the adversarial sender in each execution, it runs the extractor for Com on $\{\mathsf{Com}_{i,j}, c_i, d_{i,j}, e_{i,j}\}_{i \in [m], j \in [n]}$ to obtain $\{\pi_{i,j,2}, \mathsf{msg}_{S,\mathsf{inp},i}, s_{i,j}, \mathsf{msg}_{S,\mathsf{rand},i,j}\}_{i \in [m], j \in [n]}$.
   (b) **Check Phase.** It performs the same checks that an honest receiver performs and if any of these checks fail, it instructs the ideal functionality to deliver $\perp$ to the receiver. In addition to these checks, Sim performs the following checks:
      i. It initializes an empty set I_1. For each $i \in [m]$ and $j \in [n]$, it checks if $\pi_{i,j,2} = \Pi_{i,2}(\pi_{i,j,1}, (\mathsf{msg}_{S,\mathsf{inp},i}, \mathsf{msg}_{S,\mathsf{rand},i,j}); s_{i,j})$. If there is some $j \in [n]$ such that the above check does not pass, it adds i to I_1.
      ii. It constructs an inconsistency graph G on m vertices where it adds an edge between (i, i') in G if $\mathsf{msg}_{S,\mathsf{inp},i}$ and $\mathsf{msg}_{S,\mathsf{inp},i'}$ do not pass the pairwise consistency check P. It then runs a 2-approximation algorithm for the minimum vertex cover for this graph G and calls this set as I_2.
      iii. It initializes another empty set J_1. For each $j \in [n]$, if $(\mathsf{msg}_{S,\mathsf{rand},1,j}, \ldots, \mathsf{msg}_{S,\mathsf{rand},m,j})$ do not pass the global predicate check P', it adds j to J_1.

iv. If $|I_1| \geq \lambda$, or $|I_2| \geq \lambda$, or $|J_1| \geq \lambda$, it aborts and instructs the ideal functionality to deliver $\perp$ to the honest receiver.

v. It sends $I_1 \cup I_2$ as the second set of server corruptions to $\mathsf{Sim}_{\Phi,S}$ (which is given by T in the definition) and provides $\{\mathsf{msg}_{S,\mathsf{inp},i}, \mathsf{msg}_{S,\mathsf{rand},i,j}\}_{i \notin (K_1 \cup I_1 \cup I_2), j \notin J_1 \cup L_2}$ as the first round messages to the honest servers from the corrupted sender. $\mathsf{Sim}_{\Phi,S}$ extracts the adversarial sender input y and sends this input to the ideal functionality.[18] Sim_S intercepts this query and forwards this to its own ideal functionality along with the instruction to deliver this output to the honest receiver. Sim_S forwards whatever it receives from the environment to $\mathsf{Sim}_{\Phi,S}$ and $\mathcal{A}$.

Proof of Indistinguishability. We show that the real execution and the ideal world execution are computationally indistinguishable via a hybrid argument.

- $\underline{\mathsf{Hyb}_0}$: This corresponds to the real execution described in Definition 3.3.

- $\underline{\mathsf{Hyb}_1}$: In this hybrid, we answer all the random oracle queries to H_1 and H_2 by the adversary using uniformly chosen random values from the co-domain. Furthermore, we choose a random $\mathsf{tag}_R \leftarrow \{0,1\}^\lambda$ and if the adversary makes any query to H_1 or H_2 that starts with tag_R before receiving the message from the honest receiver, we abort. Since tag_R is a uniformly chosen string of length λ, the probability that we abort is at most $q \cdot 2^{-\lambda}$ where q is the total number of queries made by the adversary. Since q is polynomial in λ, we infer that Hyb_1 is statistically close to Hyb_0.

- $\underline{\mathsf{Hyb}_2}$: We make the following changes in this hybrid.

  1. We sample a set K_1 as a subset of $[m]$ where each element is independently included in K_1 with probability p_m. We program the output of the random oracle H_1 to output K_1 when the honest receiver makes its query.
  2. If $|K_1| \geq 2p_m m$, we abort.

  This hybrid is statistically close to Hyb_1 using standard Chernoff bounds.

- $\underline{\mathsf{Hyb}_3}$: We make the following changes in this hybrid.

  1. On receiving the second round message from the adversarial sender in each execution, we run the extractor for Com on $\{\mathsf{Com}_{i,j}, c_i, d_{i,j}, e_{i,j}\}_{i \in [m], j \in [n]}$ to obtain $\{\pi_{i,j,2}, \mathsf{msg}_{S,\mathsf{inp},i}, s_{i,j}, \mathsf{msg}_{S,\mathsf{rand},i,j}\}_{i \in [m], j \in [n]}$.
  2. We initialize an empty set I_1. For each $i \in [m]$ and $j \in [n]$, we check if $\pi_{i,j,2} = \Pi_{i,2}(\pi_{i,j,1}, (\mathsf{msg}_{S,\mathsf{inp},i}, \mathsf{msg}_{S,\mathsf{rand},i,j}); s_{i,j})$. If there is some $j \in [n]$ such that the above check does not pass, we add i to I_1.
  3. We construct an inconsistency graph G on m vertices where we add an edge between (i, i') in G if $\mathsf{msg}_{S,\mathsf{inp},i}$ and $\mathsf{msg}_{S,\mathsf{inp},i'}$ do not pass the pairwise consistency check P. We run a 2-approximation algorithm for the minimum vertex cover for this graph G and call this set as I_2.

[18] Note that the adversarial sender input is solely determined by the messages $\{\mathsf{msg}_{S,\mathsf{inp},i}\}$.

4. We initialize another empty set J_1. For each $j \in [n]$, if $(\mathsf{msg}_{S,\mathsf{rand},1,j}, \ldots, \mathsf{msg}_{S,\mathsf{rand},m,j})$ do not pass the global predicate check P', we add j to J_1.
5. If $|I_1| \geq \lambda$, or $|I_2| \geq \lambda$, or $|J_1| \geq \lambda$, we abort.

In Lemma 5.7, we show that Hyb_2 and Hyb_3 are statistically close.

- $\underline{\mathsf{Hyb}_4}$: We make the following changes in this hybrid.

  1. For each $i \notin K_1$ and each $j \in [n]$, we replace the messages inside the commitments a_i and $b_{i,j}$ to be dummy values.

  It follows from the computational hiding property of the commitment scheme Com that Hyb_4 and Hyb_5 are computationally indistinguishable.

- $\underline{\mathsf{Hyb}_5}$: We make the following changes in this hybrid for each execution initiated by the adversarial sender.

  1. On receiving the second round message from the sender, we compute the sets I_1, I_2, J_1 as before.
  2. In the output computation step performed by the honest receiver,
     (a) For each $j \notin L_2 \cup J_1$ and for each $i \in K_1 \cup I_1 \cup I_2$, we replace the output of $\mathsf{out}_{\Pi_i}(\pi_{i,j,2}, r_{i,j})$ with a default value $\perp$.
     (b) For each $j \in J_1$, we replace α_j while computing the $\mathsf{Majority}$ function with $\perp$.

  In Lemma 5.8, we show that Hyb_4 and Hyb_5 are identically distributed.

- $\underline{\mathsf{Hyb}_6}$: We make the following changes in this hybrid.

  1. For each $i \notin K_1$ and $j \in [n]$, we compute $\pi_{i,j,1}$ using the simulator $\mathsf{Sim}_{\Pi_i,S}$. We use this to generate the first round message from the receiver.
  2. On receiving the second round message from the sender, we compute the sets I_1, I_2, and J_1 as above.
  3. For each $i \notin K_1 \cup I_1 \cup I_2$ and for each $j \notin J_1 \cup L_2$, we send $\pi_{i,j,2}$ to $\mathsf{Sim}_{\Pi_i,S}$ along with the explanation $(\mathsf{msg}_{S,\mathsf{inp},i}, \mathsf{msg}_{S,\mathsf{rand},i,j}, s_{i,j})$ and if this instructs the ideal functionality to deliver the output to the receiver. If yes, we compute the output using the honest receiver shares.

  In Lemma 5.9, we prove that $\mathsf{Hyb}_5 \approx_c \mathsf{Hyb}_6$ based on the reusable security of $\{\Pi_i\}_{i\in[n]}$ against explainable senders.

- $\underline{\mathsf{Hyb}_7}$: We make the following changes in this hybrid.

  1. We run the simulator $\mathsf{Sim}_{\Phi,S}$ for the protocol Φ by corrupting the receiver client and provide the first set of server corruptions given by $T_1 = K_1$. The simulator provides with $\{\mathsf{msg}_{R,\mathsf{inp},i}\}_{i\in K_1}$. We use this to generate the first round message in the protocol.
  2. On receiving the second round message from $\mathcal{A}$ for each execution, we compute the sets I_1, I_2 and J_1 as before.

3. We continue our interaction with $\mathsf{Sim}_{\Phi,S}$ and corrupt the set of servers given by $T = I_1 \cup I_2$. We extract $\{\mathsf{msg}_{S,\mathsf{inp},i}, \mathsf{msg}_{S,\mathsf{rand},i,j}\}_{i \notin (K_1 \cup I_1 \cup I_2), j \notin L_2 \cup J_1}$ and provide them as the first round messages from the malicious sender to the honest servers.
4. $\mathsf{Sim}_{\Phi,S}$ extracts the adversarial sender input y and we forward this to the ideal functionality. We compute the output of f using the honest receiver input x.
5. For each $j \notin J_1 \cup L_2$, we set $\alpha_j = f(x,y)$ and use this to compute the output of the protocol as before.

In Lemma 5.10, we use the reusable security against verifiable senders property of the protocol Φ to show that $\mathsf{Hyb}_6 \approx_c \mathsf{Hyb}_7$. We note that Hyb_7 is identical to the output of the ideal execution as $\mathsf{Majority}(\{\alpha_j\}_{j \notin L_2})$ is guaranteed to output $f(x,y)$.

Lemma 5.7. $\mathsf{Hyb}_2 \approx_s \mathsf{Hyb}_3$.

Proof. We note that the only difference in Hyb_2 and Hyb_3 is that in Hyb_3 if $|I_1|$ or $|I_2|$ or $|J_1|$ is greater than or equal to λ, we abort. We show that if this event happens, then we abort in Hyb_2 as well except with negligible probability. The proof of this claim when $|I_1| \geq \lambda$ or $|I_2| \geq \lambda$ is identical to the proof of Lemma 5.4. We only consider the case that $|J_1| \geq \lambda$.

Consider any query made by the adversary to the random oracle H_2. If for that particular query, $|J_1| \geq \lambda$, then we show that $L_2 \cap J_1 \neq \emptyset$ with overwhelming probability. If this happens, then we infer that the honest receiver client in Hyb_2 also aborts.

$$
\begin{aligned}
\Pr[|J_1 \cap L_2| = 0] &= (1-p_n)^{|J_1|} \\
&\leq (1-p_n)^{\lambda} \\
&= (1-\frac{\lambda}{2n})^{\lambda} \\
&\leq \exp(-\lambda)
\end{aligned}
$$

as $n = 4\lambda$. By a standard union bound over all the queries made by $\mathcal{A}$ to the random oracle H_2, we get that the probability that adversary makes a query to H_2 such that $|J_1 \cap L_2| = 0$ is $q \cdot \exp(-\lambda)$ where q is the total number of queries made by $\mathcal{A}$ to the random oracle. Since q is polynomial in λ, we infer that Hyb_2 and Hyb_3 are statistically close.

□

Lemma 5.8. $\mathsf{Hyb}_4 \equiv \mathsf{Hyb}_5$

Proof. Note that for each $j \notin L_2 \cup J_1$, $(\mathsf{msg}_{S,\mathsf{rand},1,j}, \ldots, \mathsf{msg}_{S,\mathsf{rand},m,j})$ satisfy the global consistency predicate P'. Further, for each such j, and for any $i, i' \in [m] \setminus K_1 \cup I_1 \cup I_2$, $\mathsf{msg}_{S,\mathsf{inp},i}$ and $\mathsf{msg}_{S,\mathsf{inp},i'}$ satisfy the local consistency predicate P'. Further, $|K_1 \cup I_1 \cup I_2| \leq t$. Thus, these messages have the same properties as that of the messages generated by a verifiable adversary (see Definition 4.2). Hence, it follows from the error correction property of Φ that the changes made in Step-2.(a) of this hybrid does not affect the output. To see why the changes described in step-2.(b) does not affect the output, notice that for each $j \notin J_1 \cup L_2$, it follows from the furthermore part of the error correction property of Φ that the output of Dec remains the same and hence, all $\{\alpha_j\}_{j \notin J_1 \cup L_2}$ are equal to the same value α. Since, $|J_1| < \lambda$, $|L_2| \leq \lambda$ (with overwhelming probability from Chernoff bounds) and $n = 4\lambda$, it follows that $\mathsf{Majority}(\{\alpha_j\}_{j \notin L_2})$ is equal to α. This shows that Hyb_4 and Hyb_5 are identically distributed. □

Lemma 5.9. *Assuming the reusable security of $\{\Pi_i\}_{i\in[n]}$ against explainable senders, we have* $\mathsf{Hyb}_5 \approx_c \mathsf{Hyb}_6$.

Proof. Assume for the sake of contradiction that Hyb_5 and Hyb_6 are computationally distinguishable with non-negligible advantage.

Via a standard averaging argument, there exists two hybrids Hyb'_5 and Hyb'_6 and an $i \in [m] \setminus K_1$ and $j \in [n]$ such that Hyb'_5 and Hyb'_6 are distinguishable with non-negligible advantage and these two hybrids only differ in how $\pi_{i,j,1}$ is generated and the output of Π_i is computed. In Hyb'_5, it is generated using the honest receiver algorithms whereas in Hyb'_6, it is generated using the simulator.

We start interacting with the external challenger and provide $\mathsf{msg}_{R,\mathsf{inp},i}$ as the honest receiver input. The challenger responds with $\pi_{i,j,1}$ and we use it to generate the first round message in the protocol. On receiving the second round message from the adversary for a execution, we compute the sets I_1, I_2, J_1 as before. If $j \notin L_2 \cup J_1$ and if $i \notin (K_1 \cup I_1 \cup I_2)$, we forward the message $\pi_{i,j,2}$ to the external challenger along with the explanation $(\mathsf{msg}_{S,\mathsf{inp},i}, \mathsf{msg}_{S,\mathsf{rand},i,j}, s_{i,j})$. This explanation is guaranteed to be valid from the definition of the set I_1. The challenger responds with the output. We use this output to compute the output of the honest receiver for this execution. We finally output whatever the adversary outputs at the end of all executions.

If the challenger interacted with the above reduction using the honest receiver algorithms, then the output of the above reduction is identical to Hyb'_5. Else, it is identically distributed to Hyb'_6. Since we assumed that Hyb'_5 and Hyb'_6 are computationally distinguishable with non-negligible advantage, the above reduction breaks the reusable security of Π_i against explainable senders and this is a contradiction.

□

Lemma 5.10. *Assuming the reusable security against verifiable senders property of Φ, we have* $\mathsf{Hyb}_6 \approx_c \mathsf{Hyb}_7$.

Proof. Assume for the sake of contradiction that Hyb_6 and Hyb_7 are computationally distinguishable with non-negligible advantage. We show that this contradicts the reusable security against verifiable senders property of Φ.

We start interacting with the external challenger and provide x as the honest sender input. We corrupt the sender client and output K_1 to be the initial set of servers to be corrupted. The challenger responds with $\{\mathsf{msg}_{R,\mathsf{inp},i}\}_{i\in K_1}$. We use this to generate the first round message in the protocol. On receiving the second round message for each execution from the adversarial sender, we compute the sets I_1, I_2 and J_1 as before. We set $I_1 \cup I_2$ as the second set of corrupted servers for this execution. We extract $\{\mathsf{msg}_{S,\mathsf{inp},i}, \mathsf{msg}_{S,\mathsf{rand},i,j}\}_{i\notin(K_1\cup I_1\cup I_2), j\notin L_2\cup J_1}$ and provide them as the first round messages from the malicious sender to the honest servers. The challenger responds with the output $\{\alpha_j\}_{j\notin L_2\cup J_1}$. We use this compute the receiver's output in this execution. We finally output whatever the adversary outputs at the end of all executions.

If the messages generated by the challenger are as per the honest protocol execution then the output of the above reduction is identically distributed to Hyb_6. Else, it is identically distributed to Hyb_7 (note that the furthermore part of error correction property ensures that the value of all α_j's are the same). Furthermore, the adversary that is emulated by the above reduction is verifiable (this follows from the definition of I_2 and J_1) and $|K_1 \cup I_1 \cup I_2| \leq t$. Thus, if Hyb_6 and Hyb_7 are computationally distinguishable with non-negligible advantage, then the above reduction breaks the reusable security of the protocol Φ against verifiable senders and this is a contradiction. □

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.[19]

- 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.
- 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 6.1 (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.*

[19] 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.

- ***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 5.*

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

Figure 5: Descriptions of Real and Ideal experiments.

6.2 Construction

Building Blocks. The construction makes use of the following building blocks:

- A straight-line extractable commitment $(\mathsf{Com}, \mathsf{Open}, \mathsf{Extract})$ that is additively homomorphic. We require the commitment to be computationally hiding and statistically binding. Furthermore, the message space $\mathcal{M}$ forms a group with operation " $+$ " and the randomness space forms a group $\mathcal{R}$ with group operation "$+$" and the commitment space $\mathcal{C}$ forms a group with operation " $\cdot$ ". If $\mathsf{com}_1 = \mathsf{Com}(m_1; r_1)$ and $\mathsf{com}_2 = \mathsf{Com}(m_2; r_2)$, then we have:

$$\mathsf{com}_1 \cdot \mathsf{com}_2 = \mathsf{Com}(m_1 + m_2; r_1 + r_2)$$

  Such additively homomorphic commitments are known from standard cryptographic assumptions such as DDH [ElG86], QR [GM82], and LWE [Reg05].

- A t-out-of-m Shamir secret sharing scheme $(\mathsf{Share}, \mathsf{Rec})$ that can corrupt up to t errors. We set $t = 2\lambda$ and $m = O(t)$.

- A m-party MPC protocol Φ that is secure against t malicious corruptions for computing a function f' that takes $\{x_{i,j}\}_{j \in [n]}$ from party P_i and does the following:

  - It computes $x_j = \mathsf{Rec}(x_{1,j}, \ldots, x_{m,j})$ for each $j \in [n]$.
  - It outputs $f(x_1, \ldots, x_n)$ to every party.

  We say that two views view_i and $\mathsf{view}_{i'}$ of parties P_i and $P_{i'}$ in this MPC protocol are pairwise consistent if the messages sent by party P_i (resp. $P_{i'}$) to $P_{i'}$ (resp. P_i) in view_i (resp. $\mathsf{view}_{i'}$) are correctly represented as incoming messages in $\mathsf{view}_{i'}$ (resp. view_i).

- A hash function $H : \{0,1\}^* \to (\{0,1\}^{k_m})^m$ that is modelled as a random oracle. Here, k_m is the number of random bits to needed to toss a biased coin that outputs 1 with probability $p_m = \frac{\lambda}{2m}$. We model the output of hash functions H as subset of $[m]$ where each element of the set is included independently with probability p_m.

Description of the Protocol. The formal description of the protocol appears in Figure 6. $(\mathsf{Com}, \mathsf{Open}, \mathsf{Extract})$ for this construction are identical to the corresponding algorithms in the additively homomorphic commitment scheme.

- Prove : On input $(\mathsf{com}_1, \ldots, \mathsf{com}_n)$, a function f and $(\mathsf{Open}(\mathsf{com}_1), \ldots, \mathsf{Open}(\mathsf{com}_n))$, Prove does the following:
  1. Let $x_1, \ldots, x_n$ be the messages committed inside $\mathsf{com}_1, \ldots, \mathsf{com}_n$.
  2. For each $j \in [n]$,
     (a) It chooses random elements $r_{1,j}, \ldots, r_{t,j}$ from $\mathcal{M}$ and computes $\mathsf{com}^{(r)}_{i,j} \leftarrow \mathsf{Com}(r_{i,j})$ for each $i \in [t]$.
     (b) For each $i \in [m]$, it computes the commitment to the i-th share $\mathsf{com}_{i,j}$ by homomorphically applying the Share operation on $(\mathsf{com}_j, \{\mathsf{com}^{(r)}_{i,j}\}_{i\in[t]})$. Let this share be denoted by $x_{i,j}$.
  3. It runs the MPC protocol Φ "in its head" with the private input of P_i being $\{x_{i,j}\}_{j\in[n]}$. Let $\mathsf{view}_1, \ldots, \mathsf{view}_m$ be the view of the parties.
  4. It generates $\overline{\mathsf{com}}_i \leftarrow \mathsf{Com}(\mathsf{view}_i)$ for each $i \in [m]$.
  5. It computes $K := H(\mathsf{tag}, \{\mathsf{com}_j, \{\mathsf{com}^{(r)}_{i',j}, \overline{\mathsf{com}}_i, \mathsf{com}_{i,j}\}_{i'\in[t], i\in[m]}\}_{j\in[n]})$ where $\mathsf{tag} \leftarrow \{0,1\}^\lambda$ and interprets K as a subset of $[m]$.
  6. It sets $\pi = (\mathsf{tag}, \{\overline{\mathsf{com}}_i, \{\mathsf{com}^{(r)}_{i',j}, \mathsf{com}_{i,j}\}_{j\in[n]}\}_{i'\in[t], i\in[m]}, \{\mathsf{Open}(\overline{\mathsf{com}}_i),$
     $\{\mathsf{Open}(\mathsf{com}_{i,j})\}_{j\in[n]}\}_{i\in K})$.
- Verify : On input $(\mathsf{com}_1, \ldots, \mathsf{com}_n)$, a function f and the proof π, Verify does the following:
  1. It checks if $\mathsf{com}_{i,j}$ is obtained as a result of homomorphically applying Share on $\mathsf{com}_j, \{\mathsf{com}^{(r)}_{i,j}\}_{i\in[t]}$.
  2. It recomputes K as described in Step 5 of Prove and checks if all the openings are correct.
  3. For each $i \in K$, it checks if view_i in $\overline{\mathsf{com}}_i$ is correctly computed as per the protocol specification when starting with the private inputs $\{x_{i,j}\}_{j\in[n]}$ that is committed in $\{\mathsf{com}_{i,j}\}_{j\in[n]}$. It also checks if output of the party P_i in view_i is 1.
  4. For each $i, i' \in K$, it checks if view_i and $\mathsf{view}_{i'}$ are pairwise consistent.
  5. If any of the above checks fail, it outputs 0. Else, it outputs 1.

Figure 6: Construction of Non-Interactive Reusable Commit-and-Prove Protocol

6.3 Proof of Security

The completeness of the protocol can be easily verified and we now show soundness and reusable zero-knowledge property.

6.3.1 Soundness

Let P^* be a malicious prover that wins the soundness game described in Definition 6.1 with non-negligible probability $\mu(\lambda)$. Consider the following sequence of hybrids.

- $\underline{\mathsf{Hyb}_0}$: This corresponds to the soundness game with the prover P^* and by assumption, P^* wins the game with probability at least $\mu(\lambda)$.

- $\underline{\mathsf{Hyb}_1}$: We make the following changes in this hybrid.

  1. On receiving the proof π, we run the extractor $\mathsf{Extract}$ for Com on $\{\overline{\mathsf{com}}_i, \{\mathsf{com}_{i,j}\}_{j\in[n]}\}_{i\in[m]}$ to obtain $\mathsf{view}_1, \ldots, \mathsf{view}_m$ and $\{x_{i,j}\}_{i\in[m],j\in[n]}$.
  2. We initialize two empty sets I_1 and I_2.
  3. For each $i \in [m]$, we check if view_i is generated honestly as per the protocol specification when starting with the private inputs $\{x_{i,j}\}_{j\in[n]}$. If not, we add i to I_1.
  4. We construct an inconsistency graph G on m vertices where we add an edge between (i, i') in G if view_i and $\mathsf{view}_{i'}$ are not pairwise consistent. We run a 2-approximation algorithm for the minimum vertex cover for this graph G and call this set as I_2.
  5. If $|I_1| \geq \lambda$ or $|I_2| \geq \lambda$, we instruct the verifier to output 0.

  Via an identical argument to Lemma 5.4, we note that the probability that P^* wins the game in Hyb_1 is at least $\mu(\lambda) - \mathsf{negl}(\lambda)$.

- $\underline{\mathsf{Hyb}_2}$: We make the following changes in this hybrid.

  1. On receiving the proof π, we compute the sets I_1 and I_2 as described before.
  2. We instruct the verifier to output 0 if $K \not\subseteq I_1 \cup I_2$.

  Let E be the event that P^* wins the soundness game in Hyb_1. Observe that every party outside $I_1 \cup I_2$ behaves honestly in the protocol Φ. Note that since $|I_1 \cup I_2| \leq t$, and $(\mathsf{Share}, \mathsf{Rec})$ can corrupt up to t errors, we infer that for any choice of inputs of the parties in $I_1 \cup I_2$, the output of the function f' is going to be 0 if E happens. Since, we assumed that the MPC protocol Φ is perfectly secure against t-malicious corruptions, it follows that the output of each honest party (i.e., the set of parties not in $I_1 \cup I_2$) is going to be 0. Thus, if $K \not\subseteq I_1 \cup I_2$, then the verifier in Hyb_1 outputs 0 as well. Therefore, the probability that P^* wins the soundness game in Hyb_2 is at least $\mu(\lambda) - \mathsf{negl}(\lambda)$.

  We now argue that the probability that $K \subseteq I_1 \cup I_2$ is negligible and this contradicts the assumption that $\mu(\lambda)$ is non-negligible as the probability that P^* wins the game in Hyb_2 is upper bounded by the probability that $K \subseteq I_1 \cup I_2$. Consider any query that P^* makes to the random oracle such that $|I_1| \leq \lambda$ and $|I_2| \leq \lambda$. Note that in this case, the probability that $K \subseteq I_1 \cup I_2$, is at most by $(1 - p_m)^{m-2\lambda} \leq \exp(-\lambda)$ as $m = O(t)$, $t = 2\lambda$, and $p_m = \frac{\lambda}{2m}$. From a standard union bound, the probability that P^* makes any query to the random oracle such that $K \subseteq I_1 \cup I_2$ is at most $q \cdot \exp(-\lambda)$ and this completeness the proof of soundness.

6.3.2 Reusable Zero-Knowledge

Let V^* be an adversarial verifier. We start with the description of Sim.

Description of Sim.

1. Sim answers all the random oracle queries of the adversary using uniformly chosen random values from the co-domain. For each execution started by $\mathcal{A}$, it chooses a random $\mathsf{tag} \leftarrow \{0,1\}^\lambda$ and if the adversary makes any query to H that starts with tag before receiving the proof, then it aborts.

2. For each execution the adversary starts, it samples a set K as a subset of $[m]$ where each element is independently included in K with probability p_m. It programs the output of the random oracle H to output K when queried on the proof it generates. If $|K| \geq 2p_m m$, it aborts.

3. For each $i \in K$ and for each $j \in [n]$, it chooses a random element $x_{i,j}$ from the message space and generates $\mathsf{com}_{i,j}$ as a commitment to this element. Sim starts running the (semi-honest) simulator Sim_Φ by corrupting the subset of the parties given by K and provides $\{x_{i,j}\}_{i\in K, j\in[n]}$ as their private inputs and 1 as the output. It generates $\{\mathsf{com}_{i,j}\}_{i\in K,j\in[n]}$ using these values

4. Sim_Φ returns $\{\mathsf{view}_i\}_{i\in K}$ and Sim uses this to generate $\{\overline{\mathsf{com}}_i\}_{i\in K}$. For each $i \notin K$, it generates $\overline{\mathsf{com}}_i$ as a commitment to a dummy value.

5. Let T be a superset of K of size t. Sim generates $\mathsf{com}_{i,j}$ for each $i \in T \setminus K$, $j \in [n]$ as commitments to dummy values.

6. For each $i \notin T$, $j \in [n]$, Sim generates $\mathsf{com}_{i,j}$ homomorphically from com_j by fixing the shares in the set T to be the values inside $\{\mathsf{com}_{i,j}\}_{i\in T,j\in[n]}$. It also computes $\{\mathsf{com}^{(r)}_{i',j}\}_{i'\in[t],j\in[n]}$ homomorphically from $\{\mathsf{com}_{i,j}\}_{i\in T,j\in[n]}$ and com_j.

7. It computes the rest of the components in the proof as per the protocol specification.

Proof of Indistinguishability. We now argue that the real and the ideal experiments are computationally indistinguishable via a hybrid argument.

- $\underline{\mathsf{Hyb}_0}$: This corresponds to the output of the real experiment described in Definition 6.1.

- $\underline{\mathsf{Hyb}_1}$: In this hybrid, we answer all the random oracle queries of the adversary using uniformly chosen random values from the co-domain. For each execution started by $\mathcal{A}$, we choose a random $\mathsf{tag} \leftarrow \{0,1\}^\lambda$ and if the adversary makes any query to H that starts with tag before receiving the proof, then we abort. Since tag is uniformly chosen, the probability that we abort in this hybrid is at most $q \cdot 2^\lambda$ where q is the total number of queries that the adversary makes.

- $\underline{\mathsf{Hyb}_2}$: We make the following changes in this hybrid. For each execution that the adversary starts,

1. We sample a set K as a subset of $[m]$ where each element is independently included in K with probability p_m. We program the output of the random oracle H to output K when the honest receiver makes its query.
2. If $|K| \geq 2p_m m$, we abort.

This hybrid is statistically close to Hyb_1 using standard Chernoff bounds.

- $\underline{\mathsf{Hyb}_3}$: In this hybrid, we make the following changes.

  1. Let T be a superset of K of size t.
  2. For each $i \notin T$, $j \in [n]$, we generate $\mathsf{com}_{i,j}$ homomorphically from com_j by fixing the shares in the set T to be the values inside $\{\mathsf{com}_{i,j}\}_{i \in T, j \in [n]}$. We also compute $\{\mathsf{com}^{(r)}_{i',j}\}_{i' \in [t], j \in [n]}$ homomorphically from $\{\mathsf{com}_{i,j}\}_{i \in T, j \in [n]}$ and com_j.

  This hybrid is identically distributed to Hyb_2 from the additive homomorphism property of the commitments.

- $\underline{\mathsf{Hyb}_4}$: In this hybrid, we make the following changes for each execution that the adversary starts.

  1. For each $i \in T \setminus K$, $j \in [n]$, we generate $\mathsf{com}_{i,j}$ as commitments to a dummy value.
  2. For each $i \notin K$, we generate $\overline{\mathsf{com}}_i$ as a commitment to a dummy value.

  This hybrid is computationally indistinguishable from the previous hybrid from the hiding property of Com.

- $\underline{\mathsf{Hyb}_5}$: We make the following changes in this hybrid. For each execution that the adversary starts,

  1. We start running the semi-honest simulator Sim_Φ by corrupting the set K of the parties and provide $\{x_{i,j}\}_{i \in K, j \in [n]}$ as the set of private inputs of the corrupt parties and the output to be 1.
  2. Sim_Φ provides with $\{\mathsf{view}_i\}_{i \in K}$ and we use this to generate $\{\overline{\mathsf{com}}_i\}_{i \in K}$.

  We note that Hyb_4 and Hyb_5 are identically distributed from the semi-honest security of the protocol Φ.

- $\underline{\mathsf{Hyb}_6}$: In this hybrid, for each $i \in K$, $j \in [n]$, we generate $\mathsf{com}_{i,j}$ as commitments to random values. We note that Hyb_5 and Hyb_6 are identically distributed from the perfect privacy of Shamir secret sharing scheme. Observe that Hyb_6 is identically distributed to the output of the ideal experiment in Definition 6.1.

Reusable Commit-and-Prove with Updateable State. In Appendix B, we give a construction of a non-interactive commit-and-prove protocol in the random oracle model (without the additional assumption of additively homomorphic commitments) in a weaker setting where the prover and the verifier maintain a state that is updated at the end of each execution.

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 7.1. *Assume black-box access to:*

1. *A (non-reusable) one-sided NISC protocol.*
2. *A non-interactive reusable commit-and-prove protocol satisfying Definition 6.1.*

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

7.1 Construction

Let $f = (f_0, f_1)$ be a two-party functionality. For each $\beta \in \{0,1\}$, f_β takes (i) the offline inputs x_0^{off} and x_1^{off} from P_0 and P_1 respectively, (ii) a common public online input $x_{\mathsf{pub}}^{\mathsf{on}}$, (iii) 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.

Building Blocks. The protocol makes use of the following building blocks.

1. For each $\beta \in \{0,1\}$, a reusable verifiable client-server protocol $\Phi^{(\beta)} = (\mathsf{Share}_R^{(\beta)}, \mathsf{ShareInp}_S^{(\beta)}, \mathsf{ShareRand}_S^{(\beta)}, \mathsf{Eval}^{(\beta)}, \mathsf{Dec}^{(\beta)})$ w.r.t. pairwise predicate P and global predicate P' for computing the function f_β. We require this protocol to be secure against against $t = 4\lambda$ server corruptions (see Definition 4.3). 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 $(\mathsf{Share}_{(t,m)}, \mathsf{Rec}_{(t,m)})$). Some remarks are in order.
   (a) We note that Share_R and $\mathsf{ShareInp}_S$ take the private input of the receiver and sender respectively and run $\mathsf{Share}_{(t,m)}$. Hence, we assume that there is a reconstruction algorithm Rec that takes the shares output by either Share_R or $\mathsf{ShareInp}_S$ and runs $\mathsf{Rec}_{(t,m)}$ on these shares.
   (b) Recall from Remark 4.4 that Eval algorithm does not compute any cryptographic operations.
2. For each $\beta \in \{0,1\}$, a NISC protocol $(\Pi_{i,1}^{(\beta)}, \Pi_{i,2}^{(\beta)}, \mathsf{out}_{\Pi_i}^{(\beta)})$ for computing $\mathsf{Eval}^{(\beta)}(i,\cdot,\cdot)$ (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 5.3, we infer that this protocol satisfies reusable security against explainable senders and standard security against malicious receivers.
3. A non-interactive reusable commit-and-prove protocol $(\mathsf{Com}, \mathsf{Open}, \mathsf{Extract}, \mathsf{Prove}, \mathsf{Verify})$ that satisfies Definition 6.1.
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 the Protocol. The formal description of the first two rounds of the protocol appears in Figure 7 and the output computation is given in Figure 8. At a high-level, the two-sided protocol runs the one-sided NISC protocol given in Section 5 in both directions and uses the commit-and-prove protocol to show the consistency of the offline inputs across these two executions.

7.2 Proof of Security

Let $\mathcal{A}$ be a malicious adversary that corrupts the party $P_{1-\beta}$ for some $\beta \in \{0,1\}$. We start with the description of the simulator Sim.

Description of Sim. Sim runs Sim_S of the one-sided NISC protocol to generate all the messages when P_β acts as the receiver and runs Sim_R to generate all the messages on when P_β acts as the sender. Additionally, it uses the simulator for the commit-and-prove protocol to generate the proof $\pi^{(\beta)}$ and checks if $g_{\mathsf{zk}}[x^{\mathsf{on}}_{\mathsf{pub}}]$ predicate holds on the commitments generated by $P_{1-\beta}$ (by extracting the committed values) and if does not hold, it aborts.

Proof of Indistinguishability. We show that the real and the ideal executions are computationally indistinguishable via a hybrid argument.

- $\underline{\mathsf{Hyb}_0}$: This corresponds to the real execution generate with respect to the adversary $\mathcal{A}$ that corrupts $P_{1-\beta}$.

- $\underline{\mathsf{Hyb}_1}$: In this hybrid, we answer all the random oracle queries made to H_1 and H_2 by the adversary using uniformly chosen random values from the co-domain. Furthermore, we choose a random $\mathsf{tag}_R^{(\beta)}, \mathsf{tag}_S^{(\beta)} \leftarrow \{0,1\}^\lambda$ and if the adversary makes any query to H_1 or H_2 that starts with $\mathsf{tag}_R^{(\beta)}$ or $\mathsf{tag}_S^{(\beta)}$ before receiving the first or second round message from the honest P_β respectively, we abort. Since $\mathsf{tag}_R^{(\beta)}, \mathsf{tag}_S^{(\beta)}$ are uniformly chosen strings of length λ, the probability that we abort is at most $2q \cdot 2^{-\lambda}$ where q is the total number of queries made by the adversary. Therefore, Hyb_1 is statistically close to Hyb_0.

- $\underline{\mathsf{Hyb}_2}$: We make the following changes to this hybrid.

  1. We sample a set $K_1^{(\beta)}$ as a subset of $[m]$ where each element is independently included in $K_1^{(\beta)}$ with probability p_m. We program the output of the random oracle H_1 to output $K_1^{(\beta)}$ when queried on the first round message generated by P_β.
  2. For each session with $\mathcal{A}$, we sample sets $L_1^{(\beta)}$ and $L_2^{(\beta)}$ as subsets of $[m]$ and $[n]$ respectively where each element is independently included in $L_1^{(\beta)}$ and $L_2^{(\beta)}$ with probability p_m and p_m respectively. We program the outputs of the random oracles H_1 and H_2 to output $L_1^{(\beta)}$ and $L_2^{(\beta)}$ when the queried on the second round message generated by P_β.
  3. If $|K_1^{(\beta)}| \geq 2p_m m$, or $|L_1^{(\beta)}| \geq 2p_m m$. or $|L_2^{(\beta)}| \geq 2p_n n$, we abort.

  This hybrid is statistically close to Hyb_1 using standard Chernoff bounds.

- $\underline{\mathsf{Hyb}_3}$: We make the following changes in this hybrid.

- **Round-1:** To generate the first round message using the offline private input x_β^{off}, P_β does the following:

  1. It computes $(\mathsf{msg}^{(\beta)}_{R,\mathsf{inp},1}, \ldots, \mathsf{msg}^{(\beta)}_{R,\mathsf{inp},m}) \leftarrow \mathsf{Share}^{(\beta)}_R(x^{\mathsf{off}}_\beta)$.
  2. For each $i \in [m]$ and $j \in [n]$,
     (a) It samples a uniform random tape $r^{(\beta)}_{i,j}$ to be used in the protocol $\Pi^{(\beta)}_i$.
     (b) It computes $\pi^{(\beta)}_{i,j,1} := \Pi^{(\beta)}_{i,1}(\mathsf{msg}^{(\beta)}_{R,\mathsf{inp},i}; r^{(\beta)}_{i,j})$, $a^{(\beta)}_i \leftarrow \mathsf{Com}(\mathsf{msg}^{(\beta)}_{R,\mathsf{inp},i})$ and $b^{(\beta)}_{i,j} \leftarrow \mathsf{Com}(r^{(\beta)}_{i,j})$.
  3. It computes $K_1^{(\beta)} = H_1(\mathsf{tag}^{(\beta)}_R, \{\pi^{(\beta)}_{i,j,1}, a^{(\beta)}_i, b^{(\beta)}_{i,j}\}_{i\in[m],j\in[n]})$ where $\mathsf{tag}^{(\beta)}_R \leftarrow \{0,1\}^\lambda$ and interprets $K_1^{(\beta)}$ as a subset of $[m]$.
  4. It sends $(\{\pi^{(\beta)}_{i,j,1}, a^{(\beta)}_i, b^{(\beta)}_{i,j}\}_{i\in[m],j\in[n]}, \mathsf{tag}^{(\beta)}_R, \{\mathsf{Open}(a^{(\beta)}_i), \mathsf{Open}(b^{(\beta)}_{i,j})\}_{i\in K_1^{(\beta)},j\in[n]})$ as the first round message.

- **Round-2:** To generate the second round message using the online inputs $(x^{\mathsf{on}}_{\mathsf{pub}}, x^{\mathsf{on}}_{1-\beta})$, $P_{1-\beta}$ does the following:

  1. **Check Phase:**
     (a) It recomputes $K_1^{(\beta)}$ as in step-3 of round-1 and checks if the openings are valid.
     (b) For each $i \in K_1^{(\beta)}$ and for each $j \in [n]$, it checks if $\pi^{(\beta)}_{i,j,1} = \Pi^{(\beta)}_{i,1}(\mathsf{msg}^{(\beta)}_{R,\mathsf{inp},i}; r^{(\beta)}_{i,j})$.
     (c) For each $i, i' \in K_1^{(\beta)}$, it checks if $\mathsf{msg}^{(\beta)}_{R,\mathsf{inp},i}$ and $\mathsf{msg}^{(\beta)}_{R,\mathsf{inp},i'}$ pass the pairwise consistency check P.
  2. If any of the above checks fail, it aborts.
  3. Else, it computes $(\mathsf{msg}^{(1-\beta)}_{S,\mathsf{inp},1}, \ldots, \mathsf{msg}^{(1-\beta)}_{S,\mathsf{inp},m}) \leftarrow \mathsf{ShareInp}^{(\beta)}_S((x^{\mathsf{off}}_{1-\beta}, x^{\mathsf{on}}_{\mathsf{pub}}, x^{\mathsf{on}}_{1-\beta}))$.
  4. For each $j \in [n]$, it independently runs $\mathsf{ShareRand}^{(\beta)}_S$ to obtain $(\mathsf{msg}^{(1-\beta)}_{S,\mathsf{rand},1,j}, \ldots, \mathsf{msg}^{(1-\beta)}_{S,\mathsf{rand},m,j})$.
  5. For each $i \in [m]$ and $j \in [n]$,
     (a) It samples a uniform random tape $s^{(1-\beta)}_{i,j}$ to be used in the protocol $\Pi^{(\beta)}_i$.
     (b) It computes $\pi^{(1-\beta)}_{i,j,2} := \Pi^{(\beta)}_{i,2}(\pi^{(\beta)}_{i,j,1}, (\mathsf{msg}^{(1-\beta)}_{S,\mathsf{inp},i}, \mathsf{msg}^{(1-\beta)}_{S,\mathsf{rand},i,j}); s^{(1-\beta)}_{i,j})$ and $\mathsf{com}^{(1-\beta)}_{i,j} \leftarrow \mathsf{Com}(\pi^{(1-\beta)}_{i,j,2})$.
     (c) It computes $c^{(1-\beta)}_i \leftarrow \mathsf{Com}(\mathsf{msg}^{(1-\beta)}_{S,\mathsf{inp},i})$, $d^{(1-\beta)}_{i,j} \leftarrow \mathsf{Com}(s^{(1-\beta)}_{i,j})$, and $e^{(1-\beta)}_{i,j} \leftarrow \mathsf{Com}(\mathsf{msg}^{(1-\beta)}_{S,\mathsf{rand},i,j})$.
  6. It computes $L_1^{(1-\beta)} = H_1(\mathsf{tag}^{(1-\beta)}_S, \{\mathsf{com}^{(1-\beta)}_{i,j}, c^{(1-\beta)}_i, d^{(1-\beta)}_{i,j}, e^{(1-\beta)}_{i,j}\}_{i\in[m],j\in[n]})$ and $L_2^{(1-\beta)} = H_2(\mathsf{tag}^{(1-\beta)}_S, \{\mathsf{com}^{(1-\beta)}_{i,j}, c^{(1-\beta)}_i, d^{(1-\beta)}_{i,j}, e^{(1-\beta)}_{i,j}\}_{i\in[m],j\in[n]})$ where $\mathsf{tag}^{(1-\beta)}_S \leftarrow \{0,1\}^\lambda$ and interprets $L_1^{(1-\beta)}$ as a subset of $[m]$ and $L_2^{(1-\beta)}$ as a subset of $[n]$.
  7. It computes $\pi^{(1-\beta)} \leftarrow \mathsf{Prove}(\{a^{(1-\beta)}_i\}_{i\notin K_1^{(1-\beta)}}, \{c^{(1-\beta)}_i\}_{i\notin L_1^{(1-\beta)}}, \{\mathsf{Open}(a^{(1-\beta)}_i)\}_{i\notin K_1^{(1-\beta)}},$
     $\{\mathsf{Open}(c^{(1-\beta)}_i)\}_{i\notin L_1^{(1-\beta)}}, g_{\mathsf{zk}}[x^{\mathsf{on}}_{\mathsf{pub}}])$ where $g_{\mathsf{zk}}[x^{\mathsf{on}}_{\mathsf{pub}}](t_1, \ldots, t_k, t'_1, \ldots, t'_{k'})$ does the following:
     (a) $x^{\mathsf{off}} = \mathsf{Rec}(t_1, \ldots, t_k)$.
     (b) $(\overline{x}^{\mathsf{off}}, \overline{x}^{\mathsf{on}}_{\mathsf{pub}}, x^{\mathsf{on}}) = \mathsf{Rec}(t'_1, \ldots, t'_{k'})$.
     (c) Output 1 iff $x^{\mathsf{off}} = \overline{x}^{\mathsf{off}}$ and $\overline{x}^{\mathsf{on}}_{\mathsf{pub}} = x^{\mathsf{on}}_{\mathsf{pub}}$.
  8. It sends
     (a) $\{\mathsf{com}^{(1-\beta)}_{i,j}, c^{(1-\beta)}_i, d^{(1-\beta)}_{i,j}, e^{(1-\beta)}_{i,j}\}_{i\in[m],j\in[n]}$, $\mathsf{tag}^{(1-\beta)}_S$, $\pi^{(1-\beta)}$.
     (b) $\{\mathsf{Open}(\mathsf{com}^{(1-\beta)}_{i,j}), \mathsf{Open}(c^{(1-\beta)}_i), \mathsf{Open}(d^{(1-\beta)}_{i,j}), \mathsf{Open}(e^{(1-\beta)}_{i,j})\}_{i\in L_1^{(1-\beta)},j\in[n]}$.
     (c) $\{\mathsf{Open}(e^{(1-\beta)}_{i,j})\}_{i\in[m],j\in L_2^{(1-\beta)}}$ and $\{\mathsf{Open}(\mathsf{com}^{(1-\beta)}_{i,j})\}_{i\in[m],j\notin L_2^{(1-\beta)}}$.

Figure 7: First Two Rounds of Reusable Black-Box Two-Sided NISC Protocol

- **Output Computation:** To compute the output, P_β does the following:
  1. **Check Phase:**
     (a) It recomputes $L_1^{(1-\beta)}$ and $L_2^{(1-\beta)}$ as in Step-6 of round-2 and checks if all the openings are valid.
     (b) It checks if $\mathsf{Verify}(\{a_i^{(1-\beta)}\}_{i \notin K_1^{(1-\beta)}}, \{c_i^{(1-\beta)}\}_{i \notin L_1^{(1-\beta)}}, g_{\mathsf{zk}}[x_{\mathsf{pub}}^{\mathsf{on}}], \pi^{(1-\beta)}) = 1$.
     (c) Using the openings to the commitments $\mathsf{com}_{i,j}^{(1-\beta)}$, $c_i^{(1-\beta)}$, $d_{i,j}^{(1-\beta)}$ and $e_{i,j}^{(1-\beta)}$ given by $P_{1-\beta}$,
        i. It checks if for each $i \in L_1^{(1-\beta)}$ and $j \in [n]$ that $\pi_{i,j,2}^{(1-\beta)} = \Pi_{i,2}^{(\beta)}(\pi_{i,j,1}^{(\beta)}, (\mathsf{msg}_{S,\mathsf{inp},i}^{(1-\beta)}, \mathsf{msg}_{S,\mathsf{rand},i,j}^{(1-\beta)}); s_{i,j}^{(1-\beta)})$.
        ii. For each $i, i' \in L_1^{(1-\beta)}$, it checks if $\mathsf{msg}_{S,\mathsf{inp},i}^{(1-\beta)}$ and $\mathsf{msg}_{S,\mathsf{inp},i'}^{(1-\beta)}$ pass the pairwise consistency check P.
        iii. For each $j \in L_2^{(1-\beta)}$, it checks if $\{\mathsf{msg}_{S,\mathsf{rand},i,j}^{(1-\beta)}\}_{i\in[m]}$ pass the global predicate check P'.
  2. If any of the above checks fail, it aborts.
  3. Else, for each $j \in [n] \setminus L_2^{(1-\beta)}$,
     (a) It computes $\mathsf{msg}_{2,i,j}^{(\beta)} \leftarrow \mathsf{out}_{\Pi_i}^{(\beta)}(\pi_{i,j,2}^{(1-\beta)}, r_{i,j}^{(\beta)})$ for each $i \in [m]$.
     (b) It computes $\alpha_j^{(\beta)} = \mathsf{Dec}^{(\beta)}(\{\mathsf{msg}_{2,i,j}^{(\beta)}\}_{i\in[m]})$.
  4. It outputs $\mathsf{Majority}(\{\alpha_j^{(\beta)}\}_{j\in[n]\setminus L_2^{(1-\beta)}})$.

Figure 8: Output Computation in Our Two-sided NISC

1. On receiving the first round message from $P_{1-\beta}$, for each $i \in [m]$ and $j \in [n]$, we run the extractor $\mathsf{Extract}$ of the commitment scheme on $a_i^{(1-\beta)}$ to extract $\mathsf{msg}_{R,\mathsf{inp},i}^{(1-\beta)}$ and on $b_{i,j}^{(1-\beta)}$ to extract $r_{i,j}^{(1-\beta)}$.
2. We initiate empty sets $I_1^{(1-\beta)}$ and $I_2^{(1-\beta)}$.
3. For each $i \in [m]$, we check if for all $j \in [n]$, $\pi_{i,j,1}^{(1-\beta)} = \Pi_{i,1}^{(\beta)}(\mathsf{msg}_{R,\mathsf{inp},i}^{(1-\beta)}; r_{i,j}^{(1-\beta)})$. If not, we add i to $I_1^{(1-\beta)}$.
4. We construct an inconsistency graph G on m vertices where we add an edge between (i, i') in G if $\mathsf{msg}_{R,\mathsf{inp},i}^{(1-\beta)}$ and $\mathsf{msg}_{R,\mathsf{inp},i'}^{(1-\beta)}$ do not pass the pairwise consistency check P. We run a 2-approximation algorithm for the minimum vertex cover for this graph G and call this set as $I_2^{(1-\beta)}$.
5. If $|I_1^{(1-\beta)}| \geq \lambda$ or $|I_2^{(1-\beta)}| \geq \lambda$, we abort.

Via an identical argument to the one made in Lemma 5.4, we infer that Hyb_2 and Hyb_3 are statistically close.

- $\underline{\mathsf{Hyb}_4}$: We make the following changes in this hybrid.

1. On receiving the second round message from $P_{1-\beta}$ in each session, we run the extractor for Com on $\{\mathsf{Com}_{i,j}^{(1-\beta)}, c_i^{(1-\beta)}, d_{i,j}^{(1-\beta)}, e_{i,j}^{(1-\beta)}\}_{i\in[m],j\in[n]}$ to obtain $\{\pi_{i,j,2}^{(1-\beta)}, \mathsf{msg}_{S,\mathsf{inp},i}^{(1-\beta)}, s_{i,j}^{(1-\beta)}, \mathsf{msg}_{S,\mathsf{rand},i,j}^{(1-\beta)}\}_{i\in[m],j\in[n]}$.

2. We initialize an empty set $\overline{I}_1^{(1-\beta)}$. For each $i \in [m]$ and $j \in [n]$, we check if $\pi_{i,j,2}^{(1-\beta)} = \Pi_{i,2}^{(\beta)}(\pi_{i,j,1}^{(\beta)}, (\mathsf{msg}_{S,\mathsf{inp},i}^{(1-\beta)}, \mathsf{msg}_{S,\mathsf{rand},i,j}^{(1-\beta)}); s_{i,j}^{(1-\beta)})$. If not, we add i to $\overline{I}_1^{(1-\beta)}$.
3. We construct an inconsistency graph G on m vertices where we add an edge between (i, i') in G if $\mathsf{msg}_{S,\mathsf{inp},i}^{(1-\beta)}$ and $\mathsf{msg}_{S,\mathsf{inp},i'}^{(1-\beta)}$ do not pass the pairwise consistency check P. We run a 2-approximation algorithm for the minimum vertex cover for this graph G and call this set as $\overline{I}_2^{(1-\beta)}$.
4. We initialize another empty set $J_1^{(1-\beta)}$. For each $j \in [n]$, if $(\mathsf{msg}_{S,\mathsf{rand},1,j}^{(1-\beta)}, \ldots, \mathsf{msg}_{S,\mathsf{rand},m,j}^{(1-\beta)})$ do not pass the global predicate check, we add j to $J_1^{(1-\beta)}$.
5. If $|\overline{I}_1^{(1-\beta)}| \geq \lambda$, or $|\overline{I}_2^{(1-\beta)}| \geq \lambda$, or $|J_1^{(1-\beta)}| \geq \lambda$, we abort.

Using an identical argument to Lemma 5.7, we can show that Hyb_3 and Hyb_4 are statistically close.

- $\underline{\mathsf{Hyb}_5}$: We make the following changes in this hybrid. For each session with adversarial $P_{1-\beta}$,

  1. We generate $\pi^{(\beta)}$ as $\mathsf{Sim}_{\mathsf{ZK}}(1^\lambda, \{a_i^{(\beta)}\}_{i \notin K_1^{(\beta)}}, \{c_i^{(\beta)}\}_{i \notin L_1^{(\beta)}}, g_{\mathsf{zk}}[x_{\mathsf{pub}}^{\mathsf{on}}])$.

In Lemma 7.2, we rely on the reusable zero-knowledge property of the commit-and-prove protocol to prove that $\mathsf{Hyb}_5 \approx_c \mathsf{Hyb}_4$.

- $\underline{\mathsf{Hyb}_6}$: We make the following changes in this hybrid for each session with $\mathcal{A}$.

  1. For each $j \in L_2^{(\beta)}$ and $i \in [m] \setminus (L_1^{(\beta)} \cup I_1^{(\beta)} \cup I_2^{(\beta)})$, we change the message inside $\mathsf{com}_{i,j}^{(\beta)}$ to be a dummy value.
  2. For each $i \notin L_1^{(\beta)}$ and each $j \in [n]$, we replace the message inside $c_i^{(\beta)}$ and $d_{i,j}^{(\beta)}$ with a dummy value.
  3. For each $i \notin L_1^{(\beta)}$ and each $j \in [n] \setminus L_2^{(\beta)}$, we replace the message inside $e_{i,j}^{(\beta)}$ with a dummy value.

The computational indistinguishability between Hyb_5 and Hyb_6 directly reduces to the hiding property of Com.

- $\underline{\mathsf{Hyb}_7}$: We make the following changes in this hybrid for each session with $\mathcal{A}$.

  1. For each $i \notin K_1^{(\beta)}$ and each $j \in [n]$, we replace the messages inside the commitments $a_i^{(\beta)}$ and $b_{i,j}^{(\beta)}$ to be dummy values.

It follows from the hiding property of the commitment scheme Com that Hyb_6 and Hyb_7 are computationally indistinguishable.

- $\underline{\mathsf{Hyb}_8}$: We make the following changes in this hybrid.

  1. We compute the sets $I_1^{(1-\beta)}$ and $I_2^{(1-\beta)}$ as before.

2. For each $i \notin (L_1^{(\beta)} \cup I_1^{(1-\beta)} \cup I_2^{(1-\beta)})$ and $j \notin L_2^{(\beta)}$, we generate $\pi_{i,j,2}^{(\beta)}$ using the simulator $\mathsf{Sim}_{\Pi_i^{(1-\beta)},R}$ on input $\pi_{i,j,1}^{(1-\beta)}$ and $\mathsf{msg}_{2,i,j}^{(1-\beta)}$ (which is computed using the honest sender shares). We generate commitments $\{\mathsf{com}_{i,j}^{(\beta)}\}_{i \notin (L_1^{(\beta)} \cup I_1^{(1-\beta)} \cup I_2^{(1-\beta)}), j \notin L_2^{(\beta)}}$ using these values.

Via an identical argument to the proof of Lemma 5.5, we can show that Hyb_7 and Hyb_8 are computationally indistinguishable from the security of $\{\Pi_i^{(1-\beta)}\}_{i \in [m]}$ against malicious receivers.

- $\underline{\mathsf{Hyb}_9 :}$ We make the following changes in this hybrid for each session initiated by $\mathcal{A}$.

1. On receiving the second round message from $P_{1-\beta}$, we compute the sets $\overline{I}_1^{(1-\beta)}, \overline{I}_2^{(1-\beta)}, J_1^{(1-\beta)}$ as before.
2. For each $j \notin L_2^{(1-\beta)} \cup J_1^{(1-\beta)}$ and for each $i \in K_1^{(\beta)} \cup \overline{I}_1^{(1-\beta)} \cup \overline{I}_2^{(1-\beta)}$, we replace the output of $\mathsf{out}_{\Pi_i}(\pi_{i,j,2}^{(1-\beta)}, r_{i,j}^{(\beta)})$ with a default value $\perp$.
3. For each $j \in J_1^{(1-\beta)}$, we replace $\alpha_j^{(\beta)}$ in the output computation of P_β with $\perp$.

Using a similar proof to Lemma 5.8, we can prove that Hyb_9 and Hyb_8 are identically distributed.

- $\underline{\mathsf{Hyb}_{10} :}$ We make the following changes in this hybrid.

1. For each $i \notin K_1^{(\beta)}$ and $j \in [n]$, we compute $\pi_{i,j,1}^{(\beta)}$ using the simulator $\mathsf{Sim}_{\Pi_i^{(\beta)},S}$. We use this to generate the first round message from P_β.
2. On receiving the second round message from $P_{1-\beta}$ in each session, we compute the sets $\overline{I}_1^{(1-\beta)}, \overline{I}_2^{(1-\beta)}$, and $J_1^{(1-\beta)}$ as above.
3. For each $i \notin K_1^{(\beta)} \cup \overline{I}_1^{(1-\beta)} \cup \overline{I}_2^{(1-\beta)}$ and for each $j \notin J_1^{(1-\beta)} \cup L_2^{(1-\beta)}$, we send $\pi_{i,j,2}^{(1-\beta)}$ along with the explanation $(\mathsf{msg}_{S,\mathsf{inp},i}^{(1-\beta)}, \mathsf{msg}_{S,\mathsf{rand},i,j}^{(1-\beta)}, s_{i,j}^{(1-\beta)})$ to $\mathsf{Sim}_{\Pi_i^{(\beta)},S}$ and if this instructs the ideal functionality to deliver the output to the receiver, we compute the output using the honest receiver shares.

A similar argument as given in Lemma 5.9 shows that $\mathsf{Hyb}_9 \approx_c \mathsf{Hyb}_{10}$ assuming the reusable security of $\{\Pi_i^{(\beta)}\}_{i \in [n]}$ against explainable senders.

- $\underline{\mathsf{Hyb}_{11}}$: We make the following changes in this hybrid.

1. On receiving the second round message from $P_{1-\beta}$, we check if $\mathsf{Verify}(\{a_i^{(1-\beta)}\}_{i \notin K_1^{(1-\beta)}}, \{c_i^{(1-\beta)}\}_{i \notin L_1^{(1-\beta)}}, g_{\mathsf{zk}}[x_{\mathsf{pub}}^{\mathsf{on}}], \pi^{(1-\beta)}) = 1$.
2. We extract $\{\mathsf{msg}_{R,\mathsf{inp},i}^{(1-\beta)}\}_{i \notin K_1^{(1-\beta)}}$ and $\{\mathsf{msg}_{S,\mathsf{inp},i}^{(1-\beta)}\}_{i \notin L_1^{(1-\beta)}}$ as before.
3. If yes, we check if $g_{\mathsf{zk}}((\{\mathsf{msg}_{R,\mathsf{inp},i}^{(1-\beta)}\}_{i \notin K_1^{(1-\beta)}}), (\{\mathsf{msg}_{S,\mathsf{inp},i}^{(1-\beta)}\}_{i \notin L_1^{(1-\beta)}})) = 1$. If not, we abort.

We note that $\mathsf{Hyb}_{10} \approx_c \mathsf{Hyb}_{11}$ from the soundness of the commit-and-prove protocol.

- $\underline{\mathsf{Hyb}_{12}}$: We make the following changes in this hybrid.

  1. We receive the first round message from the receiver and compute the sets $I_1^{(\beta)}$ and $I_2^{(\beta)}$ respectively.
  2. For each session with $\mathcal{A}$, we run the simulator $\mathsf{Sim}_{\Phi^{(1-\beta)},R}$ for the protocol $\Phi^{(1-\beta)}$ by corrupting the receiver client and the set of servers given by $L_1^{(\beta)} \cup I_1^{(\beta)} \cup I_2^{(\beta)}$. We request $|[n] \setminus L_2^{(\beta)}|$ number of instances of the random shares that are output by $\mathsf{ShareRand}_S^{(1-\beta)}$ for a fixed set of input shares of the sender (see Remark 4.5). $\mathsf{Sim}_{\Phi^{(1-\beta)},R}$ returns $\{\mathsf{msg}_{S,\mathsf{inp},i}^{(\beta)}, \mathsf{msg}_{S,\mathsf{rand},i,j}^{(\beta)}\}_{i \in L_1 \cup I_1^{(\beta)} \cup I_2^{(\beta)}, j \notin L_2^{(\beta)}}$.
  3. For each $j \in L_2^{(\beta)}$, we run independent instances of $\mathsf{ShareRand}_S^{(1-\beta)}$ to generate $\{\mathsf{msg}_{S,\mathsf{rand},i,j}^{(\beta)}\}_{i \in [m]}$.
  4. We return $\{\mathsf{msg}_{R,\mathsf{inp},i}^{(1-\beta)}\}_{i \notin (L_1^{(\beta)} \cup I_1^{(\beta)} \cup I_2^{(\beta)})}$ as the first round message sent by the corrupt receiver to the honest senders. $\mathsf{Sim}_{\Phi^{(1-\beta)},R}$ queries the ideal functionality on the malicious receiver input $x_{1-\beta}^{\mathsf{off}}$ which we forward it to our trusted party. On receiving the output of $f_{1-\beta}$ from the functionality, we forward it to $\mathsf{Sim}_{\Phi^{(1-\beta)},R}$.
  5. The simulator $\mathsf{Sim}_{\Phi^{(1-\beta)},R}$ returns the second round messages $\{\mathsf{msg}_{2,i,j}^{(1-\beta)}\}_{i \notin (L_1^{(\beta)} \cup I_1^{(\beta)} \cup I_2^{(\beta)}), j \notin L_2^{(\beta)}}$ from the honest servers to the corrupt receiver. We give this as the output of the server emulations to $\mathsf{Sim}_{\Pi_i^{(1-\beta)},R}$ to generate the second round messages $\pi_{i,j,2}^{(\beta)}$.

  Using an identical argument to Lemma 5.6, we can show that Hyb_{11} and Hyb_{12} are computationally indistinguishable from the security of Φ against verifiable receivers.

- $\underline{\mathsf{Hyb}_{13}}$: We make the following changes in this hybrid.

  1. We run the simulator $\mathsf{Sim}_{\Phi^{(\beta)},S}$ for the protocol Φ by corrupting the sender client and provide the first set of server corruptions given by $T_1 = K_1^{(\beta)}$. The simulator provides with $\{\mathsf{msg}_{R,\mathsf{inp},i}^{(\beta)}\}_{i \in K_1^{(\beta)}}$. We use this to generate the first round message in the protocol.
  2. On receiving the second round message from $\mathcal{A}$ for each session, we compute the sets $\overline{I}_1^{(1-\beta)}, \overline{I}_2^{(1-\beta)}$ and $J_1^{(1-\beta)}$ as before.
  3. We continue our interaction with $\mathsf{Sim}_{\Phi^{(\beta)},S}$ and corrupt the set of servers given by $T = \overline{I}_1^{(1-\beta)} \cup \overline{I}_2^{(1-\beta)}$. We extract $\{\mathsf{msg}_{S,\mathsf{inp},i}^{(1-\beta)}, \mathsf{msg}_{S,\mathsf{rand},i,j}^{(1-\beta)}\}_{i \notin (K_1^{(\beta)} \cup \overline{I}_1^{(1-\beta)} \cup \overline{I}_2^{(1-\beta)}), j \notin L_2^{(1-\beta)} \cup J_1^{(1-\beta)}}$ and provide them as the first round messages from the malicious sender to the honest servers.
  4. $\mathsf{Sim}_{\Phi^{(\beta)},S}$ extracts the adversarial sender input $(x_{1-\beta}^{\mathsf{off}}, x_{\mathsf{pub}}^{\mathsf{on}}, x_{1-\beta}^{\mathsf{on}})$ and we forward $(x_{\mathsf{pub}}^{\mathsf{on}}, x_{1-\beta}^{\mathsf{on}})$ to the ideal functionality. We compute the output of f_β using the honest receiver input x_β^{off}.
  5. For each $j \notin J_1^{(1-\beta)} \cup L_2^{(1-\beta)}$, we set α_j to be equal to the output of f_β and use this to compute the output of the protocol as before.

  Using a similar proof as in Lemma 5.10, we can rely on the reusable security against verifiable senders property of the protocol $\Phi^{(\beta)}$ to show that $\mathsf{Hyb}_{12} \approx_c \mathsf{Hyb}_{13}$. We note that Hyb_{13} is identical to the output of the ideal execution as $\mathsf{Majority}(\{\alpha_j^{(\beta)}\}_{j \notin L_2^{(\beta)}})$ is guaranteed to output to be the output of f_β and furthermore, it follows that $x_{1-\beta}^{\mathsf{off}}$ that is extracted by simulators in Hyb_{12} and Hyb_{13} and the value of $x_{\mathsf{pub}}^{\mathsf{on}}$ used by P_0 and P_1 are identical due to the error correction property of $(\mathsf{Share}_{(t,m)}, \mathsf{Rec}_{(t,m)})$ and the fact that the output of $g_{\mathsf{zk}}[x_{\mathsf{pub}}^{\mathsf{on}}]$ is 1.

Lemma 7.2. *Assuming the reusable zero-knowledge property of the commit-and-prove protocol, we have that* $\mathsf{Hyb}_4 \approx_c \mathsf{Hyb}_5$.

Proof. Assume for the sake of contradiction that Hyb_4 and Hyb_5 are computationally distinguishable. We show that this contradicts the reusable zero-knowledge property.

We start interacting with the challenger and produce $\{\mathsf{msg}_{R,\mathsf{inp},i}^{(\beta)}\}_{i \in [m] \setminus K_1^{(\beta)}}$ as the first set of challenge messages. The challenger returns $\{a_i^{(\beta)}\}_{i \notin K_1^{(\beta)}}$ and we use it to generate the first round message in the protocol. For each execution initiated with the adversary, we generate $\{\mathsf{msg}_{S,\mathsf{inp},i}^{(\beta)}\}_{i \notin L_1^{(\beta)}}$ as the second set of challenge messages. We set the functionality f to be $g_{\mathsf{zk}}[x_{\mathsf{pub}}^{\mathsf{on}}]$. We obtain $\{c_i^{(\beta)}\}_{i \notin L_i^{(\beta)}}$ and the proof $\pi^{(\beta)}$ and use it to generate the final round message. We finally output the view of the adversary at the end of all executions.

If the challenger generated the proofs as per the real experiment, then the output of the reduction is identically distributed to Hyb_4. Else, it is distributed identically to Hyb_5. Since we assumed that Hyb_4 and Hyb_5 are computationally distinguishable, the above reduction breaks the reusable zero-knowledge property and this is a contradiction.

□

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.

[ElG86] Taher ElGamal. On computing logarithms over finite fields. In Hugh C. Williams, editor, *CRYPTO'85*, volume 218 of *LNCS*, pages 396–402. Springer, Heidelberg, August 1986.

[GM82] Shafi Goldwasser and Silvio Micali. Probabilistic encryption and how to play mental poker keeping secret all partial information. In *14th ACM STOC*, pages 365–377. ACM Press, May 1982.

[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.

[IKP10] Yuval Ishai, Eyal Kushilevitz, and Anat Paskin. Secure multiparty computation with minimal interaction. In Tal Rabin, editor, *CRYPTO 2010*, volume 6223 of *LNCS*, pages 577–594. Springer, Heidelberg, August 2010.

[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.

[Reg05] Oded Regev. On lattices, learning with errors, random linear codes, and cryptography. In Harold N. Gabow and Ronald Fagin, editors, *37th ACM STOC*, pages 84–93. ACM Press, May 2005.

[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.

A Definition of NISC

This section is mostly taken verbatim from [IKSS22b].

Let $\mathcal{A}$ be a malicious adversary corrupting either the sender or the receiver in the real protocol execution. In the following, we use $\mathsf{Real}(1^\lambda, \Pi, \mathcal{A}, x)$ to denote the joint distribution of the output of the adversary and the output of the honest party (when its input is x) in the real execution of the protocol.

Let Sim be a malicious adversary that is corrupting either the sender or the receiver in the ideal protocol execution. Recall that in the ideal execution, the parties have access to a trusted functionality for computing f that delivers the output to the receiver. The honest party sends its inputs to this trusted functionality, whereas the adversarial party can send an arbitrary value. The functionality computes the output and sends it to Sim. If Sim corrupts the sender, then it can send the instruction to the trusted party whether to deliver this output to the honest receiver or abort. In the latter case, the output of the honest party is abort and in the former, the honest party outputs whatever it receives from the trusted party. We use $\mathsf{Ideal}(1^\lambda, \Pi, \mathsf{Sim}, x)$ to denote the joint distribution of the output of Sim and the output of the honest party when its input is x in the ideal execution of the protocol.

Definition A.1. *A NISC protocol $(\Pi_1, \Pi_2, \mathsf{out}_\Pi)$ is said to compute a function f with security against malicious adversaries if for any non-uniform PPT real world adversary $\mathcal{A}$ that is corrupting either the sender or the receiver, there exists a non-uniform PPT ideal world adversary Sim that is corrupting the same party as that of $\mathcal{A}$ such that for any input x of the uncorrupted party, we have:*

$$\mathsf{Real}(1^\lambda, \Pi, \mathcal{A}, x) \approx_c \mathsf{Ideal}(1^\lambda, \Pi, \mathsf{Sim}, x)$$

We can modify the above definition in the UC-style setting where an environment $\mathcal{Z}$ provides the inputs to the parties and is provided with the outputs of all the parties.

B Commit-and-Prove with Updatable State

In this section, we give a construction of a commit-and-prove protocol satisfying Definition 6.1 in a setting where the prover and the verifier maintain a state that is updated at the end of every execution. This setting is weaker than the one previously considered. The syntax for this setting is almost the same as the previous one except that the Prove algorithm now outputs a public state $\mathsf{st_{pub}}$ and a private state $\mathsf{st_{priv}}$ in addition to the proof π. The prover updates its stored state with $\mathsf{st_{priv}}$ and sends $\mathsf{st_{pub}}$ to the verifier along with the proof π. The verifier updates its previously stored state with $\mathsf{st_{pub}}$ if the proof is correct. Else, it does not make any changes to its stored state. We require this updatable state model commit-and-prove to satisfy the following notion of reusable soundness.

Definition B.1 (Reusable Soundness). *Let P^* be a non-uniform PPT prover. We require the probability that P^* wins the following reusable soundness game to be negligible.*

- $(\mathsf{com}_1, \dots, \mathsf{com}_\ell) \leftarrow P^*(1^\lambda)$.
- *Set verifier's current state* $\mathsf{st_{pub}}$ *to be* $\mathsf{com}_1, \dots, \mathsf{com}_\ell$.
- *Repeat the following until P^* outputs the special symbol* STOP*:*
  1. $(\mathsf{com}_{\ell+1}, \dots, \mathsf{com}_n, f, \pi, \mathsf{st}'_{\mathsf{pub}}) \leftarrow P^*(1^\lambda)$.
  2. *Let $(x_1, \dots, x_n)$ be the output of* $\mathsf{Extract}$ *on inputs* $\mathsf{com}_1, \dots, \mathsf{com}_n$ *respectively.*
  3. *If $f(x_1, \dots, x_n) = 0$ and* $\mathsf{Verify}(\mathsf{st_{pub}}, \mathsf{com}_{\ell+1}, \dots, \mathsf{com}_n, f, \pi) = 1$*, then the prover wins this game.*
  4. *Else, if* $\mathsf{Verify}(\mathsf{st_{pub}}, \mathsf{com}_{\ell+1}, \dots, \mathsf{com}_n, f, \pi) = 1$*, we update the current value of verifier's public state* $\mathsf{st_{pub}}$ *with* $\mathsf{st}'_{\mathsf{pub}}$.

The main theorem we show here is:

Theorem B.2. *Assume black-box access to a straight-line extractable commitment scheme* $(\mathsf{Com}, \mathsf{Open}, \mathsf{Extract})$ *that is computationally hiding and statistically binding. Then, in the random oracle model, there exists a non-interactive reusable commit-and-prove protocol in the updatable state model.*

As such commitments can be constructed unconditionally in the random oracle model, the above theorem gives a random oracle based construction of a non-interactive reusable commit-and-prove protocol in the updatable state model.

Building Blocks. The construction makes use of the following building blocks. Let us fix n to be the maximum arity of the predicate functions f that can provided as inputs to Prove.[20]

1. A straight-line extractable commitment $(\mathsf{Com}, \mathsf{Open}, \mathsf{Extract})$ that is computationally hiding and statistically binding. We assume that the message space $\mathcal{M}$ forms a group with operation '$+$'.
2. A pairwise verifiable t-out-of-m secret sharing scheme $(\mathsf{Share}_{t,m}, \mathsf{Rec}_{(t,m)})$ that can correct up to t errors. Let P be the corresponding pairwise predicate.

[20] Unlike the previous construction, here we need to fix the maximum arity of the predicate functions during the setup phase.

3. A m-party MPC protocol Φ that is secure against t-malicious corruptions for computing a function f' that takes $\{x_{i,j}\}_{j\in[n]}$ from party P_i and does the following:

   (a) For each $j \in [m]$, it computes $\mathsf{Rec}_{(t,m)}(x_{1,j}, \ldots, x_{m,j})$.

   (b) It computes $z = f(x_1, \ldots, x_n)$.

   (c) For each $j \in [n]$, it computes $(y_{1,j}, \ldots, y_{m,j}) \leftarrow \mathsf{Share}_{(t,m)}(x_j)$.

   (d) It outputs $(z, \{y_{i,j}\}_{j\in[n]})$ to P_i.

   We set $t = 6n^2\lambda$ and $m = O(t)$.

4. A hash function $H : \{0,1\}^* \to (\{0,1\}^{k_m})^m$ that is modelled as a random oracle. Here, k_m is the number of random bits to needed to toss a biased coin that outputs 1 with probability $p_m = \frac{\lambda n}{2m}$. We model the output of hash functions H as subset of $[m]$ where each element of the set is included independently with probability p_m.

Description of the Protocol. The formal description of the protocol appears in Figure 9.

Completeness. This can be easily observed from the construction.

Reusable Soundness. Let P^* be a malicious prover that interacts with the honest verifier in multiple rounds of interaction. We say that P^* wins the reusable soundness game if there exists some round of interaction such that P^* produces an accepting proof but the statement is false. We show that for any non-uniform PPT prover P^*, the probability that P^* can win the reusable soundness game is negligible.

Let $(\mathsf{com}_1, \ldots, \mathsf{com}_\ell)$ be the initial commitments that the prover sends. Let $\mathsf{com}_{\ell+1}, \ldots, \mathsf{com}_n$ be the new commitments that the prover sent in this particular round. Parse com_j as $(\mathsf{tag}, \mathsf{com}_{1,j}, \ldots, \mathsf{com}_{m,j}, \mathsf{Open}(\mathsf{com}_{i,j})_{i\in K_i})$ and let $x_{i,j} = \mathsf{Extract}(\mathsf{com}_{i,j})$ for each $i \in [m]$ and $j \in [n]$. If the proof is accepting, then we prove that:

1. $f(x_1, \ldots, x_n) = 1$.

2. Let $\mathsf{st}_{\mathsf{pub}}$ be equal to $\{\widetilde{\mathsf{com}}_{i,j}\}_{i\in[m],j\in[\ell]}$ and let $y_{i,j} = \mathsf{Extract}(\widetilde{\mathsf{com}}_{i,j})$. Then, $\mathsf{Rec}_{(t,m)}(y_{1,j}, \ldots, y_{m,j}) = x_j$ for each $j \in [\ell]$.

except with negligible probability. This is sufficient to show reusable soundness.

Let P^* be a prover such that the above invariant does not hold with non-negligible probability $\mu(\lambda)$. Consider the following sequence of hybrids.

- $\underline{\mathsf{Hyb}_0}$: This corresponds to the actual reusable soundness game where the prover P^* breaks the above invariant with non-negligible probability $\mu(\lambda)$.

- $\underline{\mathsf{Hyb}_1}$: We make the following changes in this hybrid.

  1. Parse the current value of $\mathsf{st}_{\mathsf{pub}}$ as $\mathsf{com}_1, \ldots, \mathsf{com}_\ell$.
  2. On receiving the proof π and the commitments $\{\mathsf{com}_j\}_{j\in[\ell+1,n]}$, we parse com_j as $(\mathsf{tag}, \mathsf{com}_{1,j}, \ldots, \mathsf{com}_{m,j}, \mathsf{Open}(\mathsf{com}_{i,j})_{i\in K_i})$.
  3. For each $j \in [n]$,

- **Commitment** Com'**:** To commit to a message $x \in \mathcal{M}$, we do the following:
  1. We compute $(x_1, \ldots, x_m) \leftarrow \mathsf{Share}_{(t,m)}(x)$.
  2. We compute $\mathsf{com}_i \leftarrow \mathsf{Com}(x_i)$ for each $i \in [m]$.
  3. We choose a random $\mathsf{tag} \leftarrow \{0,1\}^\lambda$.
  4. We compute $K = H(\mathsf{tag}, \{\mathsf{com}_i\}_{i\in[m]})$.
  5. We set $\mathsf{com} = (\mathsf{tag}, \{\mathsf{com}_i\}_{i\in[m]}, \{\mathsf{Open}(\mathsf{com}_i)\}_{i\in K})$.
- **Commitment Validity:** On input $\mathsf{com} = (\mathsf{tag}, \{\mathsf{com}_i\}_{i\in[m]}, \{\mathsf{Open}(\mathsf{com}_i)\}_{i\in K})$, we do the following:
  1. We recompute the set K as given in Step-4 of the commitment algorithm.
  2. For each $i \in K$, we check if the openings provided are correct. Let $\{x_i\}_{i\in K}$ be the messages inside the openings to the commitments $\{\mathsf{com}_i\}_{i\in K}$.
  3. For each $i, i' \in K$, we check if x_i and $x_{i'}$ satisfy the pairwise predicate check P.
  4. If all the above checks pass, we say that the commitment is valid.
- **Opening** Open'**:** It consists of $(\mathsf{Open}(\mathsf{com}_1), \ldots, \mathsf{Open}(\mathsf{com}_m))$.
- **Extraction** $\mathsf{Extract}'$**:** It checks if the commitment is valid and if yes, it outputs $\mathsf{Rec}_{(t,m)}(\mathsf{Extract}(\mathsf{com}_1), \ldots, \mathsf{Extract}(\mathsf{com}_m))$
- Prove : On input $(\mathsf{com}_1, \ldots, \mathsf{com}_n)$ (which are outputs of Com'), a function f and $(\mathsf{Open}(\mathsf{com}_1), \ldots, \mathsf{Open}(\mathsf{com}_n))$, Prove does the following:
  1. For each $j \in [m]$, it parses com_j as $(\mathsf{tag}, \mathsf{com}_{1,j}, \ldots, \mathsf{com}_{m,j}, \mathsf{Open}(\mathsf{com}_{i,j})_{i\in K_i})$.
  2. It runs the MPC protocol Φ "in its head" with the private input of P_i being $\{x_{i,j}\}_{j\in[n]}$ where $x_{i,j}$ is the message inside $\mathsf{com}_{i,j}$. Let $\mathsf{view}_1, \ldots, \mathsf{view}_m$ be the view of the parties and let $\mathsf{out}_i = (z, \{y_{i,j}\}_{j\in[n]})$ be the output of P_i in the protocol.
  3. It generates $\overline{\mathsf{com}}_i \leftarrow \mathsf{Com}(\mathsf{view}_i)$ for each $i \in [m]$, $\mathsf{com}_z \leftarrow \mathsf{Com}(z)$.
  4. For each $j \in [n]$, it computes $\widetilde{\mathsf{com}}_{i,j} = \mathsf{Com}(y_{i,j})$ for each $i \in [m]$. It then chooses a random string $\widetilde{\mathsf{tag}}_j \leftarrow \{0,1\}^\lambda$ and computes $\widetilde{K}_j = H(\widetilde{\mathsf{tag}}_j, \{\widetilde{\mathsf{com}}_{i,j}\}_{i\in[m]})$. It sets $\widetilde{\mathsf{com}}_j = (\widetilde{\mathsf{tag}}_j, \{\widetilde{\mathsf{com}}_{i,j}\}_{i\in[m]}, \{\mathsf{Open}(\widetilde{\mathsf{com}}_{i,j})\}_{i\in\widetilde{K}_j})$.
  5. It computes $K := H(\mathsf{tag}, \{\mathsf{com}_j, \widetilde{\mathsf{com}}_j, \mathsf{com}_z, \{\overline{\mathsf{com}}_i\}_{i\in[m]}\}_{j\in[n]})$ where $\mathsf{tag} \leftarrow \{0,1\}^\lambda$ and interprets K as a subset of $[m]$.
  6. It sets
     (a) $\pi = (\mathsf{tag}, \{\overline{\mathsf{com}}_i, \mathsf{com}_z, \{\widetilde{\mathsf{com}}_j\}_{j\in[n]}\}_{i\in[m]}, \mathsf{Open}(\mathsf{com}(z)), \{\mathsf{Open}(\overline{\mathsf{com}}_i),$
     $\{\mathsf{Open}(\mathsf{com}_{i,j}), \mathsf{Open}(\widetilde{\mathsf{com}}_{i,j})\}_{j\in[n]}\}_{i\in K})$.
     (b) $\mathsf{st_{pub}} = \{\widetilde{\mathsf{com}}_j\}_{j\in[n]}$.
     (c) $\mathsf{st_{priv}} = \{\mathsf{Open}'(\widetilde{\mathsf{com}}_j)\}_{j\in[n]}$.
- Verify : On input $(\mathsf{com}_1, \ldots, \mathsf{com}_n)$, a function f, the proof π and $\mathsf{st_{pub}}$, Verify does the following:
  1. It checks if $(\mathsf{com}_1, \ldots, \mathsf{com}_n)$ is equal to the current value of $\mathsf{st_{pub}}$ and if $\widetilde{\mathsf{com}}_j$ is valid for each $j \in [n]$.
  2. It recomputes K as described in Step 5 of Prove and checks if all the openings are correct.
  3. For each $i \in K$, it checks if view_i in $\overline{\mathsf{com}}_i$ is correctly computed as per the protocol specification when starting with the private inputs $\{x_{i,j}\}_{j\in[n]}$ that is committed in $\{\mathsf{com}_{i,j}\}_{j\in[n]}$. It also checks if output z and $\{y_{i,j}\}_{j\in[n]}$ are correctly computed form view_i. It finally checks if $z = 1$.
  4. For each $i, i' \in K$, it checks if view_i and $\mathsf{view}_{i'}$ are pairwise consistent.
  5. If any of the above checks fail, it outputs 0. Else, it outputs 1. If the output is 1, it updates $\mathsf{st_{pub}} = \{\widetilde{\mathsf{com}}_1, \ldots, \widetilde{\mathsf{com}}_n\}$.

Figure 9: Construction of Non-Interactive Reusable Commit-and-Prove Protocol with Updatable State

(a) We run $\mathsf{Extract}(\mathsf{com}_{i,j})$ for each $i \in [m]$ to obtain $\{x_{i,j}\}_{i \in [m]}$.

(b) We construct an inconsistency graph G_j on $[m]$ vertices and we add an edge between $i, i' \in [m]$ if $x_{i,j}$ and $x_{i',j}$ do not pass the pairwise consistency predicate. We run a 2-approximation algorithm for the vertex cover on G_j and call the resultant vertex cover to be $I_{2,j}$.

(c) If $|I_{2,j}| \geq n\lambda$, we abort.

Using an identical argument to Lemma 5.4, we can show that the probability that the invariant does not hold in Hyb_1 is at least $\mu(\lambda) - \mathsf{negl}(\lambda)$.

- $\underline{\mathsf{Hyb}_2}$: We make the following changes in this hybrid.
  1. Let the current value of $\mathsf{st}_{\mathsf{pub}}$ be $\mathsf{com}_1, \ldots, \mathsf{com}_\ell$.
  2. On receiving the proof π, we run the extractor $\mathsf{Extract}$ for Com on $\{\overline{\mathsf{com}}_i, \{\mathsf{com}_{i,j}, \widetilde{\mathsf{com}}_{i,j}\}_{j \in [n]}\}_{i \in [m]}$, com_z to obtain $\mathsf{view}_1, \ldots, \mathsf{view}_m$, $\{x_{i,j}, y_{i,j}\}_{i \in [m], j \in [n]}$, and z.
  3. We initialize two empty sets I_1 and I_2.
  4. For each $i \in [m]$, we check if view_i is generated honestly as per the protocol specification when starting with the private inputs $\{x_{i,j}\}_{j \in [n]}$. We also check if the output z and $\{y_{i,j}\}_{j \in [n]}$ are correctly computed form view_i. We finally check if $z = 1$. If any of these checks fail, we add i to I_1.
  5. We construct an inconsistency graph G on m vertices where we add an edge between (i, i') in G if view_i and $\mathsf{view}_{i'}$ are not pairwise consistent. We run a 2-approximation algorithm for the minimum vertex cover for this graph G and call this set as I_2.
  6. If $|I_1| \geq \lambda n^2$ or $|I_2| \geq \lambda n^2$, we instruct the verifier to output 0.

  Via an identical argument to Lemma 5.4, we note that the probability that the invariant does not hold in Hyb_2 is at least $\mu(\lambda) - \mathsf{negl}(\lambda)$.

- $\underline{\mathsf{Hyb}_2}$: We make the following changes in this hybrid.
  1. On receiving the proof π, we compute the sets I_1, I_2 and $\{I_{2,j}\}_{j \in [n]}$ as described before.
  2. Let $I = I_1 \cup I_2 \cup \{I_{2,j}\}_{j \in [n]}$. Note that $|I| \leq 3n^2\lambda$.
  3. If $K \subseteq I$, we abort.

  We now argue that the probability that $K \subseteq I$ is negligible. Consider any query that P^* makes to the random oracle such that $|I| \leq 3n^2\lambda$. Note that in this case, the probability that $K \subseteq I$, is at most by $(1 - p_m)^{m - |I|} \leq \exp(-\lambda)$ as $m = O(t)$, $t = 6\lambda n^2$ and $p_m = \frac{\lambda n}{2m}$. From a standard union bound, the probability that P^* makes any query to the random oracle such that $K \subseteq I$ is at most $q \exp(-\lambda)$. Hence, the invariant does not hold in Hyb_2 with probability at least $\mu(\lambda)$.

  We now argue that this contradicts the security of Φ. Note that since $|I| \leq t$, we infer that from the security of Φ, every party outside the set I correctly computes the output of f'. If the first part of the invariant does not hold, then since $(\mathsf{Share}_{(t,m)}, \mathsf{Rec}_{(t,m)})$ can corrupt up to t errors, we infer that for any choice of inputs of the parties in I, the output z of the function f' is going to be 0. Since $K \not\subseteq I$, the verifier is going to reject the proof. Furthermore, since the output of f' is correctly computed by every party outside I, it follows from the error correction property of $(\mathsf{Share}_{(t,m)}, \mathsf{Rec}_{(t,m)})$ that the second part of the invariant holds.

Reusable Zero-Knowledge. To prove zero-knowledge, we consider a slightly weaker definition where the simulator obtains some more information about the commitments $\mathsf{com}_1, \ldots, \mathsf{com}_n$ that are output by Com'. Specifically, instead of only obtaining the openings to the set $K_1, \ldots, K_n$ respectively, we allow the simulator to obtain the openings to a larger set K such that $|K| \leq t$. Specifically, the simulator specifies this set K before hand, and the challenger provides the openings to the shares indexed by this set K. Hence, this does not induce selective opening issues. We note that since $|K| \leq t$, this does not affect the computational hiding property of the commitment Com'. Note that this weaker definition is sufficient for reusable two-sided NISC protocol.

We provide a sequence of hybrids and the description of the simulator is identical to the final hybrid in this sequence.

- $\underline{\mathsf{Hyb}_0}$: This corresponds to the output of the real experiment described in Definition 6.1.

- $\underline{\mathsf{Hyb}_1}$: In this hybrid, we answer all the random oracle queries of the adversary using uniformly chosen random values from the co-domain. At the beginning of the adversarial execution, we choose random tags $\mathsf{tag}_1, \ldots, \mathsf{tag}_\ell \leftarrow \{0,1\}^\lambda$ which are used in generating the commitments to $x_1, \ldots, x_\ell$. For the i-th execution started by $\mathcal{A}$, we retrieve the tag $\mathsf{tag}^{(i)} \leftarrow \{0,1\}^\lambda$ to be used in this execution (which was sampled in the $(i-1)$-th execution) and sample $\mathsf{tag}^{(i+1)} \leftarrow \{0,1\}^\lambda$ to be used in the next execution. We also sample $\mathsf{tag}^{(k)}_{\ell+1}, \ldots, \mathsf{tag}^{(k)}_n, \widetilde{\mathsf{tag}}^{(k)}_1, \ldots, \widetilde{\mathsf{tag}}^{(k)}_\ell \leftarrow \{0,1\}^\lambda$ for $k \in \{i+1\}$ and retrieve the previous sampled values for $k = i$. These tags are used in generating the commitments to $x_{\ell+1}, \ldots, x_n$ and $\{(y_{1,j}, \ldots, y_{m,j})\}_{j \in [\ell]}$ in the i-th and $(i+1)$-th executions. If the adversary makes any query to H that starts with $\mathsf{tag}^{(i)}, \mathsf{tag}^{(i+1)}, \mathsf{tag}_1, \ldots, \mathsf{tag}_\ell, \{\mathsf{tag}^{(k)}_{\ell+1}, \ldots, \mathsf{tag}^{(k)}_n, \widetilde{\mathsf{tag}}^{(k)}_1, \ldots, \widetilde{\mathsf{tag}}^{(k)}_\ell\}_{k \in \{i,i+1\}}$ before receiving this information from the prover, then we abort. Since these tags are uniformly chosen, the probability that we abort in this hybrid is at most $6nq \cdot 2^\lambda$ where q is the total number of queries that the adversary makes.

- $\underline{\mathsf{Hyb}_2}$: We make the following changes in this hybrid. At the beginning of the execution, we sample sets $K_1, \ldots, K_\ell$ as subsets of $[m]$ where each element of $[m]$ is independently included in these sets with probability p_m. We program the output of the random oracle H to output these sets when the honest prover makes its query to generate the commitments $\mathsf{com}_1, \ldots, \mathsf{com}_\ell$. For the i-th execution that the adversary starts,

  1. We retrieve the sets corresponding to i-th execution and sample sets for the $(i+1)$-th execution. We thus obtain $K^{(i)}, K^{(i+1)}, \{K^{(k)}_{\ell+1}, \ldots, K^{(k)}_n,$ $\widetilde{K}^{(k)}_1, \ldots, \widetilde{K}^{(k)}_\ell\}_{k \in \{i,i+1\}}$ as a subset of $[m]$ where each element is independently included in these sets with probability p_m. We program the output of the random oracle to output $K^{(i)}$ when generating the proof for the i-th execution and $K^{(i+1)}$ when generating the proof for the $(i+1)$-th execution. We program the output of the random oracle H to output $K^{(k)}_{\ell+1}, \ldots, K^{(k)}_n, \widetilde{K}^{(k)}_1, \ldots, \widetilde{K}^{(k)}_\ell$ when generating the commitments to $x_{\ell+1}, \ldots, x_n, (y_{1,1}, \ldots, y_{m,1}), \ldots, (y_{1,\ell}, \ldots, y_{m,\ell})$ in k-th execution (where $k \in \{i, i+1\}$) respectively.

  If the size of any of the above sampled sets is $\geq 2p_m m$, we abort. This hybrid is statistically close to Hyb_1 using standard Chernoff bounds.

- $\underline{\mathsf{Hyb}_3}$: In this hybrid, we make the following changes in each execution started by the adversarial verifier V^*. For the i-th execution that the adversary starts,

  1. Let $K_i = K^{(i)} \cup K_1 \cup \ldots K_\ell \cup \{K_{\ell+1}^{(k)} \cup \ldots \cup K_n^{(k)} \cup \widetilde{K}_1^{(k)} \cup \ldots \cup \widetilde{K}_\ell^{(k)}\}_{k \in \{i\}}$ and $K_{i+1} = K^{(i+1)} \cup K_1 \cup \ldots K_\ell \cup \{K_{\ell+1}^{(k)} \cup \ldots \cup K_n^{(k)} \cup \widetilde{K}_1^{(k)} \cup \ldots \cup \widetilde{K}_\ell^{(k)}\}_{k \in \{i+1\}}$. Set $K = K_i \cup K_{i+1}$ Note that the $|K| \leq 5n^2\lambda < t$.
  2. For each $i \notin K$, $j' \in [\ell]$, we generate $\widetilde{\mathsf{com}}_{i,j'}$ as commitments to dummy values.
  3. For each $i \notin K_i$, we generate $\overline{\mathsf{com}}_i$ as a commitment to a dummy value.

  This hybrid is computationally indistinguishable from the previous hybrid from the hiding property of Com.

- $\underline{\mathsf{Hyb}_4}$: We make the following changes in this hybrid for each execution that the adversary starts. Specifically, in the i-th execution that the adversary starts,

  1. We start running the semi-honest simulator Sim_Φ by corrupting the set K_i of the parties and provide $\{x_{i,j}\}_{i \in K_i, j \in [n]}$ as the set of private inputs of the corrupt parties and provide the output z to be 1 and $\{y_{i,j}\}_{i \in K_i, j \in [\ell]}$ computed honestly.
  2. Sim_Φ provides with $\{\mathsf{view}_i\}_{i \in K_i}$ and we use this to generate $\{\overline{\mathsf{com}}_i\}_{i \in K_i}$. We generate the rest of the components in the proof as before.

  We note that Hyb_4 and Hyb_3 are identically distributed from the semi-honest security of the protocol Φ. In the reduction, we use the output of the honest parties that is provided by the challenger to compute $\{\widetilde{\mathsf{com}}_{i,j'}\}_{i \in K, j' \in [\ell]}$.

- $\underline{\mathsf{Hyb}_5}$: In this hybrid, we replace $\{y_{i,j'}\}_{i \in K, j' \in [\ell]}$ with uniformly chosen random values. We note that Hyb_4 and Hyb_5 are identically distributed from the perfect privacy of $(\mathsf{Share}_{(t,m)}, \mathsf{Rec}_{(t,m)})$. The output of Hyb_5 corresponds to the view of the adversarial verifier V^* in the ideal experiment.