Multiparty Reusable Non-Interactive Secure Computation

Fabrice Benhamouda[1] and Huijia Lin[2]

[1] Algorand Foundation, New York, US
[2] University of Washington, Seattle, US

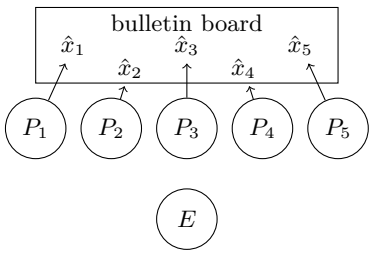
Abstract. Reducing interaction in Multiparty Computation (MPC) is a highly desirable goal in cryptography. It is known that 2-round MPC can be based on the minimal assumption of 2-round Oblivious Transfer (OT) [Benhamouda and Lin, Garg and Srinivasan, EC 2018], and 1-round MPC is impossible in general. In this work, we propose a natural "hybrid" model, called **multiparty reusable Non-Interactive Secure Computation Market (mrNISC)**. In this model, parties publish encodings of their private inputs x_i at the beginning, once and for all. Later, any subset I of them can compute *on-the-fly* a function f on their inputs $\boldsymbol{x}_I = \{x_i\}_{i \in I}$ by just sending a single message to a stateless evaluator, conveying the result $f(\boldsymbol{x}_I)$ and nothing else. Importantly, the input encodings can be *reused* in any number of on-the-fly computations, and the same classical simulation security guaranteed by multi-round MPC, is achieved. In short, mrNISC has minimal yet "tractable" interaction pattern.
We initiate the study of mrNISC on several fronts. First, we formalize the security of mrNISC protocols in both a UC definition and a game-based definition. Second, we construct mrNISC protocols in the plain model with semi-honest and semi-malicious security based on bilinear groups. Third, we demonstrate the power of mrNISC by showing two applications: non-interactive MPC (NIMPC) with reusable setup and a distributed version of program obfuscation. In addition, at the core of our construction of mrNISC is a witness encryption scheme for a special language that verifies Non-Interactive Zero-Knowledge (NIZK) proofs of the validity of computations over committed values, which we believe is of independent interest.

1 Introduction

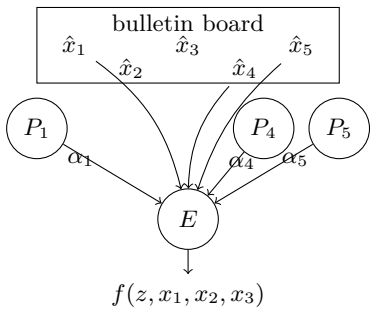
Reducing interaction in Multiparty Computation (MPC) is a highly desirable goal in cryptography, both because each round of communication is expensive and because the liveliness of parties is hard to guarantee, especially when the number of participants is large. Contrary to throughput, latency is now essentially limited by physical constraints, and the time taken by a round of communication cannot be significantly reduced anymore. Moreover, non-interactive primitives are more versatile and more amenable to be used as powerful building blocks. Recent works [11, 27] constructed 2-round MPC protocols from the minimal primitive of 2-round Oblivious Transfer (OT), where in each round all participants

simultaneously broadcast one message. Is it possible to further reduce interaction? The answer is no in general as any non-interactive (i.e., one-round) protocol is susceptible to the so-called residual attack, and cannot achieve the classical simulation security.

In this work, we introduce and study a natural "hybrid" model, between the 2-round and the 1-round settings, which gets us close to having non-interactive protocols while still providing classical security guarantees. We call this model **multiparty reusable Non-Interactive Secure Computation (mrNISC) Market**. In this model, parties publish encodings of their private inputs x_i on a public bulletin board at the beginning, once and for all. Later, any subset I of them can compute *on-the-fly* a function f on their inputs $\boldsymbol{x}_I = \{x_i\}_{i \in I}$ by just sending a single public message to a stateless evaluator, conveying the result $f(\boldsymbol{x}_I)$ and nothing else. Importantly, the input encodings are reusable across any number of computation sessions, and are generated independently of any information of the computation sessions, from the number or set of participants, to the run-time and function of the computation. Figure 1 depicts the setting. The security guarantee is that an adversary corrupting a subset of parties across multiple computation sessions learns no information about the private inputs of honest parties, except for the information it can derive from the outputs of the computations in which the honest parties participated.



(a) Parties P_i publish the encodings $\hat{x}_i$ of their inputs x_i. (Step done a single time, usable for multiple computations.)



(b) Parties P_1, P_4, P_5 want to let the evaluator E compute $f(z, x_1, x_2, x_3)$ by each sending a single message α_i.

Fig. 1: mrNISC market (z is a public input to the function)

Our Contributions. We initiate the study of mrNISC at the following fronts:

Modeling: We introduce the mrNISC market model and formalize both UC security of mrNISC protocols through an ideal mrNISC market functionality, and a simpler game-based security notion that implies UC security.

Construction: We construct mrNISC protocols based on the standard SXDH assumption in asymmetric bilinear groups. While some existing 2-round MPC protocols can be used to construct mrNISC protocols, the resulting

constructions rely on a CRS and multi-key fully-homomorphic encryption (MKFHE) [4,21,41], or on the strong primitives of indistinguishability obfuscation [24] or witness encryption [25,31]. Our new construction gives the first plain-model mrNISC protocols with semi-honest (and semi-malicious) security based on standard assumptions. For malicious security, reliance on some trusted setups is inevitable. We use a CRS.

Techniques: At the core of our construction is a witness encryption (WE) scheme for a special language that verifies non-interactive zero-knowledge (NIZK) proofs of the validity of computations over committed values. We construct it from bilinear groups. This significantly extends the range of languages for which we know how to construct WE from standard assumptions, which is a result of independent interest.

Applications: We demonstrate the power of mrNISC protocols in two cryptographic applications. First mrNISC allows to generically transform non-interactive MPC protocols [9] using correlated randomness into non-interactive MPC protocols in the PKI plus CRS model. Second, mrNISC enables a secret-sharing analogue of Virtual Black-Box program obfuscation [8] — called secret sharing VBB.

In summary, we propose and formalize the mrNISC market model, which has minimal yet "tractable" interaction pattern. We construct the first plain-model mrNISC protocols with semi-honest and semi-malicious security from standard assumptions, using an interesting new tool, WE for NIZK of commitments, that we implement from bilinear groups. We show two applications of mrNISC and believe there will be more.

Relation with Prior Works on MPC with Minimal Interaction. Our mrNISC model naturally generalizes the beautiful notion of reusable NISC by Ishai et al. [38] (and further studied in [2,5,6,16,19]) to the multiparty setting. Reusable NISC is a sender-receiver secure computation protocol, where a receiver publishes a reusable encoding of its input x, so that, any sender can enable it to obtain $f(x, y)$ for any f and y by sending just one message. Both reusable NISC and mrNISC insist on achieving the standard UC simulation security. This differentiates them from the notion of non-interactive MPC (NIMPC) [9] and Private Simultaneous Messages (PSM) [23,36], where the protocols send truly a single message per participant, but necessarily have to settle for weaker security guarantees (e.g., permitting residual attacks in NIMPC) or restricted corruption patterns (e.g., only the evaluator is corrupted in PSM). mrNISC avoids such compromise by asking the parties to send an initial input encoding. Moreover, NIMPC implies obfuscation and is much stronger than classical MPC.

There are, however, important differences between reusable NISC and mrNISC. In reusable NISC only the receiver obtains the output, whereas mrNISC has public output reconstruction (any evaluator, without secrets, seeing all messages can recover the output). Reusable NISC is much better understood: For semi-honest security, it can be constructed from the minimal primitive of OT [38], and for malicious security, from a standard arithmetic generalization of OT known as

oblivious linear-function evaluation [20]. In contrast, prior to our work, mrNISC is only known under MKFHE (or strong obfuscation-related primitives).

Comparison with MKFHE-based Constructions. Our construction diversifies the assumptions and removes CRS in the semi-honest and semi-malicious setting, compared with MKFHE-based mrNISC protocols. It also makes the conceptual point that such reusable input encodings that support arbitrary computations can be implemented without homomorphic encryption (nor strong primitives). This raises interesting future research questions: What is the minimal primitive for mrNISC? Is OT enough? Can we have concretely efficient mrNISC protocols?

Our work can be morally compared with the work of Garg and Srinivasan [26] that constructed the first 2-round MPC protocols without MKFHE, based on bilinear groups. Their work opened the door to settling the minimal primitive for 2-round MPC [11, 27] and followup constructions with various properties Our work is a first step towards investigating similar interesting questions in the natural and practical model of mrNISC.

A more detailed comparison with prior works can be found in Section F.

1.1 Our Results in More Detail

mrNISC Scheme. We start with defining a mrNISC scheme consisting of three algorithms (input encoding Com, computation Encode, and output Eval), which immediately yields an MPC protocol with the minimal interaction pattern, called an mrNISC protocol. We then formalize its security in both a game-based security definition and a UC-security definition, where the latter requires UC-implementing an ideal mrNISC-market functionality. We show that the former implies the latter.

We give both definitions since they each has its own advantages: UC security is the strongest security notion for MPC protocols, and implies security under composition; the formulation of the ideal mrNISC market functionality captures dynamic choices that we envision a "MPC market" should have (details below), while giving a simple interface for using our protocols as components in others protocols. On the other hand, the UC framework is arguably cumbersome, and our game-based security definition is more succinct, self-contained, and easier to manipulate. By showing that our game-based security implies UC security, we have the best of both sides.

- Input Encoding: A party P_i encodes its private input x_i by invoking $(\hat{x}_i, s_i) \leftarrow \mathsf{Com}(1^\lambda, x_i)$. It then publishes the encoding $\hat{x}_i$ and keeps the secret state s_i.
- Computation: In order for a set of parties $\{P_i\}_{i\in I}$ to compute the functionality f on their private inputs $\boldsymbol{x}_I$ and a public input z, each party of the set generates a computation encoding $\alpha_i \leftarrow \mathsf{Encode}(z, \{\hat{x}_j\}_{j\in I}, s_i)$ using its secret state s_i, and sends α_i to the evaluator.
- Output: The evaluator reconstructs the output $y = \mathsf{Eval}(z, \{\hat{x}_i\}_{i\in I}, \{\alpha_i\}_{i\in I})$. (Note that reconstruction is *public* as the evaluator has no secret state.) Correctness requires that $y = f(z, \{x_i\}_{i\in I})$ when everything is honestly computed.

It is easy to see that an mrNISC scheme for f immediately gives an mrNISC protocol for f. Simulation-security requires that the view of an adversary corrupting the evaluator and any subset of parties, can be simulated using just the outputs of the computations that honest parties participate in. However, the same security intuition can be formalized in many settings with different flexibility. In the simplest *selective setting*, where the function f, parties' inputs $x_1, \ldots, \ldots, x_m$, and computations represented by $y^1 = f(z^1, \boldsymbol{x}_{I^1}), \ldots, y^K = f(z^K, \boldsymbol{x}_{I^K})$ are all chosen selectively at the beginning, the view of any subset $\bar{H}$ of corrupted parties should be simulatable by a universal simulator $\mathcal{S}$ as follows.

$$\text{Selective Security:} \quad \left\{ \{x_i, r_i\}_{i\in\bar{H}},\ \{\hat{x}_i\}_{i\in H},\ \{\alpha_i^1\}_{i\in I^1\cap H},\ \ldots,\ \{\alpha_i^K\}_{i\in I^K\cap H} \right\} \approx \left\{ \mathcal{S}\left(\left(y^1, z^1, I^1\right), \ldots, \left(y^K, z^K, I^K\right)\right) \right\},$$

where $\{x_i, r_i\}_{i\in\bar{H}}$ are the inputs and randomness of corrupted parties, and $\hat{x}_i$ and α_i^k are an input encoding and a computation encoding from the honest party P_i ($i \in H$). When the randomness r_i's of the corrupted parties are randomly sampled, the above definition captures *semi-honest* security, and when the randomness r_i's are arbitrary (chosen by the adversary), it captures *semi-malicious* security [4].

Dynamics in the mrNISC Market. The simple selective setting has several drawbacks undesirable for capturing a dynamic mrNISC market we envision. In a mrNISC market, we want to allow all parties' inputs x_i and computations specified by (z^k, I^k) to be chosen adaptively, instead of selectively. Parties have access to a common public bulletin board, but otherwise should only use asynchronous point-to-point authenticated channels, instead of (possibly simultaneous) broadcast channel. In a computation session (z^k, I^k), honest parties in I^k may opt in or out of participation, instead of assuming that all honest parties must participate; and the output is revealed only when all parties in I^k participate. We capture these features by defining the mrNISC-market ideal functionality in the UC framework [14] in Section B.2. In an UC-execution of our ideal functionality, with honest parties and adversary/simulator, the UC environment (that may collude with the simulator) makes all aforementioned choices adaptively and controls the order of message delivery. Clearly, selective security is insufficient for implementing the mrNISC-market ideal functionality. We thus formalize a game-based adaptive security of mrNISC schemes, Definition 2.1 in the overview overview (Section 2.1) and we show that it implies UC-security. We emphasize that our adaptive security is different from security against adaptive corruptions.

Lemma 1.1 (Informal). *An adaptively semi-honestly or semi-maliciously private mrNISC scheme for a function f implies a protocol that semi-honestly or semi-maliciously UC-implements the mrNISC-market ideal functionality for f in the plain model.*

Following standard techniques [4], we can transform semi-malicious UC secure protocols in the plain model into malicious UC-secure protocols in the CRS model using malicious UC-secure NIZK.

Plain-Model mrNISC from Bilinear Groups. We construct an mrNISC scheme for any function in P in the plain model from bilinear maps.

Theorem 1.2 (Informal). *Our construction in Appendix C gives an adaptively semi-maliciously secure mrNISC scheme in the plain model for any function in* P *based on the SXDH assumption on asymmetric bilinear groups.*

Our construction follows the round collapsing approach for constructing 2-round MPC protocols developed in recent works starting from [24], in particular, the approach that uses WE plus NIZK in [31]. We observe that they only need witness encryption for a special language that verifies NIZK proofs for the validity of computation over committed values. We then construct a commitment scheme Com, a NIZK proof system NIZK, and a WE scheme for the language $\mathcal{L}_{\mathsf{Com},\mathsf{NIZK}}$ of statements of form $X = (\mathsf{crs}, c_1, \ldots, c_k, G, y)$ (where crs is a CRS of NIZK, every c_i is a commitment of Com, and G is an arbitrary polynomial-sized circuit). The statement is true iff there exists a NIZK proof π (i.e., the witness) proving w.r.t. crs that G evaluated on the values $x_1, \ldots, x_k$ committed in $c_1, \ldots, c_k$ through Com outputs y, i.e.: $G(x_1, \ldots, x_k) = y$. We refer to such a triple $(\mathsf{Com}, \mathsf{NIZK}, \mathsf{WE})$ as WE for NIZK of commitments, and believe that it is of independent interest and may be useful for other applications.

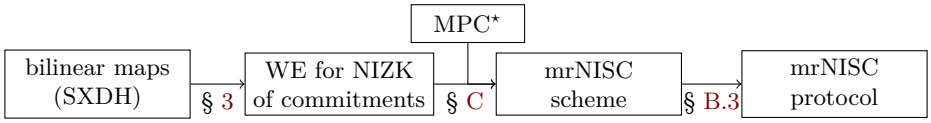


Fig. 2: Construction of mrNISC schemes and protocols (mrNISC protocols implement the mrNISC market ideal functionality; MPC* is an MPC with some special properties defined in Appendix A.7)

Applications. We show two cryptographic applications of our mrNISC scheme. A summary of the applications of mrNISC schemes can be found in Fig. 3.

Our first application is in the topic of Non-Interactive MPC [9] (NIMPC). The NIMPC model permits even less interaction between parties than the mrNISC model: to jointly compute a function, each party sends just a single message to an

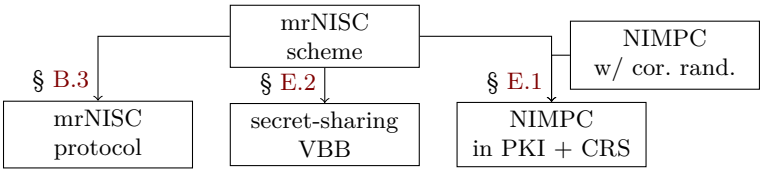


Fig. 3: Applications of mrNISC schemes (mrNISC protocols implement the mrNISC market ideal functionality)

evaluator, *without initially committing to their inputs*. In this setting, we cannot prevent the adversary from evaluating the function on all possible inputs of the corrupted parties. Thus, NIMPC protocols aims at achieving the best-possible security where honest parties' messages reveal no more information than having oracle access to the residual function $f|_{H,\{x_i\}_{i\in H}}$ with the inputs of the honest parties hardcoded inside. NIMPC is a powerful concept equivalent, under different corruption models (i.e., what set $\bar{H}$ of parties can be corrupted), to garbled circuits, Private Simultaneous Messages [23, 36] protocols, and obfuscation. Almost all NIMPC protocols are constructed in a model where parties receive *correlated randomness* sampled by a trusted third party from some distribution. However, correlated randomness is not reusable, and must be re-sampled independently for each computation session. So far, the only construction of NIMPC protocols with reusable setups is by [35], in the (reusable) PKI plus CRS model, and is based on the sub-exponentially security of IO and DDH. Using mrNISC, we give a generic transformation from any NIMPC protocols using correlated randomness to ones in the PKI plus CRS model.

Corollary 1.3. *Applying our transformation to known NIMPC protocols [9, 10], gives the following NIMPC protocols in the PKI plus CRS model assuming mrNISC for* P *and UC-NIZK for* NP*.*

1. *Fully robust NIMPC for the iterated product function* $f(x_1, \ldots, x_n) = x_1 \cdots x_n$ *over a group* $\mathbb{G}$*.*
2. *Fully robust NIMPC for* P *from multi-input functional encryption [29].*
3. *Constant-robust NIMPC for* P*.*

Here, full-robustness means that the protocol is secure no matter what subset of parties the adversary corrupts, and constant-robustness means that security only holds if the adversary corrupt a constant number of parties (each holding a $O(1)$-bit input). The first and third bullets above are achieved for the first time in the PKI plus CRS model. Moreover, we weaken the assumption needed for the second bullet from sub-exponentially secure IO in [35] to polynomially secure IO (which implies multi-input functional encryption), which is a necessary assumption. See Appendix E.1 for more details.

Our second application is a new primitive called *secret-sharing VBB* obfuscation. As the name suggests, it enables the owner of a private program M to secret share M among N servers, where the i'th server holds share M_i. Later, the servers can evaluate the program on any input x, by sending one message, called the output shares, to an evaluator who learns the output $M(x)$ and nothing else; this holds even if the evaluator colludes with all but one servers. Analogous to VBB obfuscation, the secret shares of M are reusable and security is simulation-based. While VBB is impossible in general, secret-sharing VBB can be implemented using mrNISC in a simple way, which in turn can be based on bilinear maps or LWE. Though the construction from mrNISC is simple, we found secret-sharing VBB conceptually interesting and it can be readily used to turn applications of VBB into their secret-sharing counterparts. For instance, for cryptographic

primitives, such as, IBE, ABE, PE, and FE, where a central trusted authority issues secret keys for identities, key policies, and functions respectively, we can decentralize the trusted authority by creating a secret-sharing VBB obfuscation of the key generation algorithm among multiple servers. Importantly, the servers do not need to communicate with each other and only need to send a single message to the inquirer of a key.

Comparison with HSS. The notion of secret-sharing VBB is close to (but different from) the notion of bit-fixing homomorphic sharing proposed in the recent work of [39], and to the notions of Homomorphic Secret Sharing and Function Secret Sharing (HSS/FSS) [12,13]. The main difference from the latter is that in secret-sharing VBB the evaluator may collude with all but one servers, whereas in HSS/FSS the evaluator is honest. Consequently, the security of secret-sharing VBB must hold even when all output shares are made public, whereas HSS/FSS does not guarantee security in this setting. See Appendix E.2 for more details and comparisons with these related notions.

Organization of the Paper We start by giving an overview of the techniques used for implementing mrNISC from bilinear maps as well as a formal definition of mrNISC schemes in Section 2. We define witness encryption for NIZK of commitments, and construct a scheme for NC^1 in Section 3; and show bootstrapping from NC^1 to a scheme for P in Appendix D. the UC definition of mrNISC protocols is presented in Appendix B.2, the formal constructions of UC-secure mrNISC protocols from mrNISC schemes in Appendix C, and the applications of mrNISC in Appendix E.

2 Technical Overview

2.1 Security Definition of mrNISC Schemes

As discussed in the introduction, the mrNISC market allows to adaptively choose which set I^k of parties participate in the computation and which computation z^k is performed. Furthermore, it does not assume broadcast channel, and allows some computations to be "incomplete" due to an honest participant deciding to opt out. Due to space limit, we present the ideal mrNISC market functionality in Appendix B.2, and describe here a game-based adaptive security for mrNISC scheme that implies UC security.

Definition 2.1 (Adaptive Security). *An mrNISC scheme* mrNISC *for* f *is semi-honest (or semi-malicious) private if there exists a PPT simulator* $\mathcal{S}$*, such that, for all PPT adversary* $\mathcal{A}$*, the views of* $\mathcal{A}$ *in the following experiments* $\mathsf{Exp}_{\mathcal{A},\mathcal{S}}(\mathsf{Real}, \lambda, f)$ *and* $\mathsf{Exp}_{\mathcal{A},\mathcal{S}}(\mathsf{Ideal}, \lambda, f)$ *are indistinguishable.*

Experiment $\mathsf{Exp}_{\mathcal{A},\mathcal{S}}(\mathsf{Real}, \lambda, f)$**:** *The adversary* $\mathcal{A}$ *chooses the number of parties* M *and the set of honest parties* $H \subseteq [M]$*. It then interacts with a challenger in an arbitrary number of iterations until it terminates. In every iteration* k*, it can submit* one query *of one of the following three types.*

Corrupt Input Encoding: *Upon $\mathcal{A}$ sending a query $(\mathsf{input}, P_i, x_i, \rho_i)$ for a corrupt party $i \in \bar{H}$, record the input encoding $\hat{x}_i$ generated as $(\hat{x}_i, s_i) = \mathsf{Com}(1^\lambda, x_i; \rho_i)$, using input x_i and randomness ρ_i. In the semi-honest case, ρ_i must be randomly sampled, whereas in the semi-malicious case, it can be arbitrary chosen by $\mathcal{A}$.*

Honest Input Encoding: *Upon $\mathcal{A}$ choosing the input $(\mathsf{input}, P_i, x_i)$ of an honest party $i \in H$, generate $(\hat{x}_i, s_i) \leftarrow \mathsf{Com}(1^\lambda, x_i)$ and send $\hat{x}_i$ to $\mathcal{A}$.*

Honest Computation Encoding: *Upon $\mathcal{A}$ querying $(\mathsf{compute}, P_i, z, I)$ for an honest party $i \in H \cap I$, if the input encodings $\{\hat{x}_j\}_{j\in I}$ of all participants have been generated, send $\mathcal{A}$ the computation encoding $\alpha_i \leftarrow \mathsf{Encode}(z, \{\hat{x}_j\}_{j\in I}, s_i)$.*

Experiment $\mathsf{Exp}_{\mathcal{A},\mathcal{S}}(\mathsf{Ideal}, \lambda, f)$**:** *The ideal experiment proceeds identically as above, except for the following differences: Invoke $\mathcal{S}(1^\lambda, f)$.*

Corrupt Input Encoding: *Additionally send query $(\mathsf{input}, P_i, x_i, \rho_i)$ to $\mathcal{S}$.*

Honest Input Encoding: *Upon $\mathcal{A}$ choosing $(\mathsf{input}, P_i, x_i)$ for $i \in H$, send query (input, P_i) to the simulator $\mathcal{S}$ and forward to $\mathcal{A}$ the simulated input encoding $\tilde{x}_i$ generated by $\mathcal{S}$.*

Honest Computation Encoding: *Upon $\mathcal{A}$ choosing $(\mathsf{compute}, P_i, z, I)$, if this is the last honest computation encoding to be generated for computation $f(z, \star)$ with I (i.e., $\forall\ j \neq i \in I \cap H$, $\mathcal{A}$ has queried $(\mathsf{compute}, P_j, z, I)$ before), send $\mathcal{S}$ the query $(\mathsf{compute}, P_i, z, I, y)$ with the output $y = f(z, \{x_t\}_{t\in I})$; otherwise, send $\mathcal{S}$ the query $(\mathsf{compute}, P_i, z, I)$ without y. Forward to $\mathcal{A}$ the simulated computation encoding $\tilde{\alpha}_i$ generated by $\mathcal{S}$.*

Above, $\mathcal{A}$ is restricted to submit one input query for each party P_i.

We note that in the ideal world, for each computation session, the simulator is required to simulate all but the last honest computation encoding without using the output of that session. This gives the security guarantees that the output remains hidden until all honest participants have sent their computation encodings. As we describe later, additional care is needed to achieve this security.

2.2 Overview of Our mrNISC Scheme

At a high level, we can view mrNISC as 2-round MPC protocols where the first messages are reusable. We thus examine the recent 2-round MPC constructions following the round-collapsing approach [11, 24, 26, 27, 31], and make their first messages reusable, using our key tool of WE for NIZK of commitments. Then we implement the tool using bilinear pairing groups.

The Round Collapsing Approach. The *round-collapsing* approach collapses a inner MPC protocol with a polynomial L number of rounds into a 2-round protocol as follows. In the first round, each party P_i commits $c_i \leftarrow \mathsf{COM}(x_i, r_i)$ to its input x_i and some random tape r_i to be used to execute the inner MPC protocol. In the second round, each party P_i sends one garbled circuit per round $l \in [L]$ of the inner MPC protocol corresponding to the next message function of

P_i. This garbled circuit takes as input all the messages $\boldsymbol{m}^{<l} = \{m_j^\ell\}_{\ell<l}$ sent in previous rounds of the inner MPC, and outputs the next message m_i^l of P_i.

Importantly, each party P_i provides a way to compute the correct labels of its garbled circuits, in particular the labels corresponding to an inner MPC message m_j^ℓ from P_j matching its committed input and randomness x_j, r_j. The work of [31] proposed the following translation mechanism using a general purpose Witness Encryption (WE) scheme. Let k_0, k_1 be two labels of P_i's garbled circuit for round l, for the input wire corresponding to the t'th bit of P_j's message $y = m_{jt}^\ell$. Towards the goal of translating the valid bit y to k_y, P_i's garbled circuit for round $l-1$ outputs two WE encryption $\mathsf{WEnc}(X_0, k_0)$ and $\mathsf{WEnc}(X_1, k_1)$, under the statement X_y that y is the t'th bit of the message m_{jt}^ℓ computed correctly from P_j's committed input and randomness (x_j, r_j), according to the protocol specification and the right transcript of messages. The security of WE ensures that only k_y for the right bit y is revealed. This approach, however, has the issue that decryption requires decommitment to (x_j, r_j), which violates privacy of inputs. To solve the problem, the statement X_y instead requires a witness that is a NIZK proof that y is the valid bit m_{jt}^ℓ computed from (x_i, r_i).

If we had general purpose WE, the above MPC protocols would already have reusable first messages, as the first messages simply consist in a commitment to parties' inputs x_i and randomness r_i. To support multiple computations, we would just need to slightly modify the first message to commit to a PRF seed r_i which can generate pseudo-random tapes for an unbounded number of computations (instead of a fixed length random tape).

Challenge and Our Method The challenge lies in that we do not have general purpose WE from standard assumptions. Previous 2-round MPC constructions from standard assumptions circumvent this problem using weaker tools, namely functional commitment with witness encryption from OT in [11], or homomorphic proof commitment with encryption from bilinear pairing groups in [26], or achieving its effect using OT in [27]. Unfortunately, as we explain shortly, using these weaker tools kills the reusability of the first messages.

We restore the reusability of first messages by identifying that what is needed is **WE for NIZK of commitments**, where statements of WE verify NIZK proofs of the correctness of complex computations over committed values. WE for NIZK of commitments is a triple $(\mathsf{Com}, \mathsf{NIZK}, \mathsf{WE})$ of commitment, NIZK, and WE schemes. It allows to commit to any value $c \leftarrow \mathsf{Com}(v)$ and later reveal NIZK proofs π^k that $G^k(v) = y^k$ for multiple polynomial-size circuits G^k and outputs y^k. In addition, the proofs π^k can be used to decrypt encryptions $\mathsf{ct} \leftarrow \mathsf{WEnc}((c, G^k, y^k), \mathsf{m})$ of a message m with regards to a statement $X^k = (c, G^k, y^k)$, so that the message can be recovered if and only if π^k proves that $G^k(v) = y^k$.

The two key properties of WE for NIZK of commitments is *i) reusability* of commitments – one can generate an unbounded number of NIZK proofs and WE ciphertexts w.r.t. them while keeping committed values hidden (only information in the statements are revealed), and *ii) support* P *computation* – the statements $X^k = (c, G, y)$ are about the correctness of arbitrary polynomial-sized circuits. These two properties are crucial for achieving the resuability of first messages.

Our specific definition and construction of WE for NIZK of commitments has an additional bonus feature that it is "dual-mode" in the sense that in a binding mode, binding of commitments, soundness of NIZK, and semantic security of WE are all information theoretic and perfect, and in a simulation mode, the commitments are perfectly equivocable, NIZK perfectly zero-knowledge. These two modes are controlled by how the CRS is sampled and are indistinguishable. (Don't worry; in our mrNISC the CRSs are sampled by each party.) The "dual-mode" feature is not necessary for mrNISC, but might be useful for other applications.

At a first glance, this object appears to require powerful WE. Nevertheless, we show how to construct the triple of WE, NIZK, and commitment schemes together from bilinear groups, which significantly extends the range of languages for which we know how to construct witness encryption for from standard assumptions. See overview in Section 2.3.

Comparison with Tools used in Previous 2-round MPC The functional commitments with witness encryption in [11] do not have reusability (i.e., only a *single* NIZK proof can be given w.r.t. their commitments, or else committed values are revealed), and the homomorphic proof commitments with encryption of [26,27] do not support P computation. Correspondingly, the first messages in their protocols are not resuable.

In more detail, the homomorphic proof commitment with encryption of [26,27] can be viewed as a WE for NIZK of the statement that (a linear combination of) committed values is 0 or 1. This in turns allows to handle statements of the form: c_1, c_2, c_3 commit to three values v_1, v_2, v_3 such that $v_3 = \text{NAND}(v_1, v_2)$. The acute reader may remark that being able to prove NAND relations between committed values allow to actually prove any statement $X^k = (c, G^k, y^k)$, by including, in the NIZK proof, commitments to intermediate values in the computation of G^k, and proofs of correctness of every NAND gate computation w.r.t. them. This is actually the whole idea of GOS NIZK [32], on which [26] is based. However, we do not know how to construct WE for verifying such NIZK proofs. This is essentially because checking the proof requires verifying *quadratic* relations among elements in the proof. But, we do not how to construct WE verifying quadratic relations in the witness, in fact, if we knew, we would have obtained general purpuse WE.

The work of [26] showed that WE for NIZK of NAND is, nevertheless, sufficient for 2-round MPC, by first converting the inner MPC protocols with general next step functions into one whose next step functions simply computes NAND only. Such MPCs are *conforming*. More specifically, at each round, a single party computes a NAND between two values of its state masked by some *one-time pad*, and broadcasts the resulting value, and finally the other parties append this value to their states. The one-time pads are important for the security of conforming protocols, but make the first messages non-reusable: two executions of the MPC with the same first messages will make use the same one-time pad.

In summary, WE for NIZK of commitments can be seen as having the features of both homomorphic commitments with encryption (namely reusability) and of functional commitments with witness selector (namely support for P

computations). However, our construction of WE for NIZK of commitments depart significantly from their constructions and introduce new ideas.

Handling Dynamics in mrNISC market Beyond making the first flow reusable, there is another subtlety we need to take care of in the mrNISC setting. As discussed in the introduction, in the mrNISC setting, parties' inputs and computations are chosen adaptively, messages may be delivered asynchronously, and some honest parties may opt out of participating in a computation. Technically, this adaptivity means simulation of a message can only use information that is available to the simulator at the moment. In particular, messages in one computation session should be simulated independently of the outputs of other sessions. Moreover, only the last (delivered) honest message in a session can be simulated using the output of that session, as before the last honest message is delivered, the output is hidden from the simulator. This subtle issue can be solved using an inner MPC satisfying that all but the last message from honest parties can be simulated without the output. This is for instance the case with respect to the GMW protocol. We then show that the resulting collapsed mrNISC protocols provides adaptivity in that case.

2.3 Construction of WE for NIZK of Commitments

Our Key Ideas in a Nutshell. At a very high-level, our idea is to design NIZK proofs π that can be verified by a linear equation, so that we can construct WE for verifying the proofs using a *WE for linear languages*, which are essentially hash proof systems or smooth projective hash functions (see, e.g., [1]). More specifically, we would like to be able to turn verifying a NIZK proof $\boldsymbol{\pi}$ of a statement $X = (c, G, y)$ into verifying a system of linear equations $\boldsymbol{\theta} = \Gamma\boldsymbol{\pi}$, where $\boldsymbol{\theta}$ and Γ describe the linear equations and importantly depend only on the statement X (independent of the proof $\boldsymbol{\pi}$). We can then use ideas from hash proof systems to add a witness encryption for verifying the linear system. (More precisely, commitments and NIZKs are bilinear group elements, and the linear equations are on values in the exponent.)

Unfortunately, verifying known NIZK proofs require verifying equations that are quadratic in elements of the proof — the proof contains all intermediate computation values, of course *encoded*, and verification checks the correctness of computation of each gate, which is quadratic. Our key idea is leveraging the fact that that NC^1 circuits can be represented as the co-called restricted multiplication straight-line programs, where multiplication occurs between an intermediate computation value and an input element (i.e., committed values). This structure allows us to design NIZK proofs whose verification is *linear* in elements in the proof (still encoding intermediate computation values), where the linear coefficients depends on elements in the statement (containing encoding of inputs). Hence, we can use WE for linear language to obtain WE for NIZK of commitments for NC^1. Finally, we present a generic bootstrapping technique for lifting from a scheme for NC^1, to a scheme for all polynomial-size circuits P.

Our NIZK for NC^1 with linear verification equations makes use of the homomorphic commitment schemes developed in existing NIZK proofs and some

of the ideas behind these proofs [32,34]. For simplicity, our description below uses GOS homomorphic proof commitments which is based on composite-order bilinear groups. Our final solution is closer to the Groth and Sahai NIZK [34] and based on prime order bilinear groups.

WE for Linear Language. We start with witness encryption for linear languages. A linear language over $\mathbb{Z}_p$ for a prime p consists of tuples of a matrix $\Gamma \in \mathbb{Z}_p^{K\times k}$ and a vector $\boldsymbol{\theta} \in \mathbb{Z}_p^K$ in the column span of Γ. A witness for $(\boldsymbol{\theta}, \Gamma)$ is thus a vector $\boldsymbol{\pi}$ s.t. $\boldsymbol{\theta} = \Gamma\boldsymbol{\pi}$. There is an extremely simple WE scheme for linear language: A ciphertext encrypting $m \in \mathbb{Z}_p$ consists of $\boldsymbol{\alpha}^T\Gamma$ and $\boldsymbol{\alpha}^T\boldsymbol{\theta} + m$ for a random row vector $\boldsymbol{\alpha}^T$. When the statement is false, that is, $\boldsymbol{\theta}$ is outside the column span of Γ, $\boldsymbol{\alpha}^T\Gamma$ contains no information of $\boldsymbol{\alpha}^T\boldsymbol{\theta}$, which hides m.

$$\text{Linear WE} \qquad \mathsf{LWEnc}((\boldsymbol{\theta}, \Gamma), \mathsf{m}) : \ \boldsymbol{\alpha} \leftarrow \mathbb{Z}_p^K, \ \ \mathsf{ct} = \boldsymbol{\alpha}^T\theta + \mathsf{m}, \boldsymbol{\alpha}^T\Gamma$$

Can we use linear WE to verify a complex computation $G(v) = y$ over committed values v? If we had a fully homomorphic commitment scheme for which verification of the opening/decommitment is linear, we would solve the problem completely. Verifying that "c opens to v and $G(v) = y$" is equivalent to that "c' opens to y" w.r.t. c' obtained from homomorphic evaluation of G on c. Now a message m can be encrypted using linear WE w.r.t. c', y (which decides $\boldsymbol{\theta}, \Gamma$) and a proof $\boldsymbol{\pi}$ is simply an opening of c' (ignoring ZK for now). Unfortunately, we do not know how to construct such commitment scheme.

One Multiplication [26, 32]. GOS [32] constructed a commitment scheme with linear opening that can do one homomorphic multiplication, using bilinear groups. This linearity was exploited by Garg and Srinivasan in [26] to construct a WE for a NIZK of commitments for functions G corresponding to a single NAND gate (which essentially corresponds to a single multiplication).

Let $(N, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_t, e, g_1, g_2)$ describe a bilinear group of order N. We use the bracket notation $[a]_b := g_b^a$ in G_b for $a \in \mathbb{Z}_N$ – referred to as an encoding of a, and write $a[a']_b = [aa']_b$ as applying group exponentiation in G_b and $[aa']_t = [a]_1[a']_2$ as applying the pairing operation. GOS uses a composite order $N = pq$ symmetric bilinear group, where the two source groups are the same $\mathbb{G} = \mathbb{G}_1 = \mathbb{G}_2$; we simply write $[a]$ as a source group element.

The CRS of the commitment scheme contains $[h]$ for a random element in $\mathbb{Z}_N$ of *order* q. A commitment to v in $\mathbb{Z}_p$ is simply $[c] = [rh + v]$ using a random scalar $r \leftarrow \mathbb{Z}_N$. Such a commitment is perfectly binding, because h has order q, and v is in $\mathbb{Z}_p$. Given two commitments $[c_1] = [r_1h + v_1]$ and $[c_2] = [r_2h + v_2]$, we can compute a commitment of the product in the target group. Furthermore, we can prove that the product v_1v_2 is equal to some value v_{12}, and the verification is linear in the proof π:

$$\begin{aligned}
\text{One Multiplication} \quad & [c_1c_2]_t = [c_1][c_2] = [(r_1r_2h + r_1v_2 + r_2v_1)\,h + v_1v_2]_t \\
\text{Proof} \quad & [\pi] := [t_1 + t_2h] \quad \text{for } t_1 = r_1v_2 + r_2v_1,\ t_2 = r_1r_2 \\
\text{Verification} \quad & 0 \stackrel{?}{=} [c_1][c_2] - [h][\pi] - [1][v_{12}]
\end{aligned}$$

In other words, the last equation shows that $[\pi] = [t_1 + t_2h]$ is a proof for the statement "$[c_1]$ and $[c_2]$ commits to values v_1 and v_2 so that $v_1v_2 = v_{12}$."

Going beyond one Multiplication. The main issue of the above construction is that a GOS commitment only allows for the evaluation of a single multiplication gate (or equivalently a single NAND), as $[c_1 c_2]_t$ is now in the target group. To evaluate more complex function G, we need to be able to make further multiplications. The idea is that the prover can commit to $v_1 v_2$ in the source group: $[c_\times] = [r_\times h + v_1 v_2]$ and then prove that $[c_\times]$ indeed commits to the same value as $[c_1 c_2]_t$:

$$\text{Multiplication} \quad [c_\times - c_1 c_2]_t = [1][c_\times] - [c_1][c_2] = \left[(-r_1 r_2 h + r_\times - r_1 v_2 - r_2 v_1)\, h\right]_t$$

$$\text{Proof} \quad [\pi_\times] \coloneqq [t_1 + t_2 h] \quad \text{for } t_1 = r_\times - r_1 v_2 - r_2 v_1,\ t_2 = -r_1 r_2 \tag{1}$$

$$\text{Verification} \quad 0 \stackrel{?}{=} [1][c_\times] - [c_1][c_2] - [h][\pi_\times] \tag{2}$$

Furthermore, by linearity of the GOS commitment, it is also possible to prove that a commitment $[c_+] = [r_+ h + v_+]$ commits to a value v_+ that is a linear combination of values v_1 and v_2 committed in $[c_1]$ and $[c_2]$: $v_+ = \mu_1 v_1 + \mu_2 v_2$ (for some public scalars μ_1, μ_2).

$$\text{Linear} \quad [c_+ - \mu_1 c_1 - \mu_2 c_2]_t = [c_+] - \mu_1 [c_1] - \mu_2 [c_2] = \left[(r_+ - \mu_1 r_1 - \mu_2 r_2)\, h\right]_t$$

$$\text{Proof} \quad [\pi_+] \coloneqq r_+ - \mu_1 r_1 - \mu_2 r_2 \tag{3}$$

$$\text{Verification} \quad 0 \stackrel{?}{=} [c_+] - \mu_1 [c_1] - \mu_2 [c_2] - [h][\pi_{12}] \tag{4}$$

To extend to proving P computations, we can proceed as follows. To commit a bitstring v, we commit each bit individually as a GOS commitment: $[c_i] = [r_i h + v_i]$. Then, to prove that $G(v) = y$, we represent G as a sequence of linear operations and multiplications, and introduce an intermediate commitment for each intermediate result. The proof consists in these intermediate commitments $[c'_j]$, intermediate proofs that they were computed properly (using Eq. (1) or Eq. (3)) and the opening r'_o of the commitment $[c'_o] = [r'_o h + y]$ corresponding to the output of G. Verification would consist of verifying the intermediate proofs (using Eqs. (2) and (4)) and the opening of the output commitment.

The final proof would actually be a zero-knowledge proof and would in essence be a GOS or a Groth-Sahai proof [32,34]. The zero-knowledge property comes from the following two facts: (1) if h is chosen to be of order N (instead of q), commitments are fully equivocable, and (2) there is a single proof $[\pi_\times]$ (resp., $[\pi_+]$ satisfying the verification equation 1 (resp., Eq. (3)). Leveraging these two facts, a ZK simulator for a proof of, say one multiplication, can equivocate $c_1, c_2, c_\times$ to any values satisfying $\tilde{v}_\times = \tilde{v}_1 \tilde{v}_2$, the equivocation gives a fake witness for computing the unique proof.

Unfortunately, the final proof verification is not linear: if two intermediate values v_1, v_2 need to be multiplied, Eq. (2) would involve a product of the corresponding two commitments c_1, c_2, which is quadratic in the final proof.

Restricted Multiplication Program. To keep verification linear in the final proof, we remark that we just need to ensure that every multiplication involves at least one input commitment, but never two intermediate commitments (which are part of the final proof). In that case Eq. (2) becomes linear in the intermediate commitment. Hence, we can use the above ideas to verify any restricted

multiplication straight-line (RMS) computation [13, 22], which includes all NC^1 computations. Indeed, in an RMS program, the only allowed operations are linear operations over inputs or intermediate values, and multiplications of one intermediate value v'_j with one input v_i (but not of two intermediate values).

Improved NC^1 Scheme Based on SXDH. The above construction of WE for NIZK of commitments for NC^1 uses composite group order with pairings which are notoriously inefficient. In Section 3, we propose a construction solely based on the standard assumption SXDH over asymmetric bilinear groups. The construction based on Groth-Sahai is more complex, and uses vector subspaces to implement features of the subgroup structure. That's why we explain our ideas w.r.t. the simpler GOS NIZK system.

Polynomial-Size Circuits. We now present a generic bootstrapping technique from a WE scheme for NIZK of commitments for RMS to one for P. We can encode any polynomial-size computation $y = G(v)$ into a randomized encoding $o = \mathsf{RE}(G, v; \mathsf{PRF}(k))$ that reveals only y (with randomness expanded from a seed k using a PRF). Since both RE and PRF are computable in NC^1, our RMS-scheme can verify whether o is correctly computed from v, k committed in some commitments c, but cannot verify that o indeed decodes to y (which belongs to P). Instead, we use a garbled circuit to verify the latter and use WE to ensure that only labels corresponding to the correct RE encoding o are revealed. More precisely, a WE ciphertext of m w.r.t. (G, c, y) for a polynomial-size circuit G contains 1) a garbled circuit $\widehat{F}_{y,m}$ of $F_{y,m}$ that outputs m iff given an input o' that decodes to y, and 2) WE encryption (using the RMS-scheme) of labels under statements that verify the computation of o from (k, v) committed in c. Decryption requires NIZK proofs certifying the correctness of o, which allows recovering labels for o, and then m.

This allows to lift our RMS scheme to a WE scheme for NIZK of commitments for P. Plugging it into the aforementioned 2-round MPC protocols, we obtain semi-honest, in fact semi-malicious, mrNISC protocols in the CRS model from SXDH over bilinear groups; we can further remove the CRS, by letting each party sample a CRS for generating its own commitments, NIZK proofs, and WE ciphertexts, yielding protocols in the plain model. This does not hurt security because for every correctly generated CRS (that describes valid bilinear groups), the binding of commitments holds information theoretically, which in turn implies that the soundness of NIZK, and hiding of WE holds information theoretically.

Applications Due to the lack of space, we refer the reader to Section E for applications of mrNISC. At very high-level, in scenarios, where a set of parties need many copies of freshly sampled correlated randomness, we can use mrNISC to replace correlated randomness with reusable PKI and CRS setup: Parties' public key in the PKI is simply an encoding of their private PRF key, later on, they can jointly run mrNISC to sample fresh correlated randomness using the pseudorandom coins contributed to from all parties' PRF keys. In NIMPC, sampling correlated randomness and generating NIMPC message using this correlated randomness can be combined in one mrNISC computation.

3 Witness Encryption for NIZK of Commitments

In this section, we define and construct our new primitive: witness encryption (WE) for NIZK of commitments (for the complexity class P), which is the main component for the construction of our mrNISC scheme.

As explained in Section 2.3, from a high-level point of view, WE for NIZK of commitments combines the properties of homomorphic proof commitments with encryption [26] and of functional commitments with witness selector [11]. Compared with the former, it supports general statements in P (instead of a single NAND gate evaluation). Compared with the latter, it allows for zero-knowledge to hold when multiple NIZK proofs are generated.

3.1 Definition of Witness Encryption for NIZK of Commitments

We start by defining dual-mode commitment schemes (a.k.a., hybrid commitments [18]), where the CRS can be generated in two computationally indistinguishable ways: one yielding perfectly binding commitments and one yielding equivocal (a.k.a., simulatable or trapdoor) commitments. The term "dual-mode commitment" come from [40].

We may not need dual-mode commitments to construct mrNISC, but just simulatable/equivocal commitments (without a perfectly binding setup). However using dual-mode commitments significantly simplify definitions and proofs. Since our constructions achieve this stronger security notion, we use it. More precisely, without a dual-mode commitment, we could not use the standard definition of witness encryption: witness encryption indeed just ensures that ciphertexts related to a false statement (about the committed value, in our setting) cannot be decrypted. Without the dual mode, because of equivocality of the commitments, it would be possible to open any commitment to any value. Hence any statement about a committed value would be always true or always false (independently of the committed value).

Definition 3.1 (Dual-Mode Commitments). A (dual-mode) commitment scheme COM has a *binding* mode and a *simulation* mode, each involves three polynomial-time algorithms.

- Binding Setup: $\mathsf{crs} \leftarrow \mathsf{CSetup}_{\mathsf{bind}}(1^\lambda)$ on input the security parameter λ generates a binding CRS crs.
- Commitment: $(c, d) \leftarrow \mathsf{CCom}(\mathsf{crs}, v)$ on input the CRS crs and a message v in some implicitly defined message set $\mathcal{V}$,[3] generates a commitment c of v and an associated decommitment (a.k.a., opening) d.

[3] The message set $\mathcal{V}$ may depend on the CRS crs. The only required constraints are that messages in $\mathcal{V}$ have polynomial size in the security parameter λ and that testing membership to $\mathcal{V}$ can be done in polynomial-time given crs. The reason to use messages spaces more complicated than $\{0,1\}^{\mathrm{poly}(\lambda)}$ is to allow messages to be elements of some finite field $\mathbb{Z}_p$ for the definition of bilinear commitments with proofs of quadratic relations.

– Verification: $b := \mathsf{CVer}(\mathsf{crs}, c, v, d)$ on input the CRS crs, a commitment c, a message $v \in \mathcal{V}$, and a decommitment d, outputs 1 if c indeed commits to v, and 0 otherwise.

– Simulation Setup: $(\mathsf{crs}, \tau) \leftarrow \mathsf{CSetup}_{\mathsf{sim}}(1^\lambda)$ on input the security parameter λ generates a simulation CRS crs and an associated trapdoor τ.
– Commitment Simulation: $(c, \mathsf{aux}) \leftarrow \mathsf{CSimCom}(\tau)$ on input a simulation trapdoor τ, generates a simulated commitment c and some auxiliary data aux.
– Opening Simulation: $d \leftarrow \mathsf{CSimOpen}(\tau, \mathsf{aux}, v)$ on input an auxiliary data aux and a message $v \in \mathcal{V}$, generates some decommitment d corresponding to an opening of the associated commitment c to v.

satisfying the following properties:

Perfect Correctness. For every security parameter $\lambda \in \mathbb{N}$, CRS $\mathsf{crs} \leftarrow \mathsf{CSetup}_{\mathsf{bind}}(1^\lambda)$ or $(\mathsf{crs}, \tau) \leftarrow \mathsf{CSetup}_{\mathsf{sim}}(1^\lambda)$, message $v \in \mathcal{V}$, and commitment $(c, d) \leftarrow \mathsf{CCom}(\mathsf{crs}, v)$, we have: $\mathsf{CVer}(\mathsf{crs}, c, v, d) = 1$.
Setup Indistinguishability. The following two distributions are computationally indistinguishable:

$$\left\{\mathsf{crs} \leftarrow \mathsf{CSetup}_{\mathsf{bind}}(1^\lambda) \ : \ \mathsf{crs}\right\}_\lambda \approx \left\{(\mathsf{crs}, \tau) \leftarrow \mathsf{CSetup}_{\mathsf{sim}}(1^\lambda) \ : \ \mathsf{crs}\right\}_\lambda .$$

Perfect Binding in Binding Mode. For every security parameter $\lambda \in \mathbb{N}$, binding CRS $\mathsf{crs} \leftarrow \mathsf{CSetup}_{\mathsf{bind}}(1^\lambda)$, message $v \in \mathcal{V}$, commitment $(c, d) \leftarrow \mathsf{CCom}(\mathsf{crs}, v)$, message $v' \in \mathcal{V}$, bitstring d', if $v' \neq v$: $\mathsf{CVer}(\mathsf{crs}, c, v', d') = 0$.
Perfect Equivocality in Simulation Mode. For every security parameter $\lambda \in \mathbb{N}$, simulation CRS $(\mathsf{crs}, \tau) \leftarrow \mathsf{CSetup}_{\mathsf{sim}}(1^\lambda)$, message $v \in \mathcal{V}$, the following two distributions are identical:

$$\{(c, d) \leftarrow \mathsf{CCom}(\mathsf{crs}, v) \ : \ (c, d)\} ,$$
$$\{(c, \mathsf{aux}) \leftarrow \mathsf{CSimCom}(\tau),\ d \leftarrow \mathsf{CSimOpen}(\tau, \mathsf{aux}, v) \ : \ (c, d)\} .$$

We are interested in proving statements "in zero-knowledge" of the form: "c commits to some value v such that $G(v) = y$," where G is a circuit in some circuit class $\mathcal{G}$ and y is the expected output of the function. In our construction, the trapdoor of the NIZK will actually be the trapdoor of the commitment. That is why we cannot easily rely on a generic definition of NIZK and instead introduce the notion of dual-mode NIZK of commitments. The binding setup yields perfectly sound NIZK proofs, while the simulation setup yields zero-knowledge proofs.

Definition 3.2 (Dual-Mode NIZK of Commitments). Let COM be as in Definition 3.1, and $\mathcal{G}$ be a class of polynomial-size circuits. A dual-mode NIZK NIZK associated with COM for $\mathcal{G}$ consists of two polynomial-time algorithms:

– Proof: $\pi \leftarrow \mathsf{CProve}(\mathsf{crs}, c, G, v, d)$ on input the CRS crs, a commitment c, a circuit $G \in \mathcal{G}$,[4] the committed message $v \in \mathcal{V}$, the decommitment d, as defined by COM, generates a proof π that G on input the value v committed in G outputs $y = G(v)$. Refer to (c, G, y) as the statement and (v, d) the witness.

[4] We implicitly systematically assume that G has input size corresponding to the size of messages in the message set $\mathcal{V}$.

– Proof Verification: $b := \mathsf{CPVer}(\mathsf{crs}, c, G, y, \pi)$ on input the CRS crs, a statement (c, G, y), and a proof π, accepts or rejects the proof.

The algorithms satisfy the following properties:

Perfect Proof Correctness. For every security parameter $\lambda \in \mathbb{N}$, CRS $\mathsf{crs} \leftarrow \mathsf{CSetup}_{\mathsf{bind}}(1^\lambda)$ or $(\mathsf{crs}, \tau) \leftarrow \mathsf{CSetup}_{\mathsf{sim}}(1^\lambda)$, message $v \in \mathcal{V}$, circuit $G \in \mathcal{G}$, commitment $(c, d) \leftarrow \mathsf{CCom}(\mathsf{crs}, v)$ and proof $\pi \leftarrow \mathsf{CProve}(\mathsf{crs}, c, G, v, d)$, we have: $\mathsf{CPVer}(\mathsf{crs}, c, G(v), \pi) = 1$.

Perfect Soundness in Binding Mode. For every security parameter $\lambda \in \mathbb{N}$, binding CRS $\mathsf{crs} \leftarrow \mathsf{CSetup}_{\mathsf{bind}}(1^\lambda)$, message $v \in \mathcal{V}$, commitment $(c, d) \leftarrow \mathsf{CCom}(\mathsf{crs}, v)$, circuit $G \in \mathcal{G}$, incorrect output $y' \neq G(v)$, and bitstring π, we have: $\mathsf{CPVer}(\mathsf{crs}, c, y', \pi) = 0$.

Zero-Knowledge in Simulation Mode. There exists a PPT simulator algorithm CPSim, such that for any PPT adversary $\mathcal{A}$, the quantity is negligible in λ:

$$\left| \Pr\left[\begin{array}{l} (\mathsf{crs}, \tau) \leftarrow \mathsf{CSetup}_{\mathsf{sim}}(1^\lambda),\ (\mathsf{st}, v) \leftarrow \mathcal{A}(\mathsf{crs}, \tau), \\ (c, \mathsf{aux}) \leftarrow \mathsf{CSimCom}(\tau),\ d \leftarrow \mathsf{CSimOpen}(\tau, \mathsf{aux}, v) \end{array} : \mathcal{A}^{\mathsf{Prove}}(\mathsf{st}) = 1 \right] \right.$$
$$\left. - \Pr\left[\begin{array}{l} (\mathsf{crs}, \tau) \leftarrow \mathsf{CSetup}_{\mathsf{sim}}(1^\lambda),\ (\mathsf{st}, v) \leftarrow \mathcal{A}(\mathsf{crs}, \tau), \\ (c, \mathsf{aux}) \leftarrow \mathsf{CSimCom}(\tau) \end{array} : \mathcal{A}^{\mathsf{Sim}}(\mathsf{st}) = 1 \right] \right|,$$

where $\mathsf{Prove}(G) = \mathsf{CProve}(\mathsf{crs}, c, G, v, d)$ and $\mathsf{Sim}(G) = \mathsf{CPSim}(\tau, \mathsf{aux}, G, G(v))$.

We remark that our notion of zero-knowledge allows the adversary to see the trapdoor τ but not the auxiliary data aux, that is why we let the adversary consider a single simulated commitment but as many simulated proofs as it wants. The reason that aux is not given to the adversary is because we need to store a PRF key in aux, to generate the randomness for simulation, to be sure to use the same randomness if the simulation is called twice with the same circuit G in the construction for P.

Definition 3.3 (Witness Encryption for NIZK of Commitments). Let COM, NIZK, and $\mathcal{G}$ be as in Definition 3.1 and 3.2. A Witness Encryption WE associated with $\mathsf{COM}, \mathsf{NIZK}$ for $\mathcal{G}$ consists of two polynomial-time algorithms:

– Witness Encryption: $\mathsf{ct} \leftarrow \mathsf{CWEnc}(\mathsf{crs}, c, G, y, \mathsf{m})$ on input the CRS crs, a statement (c, G, y) where $G \in \mathcal{G}$, and a bitstring m, encrypts m into a ciphertext ct, under that statement.
– Witness Decryption: $\mathsf{m} := \mathsf{CWDec}(\mathsf{crs}, \mathsf{ct}, c, G, y, \pi)$ on input the CRS crs, a ciphertext ct, a statement (c, G, y), and a NIZK proof π, decrypts ct into the message m, or outputs $\perp$.

The algorithms satisfy the following properties:

Perfect Encryption Correctness. For every $\lambda \in \mathbb{N}$, CRS $\mathsf{crs} \leftarrow \mathsf{CSetup}_{\mathsf{bind}}(1^\lambda)$ or $(\mathsf{crs}, \tau) \leftarrow \mathsf{CSetup}_{\mathsf{sim}}(1^\lambda)$, message $v \in \mathcal{V}$, circuit $G \in \mathcal{G}$, commitment $(c, d) \leftarrow \mathsf{CCom}(\mathsf{crs}, v)$ and proof $\pi \leftarrow \mathsf{CProve}(\mathsf{crs}, c, G, v, d)$, bitstring m, and ciphertext $\mathsf{ct} \leftarrow \mathsf{CWEnc}(\mathsf{crs}, c, G, G(v), \mathsf{m})$, we have: $\mathsf{CWDec}(\mathsf{crs}, \mathsf{ct}, c, G, G(v), \pi) = \mathsf{m}$.

Semantic Security. For any PPT adversary $\mathcal{A}$, the following is negligible in λ:

$$\left| 2 \cdot \Pr \left[\begin{array}{l} (\mathsf{st}, \rho') \leftarrow \mathcal{A}(1^\lambda),\ \mathsf{crs} \leftarrow \mathsf{CSetup}_{\mathsf{bind}}(1^\lambda; \rho'), \\ (\mathsf{st}, v, \rho, G, y, \mathsf{m}_0, \mathsf{m}_1) \leftarrow \mathcal{A}(\mathsf{st}, \mathsf{crs}), \\ (c, d) \coloneqq \mathsf{CCom}(\mathsf{crs}, v; \rho), \\ b \leftarrow \{0,1\};\ \mathsf{ct} \leftarrow \mathsf{CWEnc}(\mathsf{crs}, c, G, y, \mathsf{m}_b) \\ \mathsf{ct} \coloneqq \perp \ \text{ if } G(v) = y \end{array} : \mathcal{A}(\mathsf{st}, \mathsf{ct}) = b \right] - 1 \right| ,$$

where the randomness ρ used to commit to v is provided by the adversary.

We remark that semantic security of our WE holds even when the binding CRS is generated semi-maliciously, i.e., the adversary chooses the random tape ρ'. This is important for our semi-malicious construction of mrNISC schemes, as the adversary generates itself the binding CRS. We also note that our construction for NC^1 actually achieves perfect semantic security for binding CRS, however, our transformation from NC^1 to P only achieves computational semantic security.

3.2 Bilinear Commitments with Proofs of Quadratic Relations

As a tool to construct witness encryption for NIZK of commitments, we first introduce the notion of bilinear commitments with proofs of quadratic relations. Such commitments essentially allow to "prove linearly and in a strong form of zero-knowledge" that one commitment $c_\times$ commits to the product of the values committed by two commitments c_1 and c_2 (quadratic proofs), and that one commitment c_+ commits to some linear combination of the values committed by two commitments c_1 and c_2 (linear proofs). These proofs are amenable to be verified by hash proof systems and can be combined to construct WE for NIZK of commitments.

Bilinear Groups and Notations. Denote by $(p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_t, e, g_1, g_2)$ a *bilinear group* where $e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_t$ is an efficiently computable bilinear map (called a pairing) such that $e(g_1, g_2) = g_T$ generates $\mathbb{G}_t$. We use the bracket notation $[a]_\iota$ to denote the element g_ι^a in group $\mathbb{G}_\iota$ for $a \in \mathbb{Z}_p$ and write $a[a']_\iota = [aa']_\iota$ as applying group exponentiation in $\mathbb{G}_\iota$ and $[aa']_t = [a]_1[a']_2$ as applying the pairing operation. This notation extends to vector and matrices. We assume the *Symmetric External Diffie-Hellman assumption (*SXDH*)* assumption over asymmetric bilinear pairing groups, which requires that the Decisional Diffie-Hellman (DDH) assumption to hold in each source group $\mathbb{G}_1$ and $\mathbb{G}_2$, namely, for any $\iota \in \{1, 2\}$, $\{[r]_\iota,\ [s]_\iota,\ [rs]_\iota\} \approx \{[r]_\iota,\ [s]_\iota,\ [t]_\iota\}$, where r, s, t are random scalars sampled from $\mathbb{Z}_p$. All vectors are denoted by bold letters and all matrices are denoted by uppercase letters.

Bilinear Commitments. Our construction starts from the SXDH-based commitment scheme used in Groth-Sahai NIZK [33]. This commitment scheme allows to commit values both in $\mathbb{G}_1$ and $\mathbb{G}_2$. The resulting commitments are dual-mode and called type-1 and type-2 commitments respectively. More formally we define:

- Binding Setup: $\mathsf{crs} \leftarrow \mathsf{QSetup}_{\mathsf{bind}}(1^\lambda)$ generates a bilinear group $(p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_t, e, g_1, g_2)$, and for $\iota \in \{1,2\}$, generates a random matrix $A_\iota \in \mathbb{Z}_p^{2\times 2}$ *of rank 1* such that the vector $\mathbf{1} := (1,1)^T \in \mathbb{Z}_p^2$ is not in the column span of A_ι, and outputs $\mathsf{crs} = (p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_t, e, g_1, g_2, [A_1]_1, [A_2]_2)$.
- Simulation Setup: $(\mathsf{crs}, \tau) \leftarrow \mathsf{QSetup}_{\mathsf{sim}}(1^\lambda)$ is identical to binding setup except that A_1 and A_2 are chosen *of rank 2*. The trapdoor is $\tau = (A_1, A_2)$. Note that $\mathbf{1}$ is in the column spans of A_1 and A_2.
- Commitment: $(\boldsymbol{c}, \boldsymbol{d}) \leftarrow \mathsf{QCom}_\iota(\mathsf{crs}, v)$ generates a type-ι commitment of a message $v \in \mathcal{V} := \mathbb{Z}_p$ as follows:

$$\boldsymbol{d} \leftarrow \mathbb{Z}_p^2 \ , \qquad \boldsymbol{c} := [\tilde{\boldsymbol{c}}]_\iota := [A_\iota \cdot \boldsymbol{d} + v \cdot \mathbf{1}]_\iota \in \mathbb{G}_\iota^2 \ .$$

- Verification: $b := \mathsf{QVer}_\iota(\mathsf{crs}, \boldsymbol{c}, v, \boldsymbol{d})$ checks whether $\boldsymbol{c}$ is a valid type-ι commitment of v as follows: it returns 1 if and only if:

$$\boldsymbol{c} \stackrel{?}{=} [A_\iota \cdot \boldsymbol{d} + v \cdot \mathbf{1}]_\iota \ . \tag{5}$$

- Commitment Simulation: $(\boldsymbol{c}, \mathbf{aux}) \leftarrow \mathsf{QSimCom}_\iota(\tau)$ simulates a type-ι commitment as follows:

$$\mathbf{aux} \leftarrow \mathbb{Z}_p^2 \ , \qquad \boldsymbol{c} := [\mathbf{aux}]_\iota \in \mathbb{G}_\iota^2 \ . \tag{6}$$

- Opening Simulation: $\boldsymbol{d} \leftarrow \mathsf{QSimOpen}_\iota(\tau = (A_1, A_2), \mathbf{aux}, v)$ opens the type-ι commitment corresponding to $\mathbf{aux}$ as follows:

$$\boldsymbol{d} := A_\iota^{-1} \cdot (\mathbf{aux} - v \cdot \mathbf{1}) \in \mathbb{Z}_p^2 \ . \tag{7}$$

We have the following lemma following directly from [33].

Lemma 3.4 (in [33]). *The two commitment schemes* $(\mathsf{QSetup}_{\mathsf{bind}}, \mathsf{QCom}_\iota, \mathsf{QVer}_\iota, \mathsf{QSetup}_{\mathsf{sim}}, \mathsf{QSimCom}_\iota, \mathsf{QSimOpen}_\iota)$ *(for* $\iota \in \{1,2\}$*) described above are both dual-mode commitments.*

Remark 3.5. Jumping ahead, for semi-malicious security of mrNISC in the plain model, we want the binding of COM, soundness of NIZK, and semantic security of WE to hold against every CRS in the support of $\mathsf{QSetup}_{\mathsf{bind}}$. This boils down to ensuring that the bilinear group generated by $\mathsf{QSetup}_{\mathsf{bind}}$ is always a valid one: p must be a prime number, g_1, g_2 generates the cyclic groups $\mathbb{G}_1$ and $\mathbb{G}_2$ of order p, and it is possible to check in polynomial time whether an element is in $\mathbb{G}_1$ or $\mathbb{G}_2$. This can be done, and we implicitly assume that this is the case.

Bilinear Commitments with Proofs of Linear Relations. We now show how to prove that a type-2 commitment $\boldsymbol{c}_+$ commits to a given linear combination of values committed in two type-2 commitments $\boldsymbol{c}_1$ and $\boldsymbol{c}_2$. Concretely, we want to prove that $\boldsymbol{c}_1, \boldsymbol{c}_2, \boldsymbol{c}_+$ respectively commit to values v_1, v_2, v_+ that satisfy the linear relation: $v_+ = \mu_1 v_1 + \mu_2 v_2$, where $\mu_1, \mu_2 \in \mathbb{Z}_p$ are some public parameters.

$$\text{Statement: } (\mathsf{Linear}, \mathsf{crs}, \{\mu_i, \boldsymbol{c}_i\}_{i\in\{1,2\}}, \boldsymbol{c}_+), \qquad \text{Witness: } (v_1, \boldsymbol{d}_1, v_2, \boldsymbol{d}_2, \boldsymbol{d}_+)$$

The main idea of the construction is to remark that the commitments are linearly homomorphic and the above statement is equivalent to prove that $[\tilde{\boldsymbol{c}}_+ - \mu_1\tilde{\boldsymbol{c}}_1 - \mu_2\tilde{\boldsymbol{c}}_2]_2$ is a commitment of 0, where for $i \in \{1, 2, +\}$, $\boldsymbol{c}_i = [\tilde{\boldsymbol{c}}_i]_2$. Hence the proof $\boldsymbol{\pi}_+$ is the opening of this commitment to the value $v = 0$:

$$[\tilde{\boldsymbol{c}}_+ - \mu_1\tilde{\boldsymbol{c}}_1 - \mu_2\tilde{\boldsymbol{c}}_2]_2 = [A_2 \cdot \boldsymbol{\pi}_+ + 0 \cdot \mathbf{1}]_2 \ .$$

Zero-knowledge comes from the fact that this value $\boldsymbol{\pi}_+$ always exist and is unique in the simulation mode, as the matrix A_2 is full rank in that mode.

Formally, the construction is as follows:

- Linear Proof: $\mathsf{QLinProve}(\mathsf{crs}, \{\mu_i, \boldsymbol{c}_i, v_i, \boldsymbol{d}_i\}_{i\in[2]}, (\boldsymbol{c}_+, \boldsymbol{d}_+))$, given information of both statement and witness, outputs:

$$\boldsymbol{\pi}_+ \coloneqq \boldsymbol{d}_+ - \mu_1\boldsymbol{d}_1 - \mu_2\boldsymbol{d}_2 \in \mathbb{Z}_p^2 \ . \tag{8}$$

- Linear Proof Verification: $\mathsf{QLinVer}(\mathsf{crs}, \{\mu_i, \boldsymbol{c}_i\}_{i\in[2]}, \boldsymbol{c}_+, \boldsymbol{\pi}_+)$ returns 1 iff:

$$[\tilde{\boldsymbol{c}}_+ - \mu_1\tilde{\boldsymbol{c}}_1 - \mu_2\tilde{\boldsymbol{c}}_2]_2 \stackrel{?}{=} [A_2 \cdot \boldsymbol{\pi}_+]_2 \ , \tag{9}$$

  where $\boldsymbol{c}_i = [\tilde{\boldsymbol{c}}_i]_2$ for $i \in \{1, 2, +\}$.

Lemma 3.6. *For any security parameter $\lambda \in \mathbb{N}$, for any CRS $\mathsf{crs} \leftarrow \mathsf{QSetup}_{\mathsf{bind}}(1^\lambda)$ or $(\mathsf{crs}, \tau) \leftarrow \mathsf{QSetup}_{\mathsf{sim}}(1^\lambda)$, messages $v_1, v_2, v_+ \in \mathbb{Z}_p$, scalars $\mu_1, \mu_2, \mu_+ \in \mathbb{Z}_p$, bitstrings $\boldsymbol{c}_1, \boldsymbol{d}_1, \boldsymbol{c}_2, \boldsymbol{d}_2, \boldsymbol{c}_+, \boldsymbol{d}_+$ s.t. $\forall i \in \{1, 2, +\}, \mathsf{QVer}_2(\mathsf{crs}, \boldsymbol{c}_i, v_i, \boldsymbol{d}_i) = 1$,*

Perfect Correctness. If $v_+ = \mu_1 v_1 + \mu_2 v_2$, a proof $\pi_+ \leftarrow \mathsf{QLinProve}(\mathsf{crs}, \{\mu_i, \boldsymbol{c}_i, v_i, \boldsymbol{d}_i\}_i, (\boldsymbol{c}_+, \boldsymbol{d}_+))$ passes verification: $\mathsf{QLinVer}(\mathsf{crs}, \{\mu_i, \boldsymbol{c}_i\}_{i\in[2]}, \boldsymbol{c}_+, \boldsymbol{\pi}_+) = 1$

Perfect Uniqueness. If $v_+ = \mu_1 v_1 + \mu_2 v_2$ and the CRS is simulated, then there is a unique vector $\boldsymbol{\pi}_+ = (\tilde{\boldsymbol{c}}_+ - \mu_1\tilde{\boldsymbol{c}}_1 - \mu_2\tilde{\boldsymbol{c}}_2)A_2^{-1} \in \mathbb{Z}_p^2$ that passes verification.

Perfect Soundness. If $v_+ \neq \mu_1 v_1 + \mu_2 v_2$ and the CRS is binding, then no vector $\boldsymbol{\pi}_+ \in \mathbb{Z}_p^2$ passes verification: $\mathsf{QLinVer}(\mathsf{crs}, \{\mu_i, \boldsymbol{c}_i\}_{i\in[2]}, \boldsymbol{c}_+, \boldsymbol{\pi}_+) = 0$ for all $\boldsymbol{\pi}_+ \in \mathbb{Z}_p^2$.

Proof. **Perfect correctness** is straightforward. **Perfect uniqueness** follows from Eq. (9) and the fact that when the CRS is simulated, the matrix A_2 is full rank. **Perfect soundness** comes from the fact that:

$$[\mu_1\tilde{\boldsymbol{c}}_1 + \mu_2\tilde{\boldsymbol{c}}_2]_2 = [A_2 \cdot (\mu_1\boldsymbol{d}_1 + \mu_2\boldsymbol{d}_2) + (\mu_1 v_1 + \mu_2 v_2) \cdot \mathbf{1}]_2 \in \mathbb{G}_2^2$$

is a (perfectly binding) commitment of $\mu_1 v_1 + \mu_2 v_2 \neq v_+$. □

Remark 3.7 (Zero-knowledge of the linear proof $\boldsymbol{\pi}_+$ in simulation mode). Perfect uniqueness of the proof $\boldsymbol{\pi}_+$ in simulation mode is a very strong form of witness indistinguishability: whatever witness $(v_1, \boldsymbol{d}_1, v_2, \boldsymbol{d}_2, \boldsymbol{d}_+)$ is used, the proof is exactly the same $\boldsymbol{\pi}_+ = (\tilde{\boldsymbol{c}}_+ - \mu_1\tilde{\boldsymbol{c}}_1 - \mu_2\tilde{\boldsymbol{c}}_2)A_2^{-1}$. To show further that it is ZK, we need to argue that $\boldsymbol{\pi}_+$ is also efficiently computable. This the case when the commitments $\boldsymbol{c}_i = [\tilde{\boldsymbol{c}}_i]_2$ are simulated with $\mathsf{QSimCom}$, as the simulator can then equivocate $\boldsymbol{c}_1, \boldsymbol{c}_2, \boldsymbol{c}_+$ to any v_1', v_2', v_+' satisfying $v_+' = \mu_1 v_1' + \mu_2 v_2'$ with decommitments $\boldsymbol{d}_1', \boldsymbol{d}_2', \boldsymbol{d}_+'$ using $\mathsf{QSimOpen}$. This gives a valid witness $(v_1', \boldsymbol{d}_1', v_2', \boldsymbol{d}_2', \boldsymbol{d}_+')$ for the statement and a simulated proof can be generated by running the honest prover algorithm $\mathsf{QLinProve}$ with this witness.

Bilinear Commitments with Proofs of Quadratic Relations. We now show how to prove that a **type-2** commitment $\boldsymbol{c}_\times$ commits to the product of values committed in a **type-1** commitment $\boldsymbol{c}_1$ and a **type-2** commitments $\boldsymbol{c}_2$. Concretely, we want to prove that $\boldsymbol{c}_1, \boldsymbol{c}_2, \boldsymbol{c}_+$ respectively commit to values $v_1, v_2, v_\times$ that satisfy the quadratic relation $v_\times = v_1 \cdot v_2$.

$$\text{Statement: } (\mathsf{Mult}, \mathsf{crs}, \{\boldsymbol{c}_i\}_{i\in\{1,2,\times\}}), \qquad \text{Witness: } (v_1, \boldsymbol{d}_1, v_2, \boldsymbol{d}_2, \boldsymbol{d}_\times) \tag{10}$$

The main idea of the construction is to construct from $\boldsymbol{c}_1 = [\tilde{\boldsymbol{c}}_1]_1$ and $\boldsymbol{c}_2 = [\tilde{\boldsymbol{c}}_2]_2$ a commitment of $v_1 \cdot v_2$. Remember that in the technical overview Section 2.3, we could multiply commitments $\boldsymbol{c}_1$ and $\boldsymbol{c}_2$ directly (by using a pairing operation) to get a commitment of $v_1 \cdot v_2$, as commitments were a single group element. Intuitively, the equivalent of this multiplication to vector of group elements $\boldsymbol{c}_1$ and $\boldsymbol{c}_2$ is the tensor product operation $\otimes$. And we want to prove that $[\mathbf{1} \otimes \tilde{\boldsymbol{c}}_\times - \tilde{\boldsymbol{c}}_1 \otimes \tilde{\boldsymbol{c}}_2]_t$ is a "commitment" of 0 in $\mathbb{G}_t$, where $\mathbf{1}$ is used as a type-1 commitment of 1.[5] Similar as multiplication of commitments in Section 2.3, computing these tensor products uses pairings.

The basic idea is then that the proof is a decommitment of this commitment $[\mathbf{1} \otimes \tilde{\boldsymbol{c}}_\times - \tilde{\boldsymbol{c}}_1 \otimes \tilde{\boldsymbol{c}}_2]_t$ to 0. Unfortunately, this would not be zero-knowledge since there are multiple possible decommitments and choosing one may reveal information about the witness $(v_1, \boldsymbol{d}_1, v_2, \boldsymbol{d}_2, \boldsymbol{d}_\times)$. To tackle this subtle issue (which does not happen with the commitments from the technical overview in Section 2.3 nor with proof of linear relations), the prover needs to rerandomize this decommitment, similarly to what is done in [33] to get perfect witness indistinguishability. This is the purpose of the vector $\boldsymbol{\rho}$ in Eq. (12).

Tensor Products. We first need to briefly recall the notion of tensor products. The tensor product of two matrices $M \in \mathbb{Z}_p^{k\times m}$ and $M' \in \mathbb{Z}_p^{k'\times m'}$ is the matrix $T = M \otimes M' \in \mathbb{Z}_p^{kk'\times mm'}$ defined as:

$$T = \begin{pmatrix} M_{1,1} \cdot M' & \cdots & M_{1,m} \cdot M' \\ \vdots & & \vdots \\ M_{k,1} \cdot M' & \cdots & M_{k,m} \cdot M' \end{pmatrix} .$$

We extensively use the following identity: if $M \in \mathbb{Z}_p^{k\times m}$, $M' \in \mathbb{Z}_p^{k'\times m'}$, $N \in \mathbb{Z}_p^{m\times n}$ and $N' \in \mathbb{Z}_p^{m'\times n'}$, then we have,

$$(M \otimes M') \cdot (N \otimes N') = (M \cdot N) \otimes (M' \cdot N') \ . \tag{11}$$

Construction. Recall that the construction essentially consists in proving that $[\mathbf{1} \otimes \tilde{\boldsymbol{c}}_\times - \tilde{\boldsymbol{c}}_1 \otimes \tilde{\boldsymbol{c}}_2]_t$ is a commitment of 0, which is what Eq. (13) below ensures.

[5] $[\mathbf{1} \otimes \tilde{\boldsymbol{c}}_\times - \tilde{\boldsymbol{c}}_1 \otimes \tilde{\boldsymbol{c}}_2]_t$ is not a type-1 commitment (using the matrix A_1) nor a type-2 commitment (using the matrix A_2) but yet another type of commitment using another matrix B (formally defined in the proof in Eq. (15)). When the CRS is binding, this matrix B is such that the commitment is also binding.

To better understand how this value is computed (in term of group elements, pairings, and exponentiations), we explicitely write it down:

$$[\mathbf{1} \otimes \tilde{\boldsymbol{c}}_\times - \tilde{\boldsymbol{c}}_1 \otimes \tilde{\boldsymbol{c}}_2]_t = \begin{pmatrix} e(g_1, c_{\times,1}) \cdot e(c_{1,1}, c_{2,1})^{-1} \\ e(g_1, c_{\times,1}) \cdot e(c_{1,1}, c_{2,2})^{-1} \\ e(g_1, c_{\times,2}) \cdot e(c_{1,2}, c_{2,1})^{-1} \\ e(g_1, c_{\times,2}) \cdot e(c_{1,2}, c_{2,2})^{-1} \end{pmatrix} \qquad \text{where } \boldsymbol{c}_i = \begin{pmatrix} c_{i,1} \\ c_{i,2} \end{pmatrix}$$

The construction is as follows:

- Quadratic Proof: $\boldsymbol{\pi}_\times \leftarrow \mathsf{QQuadProve}(\mathsf{crs}, \{\boldsymbol{c}_i, v_i, \boldsymbol{d}_i\}_{i\in[2]}, \boldsymbol{c}_\times, \boldsymbol{d}_\times)$ picks $\boldsymbol{\rho} \in \mathbb{Z}_p^4$ and outputs:

$$\boldsymbol{\pi}_\times := \begin{pmatrix} [\tilde{\boldsymbol{\pi}}_\times^\top]_2 \\ [\tilde{\boldsymbol{\pi}}_\times^\perp]_1 \end{pmatrix} = \begin{pmatrix} [-v_2 \cdot \boldsymbol{d}_1 \otimes \mathbf{1} \ + \ (\mathsf{Id} \otimes A_2) \cdot \boldsymbol{\rho}]_2 \\ [\mathbf{1} \otimes \boldsymbol{d}_\times \ - \ \tilde{\boldsymbol{c}}_1 \otimes \boldsymbol{d}_2 \ - \ (A_1 \otimes \mathsf{Id}) \cdot \boldsymbol{\rho}]_1 \end{pmatrix} , \tag{12}$$

  where $\mathsf{Id} \in \mathbb{Z}_p^{2\times 2}$ is the identity matrix. Recall that the vector $\boldsymbol{\rho}$ is used to randomize the proof so that it is uniformly random amongst the valid proofs, and hence is perfectly witness indistinguishable.
- Quadratic Proof Verification: $b := \mathsf{QQuadVer}(\mathsf{crs}, \boldsymbol{c}_1, \boldsymbol{c}_2, \boldsymbol{c}_\times, \boldsymbol{\pi}_\times)$ returns 1 if and only if:

$$[\mathbf{1} \otimes \tilde{\boldsymbol{c}}_\times - \tilde{\boldsymbol{c}}_1 \otimes \tilde{\boldsymbol{c}}_2]_t = \begin{pmatrix} [A_1 \otimes \mathsf{Id}]_1 & [\mathsf{Id} \otimes A_2]_2 \end{pmatrix} \cdot \boldsymbol{\pi}_\times \ , \tag{13}$$

  where $\mathsf{Id} \in \mathbb{Z}_p^{2\times 2}$ is the identity matrix. Note that computing $[\tilde{\boldsymbol{c}}_1 \otimes \tilde{\boldsymbol{c}}_2]_t$ involves pairing operations between elements of vectors $\boldsymbol{c}_1 \in \mathbb{G}_1^2$ and $\boldsymbol{c}_2 \in \mathbb{G}_2^2$. Computing the right hand side also involves pairing operations.

Remark 3.8. Quadratic proof verification just consists in checking a linear equation in $(\boldsymbol{c}_2, \boldsymbol{c}_\times, \boldsymbol{\pi}_\times)$. Indeed, thanks to Eq. (11), Eq. (13) is equivalent to:

$$0 = \begin{pmatrix} [\mathbf{1} \otimes \mathsf{Id}]_1 & [-\tilde{\boldsymbol{c}}_1 \otimes \mathsf{Id}]_1 & [A_1 \otimes \mathsf{Id}]_1 & [\mathsf{Id} \otimes A_2]_2 \end{pmatrix} \cdot \begin{pmatrix} [\tilde{\boldsymbol{c}}_\times]_2 \\ [\tilde{\boldsymbol{c}}_2]_2 \\ [\tilde{\boldsymbol{\pi}}_\times^\top]_2 \\ [\tilde{\boldsymbol{\pi}}_\times^\perp]_1 \end{pmatrix} .$$

Lemma 3.9. *For any security parameter $\lambda \in \mathbb{N}$, for any CRS $\mathsf{crs} \leftarrow \mathsf{QSetup}_{\mathsf{bind}}(1^\lambda)$ or $(\mathsf{crs}, \tau) \leftarrow \mathsf{QSetup}_{\mathsf{sim}}(1^\lambda)$, messages $v_1, v_2, v_\times \in \mathbb{Z}_p$, bitstrings $\boldsymbol{c}_1, \boldsymbol{d}_1, \boldsymbol{c}_2, \boldsymbol{d}_2, \boldsymbol{c}_\times, \boldsymbol{d}_\times$ such that $\forall i \in \{1, 2, \times\}, \mathsf{QVer}_i(\mathsf{crs}, \boldsymbol{c}_i, v_i, \boldsymbol{d}_i) = 1$, we have:*

Perfect Correctness. If $v_\times = v_1 v_2$, a proof $\mathsf{QQuadProve}(\mathsf{crs}, \{\boldsymbol{c}_i, v_i, \boldsymbol{d}_i\}_{i\in[2]}, (\boldsymbol{c}_\times, \boldsymbol{d}_\times))$ passes verification: $\mathsf{QQuadVer}(\mathsf{crs}, \{\mu_i, \boldsymbol{c}_i\}_{i\in[2]}, \boldsymbol{c}_\times, \boldsymbol{\pi}_\times) = 1$

Perfect Uniformity. If $v_\times = v_1 v_2$ and the CRS is simulated, then the vector $\boldsymbol{\pi}_\times$ generated by $\mathsf{QQuadProve}$ follows a uniform distribution amongst the solutions of Eq. (13).

Perfect Soundness. If $v_\times \neq v_1 v_2$ and the CRS is binding, then no $\boldsymbol{\pi}_\times \in \mathbb{Z}_p^8$ passes verification: $\mathsf{QQuadVer}(\mathsf{crs}, \boldsymbol{c}_1, \boldsymbol{c}_2, \boldsymbol{c}_\times, \boldsymbol{\pi}_\times) = 0$ for all $\boldsymbol{\pi}_\times \in \mathbb{Z}_p^8$.

Proof. To prove **perfect correctness**, we use Eqs. (11) and (12) and remark:

$$\begin{aligned}
&\mathbf{1} \otimes \tilde{\boldsymbol{c}}_\times - \tilde{\boldsymbol{c}}_1 \otimes \tilde{\boldsymbol{c}}_2 = \mathbf{1} \otimes (A_2 \boldsymbol{d}_\times + v_\times \cdot \mathbf{1}) - \tilde{\boldsymbol{c}}_1 \otimes (A_2 \boldsymbol{d}_2 + v_2 \cdot \mathbf{1}) \\
&= \mathbf{1} \otimes (A_2 \boldsymbol{d}_\times) + v_\times \cdot \mathbf{1} \otimes \mathbf{1} - \tilde{\boldsymbol{c}}_1 \otimes (A_2 \boldsymbol{d}_2) - (A_1 \boldsymbol{d}_1 + v_1 \cdot \mathbf{1}) \otimes (v_2 \cdot \mathbf{1}) \\
&= \mathbf{1} \otimes (A_2 \boldsymbol{d}_\times) - \tilde{\boldsymbol{c}}_1 \otimes (A_2 \boldsymbol{d}_2) - (A_1 \boldsymbol{d}_1) \otimes (v_2 \cdot \mathbf{1}) + (v_\times - v_1 v_2) \cdot (\mathbf{1} \otimes \mathbf{1}) \\
&= (\mathsf{Id} \otimes A_2) \cdot (\mathbf{1} \otimes \boldsymbol{d}_\times) - (\mathsf{Id} \otimes A_2) \cdot (\tilde{\boldsymbol{c}}_1 \otimes \boldsymbol{d}_2) \\
&\qquad - (A_1 \otimes \mathsf{Id}) \cdot (v_2 \boldsymbol{d}_1 \otimes \mathbf{1}) + (v_\times - v_1 v_2) \cdot (\mathbf{1} \otimes \mathbf{1}) \ . \qquad (14)
\end{aligned}$$

We conclude by remarking that $v_\times = v_1 v_2$ and that:

$$\begin{pmatrix} A_1 \otimes \mathsf{Id} & \mathsf{Id} \otimes A_2 \end{pmatrix} \cdot \begin{pmatrix} (\mathsf{Id} \otimes A_2) \cdot \boldsymbol{\rho} \\ -(A_1 \otimes \mathsf{Id}) \cdot \boldsymbol{\rho} \end{pmatrix} = 0 \ .$$

Perfect soundness follows from Eq. (14) and the fact that $\mathbf{1} \otimes \mathbf{1}$ is not in the subspace generated by the columns of the matrix

$$B \coloneqq \begin{pmatrix} A_1 \otimes \mathsf{Id} & \mathsf{Id} \otimes A_2 \end{pmatrix} \in \mathbb{Z}_p^{4 \times 8} \ , \qquad (15)$$

when the CRS is binding, because if $\boldsymbol{a}_1, \boldsymbol{a}_2 \in \mathbb{Z}_p^2$ are two vectors generating the column space of A_1 and A_2 respectively, then $(\boldsymbol{a}_1 \otimes \boldsymbol{a}_2,\ \boldsymbol{a}_1 \otimes \mathbf{1},\ \mathbf{1} \otimes \boldsymbol{a}_2,\ \mathbf{1} \otimes \mathbf{1})$ is a basis of $\mathbb{Z}_p^4$.

Finally, **perfect uniformity** comes from the fact that the kernel of the matrix B (from Eq. (15)) consists of all the vectors:

$$\begin{pmatrix} (\mathsf{Id} \otimes A_2) \cdot \boldsymbol{\rho} \\ -(A_1 \otimes \mathsf{Id}) \cdot \boldsymbol{\rho} \end{pmatrix} \ ,$$

for $\boldsymbol{\rho} \in \mathbb{Z}_p^4$, since these elements are clearly in the kernel and form a subspace of dimension 4, and the kernel is of dimension 4 as $B \in \mathbb{Z}_p^{8 \times 4}$ is of rank 4 (because A_1 is of full rank and hence $A_1 \otimes \mathsf{Id} \in \mathbb{Z}_p^{4 \times 4}$ is of full rank). □

Remark 3.10 (Zero-knowledge of the quadratic proof $\boldsymbol{\pi}_\times$ in simulation mode). Perfect uniformity in the simulation mode is a very strong form of witness indistinguishability: whatever witness is used, the proof follows exactly the same uniform distribution over solutions of Equation 13. To show that $\boldsymbol{\pi}_\times$ is zero-knowledge, it remains to argue that this distribution can be efficiently sampled. This can be done similarly as in Remark 3.7: for simulated commitments $\boldsymbol{c}_i$, the simulator can equivocate $\boldsymbol{c}_1, \boldsymbol{c}_2, \boldsymbol{c}_\times$ to any $v_1', v_2', v_\times'$ satisfying $v_\times' = v_1' v_2'$ with decommitment $\boldsymbol{d}_1', \boldsymbol{d}_2', \boldsymbol{d}_\times'$ using $\mathsf{QSimOpen}$. This gives a valid witness $(v_1', \boldsymbol{d}_1', v_2', \boldsymbol{d}_2', \boldsymbol{d}_\times')$ for the statement and a simulated proof can be generated by running the honest prover algorithm $\mathsf{QQuadProve}$ with this witness.

3.3 WE for NIZK of Commitments for $\mathbf{NC^1}$

We now describe our construction of WE for NIZK of commitments for NC^1. It follows the technical overview Section 2.3. The idea is to represent the function by a Restricted Multiplication Straight-line (RMS) Program [13, 22], which only

performs multiplications or quadratic operations between an intermediate variable and an input. We start with defining a variant of RMS where operations are done modulo some prime number p.

Definition 3.11 (RMS Programs). Let p be a prime. A Restricted Multiplication Straight-line (RMS) program modulo p with input $v = v_1 \| \cdots \| v_n \in \{0,1\}^n$ and output $y = y_1 \| \cdots \| y_m \in \{0,1\}^m$ is a sequence of the following instructions:

- Load a constant $\omega \in \mathbb{Z}_p$ into the memory value u_j: $(u_j \leftarrow \omega)$.
- Linearly combine memory values u_i and u_j into the memory value u_k: $(u_k \leftarrow \mu u_i + \mu' u_j \bmod p)$, with $(\mu, \mu') \in \mathbb{Z}_p^2 \setminus \{(0,0)\}$ a non-zero pair of constants.
- Multiply the input value v_i by the memory value u_j into the memory value u_k: $(u_k \leftarrow v_i \cdot u_j \bmod p)$.

where each memory value is written at most once and each memory value that is read was written before. The program aborts if one memory value u_k is not in $\{0,1\}$. If it does not abort, it outputs $y = y_1 \| \cdots \| y_m = u_1 \| \ldots \| u_m$.

The *size* of an RMS is the number of instructions. Furthermore, any NC^1 circuit G can be written as an RMS program of polynomial size, because deterministic branching programs can be encoded into RMS with constant overhead [13, Claim A.2]. The resulting RMS program outputs the correct value when evaluated modulo any prime number p, as when evaluated without modulo, all the memory values are in $\{0,1\}$.

Construction. Let $\mathsf{QC} = (\mathsf{QSetup}_{\mathsf{bind}}, \mathsf{QSetup}_{\mathsf{sim}}, \{\mathsf{QCom}_i, \mathsf{QVer}_i, \mathsf{QSimCom}_i, \mathsf{QSimOpen}_i\}_{i\in\{1,2\}}, \mathsf{QQuadProve}, \mathsf{QQuadVer})$ be the bilinear commitment scheme with proofs of quadratic relations from the previous section. We construct a witness encryption WE for NIZK of commitments for NC^1 below. To help differentiate type-1 and type-2 commitments, all type-ι commitments have subscript starting with ι, such as, $\boldsymbol{c}_{\iota,k}$.

- Commitment: $(c, d) \leftarrow \mathsf{CCom}(\mathsf{crs}, v)$ for $v \in \mathcal{V} := \{0,1\}^n$, generates type-1 commitments for each bit of $v = v_1 \| \ldots \| v_n$. More formally, $c = (\boldsymbol{c}_{1,1}, \ldots, \boldsymbol{c}_{1,n}$ and $d = (\boldsymbol{d}_{1,1}, \ldots, \boldsymbol{d}_{1,n})$, where for $i \in [n]$, $(\boldsymbol{c}_{1,i}, \boldsymbol{d}_{1,i}) \leftarrow \mathsf{QCom}_1(\mathsf{crs}, v_i)$.
- Verification, Commitment Simulation and Opening: just consist in running the respective algorithms $\mathsf{QVer}_1, \mathsf{QSimCom}_1, \mathsf{QSimOpen}_1$ in parallel for each commitment $\boldsymbol{c}_{1,i}$.
- Proof: $\pi \leftarrow \mathsf{CProve}(\mathsf{crs}, c, G, v, d)$, for an NC^1 circuit G represented as an RMS program with n-bit input and m-bit output works as follows. Let S_ω, S_+, and $S_\times$ be the sets of memory indexes written by constant loading, linear, and multiplication instructions respectively. We suppose that the used memory values are $u_1, \ldots, u_L$. The proof π is a tuple $(\{\boldsymbol{c}_{2,k}\}_{k\in[L]}, \{\boldsymbol{d}_{2,k}\}_{k\in[m]\cup S_\omega}, \{\boldsymbol{\pi}_k\}_{k\in S_+\cup S_\times})$ where these values are generated as follows, for each instruction
  - $(u_k \leftarrow \omega)$: generate $(\boldsymbol{c}_{2,k}, \boldsymbol{d}_{2,k}) \leftarrow \mathsf{QCom}_2(\mathsf{crs}, \omega)$.

- $(u_k \leftarrow \mu u_i + \mu' u_j \bmod p)$: compute

$$(\boldsymbol{c}_{2,k}, \boldsymbol{d}_{2,k}) \leftarrow \mathsf{QCom}_2(\mathsf{crs}, \mu u_i + \mu' u_j) \ ,$$
$$\boldsymbol{\pi}_k \coloneqq \mathsf{QLinProve}(\mathsf{crs}, (\mu, \boldsymbol{c}_{2,i}, u_i, \boldsymbol{d}_{2,i}), (\mu', \boldsymbol{c}_{2,j}, u_j, \boldsymbol{d}_{2,j}), (\boldsymbol{c}_{2,k}, \boldsymbol{d}_{2,k})) \ .$$

- $(u_k \leftarrow v_i \cdot u_j \bmod p)$: compute

$$(\boldsymbol{c}_{2,k}, \boldsymbol{d}_{2,k}) \leftarrow \mathsf{QCom}_2(\mathsf{crs}, v_i \cdot u_j) \ ,$$
$$\boldsymbol{\pi}_k \coloneqq \mathsf{QQuadProve}(\mathsf{crs}, (\boldsymbol{c}_{1,i}, v_i, \boldsymbol{d}_{1,i}), (\boldsymbol{c}_{2,j}, u_j, \boldsymbol{d}_{2,j}), (\boldsymbol{c}_{2,k}, \boldsymbol{d}_{2,k})) \ .$$

(Note that values v_i and u_j are known by the prover.)

– Proof Verification: just consists in verifying the provided openings and quadratic proofs. More formally, $\mathsf{CPVer}(\mathsf{crs}, c, G, y, \pi)$ where $y = y_1 \| \cdots \| y_m$ returns 1 if and only if all the following tests pass:
  - For every $i \in [m]$, check that $\mathsf{QVer}_2(\mathsf{crs}, \boldsymbol{c}_{2,i}, y_i, \boldsymbol{d}_{2,i}) \stackrel{?}{=} 1$.
  - For every instruction:
  * $(u_k \leftarrow \omega)$: check $\mathsf{QVer}_2(\mathsf{crs}, \boldsymbol{c}_{2,k}, \omega, \boldsymbol{d}_{2,k}) \stackrel{?}{=} 1$.
  * $(u_k \leftarrow \mu u_i + \mu' u_j \bmod p)$: check $\mathsf{QLinVer}(\mathsf{crs}, (\mu, \boldsymbol{c}_{2,i}), (\mu', \boldsymbol{c}_{2,j}), \boldsymbol{c}_{2,k}, \boldsymbol{\pi}_k) \stackrel{?}{=} 1$.
  * $(u_k \leftarrow v_i \cdot u_j \bmod p)$: check $\mathsf{QQuadVer}(\mathsf{crs}, \boldsymbol{c}_{1,i}, \boldsymbol{c}_{2,j}, \boldsymbol{c}_{2,k}, \boldsymbol{\pi}_k) \stackrel{?}{=} 1$.
– Proof Simulation: $\pi \leftarrow \mathsf{CPSim}(\tau, \mathsf{aux}, c, G, y)$ where $c = (\boldsymbol{c}_{1,1}, \ldots, \boldsymbol{c}_{1,n})$ are simulated with auxiliary data $\mathsf{aux} = (\mathbf{aux}_{1,1}, \ldots, \mathbf{aux}_{1,n})$, simulates a proof $\pi = (\{\boldsymbol{c}_{2,k}\}_{k\in[L]}, \{\boldsymbol{d}_{2,k}\}_{k\in[m]\cup S_\omega}, \{\boldsymbol{\pi}_k\}_{k\in S_+\cup S_\times})$ as follows: Run through the instructions in RMS in order and for each instruction do:
  - $(u_k \leftarrow \omega)$: generate

$$(\boldsymbol{c}_{2,k}, \mathbf{aux}_{2,k}) \leftarrow \mathsf{QSimCom}_2(\tau) \ , \quad \boldsymbol{d}_{2,k} \leftarrow \mathsf{QSimOpen}_2(\tau, \mathbf{aux}_{2,k}, \omega) \ .$$

  - $(u_k \leftarrow \mu u_i + \mu' u_j \bmod p)$: set $u'_k \coloneqq y_k$ if $k \in [m]$ or 0 otherwise, and let $u'_i, u'_j \in \mathbb{Z}_p$ be arbitrary scalars such that $\mu u'_i + \mu' u'_j = u'_k$ (which is possible as $(\mu, \mu') \neq 0$), and compute:

$$(\boldsymbol{c}_{2,k}, \mathbf{aux}_{2,k}) \leftarrow \mathsf{QSimCom}_2(\tau) \ ,$$
$$\boldsymbol{d}'_{2,\ell} \leftarrow \mathsf{QSimOpen}_2(\tau, \mathbf{aux}_{2,\ell}, u'_\ell) \quad \text{for } \ell \in \{i, j, k\} \ , \tag{16}$$
$$\boldsymbol{\pi}_k \coloneqq \mathsf{QLinProve}(\mathsf{crs}, (\mu, \boldsymbol{c}_{2,i}, u'_i, \boldsymbol{d}'_{2,i}), (\mu', \boldsymbol{c}_{2,j}, u'_j, \boldsymbol{d}'_{2,j}), (\boldsymbol{c}_{2,k}, \boldsymbol{d}'_{2,k})) \ .$$

    Note: values u'_i, u'_j, u'_k are local and may be different for different instructions.
  - $(u_k \leftarrow v_i \cdot u_j \bmod p)$: set $u'_k \coloneqq y_k$ is $k \in [m]$ or 0 otherwise, as well as $u'_i \coloneqq 1$ and $u'_j \coloneqq u'_k$ (so that $u'_k = u'_i u'_j$ — again values u'_i, u'_j, u'_k are local) and compute:

$$(\boldsymbol{c}_{2,k}, \mathbf{aux}_{2,k}) \leftarrow \mathsf{QSimCom}_2(\tau) \ ,$$
$$\boldsymbol{d}'_{1,i} \leftarrow \mathsf{QSimOpen}_1(\tau, \mathbf{aux}_{1,i}, u'_i) \tag{17}$$
$$\boldsymbol{d}'_{2,\ell} \leftarrow \mathsf{QSimOpen}_2(\tau, \mathbf{aux}_{2,\ell}, u'_\ell) \quad \text{for } \ell \in \{j, k\} \ , \tag{18}$$
$$\boldsymbol{\pi}_k \coloneqq \mathsf{QQuadProve}(\mathsf{crs}, (\boldsymbol{c}_{1,i}, u'_i, \boldsymbol{d}'_{1,i}), (\boldsymbol{c}_{2,i}, u'_j, \boldsymbol{d}'_{2,j}), (\boldsymbol{c}_{2,k}, \boldsymbol{d}'_{2,k})) \ .$$

– Witness Encryption: Looking at Eqs. (5) and (9) and Remark 3.8, we remark that the proof verification $\mathsf{CPVer}(\mathsf{crs}, c, G, y, \pi)$ is affine in the vector π. Concretely, there exists a matrix $[\Gamma_{\mathsf{crs},c,G,y}]_\star$ and a vector $[\boldsymbol{\theta}_{\mathsf{crs},c,G,y}]_\star$ (both only depend of crs, c, G, y and can be efficiently computed from these three values — the start $\star$ denotes the fact that elements are not necessarily in the same group), such that, seeing π as a vector of elements in $\mathbb{Z}_p, \mathbb{G}_1, \mathbb{G}_2$ of length β, and denoting by $\tilde{\boldsymbol{\pi}} \in \mathbb{Z}_p^\beta$ the vector derived from π by replacing every $\mathbb{G}_\iota$ element with its discrete logarithm, we have:

$$[\boldsymbol{\theta}_{\mathsf{crs},c,G,y}]_t = [\Gamma_{\mathsf{crs},c,G,y} \cdot \tilde{\boldsymbol{\pi}}]_t \ .$$

(Note: This is because: By Equation 5 and 9, verification of opening and verification of a linear proof are both linear equations whose coefficients are either constants or elements in crs*. By remark 3.8, verification of a quadratic proof is a linear equation whose coefficients are constants, or elements in* crs*, or commitments* $\boldsymbol{c}_{1,i}$ *(as in Equation 17) to the first operand in the multiplication. Since in RMS the first operand of multiplication is always an input bit,* $\boldsymbol{c}_{1,i}$ *is contained in* c*.)*

The witness encryption then just uses hash proof systems from [1]. More formally, to encrypt a bit message $\mathsf{m} \in \{0,1\}$, $\mathsf{CWEnc}(\mathsf{crs}, c, G, y, \mathsf{m})$ picks a uniformly random row vector $\boldsymbol{\alpha} \in \mathbb{Z}_p^{1\times\nu}$, where ν is the number of rows of $\Gamma_{\mathsf{crs},c,G,y}$, and outputs the ciphertext $\mathsf{ct} = ([\boldsymbol{\gamma}]_\star, [\delta]_t)$ where:

$$[\boldsymbol{\gamma}]_\star \coloneqq [\boldsymbol{\alpha} \cdot \Gamma_{\mathsf{crs},c,G,y}]_\star \ , \qquad [\delta]_t \coloneqq [\boldsymbol{\alpha} \cdot \boldsymbol{\theta}_{\mathsf{crs},c,G,y} + \mathsf{m}]_t \ .$$

– Witness Decryption: Using the notations from witness encryption, $\mathsf{CWDec}(\mathsf{crs}, \mathsf{ct}, c, G, y, \pi)$ outputs $\mathsf{m} \in \{0,1\}$ satisfying

$$[\mathsf{m}]_t = [\delta - \boldsymbol{\gamma} \cdot \tilde{\boldsymbol{\pi}}]_t \ .$$

Efficiency: The algorithms $\mathsf{CSetup}_{\mathsf{bind}}, \mathsf{CCom}, \mathsf{CVer}$ (as well as the simulators $\mathsf{CSetup}_{\mathsf{sim}}, \mathsf{CSimCom}, \mathsf{CSimOpen}$) of the resulting WE for NIZK of commitments run in time polynomial in their inputs. The algorithms $\mathsf{CProve}, \mathsf{CPVer}, \mathsf{CWEnc}, \mathsf{CWDec}$ run in time polynomial in their inputs and *exponential in the depth of the circuit* G. This exponential blow up is due to the representation by a RMS program and explains the restriction to NC^1.

Theorem 3.12. *Assuming SXDH over bilinear groups. The construction* Π *described above is a WE for NIZK of commitments for* NC^1*.*

Proof. **Perfect correctness** of the commitment, **setup indistinguishability**, **perfect binding**, and **perfect equivocality** follow directly from the fact that $(\mathsf{QSetup}_{\mathsf{bind}}, \mathsf{QSetup}_{\mathsf{sim}}, \mathsf{QCom}_1, \mathsf{QVer}_1, \mathsf{QSimCom}_1, \mathsf{QSimOpen}_1)$ is a dual-mode commitment scheme. **Perfect proof correctness** follows from perfect correctness of linear and quadratic proofs. **Perfect soundness** follows from perfect binding of type-1 and type-2 commitments as well as perfect soundness of linear and quadratic proofs. **Perfect encryption correctness** and **perfect semantic**

security follow immediately from correctness and smoothness of the hash proof systems in [1]. It remains to prove the **perfect zero-knowledge property**. This is where the uniqueness of linear proofs (Remark 3.7) and the perfect uniformity (Remark 3.10) of the quadratic proofs are used. We give a proof by games:

- Game 0 corresponds to the zero-knowledge game where proofs are honestly generated.
- Game 1 is similar to Game 0 except that all the commitments are simulated but still opened to the value a real prover would use. This game is perfectly indistinguishable from the previous one by perfect equivocality of type-1 and type-2 commitments.
- Game 2 is similar to Game 1, except that the decommitments $\boldsymbol{d}_{2,k}$ for $k \in (S_+ \cup S_\times) \setminus [m]$ (i.e., the ones which are not published) and $\boldsymbol{d}'_{\star,\star}$ used to generate the linear and quadratic proofs (see Eqs. (16) to (18)) are generated as by CPSim. By perfect equivocality of type-1 and type-2 commitments, these values $\boldsymbol{d}_{2,k}$ and $\boldsymbol{d}'_{\star,\star}$ are valid decommitments. Hence by uniqueness of linear proofs and perfect uniformity of quadratic proofs, the resulting proofs $\boldsymbol{\pi}_k$ are perfectly indistinguishable between Game 1 and Game 2.

As Game 2 corresponds to the zero-knowledge game where proofs are simulated, this conclude the proof of perfect zero-knowledge. □

Acknowledgments. The second author is supported by NSF grants CNS-1528178, CNS-1514526, CNS-1652849 (CAREER), a Hellman Fellowship, the Defense Advanced Research Projects Agency (DARPA) and Army Research Office (ARO) under Contract No. W911NF-15-C-0236, and a subcontract No. 2017-002 through Galois. The views expressed are those of the authors and do not reflect the official policy or position of the Department of Defense, the National Science Foundation, or the U.S. Government.

References

1. Abdalla, M., Benhamouda, F., Pointcheval, D.: Disjunctions for hash proof systems: New constructions and applications. In: Oswald, E., Fischlin, M. (eds.) EUROCRYPT 2015, Part II. LNCS, vol. 9057, pp. 69–100. Springer, Heidelberg (Apr 2015)
2. Afshar, A., Mohassel, P., Pinkas, B., Riva, B.: Non-interactive secure computation based on cut-and-choose. In: Nguyen, P.Q., Oswald, E. (eds.) EUROCRYPT 2014. LNCS, vol. 8441, pp. 387–404. Springer, Heidelberg (May 2014)
3. Applebaum, B., Ishai, Y., Kushilevitz, E.: Computationally private randomizing polynomials and their applications. computational complexity 15(2), 115–162 (Jun 2006), https://doi.org/10.1007/s00037-006-0211-8
4. Asharov, G., Jain, A., López-Alt, A., Tromer, E., Vaikuntanathan, V., Wichs, D.: Multiparty computation with low communication, computation and interaction via threshold FHE. In: Pointcheval, D., Johansson, T. (eds.) EUROCRYPT 2012. LNCS, vol. 7237, pp. 483–501. Springer, Heidelberg (Apr 2012)

5. Badrinarayanan, S., Garg, S., Ishai, Y., Sahai, A., Wadia, A.: Two-message witness indistinguishability and secure computation in the plain model from new assumptions. In: Takagi, T., Peyrin, T. (eds.) ASIACRYPT 2017, Part III. LNCS, vol. 10626, pp. 275–303. Springer, Heidelberg (Dec 2017)
6. Badrinarayanan, S., Jain, A., Ostrovsky, R., Visconti, I.: Non-interactive secure computation from one-way functions. In: Peyrin, T., Galbraith, S. (eds.) ASIACRYPT 2018, Part III. LNCS, vol. 11274, pp. 118–138. Springer, Heidelberg (Dec 2018)
7. Barak, B., Canetti, R., Lindell, Y., Pass, R., Rabin, T.: Secure computation without authentication. In: Shoup, V. (ed.) CRYPTO 2005. LNCS, vol. 3621, pp. 361–377. Springer, Heidelberg (Aug 2005)
8. Barak, B., Goldreich, O., Impagliazzo, R., Rudich, S., Sahai, A., Vadhan, S.P., Yang, K.: On the (im)possibility of obfuscating programs. In: Kilian, J. (ed.) CRYPTO 2001. LNCS, vol. 2139, pp. 1–18. Springer, Heidelberg (Aug 2001)
9. Beimel, A., Gabizon, A., Ishai, Y., Kushilevitz, E., Meldgaard, S., Paskin-Cherniavsky, A.: Non-interactive secure multiparty computation. In: Garay, J.A., Gennaro, R. (eds.) CRYPTO 2014, Part II. LNCS, vol. 8617, pp. 387–404. Springer, Heidelberg (Aug 2014)
10. Benhamouda, F., Krawczyk, H., Rabin, T.: Robust non-interactive multiparty computation against constant-size collusion. In: Katz, J., Shacham, H. (eds.) CRYPTO 2017, Part I. LNCS, vol. 10401, pp. 391–419. Springer, Heidelberg (Aug 2017)
11. Benhamouda, F., Lin, H.: k-round multiparty computation from k-round oblivious transfer via garbled interactive circuits. In: Nielsen, J.B., Rijmen, V. (eds.) EUROCRYPT 2018, Part II. LNCS, vol. 10821, pp. 500–532. Springer, Heidelberg (Apr / May 2018)
12. Boyle, E., Gilboa, N., Ishai, Y.: Function secret sharing. In: Oswald, E., Fischlin, M. (eds.) EUROCRYPT 2015, Part II. LNCS, vol. 9057, pp. 337–367. Springer, Heidelberg (Apr 2015)
13. Boyle, E., Gilboa, N., Ishai, Y.: Breaking the circuit size barrier for secure computation under DDH. In: Robshaw, M., Katz, J. (eds.) CRYPTO 2016, Part I. LNCS, vol. 9814, pp. 509–539. Springer, Heidelberg (Aug 2016)
14. Canetti, R.: Security and composition of multiparty cryptographic protocols. Journal of Cryptology 13(1), 143–202 (Jan 2000)
15. Canetti, R.: Universally composable security: A new paradigm for cryptographic protocols. In: 42nd FOCS. pp. 136–145. IEEE Computer Society Press (Oct 2001)
16. Canetti, R., Jain, A., Scafuro, A.: Practical UC security with a global random oracle. In: Ahn, G.J., Yung, M., Li, N. (eds.) ACM CCS 14. pp. 597–608. ACM Press (Nov 2014)
17. Canetti, R., Lin, H., Pass, R.: Adaptive hardness and composable security in the plain model from standard assumptions. In: 51st FOCS. pp. 541–550. IEEE Computer Society Press (Oct 2010)
18. Catalano, D., Visconti, I.: Hybrid trapdoor commitments and their applications. In: Caires, L., Italiano, G.F., Monteiro, L., Palamidessi, C., Yung, M. (eds.) ICALP 2005. LNCS, vol. 3580, pp. 298–310. Springer, Heidelberg (Jul 2005)
19. Chase, M., Dodis, Y., Ishai, Y., Kraschewski, D., Liu, T., Ostrovsky, R., Vaikuntanathan, V.: Reusable non-interactive secure computation. Cryptology ePrint Archive, Report 2018/940 (2018), https://eprint.iacr.org/2018/940
20. Chase, M., Dodis, Y., Ishai, Y., Kraschewski, D., Liu, T., Ostrovsky, R., Vaikuntanathan, V.: Reusable non-interactive secure computation. In: Shacham, H.,

Boldyreva, A. (eds.) CRYPTO 2019, Part III. pp. 462–488. LNCS, Springer, Heidelberg (Aug 2019)
21. Clear, M., McGoldrick, C.: Multi-identity and multi-key leveled FHE from learning with errors. In: Gennaro, R., Robshaw, M.J.B. (eds.) CRYPTO 2015, Part II. LNCS, vol. 9216, pp. 630–656. Springer, Heidelberg (Aug 2015)
22. Cleve, R.: Towards optimal simulations of formulas by bounded-width programs. computational complexity 1(1), 91–105 (Mar 1991), https://doi.org/10.1007/BF01200059
23. Feige, U., Kilian, J., Naor, M.: A minimal model for secure computation (extended abstract). In: 26th ACM STOC. pp. 554–563. ACM Press (May 1994)
24. Garg, S., Gentry, C., Halevi, S., Raykova, M.: Two-round secure MPC from indistinguishability obfuscation. In: Lindell, Y. (ed.) TCC 2014. LNCS, vol. 8349, pp. 74–94. Springer, Heidelberg (Feb 2014)
25. Garg, S., Gentry, C., Sahai, A., Waters, B.: Witness encryption and its applications. In: Boneh, D., Roughgarden, T., Feigenbaum, J. (eds.) 45th ACM STOC. pp. 467–476. ACM Press (Jun 2013)
26. Garg, S., Srinivasan, A.: Garbled protocols and two-round MPC from bilinear maps. In: Umans, C. (ed.) 58th FOCS. pp. 588–599. IEEE Computer Society Press (Oct 2017)
27. Garg, S., Srinivasan, A.: Two-round multiparty secure computation from minimal assumptions. In: Nielsen, J.B., Rijmen, V. (eds.) EUROCRYPT 2018, Part II. LNCS, vol. 10821, pp. 468–499. Springer, Heidelberg (Apr / May 2018)
28. Goldreich, O.: Foundations of Cryptography: Basic Tools, vol. 1. Cambridge University Press, Cambridge, UK (2001)
29. Goldwasser, S., Gordon, S.D., Goyal, V., Jain, A., Katz, J., Liu, F.H., Sahai, A., Shi, E., Zhou, H.S.: Multi-input functional encryption. In: Nguyen, P.Q., Oswald, E. (eds.) EUROCRYPT 2014. LNCS, vol. 8441, pp. 578–602. Springer, Heidelberg (May 2014)
30. Goldwasser, S., Micali, S., Rackoff, C.: The knowledge complexity of interactive proof systems. SIAM Journal on Computing 18(1), 186–208 (1989)
31. Gordon, S.D., Liu, F.H., Shi, E.: Constant-round MPC with fairness and guarantee of output delivery. In: Gennaro, R., Robshaw, M.J.B. (eds.) CRYPTO 2015, Part II. LNCS, vol. 9216, pp. 63–82. Springer, Heidelberg (Aug 2015)
32. Groth, J., Ostrovsky, R., Sahai, A.: New techniques for noninteractive zero-knowledge. Journal of the ACM (JACM) 59(3), 11 (2012)
33. Groth, J., Sahai, A.: Efficient non-interactive proof systems for bilinear groups. In: Smart, N.P. (ed.) EUROCRYPT 2008. LNCS, vol. 4965, pp. 415–432. Springer, Heidelberg (Apr 2008)
34. Groth, J., Sahai, A.: Efficient noninteractive proof systems for bilinear groups. SIAM J. Comput. 41(5), 1193–1232 (2012), https://doi.org/10.1137/080725386
35. Halevi, S., Ishai, Y., Jain, A., Komargodski, I., Sahai, A., Yogev, E.: Non-interactive multiparty computation without correlated randomness. In: Takagi, T., Peyrin, T. (eds.) ASIACRYPT 2017, Part III. LNCS, vol. 10626, pp. 181–211. Springer, Heidelberg (Dec 2017)
36. Ishai, Y., Kushilevitz, E.: Private simultaneous message protocols with applications. In: Proceedings of ISTCS. pp. 174–184 (1997)
37. Ishai, Y., Kushilevitz, E.: Randomizing polynomials: A new representation with applications to round-efficient secure computation. In: 41st FOCS. pp. 294–304. IEEE Computer Society Press (Nov 2000)

38. Ishai, Y., Kushilevitz, E., Ostrovsky, R., Prabhakaran, M., Sahai, A.: Efficient non-interactive secure computation. In: Paterson, K.G. (ed.) EUROCRYPT 2011. LNCS, vol. 6632, pp. 406–425. Springer, Heidelberg (May 2011)
39. Lin, H., Matt, C.: Pseudo flawed-smudging generators and their application to indistinguishability obfuscation. Cryptology ePrint Archive, Report 2018/646 (2018), https://eprint.iacr.org/2018/646
40. Lindell, Y.: An efficient transform from sigma protocols to NIZK with a CRS and non-programmable random oracle. In: Dodis, Y., Nielsen, J.B. (eds.) TCC 2015, Part I. LNCS, vol. 9014, pp. 93–109. Springer, Heidelberg (Mar 2015)
41. Mukherjee, P., Wichs, D.: Two round multiparty computation via multi-key FHE. In: Fischlin, M., Coron, J.S. (eds.) EUROCRYPT 2016, Part II. LNCS, vol. 9666, pp. 735–763. Springer, Heidelberg (May 2016)

Supplementary Material

A Preliminaries

A.1 Statistical and Computational Indistinguishability

A function $\mathrm{negl}\colon \mathbb{N} \to \mathbb{N}$ is negligible if for any polynomial $p\colon \mathbb{N} \to \mathbb{N}$, for any large enough $\lambda \in \mathbb{N}$, $\mathrm{negl}(\lambda) < 1/p(\lambda)$.

Definition A.1 (Indistinguishability). Let $S = \{S_\lambda\}_{\lambda\in\mathbb{N}}$ be an ensemble of subsets of $\{0,1\}^*$, where every element in set S_λ has length $\mathrm{poly}(\lambda)$. Then ensembles $X = \{X_{\lambda,w}\}_{\lambda\in\mathbb{N},w\in S_\lambda}$ and $Y = \{Y_{\lambda,w}\}_{\lambda\in\mathbb{N},w\in S_\lambda}$ are *statistically* (resp., *computationally*) *indistinguishable*, denoted as $X \approx_s Y$ (resp., $X \approx Y$), if for any arbitrary-size (resp., polynomial-size) circuit family $D = \{D_\lambda\}_{\lambda\in\mathbb{N}}$ and any polynomial-size sequence of index $\{w_\lambda \in S\}_{\lambda\in\mathbb{N}}$, there exists a negligible function negl such that, for every $\lambda \in \mathbb{N}$,

$$|\Pr[D_\lambda(w_\lambda, X_{\lambda,w_\lambda}) = 1] - \Pr[D_\lambda(w_\lambda, Y_{\lambda,w_\lambda}) = 1]| \leq \mathrm{negl}(\lambda) \ .$$

Two statistically indistinguishable ensembles are also said to be *statistically close*.

A.2 Garbled Circuit

Definition A.2 (Garbled Circuit). Let $\mathcal{C} = \{\mathcal{C}_\lambda\}_{\lambda\in\mathbb{N}}$ be a poly-size circuit class with input and output lengths n and l. A *garbled circuit* scheme GC for $\mathcal{C}$ is a tuple of four polynomial-time algorithms $\mathsf{GC} = (\mathsf{GC.Gen}, \mathsf{GC.Garble}, \mathsf{GC.Eval}, \mathsf{GC.Sim})$:

- Input Labels Generation: $\mathsf{key} \leftarrow \mathsf{GC.Gen}(1^\lambda)$ generates input labels $\mathsf{key} = \{\mathsf{key}[i,b]\}_{i\in[n],b\in\{0,1\}}$ (with $\mathsf{key}[i,b] \in \{0,1\}^\kappa$ being the input label corresponding to the value b of the i-th input wire) for the security parameter λ, input length n, and input label length κ;
- Circuit Garbling: $\widehat{C} \leftarrow \mathsf{GC.Garble}(\mathsf{key}, C)$ garbles the circuit $C \in \mathcal{C}_\lambda$ into $\widehat{C}$;
- Evaluation: $y = \mathsf{GC.Eval}(\widehat{C}, \mathsf{key}')$ evaluates the garbled circuit $\mathsf{GC.Garble}$ using input labels $\mathsf{key}' = \{\mathsf{key}'[i]\}_{i\in[n]}$ (where $\mathsf{key}'[i] \in \{0,1\}^\kappa$) and returns the output $y \in \{0,1\}^l$;
- Simulation: $(\mathsf{key}', \widetilde{C}) \leftarrow \mathsf{GC.Sim}(1^\lambda, y)$ simulates input labels $\mathsf{key}' = \{\mathsf{key}'[i]\}_{i\in[n]}$ and a garbled circuit $\widetilde{C}$ for the security parameter λ and the output $y \in \{0,1\}^l$;

satisfying the following security properties:

Correctness. For any security parameter $\lambda \in \mathbb{N}$, for any circuit $C \in \mathcal{C}_\lambda$, for any input $x \in \{0,1\}^n$, for any key in the image of $\mathsf{GC.Gen}(1^\lambda)$ and any $\widehat{C}$ in the image of $\mathsf{GC.Garble}(\mathsf{key}, C)$:

$$\mathsf{GC.Eval}(\widehat{C}, \{\mathsf{key}[i, x_i]\}_{i\in[n]}) = C(x) \ .$$

Simulatability. The following two distributions are computationally indistinguishable:

$$\left\{ (\{\mathsf{key}[i, x_i]\}_{i\in[n]}, \widehat{C}) \ : \ \begin{array}{l} \mathsf{key} \leftarrow \mathsf{GC.Gen}(1^\lambda); \\ \widehat{C} \leftarrow \mathsf{GC.Garble}(\mathsf{key}, C) \end{array} \right\}_{\lambda, C\in\mathcal{C}_\lambda, x\in\{0,1\}^n},$$

$$\left\{ (\mathsf{key}', \widehat{C}) \ : \ (\mathsf{key}', C) \leftarrow \mathsf{GC.Sim}(1^\lambda, C(x)) \right\}_{\lambda, C\in\mathcal{C}_\lambda, x\in\{0,1\}^n}.$$

We recall that garbled circuit schemes can be constructed from one-way functions.

A.3 Collision-Resistant Hash Function Family

Definition A.3 (Collision-Resistant Hash Function Family). A collision-resistant hash function family is an ensemble $\{\mathcal{HF}_\lambda\}_\lambda$ of families of functions $\mathcal{H}$ from $\{0,1\}^*$ to $\{0,1\}^{2\lambda}$, satisfying the following property:

Collision Resistance. For any PPT adversary $\mathcal{A}$, there exists a negligible function negl such that for every $\lambda \in \mathbb{N}$:

$$\Pr\left[\mathcal{H} \leftarrow \mathcal{HF}_\lambda, (m_0, m_1) \leftarrow \mathcal{A}(\mathcal{H}) \ : \ \mathcal{H}(m_0) = \mathcal{H}(m_1) \text{ and } m_0 \neq m_1\right] \leq \mathrm{negl}(\lambda).$$

A.4 Pseudorandom Functions

Definition A.4 (Pseudorandom Functions). A pseudorandom function is a deterministic polynomial-time algorithm PRF taking as input a key $K \in \{0,1\}^\lambda$ and an input $x \in \{0,1\}^{\mathrm{poly}(\lambda)}$ and outputting a bit $y \in \{0,1\}$, satisfying the following property:

Pseudorandomness. For any PPT adversary $\mathcal{A}$, there exists a negligible function negl such that for every $\lambda \in \mathbb{N}$:

$$\left|\Pr\left[K \leftarrow \{0,1\}^\lambda \ : \ \mathcal{A}^{\mathsf{PRF}(K,\cdot)}(1^\lambda) = 1\right] - \Pr\left[\mathcal{A}^{R(\cdot)}(1^\lambda) = 1\right]\right| \leq \mathrm{negl}(\lambda).$$

where R is a random oracle taking as input a bit string $x \in \{0,1\}^{\mathrm{poly}(\lambda)}$ and outputting a bit.

Remark A.5. For simplicity of notation, we often write $\mathsf{PRF}(\mathrm{fk}, z'\|\{z_1, \ldots, z_\ell\}\|\mathbf{0}) = r$ for $z' \in \{0,1\}^{\mathrm{poly}(\lambda)}$ and $z_i \in \{0,1\}^{\mathrm{poly}(\lambda)}$ to mean evaluating PRF on all inputs of form $z'||z_i$ padded with zeros to the input length if necessary, and concatenating all output bits to a ℓ-bit string r. In particular, we use $\mathsf{PRF}(\mathrm{fk}, z'\|[\ell]\|\mathbf{0})$ to generate a ℓ-bit pseudo-random string for z'. Integers $z_i \in [\ell]$ are supposed to be written in binary with the most significant bit on the left. In addition, for the sake of simplicity, we often omit $\mathbf{0}$ in the above notation.

A.5 Background on Universal Composability

In this section we recall basics of the UC framework; for full details see [15]. A large part of this introduction has been taken verbatim from [17].

The basic model of execution. Following [28,30], a protocol is represented as an interactive Turing machine (ITM), which represents the program to be run within each participant. Specifically, an ITM has three tapes that can be written to by other ITMs: the input and subroutine output tapes model the inputs from and the outputs to other programs running within the same "entity" (say, the same physical computer), and the incoming communication tapes and outgoing communication tapes model messages received from and to be sent to the network. It also has an identity tape that cannot be written to by the ITM itself. The identity tape contains the program of the ITM (in some standard encoding) plus additional identifying information specified below. Adversarial entities are also modeled as ITMs.

We distinguish between ITMs (which represent static objects, or programs) and *instances of ITMs*, or ITIs, that represent interacting processes in a running system. Specifically, an ITI is an ITM along with an identifier that distinguishes it from other ITIs in the same system. The identifier consists of two parts: A session-identifier (SID) which identifies which protocol instance the ITM belongs to, and a party identifier (PID) that distinguishes among the parties in a protocol instance. Typically the PID is also used to associate ITIs with "parties" or clusters, that represent some administrative domains or physical computers.

The model of computation consists of a number of ITIs that can write on each other's tapes in certain ways (specified in the model). The pair (SID,PID) is a unique identifier of the ITI in the system.

We assume that all ITMs are probabilistic polynomial time (PPT). An ITM is PPT if there exists a constant $c > 0$ such that, at any point during its run, the overall number of steps taken by M is at most λ^c, where n is the overall number of bits written on the *input tape* of M in this run. execution process is bounded by a polynomial, we define λ as the total number of bits written to the input tape of M, *minus the overall number of bits written by M to input tapes of other ITMs*; see [15].)

Security of protocols. Protocols that securely carry out a given task (or protocol problem) are defined in three steps, as follows. First, the process of executing a protocol in an adversarial environment is formalized. Next, an "ideal process" for carrying out the task at hand is formalized. In the ideal process the parties do not communicate with each other. Instead they have access to an "ideal functionality," which is essentially an incorruptible "trusted party" that is programmed to capture the desired functionality of the task at hand. A protocol is said to securely realize an ideal functionality if the process of running the protocol amounts to "emulating" the ideal process for that ideal functionality. Below we overview the model of protocol execution (called the *real-world model*), the ideal process, and the notion of protocol emulation.

Real-world execution. The model of computation consists of the parties running an instance of a protocol Π, an *adversary* $\mathcal{A}$ that controls the communication

among the parties, and an *environment* $\mathcal{Z}$ that controls the inputs to the parties and sees their outputs. We assume that all parties have a security parameter $\lambda \in \mathbb{N}$. (We remark that this is done merely for convenience and is not essential for the model to make sense). The execution consists of a sequence of *activations*, where in each activation a single participant (either $\mathcal{Z}$, $\mathcal{A}$, or some other ITM) is activated, and may write on a tape of at most *one* other participant, subject to the rules below. Once the activation of a participant is complete (i.e., once it enters a special waiting state), the participant whose tape was written on is activated next. (If no such party exists then the environment is activated next.)

The environment is given an external input z and is the first to be activated. In its first activation, the environment invokes the adversary $\mathcal{A}$, providing it with some arbitrary input. In the context of UC security, the environment can from now on invoke (namely, provide input to) only ITMs that consist of a single instance of protocol Π. That is, all the ITMs invoked by the environment must have the same SID and the code of Π.

Once the adversary is activated, it may read its own tapes and the outgoing communication tapes of all parties. It may either deliver a message to some party by writing this message on the party's incoming communication tape or report information to $\mathcal{Z}$ by writing this information on the subroutine output tape of $\mathcal{Z}$. For simplicity of exposition, in the rest of this paper we assume authenticated communication; that is, the adversary may deliver only messages that were actually sent. (This is however not essential as shown in [7].)

Once a protocol party (i.e., an ITI running Π) is activated, either due to an input given by the environment or due to a message delivered by the adversary, it follows its code and possibly writes a local output on the subroutine output tape of the environment, or an outgoing message on the adversary's incoming communication tape. Finally the adversary can decide to corrupt any honest party. In this case the input and the random coins used by this party are revealed to the adversary.

The protocol execution ends when the environment halts. The output of the protocol execution is the output of the environment. Without loss of generality we assume that this output consists of only a single bit.

Let $\mathsf{Real}_{\Pi,\mathcal{A},\mathcal{Z}}(\lambda, z, r)$ denote the output of the environment $\mathcal{Z}$ when interacting with parties running protocol Π on security parameter λ, input z, and randomness $r = r_{\mathcal{Z}}, r_{\mathcal{A}}, r_1, r_2, \ldots$ (where z and $r_{\mathcal{Z}}$ for $\mathcal{Z}$; $r_{\mathcal{A}}$ for $\mathcal{A}$, r_i for party P_i). Let $\mathsf{Real}_{\Pi,\mathcal{A},\mathcal{Z}}(\lambda, z)$ be the random variable describing $\mathsf{Real}_{\Pi,\mathcal{A},\mathcal{Z}}(\lambda, z, r)$ where r is uniformly chosen. Let $\mathsf{Real}_{\Pi,\mathcal{A},\mathcal{Z}}$ denote the ensemble $\{\mathsf{Real}_{\Pi,\mathcal{A},\mathcal{Z}}(\lambda, z)\}_{\lambda\in\mathbb{N},z\in\{0,1\}^*}$.

Ideal functionalities and ideal protocols. Security of protocols is defined by comparing the protocol execution to an *ideal protocol* for carrying out the task at hand. A key ingredient in the ideal protocol is the *ideal functionality* that captures the desired functionality, or the specification, of that task. The ideal functionality is modeled as another ITM (representing a "trusted party") that interacts with the parties and the adversary. More specifically, in the ideal protocol for functionality $\mathcal{F}$ all parties simply hand their inputs to an ITI running

$\mathcal{F}$. We will simply call this ITI $\mathcal{F}$. The SID of $\mathcal{F}$ is the same as the SID of the ITIs running the ideal protocol and the PID of $\mathcal{F}$ is null.

In addition, $\mathcal{F}$ can interact with the adversary according to its code. Whenever $\mathcal{F}$ outputs a value to a party, the party immediately copies this value to its own output tape. We call the parties in the ideal protocol dummy parties. Let $\Pi(\mathcal{F})$ denote the ideal protocol for functionality $\mathcal{F}$.

A.6 Network and Corruption Model Used, and Definition of UC-Security

Here, we specify the network and corruption model we use for mrNISC protocols and recall UC security definition.

Network Model. We assume that parties have access to both a broadcast channel and point to point channels and all channels are authenticated. Assuming authenticated channel is without loss of generality, as in our mrNISC protocols, parties' broadcast their first messages, and hence each party can additionally broadcast a verification key of a signature scheme, and use signatures to ensure the integrity of their messages sent later. We remark that in the UC model, the schedule of message delivery is controlled by the adversary (and hence a message sent through the broadcast channel is not guaranteed to be delivered at the same time to all recipients, if delivered at all).

Corruption Models. We consider 1) static corruption, meaning that the adversary chooses the set of corrupted parties $\bar{H}$ at the beginning of the execution, and 2) security with aborts, meaning that the adversary obtains the outputs first, and may prevent the honest parties from obtaining the outputs.

Three standard types of adversarial behaviors are considered: semi-honest, malicious, and semi-malicious adversaries. We assume familiarity with the first two types, and briefly describe the third type. Semi-malicious adversaries follow the protocol specification (like semi-honest adversaries), but may choose arbitrary random tape to its advantage (like malicious adversaries). In slight more detail, a semi-malicious adversary is model as follows: After sending each message m as a corrupted party P_j to an honest party P_i, the adversary outputs on a special output tape, an input x_j and a random tape ρ_j, such that, m is the right next message generated by P_j according to the protocol specification on input x_j, random tape ρ_j, after receiving messages it has received so far — the pair (x_j, ρ_j) is called a *witness* of the message m. In the case that the adversary fails to output a valid witness, the message m is overwritten to $\perp$. See [4] for more detailed formalization of semi-malicious adversaries.

UC Security. A protocol Π *emulates* protocol ϕ if for any adversary $\mathcal{A}$ there exists an adversary $\mathcal{S}$ such that no environment $\mathcal{Z}$, on any input, can tell whether it is interacting with $\mathcal{A}$ and parties running Π, or it is interacting with $\mathcal{S}$ and parties running ϕ. This means that, from the point of view of the environment, running protocol Π is "just as good" as interacting with ϕ. We say that Π *securely realizes* an ideal functionality $\mathcal{F}$ if it emulates the ideal protocol $\Pi(\mathcal{F})$, where participants interacts with $\mathcal{F}$ instead of with each other.

Definition A.6. Let Π and ϕ be protocols. We say that Π UC-emulates ϕ against (static) semi-honest / semi-malicious / malicious adversaries if for every semi-honest / semi-malicious / malicious adversary $\mathcal{A}$ (corrupting a set of parties statically) there exists an adversary $\mathcal{S}$ such that for any environment $\mathcal{Z}$ that obeys the rules of interaction for UC security we have $\{\mathsf{Real}_{\Pi,\mathcal{A},\mathcal{Z}}(\lambda,z)\}_{\lambda\in\mathbb{N},z\in\{0,1\}^*} \approx \{\mathsf{Ideal}_{\phi,\mathcal{S},\mathcal{Z}}(\lambda,z)\}_{\lambda\in\mathbb{N},z\in\{0,1\}^*}$

Above $\mathsf{Ideal}_{\phi,\mathcal{S},\mathcal{Z}}(\lambda,z)$ and $\mathsf{Real}_{\Pi,\mathcal{A},\mathcal{Z}}(\lambda,z)$ are the random variables describing the outputs of the environment in the respective experiments with security parameter λ and auxiliary input z to the adversary. See definitions in Appendix A.5.

Definition A.7. Let $\mathcal{F}$ be an ideal functionality and let Π be a protocol (in the $\mathcal{G}$-Hybrid model). We say that Π UC-realizes $\mathcal{F}$ with (static) semi-honest / semi-malicious / malicious security (in $\mathcal{G}$-hybrid model) if Π UC-emulates the ideal process $\Pi(\mathcal{F})$ against (static) semi-honest / semi-malicious / malicious adversaries.

A.7 Semi-Malicious Output-Delayed Simulatability

Our mrNISC construction makes use of a special MPC protocol for which the transcript excluding the last messages can be simulated for all-but-one honest parties before knowing the output. We define it below.

Definition A.8 (MPC Protocol). An L-rounds MPC scheme for a class of functions $\mathcal{C}$ and n parties consists of two polynomial-time algorithms $\Pi = (\mathsf{Next}, \mathsf{Output})$:

- Next Message: $m_i^\ell \coloneqq \mathsf{Next}_i(1^\lambda, 1^n, z, x_i, r_i, \boldsymbol{m}^{<\ell})$ is the message broadcasted by party P_i for $i \in [n]$ in round $\ell \in [L]$, on input x_i, on random tape $r_i \in \{0,1\}^{\nu_r}$, after receiving the messages $\mathsf{Msg}^{<\ell} = \{m_j^{\ell'}\}_{j\in[n],\ell'<\ell}$, where $m_j^{\ell'}$ is the message broadcasted by party P_j on round $\ell' \in [\ell-1]$.
- Output: $y \coloneqq \mathsf{Output}(1^\lambda, 1^n, z, \mathsf{Msg})$ is the public output of the MPC protocol, for the public input z, and the transcript $\mathsf{Msg} = \{m_j^\ell\}_{j\in[n],\ell\in[L]}$.

satisfying the perfect correctness property (formally recalled in Definition A.9).

We omit 1^λ and 1^n when clear from context. We remark that anybody seeing the transcript $\mathsf{Msg} = \{m_j^\ell\}_{j\in[n],\ell\in[L]}$ can compute the output y of the function.

Security properties are defined below.

Definition A.9 (Perfect Correctness of MPC Protocol). An L-rounds MPC protocol $\Pi = (\mathsf{Next}, \mathsf{Output})$ for $\mathcal{C}$ is perfectly secure if for any security parameter $\lambda \in \mathbb{N}$, for any public input z, for any inputs $(x_1, \ldots, x_n)$,

$$\Pr\left[\bar{r} \leftarrow (\{0,1\}^{\nu_r})^n \quad : \quad \mathsf{Output}(1^\lambda, 1^n, z, \mathsf{Msg}) = f(z, x_1, \ldots, x_n)\right] = 1 \ ,$$

where $m_i^\ell = \mathsf{Next}_i(1^\lambda, 1^n, z, x_i, r_i, \mathsf{Msg}^{<\ell})$ for $i \in [n]$ and $\ell \in [L]$.

Definition A.10 (Semi-Malicious Output-Delayed Simulatability). A L-rounds MPC scheme for $\mathcal{C}$ is semi-malicious output-delayed simulatable, if there exists a PPT simulator $\mathcal{S}$, such that, for all PPT adversary $\mathcal{A}$ and $f \in \mathcal{C}$, the view of $\mathcal{A}$ in the following experiments $\mathsf{Exp}_{\mathcal{A},\mathcal{S}}(\mathsf{Real}, \lambda, f)$ and $\mathsf{Exp}_{\mathcal{A},\mathcal{S}}(\mathsf{Ideal}, \lambda, f)$ are indistinguishable: **Experiment** $\mathsf{Exp}_{\mathcal{A},\mathsf{Sim}}(\mathsf{Real}, \lambda, f)$**:**

1. The adversary $\mathcal{A}$ chooses the number of parties M, the set of honest parties $H \subseteq [M]$, the public input z, the inputs $\{x_i\}_{i\in[n]}$ of all the parties and the random tapes $\{r_i\}_{i\in\bar{H}}$ of the corrupt parties.
2. The challenger picks fresh random tapes $\{r_i\}_{i\in H}$ for the honest parties. It runs the MPC protocol with all the inputs and random tapes above, and sends the adversary the resulting transcript without the last message of the honest parties: $(\mathsf{Msg}^{<L}, \{m_i^L\}_{i\in\bar{H}})$.
3. The adversary $\mathcal{A}$ interacts with the challenger by sending queries $(\mathsf{compute}, P_i)$ for $i \in H$. Upon receiving each query, the challenger sends the last message m_i^L of P_i.

Upon receiving each query, the challenger sends the last message m_i^L of P_i. **Experiment** $\mathsf{Exp}_{\mathcal{A},\mathsf{Sim}}(\mathsf{Ideal}, \lambda, f)$**:**

1. The adversary $\mathcal{A}$ chooses the number of parties M, the set of honest parties $H \subseteq [M]$, the public input z, the inputs $\{x_i\}_{i\in[n]}$ of all the parties and the random tapes $\{r_i\}_{i\in\bar{H}}$ of the corrupt parties.
2. The challenger sends the inputs and random tapes $(\{x_i\}_{i\in\bar{H}}$ of the corrupt parties to the simulator Sim. The simulator then outputs a transcript without the last message of the honest parties: $(\mathsf{Msg}^{<L}, \{m_i^L\}_{i\in\bar{H}})$, that the challenger sends back to $\mathcal{A}$.
3. The adversary $\mathcal{A}$ interacts with the challenger by sending queries $(\mathsf{compute}, P_i)$ for $i \in H$. Upon receiving each query, if all the honest parties have not been queried yet (after this query), the challenger sends $(\mathsf{compute}, P_i)$ to Sim which answers with m_i^L that the challenger forwards to $\mathcal{A}$. If all the honest parties have been queried, the challenger sends $(\mathsf{compute}, P_i, f(x_1, \ldots, x_n))$ to Sim, which answers with m_i^L that the challenger forwards to $\mathcal{A}$.

We note that our notion of semi-malicious is weak as it forces the adversary to commit to its random tape and input from the beginning. This definition is sufficient for our purpose.

We also remark that from any semi-malicious MPC, we can construct a semi-malicious output-delayed simulatable MPC. Let f be the single-output function we consider. We define the following function f':

$$f'((x_1, t_1), \ldots, (x_n, t_n)) \coloneqq f(x_1, \ldots, x_n) \oplus t_1 \oplus \cdots \oplus t_n \ ,$$

where t_i is a bit string of the same length as the output of f, and $\oplus$ is the XOR operation. Given a L-round semi-malicious MPC Π' for f', we construct a $(L+1)$-round semi-malicious output-delayed simulatable MPC for f as follows: each party P_i with input x_i samples t_i uniformly randomly and runs Π' with

input (x_i, t_i). Then, each party P_i broadcasts in the last additional round the value t_i. The final output can be recovered using the output of Π' and the values $\{t_i\}_{i\in[n]}$.

Simulation of Π can be done just by simulating Π' with a random output. Only when the last party is corrupted, the simulator needs to know the actual output y, to program correctly the last message t_i to be revealed.

B Definitions of mrNISC Schemes and Protocols

B.1 Definition of mrNISC Schemes

Definition of mrNISC schemes has appeared in the introduction and technical overview. We repeat them here for convenient reference.

Definition B.1 (mrNISC Schemes). *An mrNISC scheme for a function f from $\bigcup_{n=1}^{\infty}(\{0,1\}^*)^{n+1}$ to $\{0,1\}^*$ (i.e., a function which can take any number ≥ 2 of inputs) consists of a tuple of polynomial-time algorithms* $\mathsf{mrNISC} = (\mathsf{Com}, \mathsf{Encode}, \mathsf{Eval})$ *with the following syntax.*

- *Input:* $(\hat{x}_i, s_i) \leftarrow \mathsf{Com}(1^\lambda, x_i)$ *on input the security parameter and an input $x_i \in \{0,1\}^\nu$, generates an input encoding $\hat{x}_i$ and a secret state s_i, where the input length ν is polynomial in the security parameter ν.*
- *Computation of $f(z, \star)$:* $\alpha_i \leftarrow \mathsf{Encode}(z, \{\hat{x}_j\}_{j\in I}, s_i)$*, on input a public input $z \in \{0,1\}^\nu$, a set of input encodings $\{\hat{x}_j\}_{j\in I}$, and the secret state s_i generated together with the i'th input encoding $\hat{x}_i$, generates the* i'th computation encoding *α_i of $f(z, x_1, \cdots, x_n)$.*
- *Output:* $y = \mathsf{Eval}(z, \{\hat{x}_i\}_{i\in[n]}, \{\alpha_i\}_{i\in[n]})$*, on input a public input $z \in \{0,1\}^\nu$, a set of input encodings $\{\hat{x}_i\}_{i\in I}$ and the corresponding set of computation encodings $\{\hat{x}_i\}_{i\in I}$, produces an output $y \in \{0,1\}^* \cup \{\perp\}$.*

Definition B.2 (Correctness). *An mrNISC scheme* mrNISC *for f is correct if: For every $\lambda \in \mathbb{N}$, every family of private inputs $\{x_i\}_{i\in I}$, every public input z,*

$$\Pr\left[\begin{array}{l}\forall i \in I,\ (\hat{x}_i, s_i) \leftarrow \mathsf{Com}(1^\lambda, x_i) \\ \forall i \in I,\ \alpha_i \leftarrow \mathsf{Encode}(z, \{\hat{x}_j\}_{j\in I}, s_i)\end{array} : \mathsf{Eval}(z, \{\hat{x}_i\}_{i\in I}, \{\alpha_i\}_{i\in I}) = f(z, \{x_i\}_{i\in I})\right] = 1\ .$$

Note that it suffices to define the correctness w.r.t. a single set of input and computation encodings, which directly implies correctness w.r.t. an arbitrary number of input and computation encodings as in the mrNISC market setting. Furthermore, since correctness is perfect, it means correctness holds even if the private and public inputs, the x_i's and z's, and the random tapes used for generating the input and computation encodings are chosen maliciously and adaptively — in short, correctness holds against semi-malicious adversaries.

Definition B.3 (Adaptive Security). *An mrNISC scheme* mrNISC *for* f *is semi-malicious (or semi-honest) private if there exists a PPT simulator* $\mathcal{S}$*, such that, for all PPT adversary* $\mathcal{A}$*, the views of* $\mathcal{A}$ *in the following experiments* $\mathsf{Exp}_{\mathcal{A},\mathcal{S}}(\mathsf{Real}, \lambda, f)$ *and* $\mathsf{Exp}_{\mathcal{A},\mathcal{S}}(\mathsf{Ideal}, \lambda, f)$ *are indistinguishable.*

Experiment $\mathsf{Exp}_{\mathcal{A},\mathcal{S}}(\mathsf{Real}, \lambda, f)$**:** *The adversary* $\mathcal{A}$ *chooses the number of parties* M *and the set of honest parties* $H \subseteq [M]$*. It then interacts with a challenger in an arbitrary number of iterations until it terminates. In every iteration* k*, it can submit* one query *of one of the following three types.*

- Corrupt Input Encoding: *Upon* $\mathcal{A}$ *sending a query* $(\mathsf{input}, P_i, x_i, \rho_i)$ *for a corrupt party* $i \in \bar{H}$*, record the input encoding* $\hat{x}_i$ *generated as* $(\hat{x}_i, s_i) = \mathsf{Com}(1^\lambda, x_i; \rho_i)$*, using input* x_i *and randomness* ρ_i*. (In the semi-honest case,* ρ_i *must be randomly sampled, whereas in the semi-malicious case, it can be arbitrary chosen by* $\mathcal{A}$*.)*
- Honest Input Encoding: *Upon* $\mathcal{A}$ *choosing the input* $(\mathsf{input}, P_i, x_i)$ *of an honest party* $i \in H$*, generate* $(\hat{x}_i, s_i) \leftarrow \mathsf{Com}(1^\lambda, x_i)$ *and send* $\hat{x}_i$ *to* $\mathcal{A}$*.*
- Honest Computation Encoding: *Upon* $\mathcal{A}$ *querying* $(\mathsf{compute}, P_i, z, I)$ *for an honest party* $i \in H \cap I$*, if the input encodings* $\{\hat{x}_j\}_{j\in I}$ *of all participants have been generated, send* $\mathcal{A}$ *the computation encoding* $\alpha_i \leftarrow \mathsf{Encode}(z, \{\hat{x}_j\}_{j\in I}, s_i)$*.*

Experiment $\mathsf{Exp}_{\mathcal{A},\mathcal{S}}(\mathsf{Ideal}, \lambda, f)$**:** *The ideal experiment proceeds identically as above, except for the following differences: Invoke* $\mathcal{S}(1^\lambda, f)$*.*

- Corrupt Input Encoding: *Additionally send query* $(\mathsf{input}, P_i, x_i, \rho_i)$ *to* $\mathcal{S}$*.*
- Honest Input Encoding: *Upon* $\mathcal{A}$ *choosing* $(\mathsf{input}, P_i, x_i)$ *for* $i \in H$*, send query* (input, P_i) *to the simulator* $\mathcal{S}$ *and forward to* $\mathcal{A}$ *the simulated input encoding* $\tilde{x}_i$ *generated by* $\mathcal{S}$*.*
- Honest Computation Encoding: *Upon* $\mathcal{A}$ *choosing* $(\mathsf{compute}, P_i, z, I)$*, if this is the last honest computation encoding to be generated for computation* $f(z, \star)$ *with* I *(i.e.,* $\forall\ j \neq i \in I \cap H$*,* $\mathcal{A}$ *has queried* $(\mathsf{compute}, P_j, z, I)$ *before), send* $\mathcal{S}$ *the query* $(\mathsf{compute}, P_i, z, I, y)$ *with the output* $y = f(z, \{x_t\}_{t\in I})$*; otherwise, send* $\mathcal{S}$ *the query* $(\mathsf{compute}, P_i, z, I)$ *without* y*. Forward to* $\mathcal{A}$ *the simulated computation encoding* $\tilde{\alpha}_i$ *generated by* $\mathcal{S}$*.*

Above, $\mathcal{A}$ *is restricted to submit one input query for each party* P_i*.*

We we say that a mrNISC protocol is private, we mean semi-maliciously private by default.

B.2 The mrNISC Market Functionality

In this section, we define the mrNISC Market functionality $\mathcal{F}_f$ for computing a general *n-party* single-output function f. Here by n-party, we mean that the function f takes n *private* inputs $(x_1, \cdots, x_n)$ from n input parties. In addition, the function takes an additional *public* input z, shared by the n input parties. An example of such a function is a universal function $f = U$ that interprets z as the actual function g to be computed and outputs $y = g(x_1, \cdots, x_n)$. As we

will see shortly, the existence of the public input allows us to consider multiple computations $f(z, \star)$ with different public inputs on the same set of private inputs.

We define the mrNISC market functionality $\mathcal{F}^f_{\mathsf{mrNISCmarket}}$, which allows an arbitrary (polynomial) number M of parties to register their inputs, and later any subset of n parties can compute $f(z, \star)$ on their private inputs using z of their choice, while enabling an evaluator to obtain the output. Importantly, the computation can happen repeatedly between different subset of parties, and using different public inputs.

Functionality $\mathcal{F}^f_{\mathsf{mrNISCmarket}}$

$\mathcal{F}^f_{\mathsf{mrNISCmarket}}$ parameterized by a function f from $\bigcup_{n=1}^{\infty} (\{0,1\}^*)^{n+1}$ to $\{0,1\}^*$, running with M parties $\mathcal{P} = \{P_1, \dots P_M\}$ for any $M \geq n$, an evaluator E, and an adversary $\mathcal{S}$, proceeds as follows. A subset of the input parties with indexes $\bar{H} \subseteq [M]$ are corrupted while the rest with indexes H are honest. (The functionality is agnostic of whether E is corrupted or not.)

Input: Upon receiving from P_i message $(\mathsf{input}, \mathsf{sid}, P_i, x_i)$ where $x_i \in \mathcal{X}^i$, record the message, send $(\mathsf{input}, \mathsf{sid}, P_i)$ to the evaluator E and the adversary $\mathcal{S}$, and ignore subsequent messages of form $(\mathsf{input}, \mathsf{sid}, P_i, \star)$.

Computation of $f(z, \star)$ with Parties in I: Upon receiving from P_i message $(\mathsf{compute}, \mathsf{sid}, P_i, z, I)$, where $z \in \mathcal{X}_0$ is a public input, and $I \subseteq \mathcal{P}$ is a subset of $|I| = n$ parties, do: Ignore this message if there is no record $(\mathsf{input}, \mathsf{sid}, P_i, x_i)$. Otherwise, forward the same message to the evaluator E and the adversary $\mathcal{S}$, and record the message.

Output: For any (z, I), upon collecting records $(\mathsf{compute}, \mathsf{sid}, P_i, z, I)$ from all *honest* input parties in I (i.e, $H \cap I$), send $(\mathsf{output}, \mathsf{sid}, y)$ to the adversary $\mathcal{S}$, where $y = f(z, \{x_i\}_{i \in I})$.
Upon collecting records $(\mathsf{compute}, \mathsf{sid}, P_i, z, I)$ from all $P_i \in I$ (honest or corrupted), and receiving $(\mathsf{deliver}, \mathsf{sid}, z, I, b_{z,I})$ from the adversary $\mathcal{S}$ where $b_{z,I} \in \{0,1\}$, send $(\mathsf{output}, \mathsf{sid}, y)$ to the evaluator, where y is overwritten to $\perp$ if $b_{z,I} = 0$. In the case where no input party in I is corrupted (i.e., $H \cap I = \emptyset$), $b_{z,I}$ must be 1.

Computation can be invoked many times with different (z, I)'s.
In addition, messages belonging to the input and computation phases can arbitrarily interleave each other.

Fig. 4: General Functionality for mrNISC Market

The mrNISC Market Functionality. Let f be a family of n-party single-output functions. The functionality $\mathcal{F}^f_{\mathsf{mrNISCmarket}}$ for computing f is defined in Figure 4. At a high-level, $\mathcal{F}^f_{\mathsf{mrNISCmarket}}$ interacts with $M \geq n$ input parties $P_1, \cdots, P_M$, an evaluator E, and an adversary $\mathcal{S}$ as follows. First, each party

P_i can register its private input x_i with the functionality. Later in a compute phase, any subset $I \subseteq [M]$ of $|I| = n$ parties can agree to compute f on their private inputs $\{x_i\}_{i\in I}$ and a public input z, and enable the evaluator E (and only E) to obtain the output $y = f(z, \{x_i\}_{i\in I})$. Importantly, the compute phase can be invoked for an arbitrary number of times, where each computation is uniquely identified by the public input used and the set of participants (z, I). Finally, since we achieve only *security with aborts*, for each computation, the functionality always delivers the output to the adversary first, who then decides whether the evaluator obtains the output or not.

B.3 UC-secure mrNISC Protocols from mrNISC Schemes

We show that given an mrNISC scheme for class $\mathcal{C}$, for any $f \in \mathcal{C}$, we construct an mrNISC protocol Π^f for computing f that UC-implements the mrNISC market functionality $\mathcal{F}^f_{\mathsf{mrNISCmarket}}$ against semi-malicious adversaries. The protocol Π^f is described in Figure 5. Note in particular that the input parties do not communicate with each other except for broadcasting the initial input encoding.

mrNISC Protocol Π^f

Π^f parameterized by a function f, running with n input parties $\mathcal{P} = \{P_1, \dots P_M\}$ and an evaluator E proceeds as follows:

Input: Every P_i upon receiving $(\mathsf{input}, \mathsf{sid}, P_i, x_i)$[a], generates $(\hat{x}_i, s_i) \leftarrow \mathsf{Com}(1^\lambda, x_i)$, broadcasts the input encoding $\hat{x}_i$, and keeps the secret s_i.
Any party, P_j or E, upon receiving $\hat{x}_i$ from P_i, ignores this message if $\hat{x}_i = \perp$[b].

Computation of $f(z, \star)$ with parties in I: Every P_i upon receiving $(\mathsf{compute}, \mathsf{sid}, P_i, z, I)$, where $i \in I \subseteq [M]$ and $|I| = n$, do: Ignore the message if P_i has not committed to its input or has not received the input encodings from other parties in I. Otherwise, it generates $\alpha_i \leftarrow \mathsf{Encode}(z, \{\hat{x}_j\}_{j\in I}, s_i)$, and sends (z, I, α_i) to E.

Output: For any (z, I), upon receiving $\hat{x}_i$ and (z, I, α_i) from all P_i with $i \in I$, the evaluator E outputs $\perp$ if any $\alpha_i = \perp$, and outputs $y = \mathsf{Eval}(z, \{\hat{x}_i\}_{i\in I}, \{\alpha_i\}_{i\in I})$ otherwise.

[a] Recall that in the UC framework, the inputs to parties, namely, messages of form $(\mathsf{input}, \mathsf{sid}, P_i, x_i)$ and $(\mathsf{compute}, \mathsf{sid}, z, I)$ are issued by the environment.
[b] Recall that in the semi-malicious model, if the adversary sends any message m without a valid explanation, the message m is overwritten to $\perp$

Fig. 5: A generic mrNISC protocol from an mrNISC scheme

Lemma B.4 (Semi-malicious UC security). *Let* mrNISC *be an mrNISC scheme for f satisfying correctness and privacy. Then the protocol Π^f described in Figure 5 UC-implements $\mathcal{F}^f_{\mathsf{mrNISCmarket}}$ against semi-malicious adversaries.*

Proof. To show that Π^f satisfies semi-malicious UC-security, we need to show that for any adversary $\mathcal{A}'$, there is a simulator $\mathcal{S}'$, such that, for every environment $\mathcal{Z}$, the output of the environment $\mathcal{Z}$ in an execution of the protocol Π^f with $\mathcal{A}'$ and honest parties $\mathcal{P} = \{P_1, \cdots, P_M\}$, is indistinguishable to its output in an execution of $\mathcal{F}^f_{\mathsf{mrNISCmarket}}$ with $\mathcal{S}'$ and $\mathcal{P}$.

Let $\mathcal{S}$ be the simulator guaranteed by the privacy property of mrNISC as in Definition B.3. We construct the simulator $\mathcal{S}'$ using $\mathcal{S}$. Externally, $\mathcal{S}'$ interacts with $\mathcal{F}^f_{\mathsf{mrNISCmarket}}$ and $\mathcal{Z}$; internally, it simulates an execution of Π^f with $\mathcal{A}$, by corrupting the same set of parties as $\mathcal{A}$ does, invoking $\mathcal{S}(f)$, and proceeding as follows:

- *Communication with $\mathcal{Z}$:* Forward all communication between $\mathcal{A}$ and $\mathcal{Z}$.
- *Honest Input Encoding:* Upon receiving $(\mathsf{input}, \mathsf{sid}, P_i)$ for an honest party P_i from $\mathcal{F}^f_{\mathsf{mrNISCmarket}}$, $\mathcal{S}'$ simulates the input encoding $\bar{x}_i$ for $\mathcal{A}$ using $\mathcal{S}$.
- *Corrupted Input Encoding:* Upon receiving $\bar{x}_j$ for a corrupted party P_j from $\mathcal{A}$. Check whether $\mathcal{A}$ outputs a valid explanation (x_j, ρ_j), such that, $(\bar{x}_j, s_j) = \mathsf{Com}(1^\lambda, x_j;\ \rho_j)$. If not, ignore the message (recall that in the semi-malicious model, in the real world, when $\mathcal{A}$ fails to output a valid explanation, its message $\bar{x}_j$ is replaced with $\perp$ and honest parties receiving $\perp$ would ignore this input encoding, as $\mathcal{S}'$ does here). If yes, register this input $(\mathsf{input}, \mathsf{sid}, P_j, x_j)$ to $\mathcal{F}^f_{\mathsf{mrNISCmarket}}$.
- *Honest Computation Encoding:* Upon receiving $(\mathsf{compute}, \mathsf{sid}, P_i, z, I)$ for an honest party P_i from $\mathcal{F}^f_{\mathsf{mrNISCmarket}}$,
  - if *not all* honest parties in I have sent such a message, simulate the computation encoding $\bar{\alpha}_i$ using $\mathcal{S}$;
  - if *all* honest parties in I have sent such a message, wait for message $(\mathsf{output}, \mathsf{sid}, z, I, y)$ from $\mathcal{F}^f_{\mathsf{mrNISCmarket}}$, and then simulate the computation encoding $\bar{\alpha}_i$ using $\mathcal{S}$ with input y.
- *Corrupted Computation Encoding:* Upon receiving $(z, I, \bar{\alpha}_j)$ from a corrupted party P_j controlled by $\mathcal{A}$, send $(\mathsf{compute}, \mathsf{sid}, P_j, z, I)$ to $\mathcal{F}^f_{\mathsf{mrNISCmarket}}$. If $\mathcal{A}$ fails to output a valid explanation of $\bar{\alpha}_j$ — that is, $(x_j, \rho_{j,1}, \rho_{j,2})$, such that, $(\bar{x}_j, s_j) = \mathsf{Com}(1^\lambda, x_j;\ \rho_{j,1})$ and $\bar{\alpha}_j = \mathsf{Encode}(f, z, \{\bar{x}_t\}_{t\in I}, s_j; \rho_{j,2})$ — set $b_{z,I}$, which is initialized to 1, to 0.
- *Output:* For any (z, I), upon $\mathcal{A}$ delivering all input encodings $\{\bar{x}_t\}_{t\in I}$ and all computation encodings $\{\alpha_t\}_{t\in I}$ related to (z, I) to the evaluator E, send $(\mathsf{deliver}, \mathsf{sid}, z, I, b_{z,I})$ to $\mathcal{F}^f_{\mathsf{mrNISCmarket}}$.

To argue that the outputs of the environment $\mathcal{Z}$ in the real and ideal worlds are indistinguishable, it suffices to argue that the view of $\mathcal{A}$ and the outputs of the evaluator E if not corrupted in the real world, is indistinguishable to the view of $\mathcal{A}$ in simulation by $\mathcal{S}'$ and the output of E in the ideal world. This is because in the UC framework, the only information the environment $\mathcal{Z}$ observes is the inputs/outputs of all parties and its communicates with $\mathcal{A}/\mathcal{S}'$. It follows from the privacy property of mrNISC that the real-world view of $\mathcal{A}$ is indistinguishable to its view simulated by $\mathcal{S}$, employed by $\mathcal{S}'$. Furthermore, by construction of the protocol Π^f and the simulator $\mathcal{S}'$, whenever $\mathcal{A}'$ outputs

an input encoding $\bar{x}_j$ without a valid explanation, the input encoding is ignore. Moreover, whenever $\mathcal{A}'$ outputs a computation encoding $\bar{\alpha}_j$ for computation (z, I) without a valid explanation, it leads to the evaluator outputting $\perp$ for that computation. Otherwise, if the input and computation encodings $\{\bar{x}_j, \alpha_j\}_{j \in I}$ related to a computation (z, I) do not contain any $\perp$, in the real world, evaluator would output the valid output $y = f(z, \{x_j\}_{j \in I})$ by the correctness of mrNISC, while in the ideal world, $\mathcal{S}'$ sends the $\mathsf{deliver}$ message with flag $b_{z,I} = 1$ to $\mathcal{F}^f_{\mathsf{mrNISCmarket}}$, who sends E the correct output y. In summary, we conclude that Π^f satisfies semi-malicious UC security. □

Next, we observe that using ZK, our semi-malicious UC-secure protocols can be "lifted" to be malicious UC-secure. If NIZK is used, the protocols remain non-interactive.

Lemma B.5 (Malicious UC Security [4]). *There is a generic transformation that turns Π^f into $\Pi^{f'}$ that UC-implements $\mathcal{F}^f_{\mathsf{mrNISCmarket}}$ against malicious adversaries in the $\mathcal{F}_{\mathsf{ZK}}$-hybrid model, for any $f \in \mathcal{C}$.*

Proof Sketch. The work of [4] presents a transformation that compiles any semi-malicious UC-secure protocol Π using the broadcast (authenticated) channel into a malicious UC-secure protocol Π' in the ideal zero-knowledge $\mathcal{F}_{\mathsf{ZK}}$ hybrid model. The transformation is very simple: Π' is identical to Π except that each party after sending each message proves using $\mathcal{F}_{\mathsf{ZK}}$ that the message correctly generated with respect to a pair of input and random tape (x, ρ), and all the messages this party has received so far. We observe that though our protocol uses both broadcast and P2P channels, the same transformation still applies. This is because each party P_i first broadcast its input encoding $\hat{x}_i$, in which case it can invoke $\mathcal{F}_{\mathsf{ZK}}$ to prove the correctness of $\hat{x}_i$ as in [4]. Later, when party P_i sends an encoding α_i for computation (z, I) to the evaluator E. It can again invoke $\mathcal{F}_{\mathsf{ZK}}$ and prove to E alone that the encoding is well-formed. Note that E can verify this statement as it knows all messages P_i has received so far (in fact, E knows all messages ever sent). □

C Construction of mrNISC Schemes

Let us now show our construction of mrNISC schemes. Our transformation uses the following building blocks:

- A WE scheme for NIZK of commitments for P, $\Pi = (\mathsf{CSetup}_{\mathsf{bind}}, \mathsf{CSetup}_{\mathsf{sim}}, \mathsf{CCom}, \mathsf{CVer}, \mathsf{CSimCom}, \mathsf{CSimOpen}, \mathsf{CProve}, \mathsf{CPVer}, \mathsf{CPSim}, \mathsf{CWEnc}, \mathsf{CWDec})$. (Defined in Section 3 and constructed in Section 3 and D).
- A semi-malicious output-delayed simulatable L-round MPC protocol $\Pi = (\mathsf{Next}, \mathsf{Output})$ for f, as defined in Appendix A.7. Output-delayed simulatability ensures that the transcript excluding the last messages can be simulated for all-but-one honest parties before knowing the output. The reason behind this requirement is that in an mrNISC protocol, only when all the honest parties

agreed to provide a computation encoding, the adversary (and so the simulator) should be able to learn the output.
- A garbled circuit scheme $\mathsf{GC} = (\mathsf{GC.Gen}, \mathsf{GC.Garble}, \mathsf{GC.Eval}, \mathsf{GC.Sim})$ for P.

Our mrNISC scheme is constructed as follows:

- Input: $(\hat{x}_i, s_i) \leftarrow \mathsf{Com}(1^\lambda, x_i)$ samples a PRF key $\mathrm{fk}_i \leftarrow \{0,1\}^\lambda$, generates a binding CRS $\mathsf{crs}_i \leftarrow \mathsf{CSetup}_{\mathsf{bind}}(1^\lambda)$, and uses it to commit to $x_i \| \mathrm{fk}_i$:

$$(c_i, d_i) \leftarrow \mathsf{CCom}(\mathsf{crs}_i, x_i \| \mathrm{fk}_i) \ .$$

Finally, it sets $\hat{x}_i := (\mathsf{crs}_i, c_i)$ and $s_i := (\hat{x}_i, x_i, \mathrm{fk}_i, d_i)$.
- Computation of $f(z, \star)$: $\alpha_i \leftarrow \mathsf{Encode}(f, z, \{\hat{x}_j\}_{j\in[n]}, s_i)$ proceeds as follows:[6]
  - For $\ell \in [L]$, generate input labels that will be used to garble the evaluation circuit F_i^ℓ defined in Fig. 6:

$$(\mathbf{stateKey}_i^\ell, \ \{\mathbf{msgKey}_{i,j}^\ell\}_j) \leftarrow \mathsf{GC.Gen}(1^\lambda) \ .$$

    For $\ell = 1$, all the input labels are empty, as F_i^1 does not take any input. We also define $\mathbf{stateKey}_i^{L+1}$ and $\{\mathbf{msgKey}_{i,j}^{L+1}\}_j$ to be empty strings.
  - For $\ell \in [L]$, garble the evaluation circuit F_i^ℓ:

$$\hat{\mathsf{F}}_i^\ell \leftarrow \mathsf{GC.Garble}((\mathbf{stateKey}_i^\ell, \ \{\mathbf{msgKey}_{i,j}^\ell\}_{j\in[n]}), \ \mathsf{F}_i^\ell) \ .$$

  - Set $\alpha_i := \{\hat{\mathsf{F}}_i^\ell\}_{\ell\in[L]}$.
- Output: $y = \mathsf{Eval}(f, z, \{\hat{x}_i\}_{i\in[n]}, \{\alpha_i\}_{i\in[n]})$ proceeds as follows in L iterations, for $\ell = 1, \ldots, L$:
  - Evaluate the garbled circuits for round ℓ, for $i \in [n]$:

$$\Big(\mathbf{stateKey}_i'^{\ell+1}, \ \{\mathsf{ct}_{i,j,k,b}^\ell\}_{j,k,b}, \ m_i^\ell, \ \{\pi_{i,k}^\ell\}_k\Big) \\ := \mathsf{GC.Eval}(\hat{\mathsf{F}}_i, \ (\mathbf{stateKey}_i'^\ell, \ \{\mathbf{msgKey}_{i,j}^\ell[m_j^{\ell-1}]\}_{j\in[n]})) \ .$$

    We recall that for round $\ell = 1$, all the input labels are empty strings, so the evaluation can be performed.
  - If $\ell \neq L$, decrypt the input labels for the next round, for $i, j \in [n]$ and $k \in [\nu_m]$, define $G_{j,k}^\ell$ as in Fig. 6 and compute:

$$\mathbf{msgKey}_{i,j}^{\ell+1}[m_j^\ell] := \left\{\mathsf{CWDec}(\mathsf{crs}_j, \ \mathsf{ct}_{i,j,k,m_{j,k}^\ell}^{\ell+1}, \ c_j, \ G_{j,k}^\ell, \ m_{j,k}^\ell, \ \pi_{i,j,k}^\ell)\right\}_{k\in[\nu_m]} \ .$$

    At the end, Eval got the full transcript of the inner MPC $\mathsf{Msg} = \{m_j^\ell\}_{j\in[n],\ell\in[L]}$ and set $y := \mathsf{Output}(z, \mathsf{Msg})$.

The correctness of the mrNISC scheme is straightforward: it follows from the perfect correctness properties of the inner MPC protocol, of the WE for NIZK for commitments, and of the garbled circuit scheme.

We now state the security of our mrNISC.

[6] For simplicity, we suppose that the set of parties participating in the computation is $I = [n]$.

Circuit F_i^ℓ

Hardwired Values: 1^λ, ℓ, i, z, $\{\hat{x}_j = (\mathsf{crs}_j, c_j)\}_{j\in[n]}$, $s_i = (\hat{x}_i, x_i, \mathrm{fk}_i, d_i)$, $\mathbf{stateKey}_i^{\ell+1}$, $\{\mathbf{msgKey}_{i,j}^{\ell+1}\}_{j\in[n]}$.

Inputs: $(\mathsf{Msg}^{<\ell-1}, \boldsymbol{m}^{\ell-1})$ where for $\ell > 1$:

- The input messages $\mathsf{Msg}^{<\ell-1}$ are the messages of protocol Π of the first $\ell-2$ rounds. Corresponding garble labels are denoted by $\mathbf{stateKey}_i^\ell$.
- The input messages $\boldsymbol{m}^{\ell-1} := \{m_j^{\ell-1}\}_{j\in[n]}$ are the $\ell-1$ round messages of protocol Π. Corresponding garble labels are denoted by $\{\mathbf{msgKey}_{i,j}^\ell\}_{j\in[n]}$.

Procedure: (for randomized algorithms, randomness is implicitly hardwired)

1. For $j \in [n]$, define the circuit G_j^ℓ by:

$$G_j^\ell(x_j, \mathrm{fk}_j) = \mathsf{Next}_j(z, x_j, \mathsf{PRF}(\mathrm{fk}_j, z\|[\nu_r]), \mathsf{Msg}^{<\ell-1}, \boldsymbol{m}^{\ell-1}) \ ,$$

   and define the circuits $G_{j,k}^\ell$ to output the k'th bit of G_j^ℓ, for $k \in [\nu_m]$.
2. Compute the ℓ-th round message $m_i^\ell = m_{i,1}^\ell \| \cdots \| m_{i,\nu_m}^\ell$ of P_i in the inner protocol Π, and proofs $\pi_{i,k}^\ell$ that each bit has been computed correctly:

$$\begin{aligned} m_i^\ell &\coloneqq G_i^\ell(x_i, \mathrm{fk}_i) \ , \\ \pi_{i,k}^\ell &\leftarrow \mathsf{CProve}(\mathsf{crs}_i, c_i, G_{j,k}^\ell, x_i\|\mathrm{fk}_i, d_i) \quad \text{for } k \in [\nu_m] \ . \end{aligned} \tag{19}$$

3. For $j \in [n]$ and $k \in [\nu_m]$, witness encrypt labels $\mathbf{msgKey}_{i,j}^{\ell+1}[k,b]$ so that $\mathbf{msgKey}_{i,j}^{\ell+1}[m_j^\ell] = \{\mathbf{msgKey}_{i,j}^{\ell+1}[k, m_{j,k}^\ell]\}_{k\in[\nu_m}$ can be decrypted for the valid message m_j^ℓ:

$$\mathsf{ct}_{i,j,k,b}^{\ell+1} \leftarrow \mathsf{CWEnc}(\mathsf{crs}_j, c_j, G_{j,k}^\ell, b, \mathbf{msgKey}_{i,j}^{\ell+1}[k,b]) \qquad \text{for } b \in \{0,1\} \ . \tag{20}$$

   If $\ell = L$, these ciphertexts are set to be empty strings.
4. Select the input labels $\mathbf{stateKey}_i^{\ell+1}[\mathsf{Msg}^{<\ell-1}\|\boldsymbol{m}^{\ell-1}]$ for the next round $(\ell+1)$, corresponding to the messages $\mathsf{Msg}^{<\ell-1}\|\boldsymbol{m}^{\ell-1}$. If $\ell = L$, these values are set to be empty strings.

Output: $(\mathbf{stateKey}_i^{\ell+1}[\mathsf{Msg}^{<\ell-1}\|\boldsymbol{m}^{\ell-1}],\ \{\mathsf{ct}_{i,j,k,b}^{\ell+1}\}_{j,k,b},\ m_i^\ell,\ \{\pi_{i,k}^\ell\}_k)$.

Fig. 6: Circuit F_i^ℓ for the construction of mrNISC in Appendix C

Theorem C.1. *Assume that the* PRF *scheme is pseudorandom, that the* GC *scheme is simulation-secure, that the Π scheme is WE for NIZK of commitments for* P*, that the inner MPC is semi-maliciously delayed-output simulatable, the mrNISC scheme described above is private.*

Proof. The proof is similar to the security of the 2-round MPC protocol from [11]. We construct a simulator $\mathcal{S}$ for the mrNISC. We develop the simulator $\mathcal{S}$ via a sequence of hybrids $H_0, H_{1,1,1}, H_{1,1,2}, H_{1,2,1}, \ldots, H_{1,L,1}$, where H_0 is the real experiment $\mathsf{Exp}_{\mathcal{A},\mathcal{S}}(\mathsf{Real}, \lambda, f)$ and H_3 is the ideal experiment $\mathsf{Exp}_{\mathcal{A},\mathcal{S}}(\mathsf{Ideal}, \lambda, f)$, which also contains the description of Sim.

Hybrid H_0 is identical to the experiment $\mathsf{Exp}_{\mathcal{A},\mathsf{pSim}}(\mathsf{Real}, \lambda, f)$.

Hybrid $H_{1,\ell^\star,1}$ ($\ell^\star \in [L]$) is identical to H_0, except that for all queried honest computation encodings $(\mathsf{compute}, P_i, z, I)$ (we assume for the sake of simplicity that $I = [n]$ and recall that P_i is honest), $\mathcal{S}$ computes the transcript $\mathsf{Msg} = \{m_j^\ell\}_{j\in[n],\ell\in[L]}$ of the inner MPC protocol between parties $\{P_j\}_{j\in I}$ with respective inputs $\{x_j\}_{j\in I}$ and the respective random tapes $\{r_j \coloneqq \mathsf{PRF}(\mathrm{fk}_j \| [\nu_r])\}$, where fk_j can be derived from the randomness ρ_j used to commit to inputs x_j, $(\hat{x}_j, s_j) = \mathsf{Com}(1^\lambda, x_j; \rho_j)$.

1. The simulator $\mathcal{S}$ computes $\pi_{i,k}^\ell$ for $\ell \le \ell^\star$ and $\mathsf{ct}_{i,j,k,b}^{\ell+1}$ for $\ell = \ell^\star$, $j \in [n]$, $k \in [\nu_m]$, $b \in \{0,1\}$, as done by F_i^ℓ described in Fig. 6 (see Eqs. (19) and (20)). It also computes $\mathsf{ct}_{i,j,k,b}^{\ell+1}$ similarly for $\ell < \ell^\star$ for $j \in [n]$, $k \in [\nu_m]$, when $b = G_{j,k}^\ell(x_j, \mathrm{fk}_j) = m_{j,k}^\ell$, but when $b \neq m_{j,k}$, it encrypts instead an arbitrary value $\mathbf{0}$:

$$\mathsf{ct}_{i,j,k,b}^{\ell+1} \leftarrow \mathsf{CWEnc}(\mathsf{crs}_j, c_j, G_{j,k}^\ell, b, \mathbf{0}) \ . \tag{21}$$

2. Instead of honestly garbling $\mathsf{F}_i^{\ell^\star}$ for $\ell \le \ell^\star$, the simulator $\mathcal{S}$ simulates the garbled circuits:

$$((\mathbf{stateKey}_i^\ell[\mathsf{Msg}^{\le \ell-2}],\ \{\mathbf{msgKey}_{i,j}^\ell[m_j^{\ell-1}]\}_{j\in[n]}),\ \hat{\mathsf{F}}_i^\ell) \leftarrow \mathsf{GC.Sim}(1^\lambda, y_i^\ell) \ ,$$

where

$$y_i^\ell \coloneqq \left(\mathbf{stateKey}_i^{\ell+1}[\mathsf{Msg}^{\le \ell-1}],\ \{\mathsf{ct}_{i,j,k,b}^{\ell+1}\}_{j,k,b},\ m_i^\ell,\ \{\pi_{i,k}^\ell\}_k\right) \ .$$

For $\ell^\star = 1$, the only difference between $H_{1,roundstar,1}$ and H_0 is the fact that the garbled circuit $\hat{\mathsf{F}}_i^{\ell^\star}$ is simulated. As $\mathsf{F}_i^{\ell^\star}$ has no inputs, these two hybrids are indistinguishable thanks to simulation security of GC.

Hybrid $H_{1,\ell^\star,2}$ ($\ell^\star \in [L-1]$) is identical to $H_{\ell^\star,1}$, except that for all queried honest computation encodings $(\mathsf{compute}, P_i, z, I)$, $\mathsf{ct}_{i,j,k,b}^{\ell+1}$ also encrypts an arbitrary value $\mathbf{0}$ (Eq. (21)) when $\ell = \ell^\star$ and $b \neq m_{j,k}$.
This hybrid is indistinguishable from $H_{1,\ell^\star,1}$ by semantic security of the witness encryption.
Furthermore, we remark that in $H_{1,\ell^\star,2}$, for $j \in [n]$, $k \in [\nu_m]$, $\ell \le \ell^\star$, only the keys $\mathbf{stateKey}_i^{\ell+1}[\mathsf{Msg}^{\le \ell-1}]$, and $\{\mathbf{msgKey}_{i,j}^{\ell+1}[m_j^\ell]\}_{j\in[n]}$ are used, but

not the full keys $\mathbf{stateKey}_i^{\ell+1}$ and $\{\mathbf{msgKey}_{i,j}^{\ell+1}\}_{j\in[n]}$. In addition, the only difference between $H_{1,\ell^\star+1,1}$ and $H_{1,\ell^\star,2}$ is the fact that in the former $\hat{\mathsf{F}}_i^{\ell^\star+1}$ is also simulated. These two hybrids are indistinguishable thanks to simulation security of GC.

Hybrid H_2 This hybrid is identical to $H_{1,L,1}$ except that for all honest commitments $\hat{x}_i = (\mathsf{crs}_i, c_i)$, crs_i is a simulation CRS: $(\mathsf{crs}_i, \tau_i) \leftarrow \mathsf{CSetup}_{\mathsf{sim}}(1^\lambda)$.
This hybrid is indistinguishable from the previous one ($H_{1,L,1}$) under setup indistinguishability of Π.
We remark that in this hybrid, for all queried honest computation encodings $(\mathsf{compute}, P_i, z, I)$, the simulator $\mathcal{S}$ only uses d_i to compute the proofs $\{\pi_{i,k}^\ell\}_{\ell\in[L],k\in[\nu_m]}$.

Hybrid H_3 This hybrid is identical to H_2, except that all the honest commitments c_i and all the proofs $\{\pi_{i,k}^\ell\}_{\ell\in[L],k\in[\nu_m]}$ generated by the simulator are now simulated:

$$(c_i, \mathsf{aux}_i) \leftarrow \mathsf{CSimCom}(\tau_i) \ ,$$
$$\pi_{i,k}^\ell \leftarrow \mathsf{CPSim}(\mathsf{aux}_i, G_{j,k}^\ell, m_i^\ell) \ .$$

Note we slightly abuse notation as there might be many different $\pi_{i,k}$ for different honest computation encoding queries. The adversary may also query twice the same honest computation encoding query, in which case two fresh proofs are generated. This is not an issue, as the zero-knowledge property we rely one can handle such cases.
This hybrid is indistinguishable from the previous one under zero-knowledge of Π.
We remark that in this hybrid, for all honest computation encodings $(\mathsf{compute}, P_i, z, I)$, the simulator $\mathcal{S}$ only uses the input x_i to compute the transcript $\mathsf{Msg} = \{m_j^\ell\}_{j\in[n],\ell\in[L]}$ of the inner MPC protocol. We recall that the randomness used by party P_i is $r_i \coloneqq \mathsf{PRF}(\mathrm{fk}_j \| [\nu_r])$, and that fk_i is not used directly either (fk_i was committed in c_i in the previous hybrid but now commitments are simulated), but only r_i is used.

Hybrid H_4 This hybrid is identical to H_3, except that the first time an honest computation encoding is queried for some z and I, randomness r_i of all the honest parties P_i from I (i.e., $i \in H \cap I$) is chosen uniformly randomly. The subsequent queries for the same z and I will use the same randomness.
This hybrid is indistinguishable from the previous one under pseudorandomness of PRF.

Hybrid H_5 This hybrid is identical to H_4, except that for any honest computation encoding query $(\mathsf{compute}, P_i, z, I)$:

- If it is the first one for some z and I, the transcript $\mathsf{Msg}^{<L}$ without the last messages $\{m_j^L\}_{j\in H}$ is computed.
- If after this query, all the honest P_j's have been queries for z and I, then m_i^L is simulated using the output $y = f(x_1, \ldots, x_n)$. Otherwise, m_i^L is simulated without using the output y.

This hybrid is well-defined and indistinguishable from the previous one thanks to the semi-malicious output-delayed simulatability of the inner MPC.
In addition, this hybrid corresponds to $\mathsf{Exp}_{\mathcal{A},\mathsf{pSim}}(\mathsf{Ideal},\lambda,f)$, with the simulator $\mathcal{S}$ answering queries as follows:

- *Corrupt Input Encoding*: On query $(\mathsf{input},P_i,x_i,\rho_i)$ for $i\in\bar{H}$ for the first time, $\mathcal{S}$ records $(\mathsf{input},P_i,x_i,\rho_i)$.
- *Honest Input Encoding*: On query (input,P_i) for $i\in H$ for the first time, generates:

$$(\mathsf{crs}_i,\tau_i)\leftarrow\mathsf{CSetup}_{\mathsf{sim}}(1^\lambda)\ ,$$
$$(c_i,\mathsf{aux}_i)\leftarrow\mathsf{CSimCom}(\tau_i)\ .$$

  And outputs $\hat{x}_i:=(\mathsf{crs}_i,c_i)$.
- *Honest Computation Encoding*: On query $(\mathsf{compute},P_i,z,I,y)$, simulate the transcript $\mathsf{Msg}^{<L}$ without the last messages $\{m_j^L\}_{j\in H}$ if it has not already been done, and simulate m_i^L using $y=f(x_1,\ldots,x_n)$ if all P_j's have been queries, and without using y otherwise. Set $\mathbf{stateKey}_i^{L+1}[\mathsf{Msg}^{\le L}]$ and $\mathsf{ct}_{i,j,k,b}^{L+1}$ to be the empty strings. Then, for ℓ from L down to 1, compute the following (for $j\in[n]$ and $k\in[\nu_m]$):

$$\mathsf{ct}^{\ell+1}_{i,j,k,m^\ell_{i,k}}\leftarrow\mathsf{CWEnc}(\mathsf{crs}_j,c_j,G^\ell_{j,k},m^\ell_{i,k},\mathbf{msgKey}^{\ell+1}_{i,j}[k,m^\ell_{i,k}])\ ,$$
$$\mathsf{ct}^{\ell+1}_{i,j,k,1-m^\ell_{i,k}}\leftarrow\mathsf{CWEnc}(\mathsf{crs}_j,c_j,G^\ell_{j,k},1-m^\ell_{i,k},\mathbf{0})\ ,$$
$$\pi^\ell_{i,k}\leftarrow\mathsf{CPSim}(\mathsf{aux}_i,G^\ell_{j,k},m^\ell_i)\ ,$$
$$y^\ell_i:=\left(\mathbf{stateKey}^{\ell+1}_i[\mathsf{Msg}^{\le\ell-1}],\ \{\mathsf{ct}^\ell_{i,j,k,b}\}_{j,k,b},\ m^\ell_i,\ \{\pi^\ell_{i,k}\}_k\right)\ ,$$

$$((\mathbf{stateKey}^\ell_i[\mathsf{Msg}^{\le\ell-2}],\ \{\mathbf{msgKey}^\ell_{i,j}[m_j^{\ell-1}]\}_j),\ \hat{\mathsf{F}}^\ell_i)\leftarrow\mathsf{GC.Sim}(1^\lambda,y^\ell_i)\ ,$$

  and output $\alpha_i=\{\hat{\mathsf{F}}^\ell_i\}_{\ell\in[L]}$

This concludes the proof. □

D WE for NIZK of Commitments: From NC^1 to P

In Section 3, we constructed WE for NIZK of commitments for NC^1. In this section, we show the transformation from WE for NIZK of commitments for NC^1 to a scheme for P and prove its security. The construction follows the technical overview (Section 2.3). Below, we first quickly recall the notions of computational randomized encodings, which is used in this transformation.

Preliminary: Computational Randomized Encodings. We recall the definition of computational randomized encodings from [3,37].

Definition D.1 (Computational Randomized Encoding). Let $\mathcal{G}$ be a class of polynomial-size circuits. A *computational randomized encoding scheme* for $\mathcal{G}$ is a tuple of three polynomial-time algorithms $(\mathsf{CRE.Enc}, \mathsf{CRE.Dec}, \mathsf{CRE.Sim})$ with the following syntax:

- Encoding: $\hat{G} \coloneqq \mathsf{CRE.Enc}(1^\lambda, G)$ on input the security parameter, a circuit $G \in \mathcal{G}$ ($G\colon \{0,1\}^n \to \{0,1\}^\ell$), outputs another circuit called a *computational randomized encoding* $\hat{G}\colon \{0,1\}^n \times \{0,1\}^m \to \{0,1\}^s$. $\mathsf{CRE.Enc}$ is deterministic.
- Decoding: $y \coloneqq \mathsf{CRE.Dec}(1^\lambda, G, \hat{y})$ on input the security parameter, a circuit $G \in \mathcal{G}$ ($G\colon \{0,1\}^n \to \{0,1\}^\ell$), and the output of $\hat{y}$ of the randomized encoding $\hat{G}$, outputs y the output of G. $\mathsf{CRE.Dec}$ is deterministic.
- Simulation: $\hat{G} \leftarrow \mathsf{CRE.Sim}(1^\lambda, G, y)$ on input the security parameter, a circuit $G \in \mathcal{G}$ ($G\colon \{0,1\}^n \to \{0,1\}^\ell$), and an output $y \in \{0,1\}^\ell$, outputs a simulated computational randomized encoding $\hat{G}$.

and satisfying the following properties:

Perfect Correctness. For every security parameter $\lambda \in \mathbb{N}$, for every circuit $G \in \mathcal{G}$ ($G\colon \{0,1\}^n \to \{0,1\}^\ell$), for every input $v \in \{0,1\}^\ell$, for every bit string $r \in \{0,1\}^m$, if $\hat{G} = \mathsf{CRE.Enc}(1^\lambda, G)$, then $\mathsf{CRE.Dec}(1^\lambda, G, \hat{G}(v,r)) = G(v)$.

Privacy. For every circuit $G \in \mathcal{G}$ ($G\colon \{0,1\}^n \to \{0,1\}^\ell$), for every input $v \in \{0,1\}^\ell$, the following two distributions are computationally indistinguishable:

$$\left\{\hat{G} \coloneqq \mathsf{CRE.Enc}(1^\lambda, G),\ r \leftarrow \{0,1\}^m\ :\ \hat{G}(v,r)\right\}\ ,$$
$$\left\{\mathsf{CRE.Sim}(1^\lambda, G, G(v)\right\}\ .$$

We focus on computational randomized encodings in NC^1, namely where m, s are both polynomial in n, λ, and the size of G; and where $\hat{G}$ is in NC^1.

Our definition slightly diverges from [3, 37]: we introduce an explicit computational randomized encoder, as to make explicit the fact that computational randomized encoding must be efficiently computable.

Remark D.2. We require a computational randomized encoding scheme to be perfectly correct. Without this, perfect soundness (or even soundness) of the NIZK for commitments might not hold. In the construction below, nothing would prevent the adversary from choosing a PRF fk inducing some bad randomness for which correctness does not hold.

As NC^1 contains NC^0, and as the existence of a PRF in NC^1 implies the Easy PRG assumption from [3], we have the following theorem

Theorem D.3 (Corollary of [3]). *Assuming the existence of a PRF in* NC^1*, there exists a computational randomized encoding scheme (with computational randomized encodings in* NC^1*) for* P*.*

From $\mathbf{NC^1}$ to P. Our construction uses the following building blocks:

- A WE for NIZK of commitments for NC^1, $\Pi = (\mathsf{CSetup}_{\mathsf{bind}}, \mathsf{CSetup}_{\mathsf{sim}}, \mathsf{CCom}, \mathsf{CVer}, \mathsf{CSimCom}, \mathsf{CSimOpen}, \mathsf{CProve}, \mathsf{CPVer}, \mathsf{CPSim}, \mathsf{CWEnc}', \mathsf{CWDec}')$, with message space $\mathcal{V} := \{0,1\}^{n+\lambda}$.
- A PRF $\mathsf{PRF}\colon \{0,1\}^\lambda \times \{0,1\}^{\mathrm{poly}(\lambda)} \to \{0,1\}$. We recall the notation defined in Remark A.5.
- A collision-resistant family of hash functions $\mathcal{HF}$.
- A computational randomized encoding scheme $(\mathsf{CRE.Enc}, \mathsf{CRE.Dec}, \mathsf{CRE.Sim})$ for P with computational randomized encodings in NC^1.

We construct below $\Pi' = (\mathsf{CSetup}'_{\mathsf{bind}}, \mathsf{CSetup}'_{\mathsf{sim}}, \mathsf{CCom}', \mathsf{CVer}', \mathsf{CSimCom}', \mathsf{CSimOpen}', \mathsf{CProve}', \mathsf{CPVer}', \mathsf{CPSim}', \mathsf{CWEnc}, \mathsf{CWDec})$ a WE for NIZK of commitments for P with message space $\mathcal{V} := \{0,1\}^n$:

- Setup: $\mathsf{CSetup}'_{\mathsf{bind}}$ and $\mathsf{CSetup}'_{\mathsf{sim}}$ are the same as $\mathsf{CSetup}_{\mathsf{bind}}$ and $\mathsf{CSetup}_{\mathsf{sim}}$ except that they also sample a collision-resistant hash function $\mathcal{H} \leftarrow \mathcal{HF}$
- Commitment: $(c', d') \leftarrow \mathsf{CCom}'(\mathsf{crs}, v)$ for $v \in \mathcal{V} := \{0,1\}^n$, generates $\mathrm{fk} \leftarrow \{0,1\}^\lambda$, and generates

$$(c, d) \leftarrow \mathsf{CCom}(\mathsf{crs}, v \| \mathrm{fk}) \ ,$$

  and outputs $(c', d') := (c, (d, \mathrm{fk}))$.
- Verification: $\mathsf{CVer}'(\mathsf{crs}, c', v, d' = (d, \mathrm{fk}))$ returns $\mathsf{CVer}(\mathsf{crs}, c, v \| \mathrm{fk}, d)$.
- Commitment Simulation: $(c', \mathsf{aux}') \leftarrow \mathsf{CSimCom}'(\tau)$ computes

$$(c, \mathsf{aux}) \leftarrow \mathsf{CSimCom}(\tau) \ ,$$

  picks $\mathrm{fk}' \leftarrow \{0,1\}^\lambda$, and outputs $(c', \mathsf{aux}') := (c, (\mathsf{aux}, \mathrm{fk}'))$.
- Commitment Opening: $d' \leftarrow \mathsf{CSimOpen}'(\mathsf{aux}' = (\mathsf{aux}, \mathrm{fk}'), v)$ generates $\mathrm{fk} \leftarrow \{0,1\}^\lambda$, computes $d \leftarrow \mathsf{CSimOpen}(\mathsf{aux}, v \| \mathrm{fk})$ and outputs $d' := (d, \mathrm{fk})$.
- Proof: $\pi' \leftarrow \mathsf{CProve}'(\mathsf{crs}, c', G, v, d' = (d, \mathrm{fk}))$ computes $h := \mathcal{H}(G)$ as well as the computational randomized encoding $\hat{G} := \mathsf{CRE.Enc}(1^\lambda, G)$, where $\hat{G}\colon \{0,1\}^n \times \{0,1\}^m \to \{0,1\}^s$, defines the circuits $\{G'_i\}_{i\in[s]}$ and computes $\hat{y}$ and $\{\pi_i\}_{i\in[s]}$ as follows, for $i \in [s]$:

$$G'_i(v \| \mathrm{fk}) := [\hat{G}(v, \mathsf{PRF}(\mathrm{fk}, h \| [s]))]_i \ , \tag{22}$$

$$\hat{y} := \hat{G}(v, \mathsf{PRF}(\mathrm{fk}, h \| [s])) \ , \tag{23}$$

$$\pi_i \leftarrow \mathsf{CProve}(\mathsf{crs}, c', G'_i, v \| \mathrm{fk}, d) \ ,$$

  where $[x]_i$ denotes the i'th bit of the bit string x. This is possible as G' is in NC^1, as both PRF and $\hat{G}$ are. Then, CProve' outputs $\pi' := (\hat{y}, \{\pi_i\}_{i\in[s]})$.
- Proof Verification: $\mathsf{CPVer}(\mathsf{crs}, c', G, y, \pi' = (\hat{y}, \{\pi_i\}_{i\in[s]}))$ defines $\{G'_i\}_{i\in[s]}$ as in Eq. (22) and outputs 1 is and only if $\mathsf{CRE.Dec}(1^\lambda, G, \hat{y}) = y$ and $\mathsf{CPVer}(\mathsf{crs}, c', G'_i, \hat{y}_i, \pi) = 1$ for $i \in [s]$, where $\hat{y}_i$ is the i-bit of $\hat{y}$.

– Proof Simulation: $\pi' \leftarrow \mathsf{CPSim}'(\mathsf{aux}' = (\mathsf{aux}, \mathrm{fk}'), G, y)$ computes:

$$\begin{aligned} \rho &\coloneqq \mathsf{PRF}(\mathrm{fk}', G \| [\nu]) \ , \\ \hat{G} &\coloneqq \mathsf{CRE.Sim}(1^\lambda, G, y; \rho) \ , \end{aligned} \tag{24}$$

with ν being the size of the randomness used by $\mathsf{CRE.Sim}$. π' then defines $\{G'_i\}_{i\in[s]}$ as in Eq. (22), and computes for $i \in [s]$:

$$\pi'_i \leftarrow \mathsf{CPSim}'(\mathsf{aux}', G'_i, \hat{y}_i) \ , \tag{25}$$

and outputs $\pi' \coloneqq (\hat{y}, \{\pi_i\}_{i\in[s]})$. Deriving the randomness ρ from fk' is important to ensure that the same $\hat{y}$ is used each time CPSim' is called for the same circuit G.

– Witness Encryption: $\mathsf{ct} \leftarrow \mathsf{CWEnc}'(\mathsf{crs}, c', G, y, \mathsf{m})$ defines $\{G'_i\}_{i\in[s]}$ as in Eq. (22), as well as the following circuit C:

$$C(\hat{y}) = \begin{cases} \mathsf{m} & \text{if } \mathsf{CRE.Dec}(1^\lambda, G, \hat{y}) = y \ , \\ \bot & \text{otherwise.} \end{cases} \tag{26}$$

It then garbles C:

$$\begin{aligned} \mathsf{key} \coloneqq \{\mathsf{key}[i,b]\}_{i\in[s],b\in\{0,1\}} &\leftarrow \mathsf{GC.Gen}(1^\lambda) \ , \\ \widehat{C} &\leftarrow \mathsf{GC.Garble}(\mathsf{key}, C) \ , \end{aligned}$$

and generates the following ciphertexts for $i \in [s]$ and $b \in \{0,1\}$:

$$\mathsf{ct}_{i,b} \leftarrow \mathsf{CWEnc}(\mathsf{crs}, c', G'_i, b, \mathsf{key}[i,b]) \ .$$

It then outputs $\mathsf{ct}' \coloneqq (\widehat{C}, \{\mathsf{ct}_{i,b}\}_{i\in[s],b\in\{0,1\}})$.

– Witness Decryption: $\mathsf{CWDec}'(\mathsf{crs}, \mathsf{ct}', c', G, y, \pi)$ where $\pi' = (\hat{y}, \{\pi_i\}_{i\in[s]})$ and $\mathsf{ct}' = (\widehat{C}, \{\mathsf{ct}_{i,b}\}_{i\in[s],b\in\{0,1\}})$, decrypts the following ciphertexts for $i \in [s]$:

$$\mathsf{key}'[i] \coloneqq \mathsf{CWDec}(\mathsf{crs}, c', G'_i, \hat{y}_i, \pi'_i) \ .$$

Finally, it outputs $\mathsf{GC.Eval}(\widehat{C}, \mathsf{key}')$.

Theorem D.4. *Assuming Π is a WE for NIZK of commitments for* NC^1*,* PRF *is pseudorandom, $\mathcal{HF}$ is collision-resistant,* $(\mathsf{CRE.Enc}, \mathsf{CRE.Dec}, \mathsf{CRE.Sim})$ *is a computational randomized encodings, the construction Π' described above is a WE for NIZK of commitments for* P*.*

Proof. **Perfect correctness** of the commitment, **setup indistinguishability**, **perfect binding**, and **perfect equivocality** are straightforward.

Perfect proof correctness follows from perfect correctness of the computational randomized encoding.

Perfect soundness follows from perfect soundness of Π and perfect correctness of the computational randomized encoding. We remark that collisions

in hash function does not break soundness and that we do not need either to rely on the pseudorandomness property of the PRF, because the computational randomized encoding is perfectly correct.

Let us prove the **zero-knowledge property**. We do a proof by games:

- Game 0 corresponds to the zero-knowledge game where proofs are honestly generated.
- Game 1 is similar to Game 0 except we abort if the adversary queries two distinct circuits G_0 and G_1 with the same hash value: $\mathcal{H}(G_0) = \mathcal{H}(G_1)$. This game is indistinguishable from the previous one thanks to the collision resistance property of the hash function family. We can now suppose that all the values h are different.
- Game 2 is similar to Game 1, except that the proofs π'_i are simulated as in Eq. (25). The randomized encoding $\hat{G}$ and the circuits $\{G'_i\}_{i\in[s]}$ are still computed as before. This game is indistinguishable from the previous one thanks to the zero-knowledge property of Π.
- Game 3 is similar to Game 2, except that $\hat{y}$ is generated with fresh randomness $r \leftarrow \{0,1\}^m$ (for each distinct G, but the same r for the same G) instead of randomness derived from the PRF:

$$\hat{y} := \hat{G}(v,\, r) \ .$$

  This game is indistinguishable from the previous one thanks to the pseudorandomness property of PRF (and the fact that all the values h queries for a given commitment c are distinct).
- Game 4 is similar to Game 3, except that the randomized encoding $\hat{G}$ is now computed as:

$$\hat{G} \leftarrow \mathsf{CRE.Sim}(1^\lambda, G, y; \rho) \ ,$$

  where ρ is fresh randomness for each distinct G. This game is indistinguishable from the previous one thanks to the privacy property of the computational randomized encoding.
- Game 5 is similar to Game 4, except that ρ is generated as by CPSim' in Eq. (24). This game is indistinguishable from the previous one thanks to the pseudorandomness property of PRF.

As Game 5 corresponds to the zero-knowledge game where proofs are simulated, this conclude the proof of zero-knowledge.

Perfect encryption correctness follows from perfect correctness of the computational randomized encoding, perfect encryption correction of Π, and perfect correctness of the garbled circuit.

Let us now prove **encryption semantic security**. From the semantic security of Π, everything can be simulated knowing only $\{\mathsf{key}[i, \hat{y}_i]\}_i$, where $\hat{y}_i$ corresponds to the outputs of G'_i applied on the committed value of c. By perfect correctness of the randomized encodings, $\hat{y} = \hat{y}_1 \| \cdots \| \hat{y}_m$ is always decoded by $\mathsf{CRE.Dec}$ to $G(v) \neq y$. We conclude using the security of the garbled circuit scheme.

□

E Applications of mrNISC

In this section, we formally describe two applications of mrNISC — transforming NIMPC protocols with correlated randomness into NIMPC protocols in the *PKI* supplemented with a CRS model, and constructing *secret-sharing VBB* obfuscation.

E.1 NIMPC: From Correlated Randomness to the PKI Setting

Non-interactive MPC (NIMPC) [9] is a very powerful notion equivalent, under different corruption model, to garbled circuits, Private Simultaneous Messages protocols [36], and obfuscation. It enables n parties $P_1, \ldots, P_n$ to securely compute a function f on their private inputs $x_1, \ldots, x_n$ by each party P_i sending just one message, m_{i,x_i}, to an evaluator E. The key difference between NIMPC and mrNISC is that mrNISC requires parties to broadcast initially an encoding $\hat{x}_1, \ldots, \hat{x}_n$, of their inputs $x_1, \ldots, x_n$, and later computation of f on $x_1, \ldots, x_n$ is enabled by sending *another* message $\alpha_1, \ldots, \alpha_n$ to the evaluator. Therefore, if considering only a single computation, each party in an mrNISC protocol sends in total two messages $(\hat{x}_i, \alpha_i)$, whereas each party in an NIMPC protocol sends only a single message. However, since mrNISC supports reusing the input encodings, the *amortized* round complexity across many computations is just one. In addition, mrNISC provides stronger security guarantees — adversaries corrupting up to n parties learn only the outputs of the computation. In contrast, NIMPC, due to the lack of initial commitments to parties' inputs, inherently allows the adversaries to learn (in the sense of having black-box oracle access to) the *residual function* $f(\boldsymbol{x}_H, \star)$ with honest parties' input $\boldsymbol{x}_H$ hardcoded in. In other words, mrNISC and NIMPC are the optimal solutions in two different settings.

As an application of mrNISC, we present a generic transformation from any NIMPC protocol using correlated randomness into a protocol in the PKI model if the adversary is semi-malicious, or a protocol in the PKI plus a CRS if the adversary is malicious. The key difference between the PKI model and correlated randomness lies in that the PKI is *reusable*: once parties have published their public keys via the PKI, they can compute many *different* functions f^k on *different* private inputs $(x_1^k, \ldots, x_n^k)$ by just sending one message to an evaluator. On the other hand, correlated randomness cannot be reused and requires a trusted third party to sample them independently for each computation and distribute them. Beyond this key difference, PKI can also be seen as a minimal form of correlated randomness, in the sense that the key pair of each party can be sampled independently and corrupted parties can control their public keys. Our transformation preserves the efficiency of the underlying NIMPC protocol, as well as its *simulation*-security, admitting a simulator that is slower by a multiplicative polynomial factor, compared to the simulator of the underlying NIMPC. (Indistinguishability security can be viewed as simulation security with an unbounded simulator, and hence is also preserved.) Applying our transformation to known NIMPC protocols [9, 10, 35] yields their counterpart protocols in the PKI plus

CRS model, under the same assumption underlying the original protocols plus the existence of mrNISC for P and UC-NIZK for NP, which in turn can be based on either a circular secure variant of LWE or the SXDH assumption on bilinear maps.

Below, we start with describing the definitions of NIMPC with correlated randomness and in the PKI model, and then the generic transformation from the former to the latter. Finally, we note corollaries obtained by applying our transformation to known NIMPC protocols with correlated randomness.

Definition of NIMPC with Correlated Randomness. We recall the definition of NIMPC protocols from [9]. We will use the following notations: For a function $f\colon \mathcal{X} \to \Omega$ where $\mathcal{X} \coloneqq \mathcal{X}_1 \times \cdots \times \mathcal{X}_n$, we denote by $f|_{H,\boldsymbol{x}_H}$ the function f with the inputs corresponding to positions H fixed to the entries of vector $\boldsymbol{x}_H$.

Definition E.1 (NIMPC Protocol). *Let $\mathcal{C} = \{\mathcal{C}_n\}_{n\in\mathbb{N}_{>0}}$ be a family of sets $\mathcal{C}_n$ of functions of the form: $f\colon \mathcal{X}_1 \times \cdots \times \mathcal{X}_n \to \Omega$ (functions in $\mathcal{C}_n$ have n inputs, but the input sets can be different for each function). A* non-interactive secure multiparty computation (NIMPC) protocol *for $\mathcal{C}$ is a tuple of three algorithms $\mathsf{NIMPC} = (\mathsf{Setup}, \mathsf{Msg}, \mathsf{Rec})$, where:*

- *Sample correlated randomness:* $(\rho_0, \rho_1, \ldots, \rho_n) \leftarrow \mathsf{Setup}(1^\lambda, 1^n, f)$*, on input the unary representations of number of inputs n and of the security parameter λ, and (a representation of) a function $f \in \mathcal{C}_n$, outputs a tuple $(\rho_0, \rho_1, \ldots, \rho_n)$;*[7]
- *Encode Input:* $m_{i,x_i} = \mathsf{Msg}(\rho_i, x_i)$ *on input a value ρ_i and an input x_i, outputs a message m_{i,x_i};*
- *Reconstruct Output:* $y = \mathsf{Rec}(\rho_0, (m_{i,x_i})_{i=[n]})$ *on input a value ρ_0 and a tuple of n messages $(m_{i,x_i})_{i=[n]}$, outputs an element of $\Omega \cup \{\perp\}$.*

satisfying the following property:

Correctness: For any values $n \in \mathbb{N}$, security parameter $\lambda \in \mathbb{N}$, $f \in \mathcal{C}_n$, and $\boldsymbol{x} \in \mathcal{X}$, the following holds

$$\Pr\left[\begin{matrix}(\rho_0,\ldots,\rho_n) \leftarrow \mathsf{Setup}(1^\lambda, 1^n, f) \\ \forall i \in [n],\ m_{i,x_i} \leftarrow \mathsf{Msg}(\rho_i, x_i)\end{matrix} \ :\ \mathsf{Rec}(\rho_0, m_{i,x_1}, \ldots, m_{n,x_n}) = f(\boldsymbol{x})\right] = 1 \ .$$

While the previous definition is abstract, in the sequel, we will often view NIMPC protocols as protocols with n parties $P_1, \ldots, P_n$ with respective inputs $x_1, \ldots, x_n$, and an evaluator E. More precisely, $\mathsf{NIMPC} = (\mathsf{Setup}, \mathsf{Msg}, \mathsf{Rec})$ yields a protocol in three phases as follows:

Offline preprocessing. For the security parameter λ and the function $f \in \mathcal{C}_n$, a trusted party generates $(\rho_0, \rho_1, \ldots, \rho_n) \leftarrow \mathsf{Setup}(1^n, 1^\lambda, f)$ and gives ρ_i to party P_i (for $i \in [n]$) and ρ_0 to the evaluator E.

Online messages. On input x_i, party P_i computes $m_{i,x_i} \coloneqq \mathsf{Msg}(\rho_i, x_i)$ and outputs m_{i,x_i} to the evaluator E.

[7] One refers to the vector $(\rho_0, \rho_1, \ldots, \rho_n)$ as the *correlated randomness* of the parties, with ρ_0 called *public randomness*.

Reconstruction. After receiving m_{i,x_i} from all the parties P_i (for $i \in [n]$), the evaluator E computes and outputs $\mathsf{Rec}(\rho_0, m_{1,x_1}, \ldots, m_{n,x_n})$.

We now recall the notions of robustness for NIMPC protocols. Informally, $\bar{H}$-robustness for a set $\bar{H} \subseteq [n]$ of colluding parties means that if $\boldsymbol{x}_H$ represents the inputs of the honest parties, then an evaluator colluding with the parties in set $\bar{H}$ can compute the residual function $f|_{H,\boldsymbol{x}_H}$ on any input $\boldsymbol{x}_{\bar{H}}$ but cannot learn anything else about the input of the honest parties. This describes the best privacy guarantee attainable in this adversarial setting. The formal definition is stated in terms of a simulator that can generate the view of the adversary (evaluator plus the colluding parties in set $\bar{H}$) with sole oracle access to the residual function $f|_{H,\boldsymbol{x}_H}$.

Since our transformation is computational, and would always produce a NIMPC protocol with computational security, even if the starting point protocol is statistically secure. Below, we only recall computational security.

Definition E.2 (NIMPC Robustness). *Let $n \in \mathbb{N}$ be a positive integer, $H \subseteq [n]$ be a subset. An NIMPC protocol* NIMPC *is computationally* $\bar{H}$-robust *if there exists a randomized algorithm* Sim *(called a* simulator*) such that, the following distributions are computationally indistinguishable:*

$$\{\mathsf{Sim}^{f|_{H,\boldsymbol{x}_H}}(1^n, 1^\lambda, f, \bar{H})\}_{\lambda,n,f\in\mathcal{C}_n,\boldsymbol{x}_H\in\mathcal{X}_H}$$
$$\approx\{\mathsf{View}(1^n, 1^\lambda, f, \bar{H}, \boldsymbol{x}_H)\}_{\lambda,n,f\in\mathcal{C}_n,\boldsymbol{x}_H\in\mathcal{X}_H},$$

where $\mathsf{View}(1^n, 1^\lambda, f, \bar{H}, \boldsymbol{x}_H)$ *is the view of the evaluator E and of the colluding parties P_i (for $i \in \bar{H}$) from running* NIMPC *on inputs $\boldsymbol{x}_H$ for the honest parties P_i (for $i \in H$): namely, $((m_{i,x_{H,i}})_{i\in H}, \rho_0, (\rho_i)_{i\in\bar{H}})$ where $(\rho_0, \ldots, \rho_n) \leftarrow \mathsf{Setup}(1^n, 1^\lambda, f)$ and $m_{i,x_{H,i}} \leftarrow \mathsf{Msg}(\rho_i, x_{H,i})$ for $i \in H$.*

Let t be an integer which is a function of n, then an NIMPC protocol Π is computationally t-robust *if for any $n \in \mathbb{N}$ and any subset $\bar{H} \subseteq [n]$ of size at most $t = t(n)$, Π is computationally t-robust. It is computationally* fully robust*, if it is computationally n-robust.*

Note that robustness does not necessarily imply that the simulator Sim is the same for any n and $\bar{H}$ nor that it runs in polynomial time in n, λ, and $|f|$. Our transformation preserves the efficiency of the simulator.

Definition of NIMPC in the PKI model. We now recall the definition of NIMPC in the (reusable) PKI model from [35], by describing the difference in syntax and security requirements from the definition of NIMPC with correlated randomness above. Since the PKI is reusable, the robustness property must guarantee simulation of multiple computation sessions sharing the same PKI. In order to prevent adversaries from mixing-and-matching messages generated for different computation sessions, each session is identified by a unique session id sid.

Definition E.3 (NIMPC Protocol in the PKI model). *A NIMPC protocol* in the PKI model *for $\mathcal{C}$ is a tuple of three algorithms* $\mathsf{pNIMPC} = (\mathsf{pKgen}, \mathsf{pMsg}, \mathsf{pRec})$ *with the following syntax:*

- *Key Generation:* $(\mathrm{pk}, \mathrm{sk}) \leftarrow \mathsf{pKgen}(1^\lambda)$ *on input the unary representations of the security parameter samples a pair of public and secret key.*
  For n parties, where the i'th ($i \in [n]$) party holds keys $(\mathrm{pk}_i, \mathrm{sk}_i)$, the correlated randomness is set to $\rho_0 = \mathbf{pk} = \{\mathrm{pk}_j\}_{j\in[n]}$, and for every $i \in [n]$, $\rho_i = (\mathbf{pk}, \mathrm{sk}_i, i)$.
- *Encode Input:* $m_{i,x_i} \leftarrow \mathsf{pMsg}(\rho_i, x_i, f, \mathsf{sid})$ *on input a value ρ_i, an input x_i, the description of a function $f \in \mathcal{C}_n$, and a session id $\mathsf{sid} \in \{0,1\}^\lambda$, outputs a message m_{i,x_i};*
- *Reconstruct Output:* $y = \mathsf{pRec}(\rho_0, \{m_{i,x_i}\}_{i\in[n]})$ *on input a value ρ_0 and a tuple of n messages $\{m_{i,x_i}\}_{i\in[n]}$, outputs an element of $\Omega \cup \{\perp\}$.*

satisfying the following

Correctness: For any values $n \in \mathbb{N}$, security parameter $\lambda \in \mathbb{N}$, $f \in \mathcal{C}_n$, $\boldsymbol{x} \in \mathcal{X}$, and $\mathsf{sid} \in \{0,1\}^\lambda$, the following holds

$$\Pr\left[\begin{array}{ll} \forall i \in [n],\ (\mathrm{pk}_i, \mathrm{sk}_i) \leftarrow \mathsf{pKgen}(1^\lambda) & \\ \rho_0 = \mathbf{pk} = \{\mathrm{pk}_j\}_{j\in[n]} & \\ \forall i \in [n],\ \rho_i = (\mathbf{pk}, \mathrm{sk}_i, i) & : \ \mathsf{pRec}(\rho_0, (m_{i,x_i})_{i=[n]}) = f(\boldsymbol{x}) \\ \forall i \in [n],\ m_{i,x_i} \leftarrow \mathsf{pMsg}(\rho_i, x_i, f, \mathsf{sid}) & \end{array}\right] = 1 \ .$$

Next, we give simulation based security definition of robustness in the PKI model, against *semi-malicious* or *malicious* adversaries. Semi-malicious adversaries choose their public and secret keys using the pKgen algorithm with an arbitrarily chosen random tape, and malicious adversaries are not bound to follow the pKgen algorithm. Since the PKI is shared between different computations sessions, we directly define simulation of multiple sessions. The work of [35] defined indistinguishability-based security in the PKI model, which can be obtained by allowing the simulator in our definition to run in unbounded time.

Definition E.4 (NIMPC Robustness in the PKI Model). *Let $n \in \mathbb{N}$ be a positive integer, $H \subseteq [n]$ be a subset. An NIMPC protocol* NIMPC *in the PKI model is semi-maliciously (or maliciously) $\bar{H}$-*robust *if there exists a randomized algorithm $\mathsf{Sim} = (\mathsf{Sim}_1, \mathsf{Sim}_2)$ (called a* simulator*) such that, for every PPT adversary $\mathcal{A}$, its views in the following experiments are computationally indistinguishable:*

Experiment $\mathsf{pExp}_{\mathcal{A},\mathsf{Sim}}(\mathsf{Real}, 1^\lambda, 1^n)$ *interacts with $\mathcal{A}(1^\lambda, 1^n)$ in following steps:*

1. *Send honestly generated $\mathbf{pk}_H$ to $\mathcal{A}$, where*

$$\forall i \in H, \qquad (\mathrm{pk}_i, \mathrm{sk}_i) \leftarrow \mathsf{pKgen}(1^\lambda) \ .$$

2. *Semi-malicious $\mathcal{A}$ outputs $(\mathbf{pk}_{\bar{H}}, \boldsymbol{r}_{\bar{H}})$. Verify that $\forall i \in \bar{H}$, $(\mathrm{pk}_i, \mathrm{sk}_i) = \mathsf{pKgen}(1^\lambda;\ r_i)$, and abort if not. (Malicious $\mathcal{A}$ does not send the explanation $\boldsymbol{r}_{\bar{H}}$.)*

3. *Interact with $\mathcal{A}$ in many iterations until it terminates. In iteration k, upon $\mathcal{A}$ choosing a function $f^k \in \mathcal{C}_n$, a set of inputs for honest parties $\boldsymbol{x}_H^k \in \mathcal{X}_H$, and a session id $\mathsf{sid}^k \in \{0,1\}^\lambda$, send $\mathcal{A}$ messages $\boldsymbol{m}_{H,\boldsymbol{x}_H}^k$ generated honestly as*

$$\forall i \in H,\ m_{i,x_i}^k \leftarrow \mathsf{pMsg}((\mathbf{pk}, \mathrm{sk}_i, i), x_i^k, f^k, \mathsf{sid}^k)\ ,$$

if sid^k was never used before, or send $\perp$ otherwise.

Experiment $\mathsf{pExp}_{\mathcal{A},\mathsf{Sim}}(\mathsf{Ideal}, 1^\lambda, 1^n)$ *interacts with $\mathcal{A}(1^\lambda, 1^n)$ in following steps:*

1. *Send simulated $\widetilde{\mathbf{pk}}_H$ to $\mathcal{A}$, where*

$$(\widetilde{\mathbf{pk}}_H,\ \mathsf{st}^0) \leftarrow \mathsf{Sim}_1(1^\lambda, 1^n, H)\ .$$

2. *Semi-malicious $\mathcal{A}$ outputs $(\mathbf{pk}_{\bar{H}}, \boldsymbol{r}_{\bar{H}})$. Verify that $\forall i \in \bar{H}$, $(\mathrm{pk}_i, \mathrm{sk}_i) = \mathsf{pKgen}(1^\lambda;\ r_i)$, and abort if not. Update $st^0 = st^0 \| (\mathbf{pk}_{\bar{H}}, \boldsymbol{r}_{\bar{H}})$. (Malicious $\mathcal{A}$ does not send the explanation $\boldsymbol{r}_{\bar{H}}$. Update $st^0 = st^0 \| \mathbf{pk}_{\bar{H}}$.)*
3. *Interact with $\mathcal{A}$ in many iterations until it terminates. In iteration k, upon $\mathcal{A}$ choosing a function $f^k \in \mathcal{C}_n$, a set of inputs for honest parties $\boldsymbol{x}_H^k \in \mathcal{X}_H$, and a session id $\mathsf{sid}^k \in \{0,1\}^\lambda$, send $\mathcal{A}$ messages $\widetilde{\boldsymbol{m}}_H^k$ simulated as*

$$(\widetilde{\boldsymbol{m}}_H^k,\ \mathsf{st}^k) \leftarrow \mathsf{Sim}_2^{f^k|_{H,\boldsymbol{x}_H^k}}(1^\lambda, 1^n, f, \mathsf{sid}^k, \mathsf{st}^{k-1})\ ,$$

if sid^k was never used before, or send $\perp$ otherwise.

From Correlated Randomness to the PKI Model. We now describe our generic transformation from a NIMPC protocol $\mathsf{NIMPC} = (\mathsf{Setup}, \mathsf{Msg}, \mathsf{Rec})$ for $\mathcal{C}$ with correlated randomness to a NIMPC protocol $\mathsf{pNIMPC} = (\mathsf{pKgen}, \mathsf{pMsg}, \mathsf{pRec})$ in the PKI model for the same function class, and show that for any $n \in \mathbb{N}$ and any $H \subseteq [n]$ if NIMPC is $\bar{H}$-robust with a simulator Sim, then pNIMPC is semi-malicious $\bar{H}$-robust with a simulator pSim with only additive polynomial slow-down.

Our transformation uses the following building blocks:

- An mrNISC scheme $\mathsf{mrNISC} = (\mathsf{Com}, \mathsf{Encode}, \mathsf{Eval})$ for the function Setup' described in Figure 7 with simulator $\mathcal{S}$.
- A PRF $\mathsf{PRF}\colon \{0,1\}^\lambda \times \{0,1\}^{\mathrm{poly}(\lambda)} \to \{0,1\}$. We recall the notation defined in Remark A.5.
- A garbled circuit scheme $\mathsf{GC} = (\mathsf{GC.Gen}, \mathsf{GC.Garble}, \mathsf{GC.Eval}, \mathsf{GC.Sim})$ for P.

Our pNIMPC protocol in the PKI model proceeds as follows:

- Key Generation: $\mathsf{pKgen}(1^\lambda)$ samples $(\mathrm{pk}, \mathrm{sk})$ as follows: Sample a random PRF key $\mathrm{fk} \leftarrow \{0,1\}^\lambda$, encode it using the mrNISC scheme $(\widehat{\mathrm{fk}}, s) \leftarrow \mathsf{Com}(1^\lambda, \mathrm{fk})$, and set $\mathrm{pk} = \widehat{\mathrm{fk}}$ and $\mathrm{sk} = (\mathrm{fk}, s)$.
  For n parties, where the i'th party holds keys $(\mathrm{pk}_i = \widehat{\mathrm{fk}}_i, \mathrm{sk}_i = (\mathrm{fk}_i, s_i))$, the correlated randomness is set to

$$\rho_0' = \mathbf{pk} = \widehat{\mathbf{fk}}\ , \quad \forall\ i \in [n],\ \rho_i' = (\mathbf{pk}, \mathrm{sk}_i = (\mathrm{fk}_i, s_i), i)\ .$$

– Encode Input: $\mathsf{pMsg}((\mathbf{pk}, \mathrm{sk}_i, i), x_i, f, \mathsf{sid})$ generates the message m_{i,x_i} as follows:
  - Generate an mrNISC computation encoding for the computation $\mathsf{Setup}'(z = (\mathsf{sid}, \lambda, f), \mathbf{fk})$ described in Figure 7.

$$\alpha_i \leftarrow \mathsf{Encode}(\mathsf{Setup}', z = (\mathsf{sid}, \lambda, f), \widehat{\mathbf{fk}}, s_i)$$

  - Sample garbled circuit labels exactly as done in Setup' in Figure 7.

$$\tau_i = \mathsf{PRF}(\mathrm{fk}_i,\ \mathsf{sid} \| 1 \| [\ell_\tau] \| \mathbf{0}) \ , \quad \mathsf{key}_i = \mathsf{GC.Gen}(1^\lambda, 1^{\ell_\rho}\ ;\ \tau_i) \ .$$

  - Garble the circuit for computing the Msg algorithm of NIMPC, *with the input x_i hardcoded in*. For convenience, we write Msg_{x_i} to represent this circuit, which satisfies $\mathsf{Msg}_{x_i}(\rho_i) = \mathsf{Msg}(\rho_i, x_i)$.

$$\widehat{\mathsf{Msg}}_{x_i} \leftarrow \mathsf{GC.Garble}(\mathsf{key}_i, \mathsf{Msg}_{x_i})$$

  Finally, set the message m_{i,x_i} to

$$m_{i,x_i} = (\alpha_i, \widehat{\mathsf{Msg}}_{x_i}, \mathsf{sid}, n, \lambda) \ .$$

– Reconstruct Output: $\mathsf{pRec}(\mathbf{pk}, (m_{i,x_i})_{i=[n]})$ computes the output as follows:
  - Invoke the mrNISC evaluation algorithm to compute

$$\begin{aligned} &\mathsf{Eval}(z = (\mathsf{sid}, \lambda, f), \widehat{\mathbf{fk}}, \{\alpha_j\}_{j\in[n]}) \\ &= \mathsf{Setup}'(z, \mathbf{fk}) = (\rho_0, \{\mathsf{key}_j[\rho_j]\}_{j\in[n]}) \end{aligned}$$

  - For every $j \in [n]$, evaluate the garbled circuit $\widehat{\mathsf{Msg}}_{x_j}$ with keys $\mathsf{key}_j[\rho_j]$,

$$\mathsf{GC.Eval}(\widehat{\mathsf{Msg}}_{x_j}, \mathsf{key}_j[\rho_j]) = \mathsf{Msg}_{x_j}(\rho_j) = im_{j,x_j} \ .$$

  - Invoke the reconstruct algorithm of NIMPC to compute the output,

$$\mathsf{Rec}(\rho_0, \{im_{j,x_j}\}_{j\in[n]}) = f(\boldsymbol{x}) = y \ .$$

  Output y. (If any of the Eval, $\mathsf{GC.Eval}$, Rec outputs $\perp$, output $\perp$.)

The correctness of our pNIMPC protocol follows directly from that of the underlying mrNISC, GC, NIMPC protocols, as illustrated already in the description of the reconstruction pRec algorithm.

Security of pNIMPC. We now show the security of our protocol.

Theorem E.5. *Assume that the PRF scheme is pseudorandom, the GC scheme is simulation-secure, the mrNISC scheme mrNISC is private. Let $n \in \mathbb{N}$ be a positive integer, $H \subseteq [n]$ be a subset. If NIMPC is $\bar{H}$-robust with simulator Sim, pNIMPC is semi-malicious $\bar{H}$-robust with simulator $\mathsf{pSim} = (\mathsf{pSim}_1, \mathsf{pSim}_2)$, who is a multiplicative polynomial (in λ) factor slower than Sim.*

Assume additionally UC-secure NIZK (with simulation-extractability), pNIMPC can be turned into a NIMPC protocol in the PKI plus CRS model that is malicious $\bar{H}$-robust.

The function $\mathsf{Setup}'(z = (\mathsf{sid}, \lambda, f), \mathrm{fk}_1, \ldots, \mathrm{fk}_n)$

The function proceeds as follows:

- Sample correlated randomness of NIMPC

$$r_j = \mathsf{PRF}(\mathrm{fk}_j,\ \mathsf{sid} \| 0 \| [\ell_r] \| \mathbf{0}) \ ,$$
$$(\rho_0, \rho_1, \ldots, \rho_n) = \mathsf{Setup}(1^\lambda, 1^n, f\ ;\ \oplus_{j\in[n]} r_j) \ ,$$

  where the randomness $\oplus_{j\in[n]} r_j$ has length $\ell_r = \mathrm{poly}(\lambda, n, |f|)$.
- For every $j \in [n]$, sample garbled circuit labels,

$$\tau_j = \mathsf{PRF}(\mathrm{fk}_j,\ \mathsf{sid} \| 1 \| [\ell_\tau] \| \mathbf{0}) \ ,$$
$$\mathsf{key}_j = \mathsf{GC.Gen}(1^\lambda, 1^{\ell_\rho}\ ;\ \tau_j) \ ,$$

  where $\ell_\rho = \max_{j\in[n]}(|\rho_i|) = \mathrm{poly}(\lambda, n, |f|)$ is the maximal length of ρ_i and the random tape τ_j has length $\ell_\tau = \mathrm{poly}(\lambda, n, |f|)$.

Finally, output $(\rho_0, \{\mathsf{key}_j[\rho_j]\}_{j\in[n]})$.

Fig. 7: The function Setup' used in the construction of pNIMPC

Proof. We construct the simulator pSim using the simulator Sim of NIMPC, the simulator $\mathcal{S}$ of mrNISC, and the simulator $\mathsf{GC.Sim}$ of the garbled circuits GC scheme. We develop the simulator pSim via a sequence of hybrids $H_0, \ldots, H_3$, where H_0 is the real experiment $\mathsf{pExp}_{\mathcal{A},\mathsf{pSim}}(\mathsf{Real}, 1^\lambda, 1^n)$ and H_3 is the ideal experiment $\mathsf{pExp}_{\mathcal{A},\mathsf{pSim}}(\mathsf{Ideal}, 1^\lambda, 1^n)$, which also contains the description of Sim.

Hybrid H_0 is identical to the experiment $\mathsf{pExp}_{\mathcal{A},\mathsf{pSim}}(\mathsf{Real}, 1^\lambda, 1^n)$. We describe the experiment below with details of the protocol pNIMPC filled in. Interact with $\mathcal{A}(1^\lambda, 1^n)$ in following steps:

1. Send honestly generated $\mathbf{pk}_H$ to $\mathcal{A}$, where

$$\forall i \in H, \qquad (\mathrm{pk}_i = \hat{\mathrm{fk}}_i, \mathrm{sk}_i = (\mathrm{fk}_i, s_i)) \leftarrow \mathsf{pKgen}(1^\lambda) \ ,$$
$$\mathrm{fk}_i \leftarrow \{0,1\}^\lambda \ , \text{ and } (\hat{\mathrm{fk}}_i, s_i) \leftarrow \mathsf{Com}(1^\lambda, \mathrm{fk}_i) \ .$$

2. $\mathcal{A}$ outputs $(\mathbf{pk}_{\bar{H}}, \boldsymbol{r}_{\bar{H}})$. Verify that $\forall i \in \bar{H}$, $(\mathrm{pk}_i, \mathrm{sk}_i) = \mathsf{pKgen}(1^\lambda;\ r_i)$, and abort if not.
3. Interact with $\mathcal{A}$ in many iterations until it terminates. In iteration k, upon $\mathcal{A}$ choosing $(f^k, \boldsymbol{x}_H^k, \mathsf{sid}^k)$, and a session id sid^k, send $\mathcal{A}$ messages $\boldsymbol{m}_{H,\boldsymbol{x}_H}^k$ generated honestly as

$$\forall i \in H, \quad m_{i,x_i}^k = (\alpha_i^k, \widetilde{\mathsf{Msg}}_{x_i^k}, \mathsf{sid}^k, n, \lambda, f^k) \leftarrow \mathsf{Msg}((\mathbf{pk}, \mathrm{sk}_i, i), x_i^k, f^k, \mathsf{sid}^k) \ ,$$
$$\text{where } \alpha_i^k \leftarrow \mathsf{Encode}(z = (\mathsf{sid}^k, \lambda, f^k), \widehat{\mathbf{fk}}, s_i)$$
$$\text{and } \widehat{\mathsf{Msg}}_{x_i^k} \leftarrow \mathsf{GC.Garble}(\mathsf{key}_i, \mathsf{Msg}_{x_i^k}) \ .$$

Hybrid H_1 is identical to H_0, except that all the input and computation encodings of mrNISC related to the honest parties are simulated using the mrNISC simulator $\mathcal{S}$. More precisely, Interact with $\mathcal{A}(1^\lambda, 1^n)$ in following steps: Invoke $\mathcal{S}(1^\lambda, \mathsf{Setup}'_n)$.

1. For every $i \in H$, sample a random PRF key $\mathrm{fk}_i \leftarrow \{0,1\}^\lambda$, and simulate the public key $\mathrm{pk}_i = \widetilde{\mathrm{fk}}_i$, using $\mathcal{S}$.
2. $\mathcal{A}$ outputs $(\mathbf{pk}_{\bar{H}}, \boldsymbol{r}_{\bar{H}})$. Verify that $\forall i \in \bar{H}$, $(\mathrm{pk}_i, \mathrm{sk}_i) = \mathsf{pKgen}(1^\lambda;\ \nu_i)$, that is, $\nu_i = (\mathrm{fk}_i, \nu'_i)$ such that $(\mathrm{pk}_i = \hat{\mathrm{fk}}_i, \mathrm{sk}_i = s_i) = \mathsf{Com}(1^\lambda, \mathrm{fk}_i; \nu'_i)$, and abort if not.
3. Interact with $\mathcal{A}$ in many iterations until it terminates. In iteration k, upon $\mathcal{A}$ choosing $(f^k, \boldsymbol{x}_H^k, \mathsf{sid}^k)$, and a session id sid^k, send $\mathcal{A}$ messages $\boldsymbol{m}^k_{H,\boldsymbol{x}_H}$ generated as follows:
   (a) For every $i \in H$, sample the garbled circuit labels key_i^k honestly as
   $$\tau_i^k = \mathsf{PRF}(\mathrm{fk}_i,\ \mathsf{sid}^k \| 1 \| [\ell_\tau] \| \mathbf{0}) \ , \quad \widehat{\mathsf{Msg}}_{x_i^k} \leftarrow \mathsf{GC.Garble}(\mathsf{key}_i, \mathsf{Msg}_{x_i^k}) \ .$$
   (b) For every $i \in H$, simulate the computation encoding $\widetilde{\alpha}_i^k$ using $\mathcal{S}$. For the last one to be simulated, send $\mathcal{S}$ the correct output of the computation
   $$\mathsf{Setup}'(z = (\mathsf{sid}^k, \lambda, f^k), \mathbf{fk}) = (\rho_0^k, \mathsf{key}_i^k[\rho_i^k]) \ .$$

The only difference between H_0 and H_1 is whether the input and computation encodings of the honest parties are simulated using the simulators $\mathcal{S}$ of the mrNISC scheme or not. It thus follows directly from the privacy guarantee of mrNISC that these two hybrids are indistinguishable.

Hybrid H_2 is identical to H_1 except that all the garbled circuits and keys belonging to the honest parties are simulated using the simulator $\mathsf{GC.Sim}$ of GC. More precisely, H_2 interacts with $\mathcal{A}$ identically as H_1 does in Step 1 and 2, but simulates Step 3 differently as follows:

3 Interact with $\mathcal{A}$ in many iterations until it terminates. In iteration k, upon $\mathcal{A}$ choosing $(f^k, \boldsymbol{x}_H^k, \mathsf{sid}^k)$, and a session id sid^k, send $\mathcal{A}$ messages $\boldsymbol{m}^k_{H,\boldsymbol{x}_H}$ generated as follows:
   (a) For every $i \in H$, simulate the garbled circuit and keys
   $$(\widetilde{\mathsf{Msg}}_i^k, \widetilde{\mathsf{key}}_i^k) \leftarrow \mathsf{GC.Sim}(1^\lambda, im_i^k) \ , \text{ where } im_i^k = \mathsf{Msg}_{x_i^k}(\rho_i^k) \ .$$
   (b) For every $i \in H$, simulate the computation encoding $\widetilde{\alpha}_i^k$ using $\mathcal{S}$. For the last one to be simulated, send $\mathcal{S}$ the partially simulated output $(\rho_0^k, \widetilde{\mathsf{key}}_i^k)$.

It follows directly from the simulation security of the garbled circuits scheme GC that H_2 is indistinguishable to H_1.

Hybrid H_3 is identical to H_2 except that all the NIMPC messages $\{im_i^k\}_{k,i\in H}$ and the public randomness $\{\rho_0^k\}_k$ are simulated using the simulator Sim of NIMPC. More precisely, H_3 interacts with $\mathcal{A}$ identically as H_2 does in Step 1 and 2, but simulates Step 3 differently as follows:

3 In iteration k, upon $\mathcal{A}$ choosing $(f^k, \boldsymbol{x}_H^k, \mathsf{sid}^k)$, send $\mathcal{A}$ messages $\boldsymbol{m}_{H,\boldsymbol{x}_H}^k$ generated as follows:

(a) Simulate the message and public randomness of NIMPC as

$$(\widetilde{\rho}_0^k, \widetilde{\boldsymbol{im}}^k) \leftarrow \mathsf{Sim}^{f^k|_{H,\boldsymbol{x}_H^k}}(1^\lambda, 1^n, f, \bar{H}) \ .$$

(b) For every $i \in H$, simulate the garbled circuit and keys using the simulated NIMPC messages,

$$(\widetilde{\mathsf{Msg}}_i^k, \widetilde{\mathsf{key}}_i^k) \leftarrow \mathsf{GC.Sim}(1^\lambda, \widetilde{im}_i^k) \ .$$

(c) For every $i \in H$, simulate the computation encoding $\widetilde{\alpha}_i^k$ using $\mathcal{S}$. For the last one to be simulated, send $\mathcal{S}$ the partially simulated output $(\widetilde{\rho}_0^k, \widetilde{\mathsf{key}}_i^k)$.

It follows directly from the $\bar{H}$-robustness of NIMPC that H_3 is indistinguishable to H_2.

Note that in H_3, the view of the adversary $\mathcal{A}$ in Step 1 is simulated by $\mathcal{S}$ of mrNISC using no information of the functions and honest parties' inputs chosen later, and the view in each iteration k in Step 3 is simulated with oracle access to the residual function $f^k|_{H,\boldsymbol{x}_H^k}$ as required in the ideal experiment $\mathsf{pExp}_{\mathcal{A},\mathsf{pSim}}(\mathsf{Real}, 1^\lambda, 1^n)$ (with pSim implicitly defined). Therefore by a hybrid argument, we conclude that the real and the ideal experiments are indistinguishable and pNIMPC is $\bar{H}$-robust.

Run time analysis of pSim. The adversary $\mathcal{A}$ runs in time $\mathrm{poly}(\lambda)$, as a result, it engages in at most $\mathrm{poly}(\lambda)$ iterations in the experiments and all messages (e.g., the description of functions f^k) it outputs in the experiments have $\mathrm{poly}(\lambda)$ length. The run time of pSim_1 for simulating the view of $\mathcal{A}$ in Step 1 is the same as the run time of $\mathcal{S}$ of mrNISC, which is $\mathrm{poly}(\lambda)$. The run time of pSim_2 for simulating the view of $\mathcal{A}$ in interaction k of Step 2 is the run time of the NIMPC simulator Sim, $T_{\mathsf{Sim}}^{n,\bar{H}}(\lambda)$, plus other polynomial-time operations. Thus, the total run time of pSim_2 is $\mathrm{poly}(\lambda)T_{\mathsf{Sim}}^{n,\bar{H}}(\lambda)$. In summary, the simulator pSim is a polynomial factor slower than the simulator Sim of the underlying NIMPC.

Security against malicious adversaries. Assume additionally UC-secure NIZK, pNIMPC can be turned into a malicious $\bar{H}$-robust NIMPC protocol in the PKI plus CRS model, by letting the key generation algorithm additionally generate a NIZK proof π of the correctness of the public key pk produced, and set the new public key pk' to (pk, π). In the security proof, a malicious adversary participating pExp experiments does not provide witnesses $\boldsymbol{r}_{\bar{H}}$) of the correctness

of the public keys $\mathbf{pk}_{\bar{H}}$ of corrupted parties in Step 2. Nevertheless, since $\mathbf{pk}'_{\bar{H}}$ contains NIZK proofs $\{\pi_j\}_{j\in\bar{H}}$) of the correctness of $\mathbf{pk}_{\bar{H}}$, the simulator can extract witnesses from the NIZK proofs. The rest of simulation and security proof remain the same as in the semi-malicious case. □

Application of Our Transformation. In the literature, the following NIMPC protocols with correlated randomness has been presented.

1. Information theoretically fully robust NIMPC for any efficiently computable functions, with communication complexity exponential in the total input length [9].
2. Information theoretically fully robust NIMPC for the iterated product function $f(x_1, \ldots, x_n) = x_1 \cdots x_n$ over a group $\mathbb{G}$ [9].
3. Computationally fully robust NIMPC for any efficiently computable functions with indistinguishability based security (implied by exponential-time simulation), from multi-input functional encryption [29].
4. Information theoretically $O(1)$-robust NIMPC for NC^1 with domains that are the Cartesian products of *constant-sized sets*, i.e., $\mathcal{X} = \mathcal{X}_1 \times \cdots \times \mathcal{X}_n$ where $|\mathcal{X}_i| = O(1)$[8] [10].
5. Computationally $O(1)$-robust NIMPC for P with domains that are the Cartesian products of *constant-sized sets* from one-way functions [10].

Applying our transformation to the above NIMPC protocols in bullets 2, 3, 5 yields their computationally secure counterparts in the PKI plus CRS model.

Corollary E.6. *Apply our transformation above to existing NIMPC protocols 2, 3, 5 gives the following protocols in the PKI plus CRS model assuming mrNISC for* P *and UC-NIZK for* NP *(both of which can be based on a circular variant of LWE or the SXDH assumption on bilinear maps).*

1. *Computationally fully robust NIMPC for the iterated product function* $f(x_1, \ldots, x_n) = x_1 \cdots x_n$ *over a group* $\mathbb{G}$.
2. *Computationally fully robust NIMPC for any efficiently computable functions with indistinguishability based security from multi-input functional encryption.*
3. *Computationally* $O(1)$*-robust NIMPC for* P *with domains the Cartesian products of* constant-sized sets.

The first and third bullets are achieved for the first time in the PKI plus CRS model, while the second was previously achieved by [35] using *sub-exponentially* secure IO and DDH. We now achieve it using multi-input functional encryption, which in turn can be based on *polynomially secure* IO and one-way functions, assuming additionally circular secure LWE or SXDH on bilinear maps. We note that the use of polynomially secure IO is necessary, as fully robust NIMPC for P implies IO for P.

[8] If the domains the Cartesian products of exponential-sized alphabet, even 1-robust NIMPC implies IO.

E.2 Secret-Sharing VBB

In this section, we use mrNISC to construct a primitive called (N,t)-*secret-sharing VBB*. As the name suggests, secret-sharing VBB allows to secret share a function f among N servers, each holding a *function share* f_i for $i \in [N]$, and later to evaluate this function on any input x, by letting each server generate an output share $\hat{y}_i \leftarrow f_i(x)$ independently using its own function share. Output shares together reveal the actual output $f(x)$. Analogous to VBB obfuscation, the secret shares of f is reusable — one can use them to evaluate any number of inputs. Moreover, security is simulation-based: for any sequence of inputs $x_1, x_2, \ldots, x_K$, observing *all* the output shares $\{\hat{y}_{1i}\}, \{\hat{y}_{2i}\}, \ldots, \{\hat{y}_{Ki}\}$, and any subset $S \subset [N]$ of up to t function shares $\{f_i\}_{i\in S}$ reveals only the actual outputs $y_1, y_2, \ldots, y_K$ and nothing else.

Definition E.7. *A (N,t)-secret share VBB scheme* ssVBB *for a class $\mathcal{C}$ of functions consists of polynomial-time algorithms* $(\mathsf{VShare}, \mathsf{Eval}, \mathsf{Recon})$ *with the following syntax:*

- *Function Sharing:* $(f_1, \ldots, f_N) \leftarrow \mathsf{VShare}(1^\lambda, f)$*, on input the security parameter λ, and the description of a function f, generates N function shares $f_1, \ldots, f_N$.*
- *Output Sharing:* $\hat{y}_i \leftarrow \mathsf{VEncode}(x, f_i)$ *on input an input x and a function share f_i, generates an output share $\hat{y}_i$.*
- *Evaluation:* $y = \mathsf{VEval}(\{y_i\}_{i\in[N]})$ *on input all output shares, reconstructs the actual output y.*

and satisfying the following properties:

Correctness. For every $\lambda \in \mathbb{N}$, every function $f \in \mathcal{C}$, and every input x

$$\Pr\left[\begin{array}{c}(f_1,\ldots,f_N) \leftarrow \mathsf{VShare}(1^\lambda, f) \\ \forall i \in [N],\ \hat{y}_i \leftarrow \mathsf{VEncode}(x, f_i)\end{array} \ :\ \mathsf{VEval}(\hat{y}_1,\ldots,\hat{y}_N) = f(x)\right] = 1\ .$$

(N,t)-Simulation Security. There exists a PPT simulator $\mathcal{S}$, such that, for every PPT adversary $\mathcal{A}$, the view of $\mathcal{A}$ in the following two experiments are indistinguishable.

Experiment $\mathsf{VExp}_{\mathcal{A},\mathcal{S}}(\mathsf{Real}, 1^\lambda)$: *Interact with $\mathcal{A}$ in two steps:*

1. *Function Shares Query: Upon $\mathcal{A}$ choosing a function $f \in \mathcal{C}$ and a subset $S \subset [N]$ of size at most t, $|S| \le t$, secret share the function f to obtain $(f_1, \ldots, f_N) \leftarrow \mathsf{VShare}(1^\lambda, f)$, and reveal $\{f_i\}_{i\in S}$ to $\mathcal{A}$.*
2. *Honest Output Share Query: Interact with $\mathcal{A}$ until it terminates. In each iteration, upon $\mathcal{A}$ choosing (x, i) for $i \notin S$[9], reply with the i'th output share $\hat{y}_i \leftarrow \mathsf{VEncode}(x, f_i)$ for evaluating x.*

Experiment $\mathsf{VExp}_{\mathcal{A},\mathcal{S}}(\mathsf{Ideal}, 1^\lambda)$: *The ideal experiment simulates the view of $\mathcal{A}$ as follows: Invoke $\mathcal{S}(1^\lambda)$.*

[9] Note that $\mathcal{A}$ can already generate output shares $\{y_i\}_{i\in S}$ itself using the function shares it obtain from the last stage.

- *Function Shares Query:* *Upon $\mathcal{A}$ choosing a function $f \in \mathcal{C}$ and a subset $S \subset [N]$ of size at most t, $|S| \leq t$, send only S to $\mathcal{S}$ who generates simulated function shares $\{\widetilde{f}_i\}_{i \in S}$, and forward them to $\mathcal{A}$.*
- *Honest Output Share Query:* *Interact with $\mathcal{A}$ until it terminates. In each iteration, upon $\mathcal{A}$ choosing (x, i) for $i \in \bar{S}$, send (x, i, y') to $\mathcal{S}$, where $y' = f(x)$ if all output shares in $\bar{S}$ for evaluating input x have been queried (i.e., $\forall j \in \bar{S}$, $\mathcal{A}$ has queried (x, j)), and $y' = \perp$ otherwise. $\mathcal{S}$ generates a simulated output share $\hat{y}_i$, which is forwarded to for $\mathcal{A}$.*

Related Notions. The primitive of secret-sharing VBB is very close to the primitive of bit-fixing homomorphic sharing proposed in the recent work of [39], except that they considered an indistinguishability based definition (analogous to IO) and requires the security to hold even when the functions shares are sampled from a conditional distribution where a few bits in the shares are fixed (i.e., $(f_1, \cdots, f_N) \leftarrow \mathsf{VShare}(1^\lambda, 1^N, f)$ conditioned on a substring $(f_1 \| \ldots \| f_N)_J = u$ of length $|J| = o(\lambda)$ being fixed to an arbitrary string u). The latter strengthening is for their specific application to constructing indistinguishability obfuscation from bilinear maps, LWE, and degree 2 pseudo flawed-smudging generator (a type of weak PRG). We observe that in the secret sharing setting, even simulation-based security can be achieved, and remove the latter strengthening to make the definition simpler and more natural. (The security of our construction of secede sharing VBB from mrNISC nevertheless holds even in the strengthened setting. Hence, our construction can also be used in their application.)

Another related notion is Homomorphic Secret Sharing (HSS) — the secret-sharing version of homomorphic encryption — and Function Secret Sharing [12,13]. The key difference from HSS / FSS is that they *do not* guarantee security when all output shares are revealed. (See Section 7.1 in [39] for more detail.) Analogous to obfuscation, secret sharing VBB considers the setting where outputs and hence output shares are all public. This is useful in applications where we want a third party to obtain the outputs of the computations, such as, enabling servers to perform spam filtering on clients' encrypted emails without involving the clients.

Construction from mrNISC. We now give a very simple construction of $(N, N-1)$-secret sharing VBB for a function class $\mathcal{C}$ from an mrNISC scheme $\mathsf{mrNISC} = (\mathsf{Com}, \mathsf{Encode}, \mathsf{Eval})$ for the following universal function $U^{\mathcal{C}}$ for class $\mathcal{C}$: $U^{\mathcal{C}}(x, v_1, \cdots, v_N)$ interprets $f = v_1 \oplus \cdots \oplus v_N$ as the description of a function $f \in \mathcal{C}$ and outputs $f(x)$.

- $\mathsf{VShare}(1^\lambda, f)$: On input the description of a function f, sample a random XOR secret sharing $v_1, \ldots, v_N$ of f (i.e., v_i's are random strings subject to $v_1 \oplus \cdots \oplus v_N = f$. For every $i \in [N]$, generate an input encoding of the i'th secret share $(\hat{v}_i, s_i) \leftarrow \mathsf{Com}(1^\lambda, v_i)$. Set the i'th function share to $f_i = (\{\hat{v}_j\}_{j \in [N]}, s_i)$, and output $(f_1, \ldots, f_N)$.
- $\mathsf{VEncode}(x, f_i)$: on input an input x and a function share f_i, parse f_i as $(\{\hat{v}_j\}_{j \in [N]}, s_i)$, generate the i'th computation encoding $\alpha_i = \mathsf{Encode}(x, \{\hat{v}_j\}_{j \in [N]}, s_i)$ for the computation $U^{\mathcal{C}}(x, v_1, \ldots, v_N)$, and output $y_i = (x, \hat{v}_i, \alpha_i)$ as the i'th output share.

– $\mathsf{VEval}(\{y_i\}_{i\in[N]})$ on input all output shares, parse each $y_i = (x, \hat{v}_i, \alpha_i)$, and reconstruct the actual output $y = \mathsf{Eval}(U^{\mathcal{C}}, x, \{\hat{v}_i\}_N, \{\alpha_i\}_N)$.

The correctness of the scheme follows directly from that of mrNISC: The latter guarantees that y output by Eval equals to $U^{\mathcal{C}}(x, v_1, \ldots, v_N)$, which by definition equals to $f(x)$.

Proof of Simulation Security. The security of our ssVBB scheme against revealing up to $N-1$ function shares follows directly from the privacy guarantee of mrNISC. This is because an adversary against ssVBB observing up to $N-1$ function shares and an arbitrary number of output shares, can be emulated by an *semi-honest* adversary against mrNISC corrupting up to $N-1$ input parties.

Theorem E.8. *For any $N \in \mathbb{N}$ and any function class $\mathcal{C}$, if* mrNISC *for $\{U_N^{\mathcal{C}}\}$ is private, then* ssVBB *for $\mathcal{C}$ described above is $(N, N-1)$-simulation secure.*

Proof. We show that the view of any adversary $\mathcal{A}$ participating in the real experiment $\mathsf{VExp}_{\mathcal{A},\mathcal{S}}(\mathsf{Real}, 1^\lambda)$ of ssVBB can be emulated by a wrapper adversary $\mathcal{A}'$ participating in the real experiment $\mathsf{Exp}_{\mathcal{A},\mathcal{S}}(\mathsf{Real}, 1^\lambda)$ of mrNISC, who is semi-honest and corrupts up to $N-1$ input parties. Thus, we can use the mrNISC simulator $\mathcal{S}'$ that simulates the view of $\mathcal{A}'$ to construct the simulator $\mathcal{S}$ for $\mathcal{A}$.

We now describe the wrapper adversary $\mathcal{A}'$ that runs $\mathcal{A}$ internally and participates in $\mathsf{Exp}_{\mathcal{A}',\mathcal{S}'}(\mathsf{Real}, 1^\lambda)$ externally.

– *Function Shares Query*: Upon $\mathcal{A}$ choosing a function $f \in \mathcal{C}$, and a subset $S \subset [N]$ of size at most $N-1$ $\mathcal{A}'$ generates the function shares in S as follows:
  - corrupt input parties in set S.
  - sample a random additive secret share $v_1, \ldots, v_N$ of f.
  - for every corrupted input party $i \in S$, generate its input encoding of v_i honestly, $(\hat{v}_i, s_i) = \mathsf{Com}(1^\lambda, v_i; \rho_i)$ using randomly string $\rho_i \leftarrow \{0,1\}^{\mathrm{poly}(\lambda)}$, and announce this input encoding externally by sending $(\mathsf{input}, P_i, v_i, \rho_i)$.
  - for every honest input party $j \in \bar{S}$, send a query $(\mathsf{input}, P_j, v_j)$ externally, and receive an honest input encoding $\hat{v}_j$.
  - Finally, set every function share f_i for $i \in S$ as $(\{\hat{v}_j\}_{j\in[N]}, s_i)$, and send them to $\mathcal{A}$.
– *Honest Output Share Query*: In each iteration, upon $\mathcal{A}$ choosing (x, i) for $i \in S$, $\mathcal{A}'$ send a query $(\mathsf{compute}, P_j, x, I = [N])$ externally, and receive an honest computation encoding α_j. It then sends $\hat{y}_i = (x, \hat{v}_i, \alpha_i)$ to $\mathcal{A}$.

It is easy to see that $\mathcal{A}'$ follows the rules in the experiment $\mathsf{Exp}_{\mathcal{A}',\mathcal{S}'}(\mathsf{Real}, 1^\lambda)$, in particular, it corrupts a strict subset S of input parties statically, and it generates all corrupt input encodings semi-maliciously, in fact, *honestly*. Therefore, by the privacy guarantee of mrNISC, there is a simulator $\mathcal{S}'$ that can simulate the view of $\mathcal{A}'$ (more specifically, $\mathcal{S}'$ can simulate the input and computation encodings of honest parties in $\bar{S}$). Thus, we can construct a simulator $\mathcal{S}$ for $\mathcal{A}$ that internally emulates both $\mathcal{S}'$ and $\mathcal{A}'$. Note that the only information that the mrNISC simulator $\mathcal{S}'$ needs to receive is the correct output $y = U_N^{\mathcal{C}}(x, v_1, \ldots, v_N) = f(x)$, whenever $\mathcal{A}$ has queried all output shares in $\bar{S}$ for evaluating a specific input x.

At this point, $\mathcal{S}$ in $\mathsf{VExp}_{\mathcal{A},\mathcal{S}}(\mathsf{Ideal}, 1^\lambda)$ indeed receives y and hence can forward it to $\mathcal{S}'$ emulated internally. The construction of $\mathcal{S}$ is mechanic and we omit details here. □

Extension to Secret-sharing Multiparty VBB. A recent work [35] proposed a generalization of obfuscation called *multiparty* obfuscation. Here, multiple function (we use function instead of program to be consistent in terminology) owners, each holding a function f_i for $i \in [J]$, can individually obfuscate their functions into $\hat{f}_i$, such that, the collection of obfuscated functions $\{\hat{f}_i\}_J$ acts as an obfuscated function of $\mathsf{Combine}(f_1, \ldots, f_J)$ according to some fixed public $\mathsf{Combine}$ function. When $J = 1$, multiparty obfuscation degenerates to the classical notion of obfuscation. One can similarly consider both VBB or indistinguishability security.

We can similarly have a secret-sharing multiparty VBB (ssMVBB), where each function owner can secret share its function into shares $f_{i,1}, \ldots, f_{i,N} \leftarrow \mathsf{VShare}(1^\lambda, f_i)$, and upload the j'th share $f_{i,j}$ to the j'th server. To evaluate an input x, each server independently generates output shares $y_j \leftarrow \mathsf{VEncode}(x, \{f_{i,j}\}_{i\in[J]})$, which together reveals the output $y = \mathsf{VEval}(\{y_j\}_{j\in[N]}) = \mathsf{Combine}(f_1, \ldots, f_J)(x)$. Security guarantees that if up to some $0 \le t < N$ servers were corrupted, given all the output shares for any number of evaluations with inputs $x_1, x_2, \ldots, x_T$, the only information revealed are the actual outputs of $\mathsf{Combine}(f_1, \ldots, f_J)$ on these inputs.

It is easy to observe that essentially the same construction from mrNISC above gives a secret sharing multiparty VBB ssMVBB scheme, with some slight modification:

- $\{f_{i,j}\}_j \leftarrow \mathsf{VShare}(1^\lambda, f_i)$ stays the same. Thus $f_{i,j} = (\{\hat{v}_{i,j}\}_j, s_{i,j})$ where $\hat{v}_{i,j}$ is the mrNISC input encoding for a XOR share of f_i and $s_{i,j}$ is the secret state associated with it.
- $\mathsf{VEncode}(x, \{f_{i,j}\}_{i\in[J]})$ generates J mrNISC computation encodings $\{\alpha_{i,j} \leftarrow \mathsf{Encode}(U, x, \{\hat{v}_{i,j}\}_{i,j}, s_{i,j})\}_i$ where $U(x, \{v_{i,j}\}_{i,j})$ is the universal function that computes $\mathsf{Combine}(f_1, \ldots, f_J)(x)$ for $f_i = \oplus_j v_{i,j}$. It sets the j'th output share to be $y_j = (x, \{v_{i,j}, \alpha_{i,j}\}_i)$.
- Finally, $\mathsf{VEval}(\{y_j\}_j)$ simply runs $\mathsf{Eval}(x, \{v_{i,j}\}_{i,j}, \{\alpha_{i,j}\}_{i,j})$ to obtain the output.

Intuitively, this scheme simply XOR secret shares each function, and encode all $J \times N$ of them using mrNISC, to evaluate an input x, the evaluator simply needs $J \times N$ computation encodings, one for each share of each function. The ssMVBB scheme puts the j'th input encodings related to all function at the j'th server, such that, if one server remains uncorrupted, all functions remain hidden.

Strengthening to Semi-Malicious Security. Above, we think of the use case of ssVBB as having an honest function owner generate the function shares and upload them to the servers, and an adversary may corrupt a subset of the servers and a subset of the function shares and many output shares. One may also consider stronger security where the adversary can *tamper* a subset S of function

shares. This tampering may be semi-malicious in the sense that the function shares are generated correctly using VShare where the randomness related to generating shares $\{f_i\}_{i\in S}$ are controlled by the adversary. More precisely, the adversaries chooses XOR shares $\{v_i\}_{i\in S}$ and random tapes $\{\rho_i\}_{i\in S}$, such that, $\{(\hat{v}_i, s_i) = \mathsf{Com}(1^\lambda, v_i;\ \rho_i)\}_{i\in S}$. On the other hand, the rest of the XOR shares and input encodings are generated randomly and honestly: $\{v_j\}_{j\in\bar{S}}$ are random subject to $\oplus_k v_k = f$ and $\{(\hat{v}_j, s_j) \leftarrow \mathsf{Com}(1^\lambda, v_j)\}_{j\in\bar{S}}$. Then the function shares are $\{(f_i = \{\hat{v}_i\}_{i\in[N]}, s_i)\}_{i\in[N]}$. Since the privacy of mrNISC guarantees security against semi-malicious adversaries. Our ssVBB constructed from it remains secure under such semi-malicious tampering of function shares.

This means our scheme satisfies the strengthened security requirement in [39] that considers the function shares sampled honestly from $(f_1, \ldots, f_N) \leftarrow \mathsf{VShare}(1^\lambda, f)$ conditioned on some substring $(f_1 \| \ldots \| f_N)_J$ of length $|J| < N$ being fixed. This is because fixing a up to $N-1$ bits in the shares can be achieved by a semi-malicious adversary tampering the same number of shares. Therefore, our ssVBB scheme is secure against such fixing attacks, and can be used in the application of [39] for constructing IO from bilinear maps, LWE, and a special type of weak PRG.

F More on Related Works

In the introduction and technical overview, we've already compared our mrNISC with related notions, and our protocols from bilinear maps with the protocols based on MKFHE. Here, we give slightly more detail.

Relation with Sender-Receiver NISC. A resuable NISC protocol [2, 5, 6, 16, 19, 38] for computing a function f is a sender-receiver MPC protocol where a receiver can publish an encoding of its input $\hat{x}$ in such a way that a sender holding an input z can send a single message so that the receiver learns $y = f(z, x)$ and nothing else. Clearly, reusable NISC is closely related with our notion of mrNISC, especially when the number of parties is just 2. However, these two notions are actually incomparable. The first difference is that in resuable NISC, the receiver may reconstruct the output y using its secret state and hence the output reconstruction is *private*, whereas in mrNISC the evaluator reconstructs the output without any secret state and hence output reconstruction is *public*. The second difference is that in mrNISC, parties must commit to their inputs before computation occurs, whereas in resuable NISC, the sender may choose its input online. We note that these two differences are intertwined: to have public reconstruction, it is necessary that parties commit to their inputs in the first round. Indeed, if a sender-receiver protocol has public output reconstruction, an adversary given the encoding of receiver's input can evaluate $f(z, x)$ for any function f and input z by generating the sender's message for f, z and reconstruct the output, violating security.

Relation with NIMPC and PSM. Proposed by [9], a NIMPC protocol for computing a function f enables a set of parties with private inputs $x_1, \ldots, x_n$ to

send a single message to an evaluator conveying only $f(x_1, \ldots, x_n)$. As explained above, this setting is inherently susceptible to the residual attack, in contrast to mrNISC in which inputs are committed. Furthermore, fully secure NIMPC implies obfuscation, hence this object is in a much more powerful league than mrNISC.

The notion of Private Simultaneous Message (PSM) proposed by [23] and named by [36] precedes the notion of NIMPC and is a special case of NIMPC, where the adversaries can only corrupt the evaluator (and not any other party), and correspondingly learns only the output of the function and nothing else. Despite of the restriction, PSM protocols still cannot be realized in the CRS model and known protocols rely on either PKI or common randomness shared by the parties but unknown to the evaluator.

Comparison with MKFHE-based mrNISC Protocol. As discussed before, our mrNISC protocols based on bilinear maps have two advantages over the MKFHE-based protocols: First, our protocols are in the plain model when the adversaries are semi-honest or even semi-malicious, whereas the MKFHE-based protocols are in the CRS model. Second, our protocols are constructed following the recently developed round-collapsing approach, which is a powerful and versatile technique that can be adapted to a variety of different settings as demonstrated in the 2-round MPC setting. On the other hand, the MKFHE-based protocols has the advantage of having succinct communication: the size of a computation message depends only on the depth of the computation, whereas the size of a computation message of our protocols scales at least linearly with the size of the computation. However, a subtle issue with the MKFHE-based protocols is that even the sizes of the CRS and input encodings grow polynomially in the depth of the computation, whereas in our protocol they depend only on the security parameter and input length (independent of any information of later computations). This subtle issue means that when using MKFHE-based protocols, all parties must *agree* ahead of time on a limit on the depth of the computations they would evaluate later, which is very limiting in a market place. This issue can be circumvented by letting parties always compute a randomized encoding of the computation, which encodes the output of the computation. However, doing so kills succinctness, making the size of the computation message scaling with the size of the computation.