Multiparty Reusable Non-Interactive Secure Computation from LWE

Fabrice Benhamouda[*] Aayush Jain[†] Ilan Komargodski[‡] Huijia Lin[§]

February, 2020

Abstract

Motivated by the goal of designing versatile and flexible secure computation protocols that at the same time require as little interaction as possible, we present new multiparty reusable Non-Interactive Secure Computation (mrNISC) protocols. This notion, recently introduced by Benhamouda and Lin (TCC 2020), is essentially two-round Multi-Party Computation (MPC) protocols where the first round of messages serves as a reusable commitment to the private inputs of participating parties. Using these commitments, any subset of parties can later compute any function of their choice on their respective inputs by just sending a single message to a stateless evaluator, conveying the result of the computation but nothing else. Importantly, the input commitments can be computed without knowing anything about other participating parties (neither their identities nor their number) and they are reusable across any number of desired computations.

We give a construction of mrNISC that achieves standard simulation security, as classical multi-round MPC protocols achieve. Our construction relies on the Learning With Errors (LWE) assumption with polynomial modulus, and on the existence of a pseudorandom function (PRF) in NC^1. We achieve semi-malicious security in the plain model and malicious security by further relying on trusted setup (which is unavoidable for mrNISC). In comparison, the only previously known constructions of mrNISC were either using bilinear maps or using strong primitives such as program obfuscation.

We use our mrNISC to obtain new Multi-Key FHE (MKFHE) schemes with threshold decryption:

- In the CRS model, we obtain threshold MKFHE for NC^1 based on LWE with only *polynomial* modulus and PRFs in NC^1, whereas all previous constructions rely on LWE with super-polynomial modulus-to-noise ratio.
- In the plain model, we obtain threshold levelled MKFHE for P based on LWE with *polynomial* modulus, PRF in NC^1, and NTRU, and another scheme for constant number of parties from LWE with sub-exponential modulus-to-noise ratio. The only known prior construction of threshold MKFHE (Ananth et al., TCC 2020) in the plain model restricts the set of parties who can compute together at the onset.

[*]Algorand. Email: `fabrice.benhamouda@gmail.com`.
[†]NTT Research and UCLA. Email: `aayushjain@cs.ucla.edu`.
[‡]Hebrew University of Jerusalem and NTT Research. Email: `ilank@cs.huji.ac.il`.
[§]UW. Email: `rachel@cs.washington.edu`.

Contents

1 Introduction

Much of the research in secure multiparty computation (MPC) is driven by the goal of minimizing interaction as much as possible. This is first motivated by the fact that network latency is often a major bottleneck to efficiency. Futhermore, having many communication rounds requires participating parties to be stateful and on-line for a long time which is difficult if not possible in some scenarios, especially when the number of participants is large. Soon after the invention of MPC [GMW87, BGW88, CCD88], a large body of works investigated constant-round MPC protocols, or even completely non-interactive ones.

The vision of non-interactive MPC is extremely fascinating. Ideally, it would allow any set of parties to jointly compute an arbitrary function of their respective secret inputs, without any prior interaction or input-dependent setup, by each sending a single message to a public bulletin board, enabling an external evaluator to compute the output of the function based only on these messages.[1] Unfortunately, it is known that such non-interactive protocols cannot satisfy the standard simulation security notion, as they are inherently susceptible to the so called residual-function attack. Therefore, at least another round of communication is necessary.

MrNISC. In a recent work, Benhamouda and Lin [BL20] introduced a hybrid model between non-interactive MPC and two-round MPC which they called *multiparty reusable Non-Interactive Secure Computation (mrNISC)*. To motivate the model, it is useful to consider the following scenario: users across the world wish to publish an encryption of their DNA on a public bulletin board, once and for all. At a later stage, for the purposes of medical analysis, a subset of them wants to compute some function on their DNAs by sending just a single public message to a doctor, who should be able to compute this function, but nothing else. Furthermore, a user may participate in an unbounded number of medical analyses, reusing the same encryption of DNA, with the same or other subsets of parties on the same or different functions.

More formally, in the mrNISC model, parties publish encodings of their private inputs x_i on a public bulletin board, once and for all, independently of each other and even independently of the total number of parties. 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 later computation sessions — each later computation can evaluate any polynomial-time function, among any polynomial-size subset of participants. The security guarantee is that an adversary corrupting a subset of parties, chosen statically at the beginning, learns no information about the private inputs of honest parties, beyond the outputs of the computations they participated in. This holds for any polynomial number of computation sessions. Throughout, each party's input, and the function and participants of each computation session are chosen adaptively by the adversary.

The work of Benhamouda and Lin [BL20] presents a general-purpose mrNISC for computing polynomial-sized circuits, whose security is based on the SXDH assumption in asymmetric bilinear groups. It is in the plain model (without any trusted setup), and satisfies semi-malicious security.[2]

[1] The reconstruction of the output is "public" in the sense that it does not require any secrets. It is w.l.o.g. to consider public output reconstruction, as one can always consider the evaluator as a participant of MPC with a dummy input and uses the all zero string as its random tape.

[2] Semi-malicious security is a strengthening of the semi-honest security wherein the adversary is allowed to choose its random tape arbitrarily. [AJL+12] showed that any protocol satisfying semi-malicious security can be made maliciously secure by additionally using Non-Interactive Zero-Knowledge proofs (NIZKs).

For malicious security, the use of some setup is inevitable; they rely on a CRS. To date, this is the only mrNISC construction in the plain model, based on well-established assumptions. Prior plain-model 2-round MPC protocols either rely on strong primitives like indistinguishability obfuscation or general-purpose witness encryption [GGHR14, GP15, GLS15, CGP15, DKR15] which have complex constructions from less well-established assumptions, or have first messages that are not reusable [GS17, GS18, BL18, GMS18, GIS18, ACGJ18, ABJ+19, QWW18], or only reusable among a fixed set of parties [AJJ20, BGMM20]. Another line of works leading to two-round MPC, using multi-key fully-homomorphic encryption (MKFHE) [AJL+12, CM15, MW16, BP16, PS16, BHP17, BJMS18], could possibly be made an mrNISC, but even then all known constructions rely on trusted setup even for semi-honest security.

1.1 Our Results

New mrNISC from LWE. Our main result is a new construction of an mrNISC. Our construction is based on the standard Learning-With-Errors (LWE) assumption with polynomial modulus as well as on a PRF in NC^1. The construction is in the plain model, and satisfies semi-malicious security.

Theorem 1.1 (mrNISC from LWE)**.** *Assuming LWE with polynomial modulus and a PRF in* NC^1*, there exists a mrNISC protocol for all polynomial-size functions. The construction is in the plain model (without any trusted setup), and satisfies semi-malicious security. For malicious security, we need to further rely on a CRS.*[3]

We emphasize that our construction requires only LWE with polynomial modulus. This is important both for efficiency as well as for security. First, having a polynomial modulus makes the sizes of keys and ciphertexts shorter. Second, for security, it is known that LWE with polynomial ratio between modulus and noise (which is our case) is at least as hard as (classical) GapSVP with polynomial approximation factor [Reg05, Pei09, MM11, MP12, BLP+13].

Unfortunately, it is not known whether PRF in NC^1 can be based on LWE with polynomial modulus-to-noise ratio, as all known constructions require super-polynomial modulus-to-noise ratio [BPR12, BLMR13, BP14]. Therefore, the above theorem can also be instantiated using a single assumption of LWE with super-polynomial modulus-to-noise ratio, which is independent of the depths of computations.

New threshold multi-key FHE schemes. We observe that mrNISC can be used to generically boost any multi-key FHE with an "unstructured" decryption function that takes as input the secret key of all participating parties, into a threshold multi-key FHE scheme by just decentralizing the decryption function.

This observation gives us new constructions of threshold multi-key FHE by instantiating the base multi-key FHE scheme with different known constructions. Specifically, we obtain the following three threshold multi-key FHE instantiations.

Theorem 1.2 (Threshold multi-key FHE in the CRS model)**.** *There exists a threshold multi-key FHE scheme in the CRS model for* NC^1 *circuits assuming LWE with* polynomial *modulus and a PRF in* NC^1*.*

The above theorem follows from the multi-key FHE schemes of [CM15, MW16], which require LWE with polynomial modulus for evaluating NC^1 circuits. Here, we rely additionally on a PRF in

[3]The CRS is needed for NIZK which exists from LWE with polynomial modulus [PS19].

NC^1. In comparison, all previous constructions of threshold multi-key FHE even for NC^1 require LWE with super-polynomial modulus-to-noise ratio. Since the latter readily implies a PRF in NC^1, our assumption is weaker.

Theorem 1.3 (Threshold multi-key FHE in the plain model)**.** *Let $d = d(\lambda)$ and $N = N(\lambda)$ be arbitrary polynomial functions of the security parameter.*

1. *There exists a threshold multi-key FHE scheme in the plain model for polynomial-size depth-d circuits and supporting N keys. The scheme is secure assuming LWE with polynomial modulus, a PRF in NC^1, and the DPSR assumption.*[4]

2. *There exists a threshold multi-key FHE scheme in the plain model for polynomial-size depth-d circuits and supporting arbitrary constant number of keys. The scheme is secure assuming LWE with sub-exponential modulus-to-noise ratio.*

The first bullet is obtained by using the multi-key FHE scheme of [LTV12]. Recently, Ananth et al. [AJJM20] obtained a similar result except that their threshold multi-key FHE definition is somewhat weak in the sense that the set of public-keys under which each evaluation is performed is fixed once and for all. On the other hand, the original vision for multi-key FHE was to support "on-the-fly" computation [LTV12] on ciphertext encrypted any subset of public-keys. All other multi-key FHE schemes were not in the plain model.

The second bullet is obtained by relying on the folklore multi-key FHE scheme obtained by nesting a constant number of FHE schemes. There was no previously-known scheme supporting constant-many keys without setup just from LWE.

Technical highlight and an open problem. Our construction is obtained in few modular steps. We first identify a "two-party" NISC protocol (denoted 2rNISC henceforth) for a particular functionality that we call "functional OT". This protocol still supports arbitrary polynomially-many parties, but only the function to be computed is specific and involves just two parties. More specifically, the two parties, acting as the OT sender and receiver, respectively, wish to compute OT with two sender's strings $(\ell_0, \ell_1) = g_1(x_1)$ computed from sender's private input x_1, and a receiver's choice bit $c = g_2(x_2)$ computed from the receiver's private input x_2, where g_1, g_2 are arbitrary public polynomial-size circuits that are different for each computation. 2rNISC enables computing ℓ_c with the sender and receiver sending a single message each. We then show that this can be generically turned into a general-purpose mrNISC. We believe that 2rNISC for the functional OT functionality is an interesting primitive that may find other applications.

Lastly, we show a construction of a 2rNISC for the functional OT functionality, from LWE with polynomial modulus-to-noise ratio and PRFs in NC^1. Our construction draws techniques from homomorphic commitments/signatures [GVW15] and 2-message statistically sender-private OT [BD18] based on LWE. At its core is a weak version of witness encryption for verifying the decommitments of homomorphic commitments, where the decommitments satisfy zero-knowledge property. This partially answers a question left open by the work of [BL20].

We believe that the above modular approach is a contribution of independent interest, as new constructions of our 2rNISC for the functional OT functionality directly yield new constructions of mrNISC. One intriguing open problem is whether it is possible to base mrNISC on DDH or

[4]DSPR stands for the *decision small polynomial ratio* assumption [LTV12] which is used to prove the security of the NTRU encryption scheme.

even CDH. Our reduction shows that, for this purpose, it suffices to build a 2rNISC for a specific functionality from DDH/CDH.

1.2 Related Works

While mrNISC is a new concept that was recently introduced by Benhamouda and Lin [BL20], it is related to (but differs from) many previously-defined variants of minimal-interaction MPC protocols. We refer to [BL20] for a comprehensive comparison and merely mention some of the most related notions. mrNISC can be viewed as a generalization of the notion of reusable NISC of Ishai et al. [IKO+11] (see also [AMPR14, CJS14, BGI+17, BJOV18, CDI+19]) from two parties to multiple parties. mrNISC differs from various completely non-interactive notions such as non-interactive MPC (NIMPC) [BGI+14] and Private Simultaneous Messages (PSM) [FKN94, GIKM98, HLP11, BGI+14, HIJ+16, HIJ+17] which inherently achieve weaker security guarantees or restrict the corruption pattern.

Apart from Benhamouda and Lin's [BL20] recent mrNISC construction from bilinear maps, all other 2-message MPC protocols either rely on strong primitives like indistinguishability obfuscation or general-purpose witness encryption [GGHR14, GLS15], or fall short of being an mrNISC. For instance, the works of Garg and Srinivasan and Benhamouda and Lin [GS18, BL18] constructed 2-round MPC protocols from any 2-round Oblivious Transfer (OT). However, both constructions are not reusable in their first message. This was recently solved by Ananth et al. [AJJ20] and Bartusek et al. [BGMM20] who constructed a 2-round MPC where the first message is reusable across polynomially-many sessions. The construction of [AJJ20] relies on LWE and the construction of [BGMM20] relies on DDH. However, both construction requires all computation sessions to be carried out by a fixed set of parties.

The concept of threshold multi-key FHE is very related to mrNISC. It is plausible that threshold multi-key FHE that are used to get 2-round MPC [AJL+12, MW16, CM15, BJMS20], could also be used to get mrNISC. However, proving it is not straightforward. For instance, as pointed out in [BL20], the current definitions of threshold decryption, e.g., [AJL+12, CM15, MW16, BJMS20] are insufficient for constructing mrNISC, as simulatability only ensures that a single partial decryption can be simulated (hence this definition does not allow to re-use ciphertexts). Even if the proof works out, it would only yield a mrNISC in the CRS model even for semi-honest security.

1.3 Organization of the Paper

We start by a technical overview in Section 2. After recalling preliminaries in Section 3, we show how to construct a 2rNISC for Functional OT in Section 4. We then present our transformation from such a 2rNISC to an mrNISC for any polynomial-time functionality in Section 5. Finally, we formally show applications in Appendix E.

2 Technical Overivew

We now give an overview of our construction of mrNISC protocols in the plain model from LWE with polynomial modulus and PRF in NC^1.

2.1 Review of Definition of mrNISC Protocols

Towards constructing mrNISC protocols, the work of [BL20] defined the notion of mrNISC schemes, with a game-based security definition. Furthermore, they showed that a mrNISC scheme immediately yields a mrNISC protocol that UC-implements an ideal mrNISC functionality that allows for any number of computations over any subsets of inputs registered by parties. Thus, in this work, we focus on implementing mrNISC schemes for polynomial-size circuits.

mrNISC Scheme. An n-party functionality $\mathcal{U}$ is a represented by a Boolean circuit that takes a public input z and n private inputs. If $\mathcal{U}$ is a universal circuit and z specifies the actual function to be computed, then this formalism allows the parties of the mrNISC to compute any function on their private inputs. An mrNISC scheme for $\mathcal{U}$, consists the following three algorithms:

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

Simulation-security requires that the view of an adversary corrupting the evaluator and a subset of parties, can be simulated using just the outputs of the computations.[5] Following [BL20], we consider static corruptions and semi-malicious security. Static corruptions restrict the adversary to corrupt a fixed subset C of parties chosen at the very beginning, and semi-malicious security [AJL+12] restricts the corrupted parties $\{P_i\}_{i\in C}$ to follow the protocol specification, but allows the adversary to choose their inputs and randomness $\{x_i, r_i\}_{i\in C}$ arbitrarily. During an execution of the mrNISC scheme for $\mathcal{U}$, honest and corrupted parties P_i can register their inputs by posting input encodings $\widehat{x}_i$. Multiple computations, each specified by (z^k, I^k), can be carried out as follows: each P_i for $i \in I^k$ sends the corresponding computation encoding α_i^k, which together reveal $y^k = \mathcal{U}(z^k, \{x_i\}_{i\in I^k})$. All the messages from the honest parties, including $\{\widehat{x}_i\}_{i\notin C}$ and $\{\alpha_i^k\}_{k, i\in I^k\setminus C}$, must be simulatable from the outputs $\{y^k\}_k$, the public information of the computations $\{z^k, I^k\}_k$, and the input and randomness of the corrupted parties $\{x_i, r_i\}_{i\in C}$. Furthermore, simulation must hold in the adaptive setting, where the input and computation encodings are interleaved and all x_i and (z^k, I^k) are chosen adaptively by the adversary.

2.2 Step 1: Reusable Functional OT from LWE

We identify a complete 2-party function, called *functional OT* $\mathcal{U}_{\text{fOT}}$, and show 1) how to construct a 2-party reusable NISC scheme for computing $\mathcal{U}_{\text{fOT}}$ in the plain model, and 2) how to bootstrap from $\mathcal{U}_{\text{fOT}}$ to general mrNISC scheme for any circuit $\mathcal{U} \in \text{P}$.

Functional OT. $\mathcal{U}_{\text{fOT}}$ takes three inputs: A public input consisting of two functions $g_1\colon \{0,1\}^{n_1} \to \{0,1\}^\lambda \times \{0,1\}^\lambda$ and $g_2\colon \{0,1\}^{n_2} \to \{0,1\}$ represented as Boolean circuits, a private input $x_1 \in$

[5]It suffices to simulate only these computations that involve at least one honest party. Computations involving only corrupted parties can be viewed as part of the internal computation of the adversary.

$\{0,1\}^{n_1}$ from a party P_1 acting as the $\mathcal{U}_{\text{fOT}}$ sender $x_2 \in \{0,1\}^{n_2}$ from a party P_2 acting as the $\mathcal{U}_{\text{fOT}}$ receiver, and computes

$$\begin{aligned}\mathcal{U}_{\text{fOT}}((g_1, g_2), x_1, x_2) \;:\; & \text{compute sender's strings } (\ell_0, \ell_1) = g_1(x_1), \\ & \text{compute receiver's choice } c = g_2(x_2), \\ & \text{output } \; y = (c, \ell_c)\end{aligned}$$

The name functional OT comes from the fact that both the OT sender's strings ℓ_0, ℓ_1 and receiver's choice bit c are functions on sender's and receiver's private inputs x_1 and x_2.

A 2rNISC scheme for computing $\mathcal{U}_{\text{fOT}}$ provides a way to encode the private input x_i of any party P_i, so that later any two parties P_i and P_j can securely compute $\mathcal{U}_{\text{fOT}}$ (acting as sender and receiver respectively) to reveal only (c, ℓ_c) computed according to arbitrarily chosen functions (g_1, g_2) and their private inputs x_i and x_j, by each sending a single message. Importantly, the encoding $\widehat{x}_i$ of P_i is reusable in any number of $\mathcal{U}_{\text{fOT}}$ computations with different parties and different functions. Note that different from classical OT where (c, ℓ_c) is private to the receiver, a 2rNISC scheme allows to reconstruct (c, ℓ_c) publicly given all messages sent. Jumping ahead, this feature serves exactly the purpose of achieving the public reconstruction property of mrNISC.

Constructing 2rNISC for $\mathcal{U}_{\text{fOT}}$. We construct 2rNISC for $\mathcal{U}_{\text{fOT}}$ in the plain model from LWE with just *polynomial modulus* and PRF in NC^1 in two steps: We start with designing a scheme $\Pi_{\text{fOT}} = (\mathsf{Com}, \mathsf{Encode}, \mathsf{Eval})$ that handles only circuits g_2 with bounded logarithmic depth $O(\log \lambda)$ (whereas the depth of g_1 is unrestricted), and then bootstrap Π to 2rNISC that handles g_2 with unbounded polynomial depth.

GSW Encryption as Homomorphic Commitments. Our 2rNISC makes use of the GSW homomorphic encryption scheme [GSW13], which can be turned into a homomorphic commitment scheme (or homomorphic trapdoor functions) as done in [GVW15]. It enables us to commit to a string $\boldsymbol{x} \in \{0,1\}^n$ in a commitment $\mathbf{C}$, and then homomorphically evaluate any circuit f on $\mathbf{C}$ to obtain a commitment $\mathbf{C}_f$ to $f(\boldsymbol{x})$. More concretely, the scheme publishes a CRS $\mathsf{crs} = \mathbf{A}$ containing a matrix of dimension $N \times M$ for $M = \Omega(N \cdot \log q)$; the matrix $\mathbf{A} = [\mathbf{B}^\top | \boldsymbol{b}_1^\top | \ldots | \boldsymbol{b}_k^\top]^\top$ consists of a random submatrix $\mathbf{B} \leftarrow \mathbb{Z}_q^{(N-k) \times M}$, together with k LWE samples $\{b_l = \boldsymbol{t}_l \mathbf{B} + \boldsymbol{e}_l\}_{l \in [k]}$ w.r.t. independently sampled secret $\boldsymbol{t}_l$ and noise $\boldsymbol{e}_l$, where $\boldsymbol{e}_1$ is sampled from a *truncated* discrete Gaussian distribution and always bounded by $|\boldsymbol{e}_l|_\infty \le B$. Committing to a binary string $\boldsymbol{x}$ simply involves encrypting each bit x_i using GSW encryption and public key $\mathbf{A}$, and the encryption randomness is the decommitment.

$$\begin{aligned}&\text{Commitment to } \boldsymbol{x}\text{: } \{\mathbf{C}_i = \mathbf{A}\mathbf{R}_i + x_i \mathbf{G}\}_i \qquad \text{Decommitment: } \{\mathbf{R}_i\}_i \\ &\text{where } \mathbf{R}_i \leftarrow \{-1,1\}^{M \times N \cdot \lceil \log q \rceil},\ \mathbf{G} \text{ the gadget matrix.}\end{aligned}$$

We note two important details: First, the matrix $\mathbf{A}$ corresponds to the public key in GSW encryption; here, we insist on it containing $k > 1$ LWE samples, where k is a parameter that scales with the input length of the parties. Second, when $\mathbf{A}$ is sampled honestly at random, it satisfies the following well-formedness with overwhelming probability: *1)* it is generated as above using some $\mathbf{B}$, $\boldsymbol{t}_l$, and B-bounded $\boldsymbol{e}_l$'s, and *2)* vectors $\boldsymbol{e}_l$'s are linearly independent over the integers. Observe that the well-formedness can be verified efficiently given the random coins used to sample $\mathbf{A}$. For any $\mathbf{A}$ satisfying property 1), commitments w.r.t. $\mathbf{A}$ are statistical binding, and in fact even extractable using the secrets $\boldsymbol{t}_l$'s. We shall see how property 2) is helpful later.

The homomorphism of GSW enables homomorphic evaluation over the commitments to obtain a commitment to $f(\boldsymbol{x})$ as follows

$$\mathsf{GSW.Eval}(f, \{\mathbf{C}_i\}) = \mathbf{C}_f = \mathbf{A}\mathbf{R}_f + f(\boldsymbol{x})\mathbf{G} \ , \qquad \text{where } \mathbf{R}_f = \mathsf{GSW.RandEval}(f, \{\mathbf{R}_i\}, \{\mathbf{C}_i\}, \boldsymbol{x}) \ .$$

The new decommitment $\mathbf{R}_f$ can be evaluated directly from $\{\mathbf{R}_i\}, \{\mathbf{C}_i\}, \boldsymbol{x}$ and in particular is linear in the original decommitments $\mathbf{R}_i$'s.

<u>From Homomorphic Commitments to 2rNISC.</u> To construct 2rNISC for functional OT, our idea is letting each player P_i commit to its input $\boldsymbol{x}$ as the input encoding, and keep the decommitment as its private state. Note that the homomorphic commitments require a CRS, but we wish to construct 2rNISC in the plain model. Thus, we let each player choose its own CRS.

$$\mathsf{Com}(1^\lambda, \boldsymbol{x}) \ : \ \widehat{\boldsymbol{x}} = (\mathbf{A}, \{\mathbf{C}_i = \mathbf{A}\mathbf{R}_i + x_i\mathbf{G}\}_i), \quad s = \{\mathbf{R}_i\}_i$$

Later two parties, P_1 acting as the sender and P_2 acting as the receiver, wish to compute functional OT w.r.t. (g_1, g_2) on their private inputs denoted as $\boldsymbol{x}_1$ and $\boldsymbol{x}_2$, and have encodings and secret states denoted as $(\widehat{\boldsymbol{x}}_b = (\mathbf{A}_b, \{\mathbf{C}_{b,i}\}), s_b = \{\mathbf{R}_{b,i}\})$ with $b = 1$ for P_1 and $b = 2$ for P_2. P_1 can privately compute sender's strings $(\ell_0, \ell_1) = g_1(\boldsymbol{x}_1)$, and P_2 the receiver's choice $c = g_2(\boldsymbol{x}_2)$. In addition, given $\widehat{\boldsymbol{x}}_2$, both parties can homomorphically evaluate g_2 to obtain a commitment $\mathbf{C}_{g_2} = \mathbf{A}_2\mathbf{R}_{g_2} + c\mathbf{G}$ to c, while P_2 additionally knows the decommitment $\mathbf{R}_{g_2}$.

At this point, we wish to have the following two components to enable computing ℓ_c non-interactively.

- *Witness Encryption of Sender's Strings* (ℓ_0, ℓ_1)*:* P_1 would like to witness encrypt ℓ_b w.r.t. the statement that, under CRS $\mathbf{A}$, $\mathbf{C}_{g_2}$ is a commitment to bit b, so that, ℓ_b is revealed given a witness that is a decommtiment to b, and is hidden if $\mathbf{C}_{g_2}$ is a commitment to $1 - b$. Then the sender's computation encoding is

$$\mathsf{Encode}((g_1, g_2), (\widehat{x}_1, \widehat{x}_2), s_1) \ : \ \alpha_1 = \{\boldsymbol{w}_b \leftarrow \mathsf{WEnc}((\mathbf{A}_2, \mathbf{C}_{g_2}, b), \ell_b)\}_{b \in \{0,1\}}$$

- *Zero-Knowledge Decommitment to Receiver's Choice c:* P_2 would like to open $\mathbf{C}_{g_2}$ to c by sending a decommitment, in a zero-knowledge way that reveals only c and nothing more about $\boldsymbol{x}_2$. Note that the basic decommitment $\mathbf{R}_{g_2}$ is not zero-knowledge and may reveal information of $\boldsymbol{x}_2$.

$$\mathsf{Encode}((g_1, g_2), (\widehat{x}_1, \widehat{x}_2), s_2 = \{\mathbf{R}_i\}_i) \ : \ \alpha_2 = (\mathbf{X}_{g_2} \leftarrow \mathsf{ZKDecom}(g_2, \mathbf{C}_{g_2}, \mathbf{R}_{g_2})) \ ,$$

  where $\mathsf{ZKDecom}$ produces a zero-knowledge decommitment $\mathbf{X}_{g_2}$.

An evaluator given (α_1, α_2) can witness decrypt to obtain ℓ_c as desired.

<u>Semi-Malicious Security and "Promise" WE and ZK Decommitments.</u> The main technical challenge is co-designing WE and ZK decommitments so that the latter can decrypt the former. For this we will draw techniques from previous works for constructing context-hiding homomorphic signatures [GVW15] and 2-message statistically sender private OT [BD18]. At the same time, we crucially rely on the fact that our 2rNISC only need to be secure against semi-malicious adversaries to simplify the requirements on WE and ZK decommitments. The key observation is that a

semi-malicious corrupted party P_2 must generate its input encoding $(\mathbf{A}_2, \{\mathbf{C}_{2,i}\}_i)$ using the honest algorithm, albeit using arbitrary randomness. This means that *i)* $\mathbf{A}_2$ must be well-formed and *ii)* $\{\mathbf{C}_{2,i}\}_i$ must be a valid commitment $\{\mathbf{A}_2\mathbf{R}_i + x_{2,i}\mathbf{G}\}_i$ to some input $\boldsymbol{x}_2$ with a decommitement $\mathbf{R}_i$ of 1/-1 values . As a result, $\mathbf{C}_{g_2} = \mathbf{A}_2\mathbf{R}_{g_2} + g_2(\boldsymbol{x}_2)\mathbf{G}$ must be a valid commitment to $g_2(\boldsymbol{x}_2) = 0/1$ with a decommitment $\mathbf{R}_{g_2}$ of small magnitude[6].

Therefore, the correctness and security of WE and ZK decommitments only need to hold w.r.t. well-formed $\mathbf{A}$ (i.e., $\mathbf{A}_2$) and valid commitment $\mathbf{C}$ (i.e., $\mathbf{C}_{g_2}$) to 0/1 with small decommitment, and does not need to hold w.r.t. ill-formed $\mathbf{A}$ or invalid commitment $\mathbf{C}$ — we refer to this as the *promise* version of WE and ZK decommitments:

- *Yes instances* $(\mathbf{A}, \mathbf{C}, b)$ contain a well-formed $\mathbf{A}$ and a valid commitment $\mathbf{C}$ to bit b, and we require the ZK property of the decommitments and correctness of WE for them.
- *No instances* $(\mathbf{A}, \mathbf{C}, b)$ contain a well-formed $\mathbf{A}$ and a valid commitment $\mathbf{C}$ to bit $1 - b$, and we require the hiding property of WE for them.

Thanks to the fact that it suffices to focus on the promise version, we manage to give a relatively simple construction of WE and ZK decommitment. Next we proceed to their description; by default, all matrices $\mathbf{A}$'s are well-formed and commitments $\mathbf{C}$'s are valid 0/1 commitments.

ZK Decommitment The context-hiding homomorphic signature schemes of [GVW15] provides a way to generate zero-knowledge decommtiments. If the committer wishes to open $\mathbf{C}_f = \mathbf{A}\mathbf{R}_f + f(\boldsymbol{x})\mathbf{G}$ to $f(\boldsymbol{x}) = b$ w.r.t. CRS $\mathbf{A}$, it constructs the matrix

$$\mathbf{D}^{(b)} = [\mathbf{A} \mid \mathbf{C}_f + (1-b)\mathbf{G}] = [\mathbf{A} \mid \mathbf{A}\mathbf{R}_f \pm \mathbf{G}] \in \mathbb{Z}_q^{N \times M'},\ M' = M + N\lceil \log q \rceil \ ,$$

and uses $\mathbf{R}_f$ as a right-trapdoor [ABB10, CHKP10] of $\mathbf{D}^{(b)}$ to sample a short B'-bounded vector $\boldsymbol{v}$, for appropriately set B' such that, $\mathbf{D}^{(b)}\boldsymbol{v} = \boldsymbol{u}$, where $\boldsymbol{u}$ is a random vector published additionally in the CRS. The vector $\boldsymbol{v}$ is the new decommitment.[7] $\boldsymbol{v}$ together with $\mathbf{A}$ and the original commitment $\{\mathbf{C}_i\}$ to $\boldsymbol{x}$ reveals no more information beyond that $f(\boldsymbol{x}) = b$, since they can be jointly simulated using only (f, b), by sampling $\mathbf{A}$ at random with a trapdoor $\mathbf{T}_\mathbf{A}$ [MP12, ABB10], $\mathbf{C}_i$'s at random, and $\boldsymbol{v}$ using $\mathbf{T}_\mathbf{A}$ as a left-trapdoor of $\mathbf{D}^{(b)}$. A random $\mathbf{A}$ is computationally indistinguishable from a well-formed $\mathbf{A}$ by LWE, and $\boldsymbol{v}$ sampled using the left or the right trapdoor is statistically close.

However, we do not know how to construct a matching WE for verifying the above ZK decommitment and need to modify the decommitment as follows. The new decommitment of $\mathbf{A}, \mathbf{C}_f$ to $f(\boldsymbol{x}) = b$ contains a short B'-bounded basis $\mathbf{X}_f \in \mathbb{Z}^{M' \times M'}$ of the lattice $\Lambda_q^\perp(\mathbf{D}^{(\mathbf{b})}) = \{\boldsymbol{z} \in \mathbb{Z}^{M'} : \mathbf{D}^{(\mathbf{b})}\boldsymbol{z} = \mathbf{0} \pmod q\}$ over the integers, that is, $\mathbf{D}^{(\mathbf{b})}\mathbf{X}_f = \mathbf{0}^{N \times M'}$ and is $\mathbf{X}_f$ has full rank over the integers. Such a basis can be sampled again using $\mathbf{R}_f$ as a right-trapdoor of $\mathbf{D}^{(b)}$, and can be simulated together with $\mathbf{A}, \{\mathbf{C}_i\}$ by sampling $\mathbf{A}$ without a trapdoor $\mathbf{T}_\mathbf{A}$ and using it as a left-trapdoor of $\mathbf{D}^{(b)}$ to sample the basis. In summary, our ZK decommitment is generated as:

$$\mathsf{ZKDecom}(f, b, \mathbf{D}^{(b)}, \mathbf{R}_f) \ : \ \mathbf{X}_f \leftarrow \mathsf{SampleRight}(\mathbf{A}, \pm\mathbf{G}, \mathbf{R_f}, \mathbf{T}_\mathbf{G}, \alpha) \ .$$

where $\mathbf{T}_\mathbf{G}$ is a trapdoor of the gadget matrix $\mathbf{G}$ and α controls the norm of the trapdoor.

[6]The magnitude scales exponentially with the depth of g_2, which is relatively small if we set the modulus to be sufficiently large.

[7]It can be verified efficiently by checking whether it has small magnitude and $\mathbf{D}^{(b)}\boldsymbol{v} = \boldsymbol{u}$

PROMISE WITNESS ENCRYPTION. To design a compatible WE that can be decrypted using the above ZK decommitments. we crucially rely on the following fundamental properties of lattices defined by a matrix $\mathbf{D} \in \mathbb{Z}_q^{N \times M'}$.

- If the lattice $\Lambda_q^{\perp}(\mathbf{D}) = \{\boldsymbol{z} \in \mathbb{Z}^{M'} : \mathbf{D}\boldsymbol{z} = \mathbf{0} \pmod q\}$ has a B'-bounded basis $\mathbf{X}$ over the integers, then vectors of form $\boldsymbol{s}\mathbf{D} + \boldsymbol{e}$ can be efficiently decoded using $\mathbf{X}$, and $\boldsymbol{s}$ can be recovered, provided that the norm of $\boldsymbol{e}$ is sufficiently smaller than q/B'.

- On the other hand if the lattice $\Lambda_q(\mathbf{D}) = \{\boldsymbol{y} \in \mathbb{Z}^{M'} : \boldsymbol{y} = \boldsymbol{s}\mathbf{D} \pmod q\}$ contains k linearly independent vectors of norm $\ll q/B'$, then vectors of form $\boldsymbol{s}\mathbf{D} + \boldsymbol{e}$ is *lossy* and $\boldsymbol{s}$ has n bits of entropy, if k is sufficiently larger than n. This is essentially because the components of $\boldsymbol{s}\mathbf{A}$ in the direction the short vectors are masked by $\boldsymbol{e}$.

The work of [BD18] relied on the above properties in their construction of two message statistically sender-private OT from LWE. We here rely on them to achieve respectively the correctness and hiding property of our promise WE. To encrypt a string ℓ_b, under a statement $(\mathbf{A}, \mathbf{C} = \mathbf{C}_f, b)$, our WE does:

$$\begin{aligned}\mathsf{WEnc}((\mathbf{A}, \mathbf{C}, b), \ell_b) \ : \ & \mathbf{D}^{(b)} = [\mathbf{A} \mid \mathbf{C} - (1-b)\mathbf{G}], \ \boldsymbol{w}_b = \boldsymbol{s}_b\mathbf{D}^{(b)} + \boldsymbol{e}_b, \\ & \widehat{\ell}_b = \mathsf{Ext}(\mathsf{sd}, \boldsymbol{s}_b) \oplus \ell_b \\ & \text{output } (\boldsymbol{w}_b, \mathsf{sd}, \widehat{\ell}_b)\end{aligned}$$

where Ext is a strong seeded extractor and sd is a randomly sampled seed, $\boldsymbol{s}_b$ is a random secret from $\mathbb{Z}_q^N$, and $\boldsymbol{e}_b$ is from a truncated discrete Gaussian distribution with appropriate parameter.

- *Correctness for Yes Instances:* For a well-formed $\mathbf{A}$ and a valid commitment $\mathbf{C} = \mathbf{A}\mathbf{R} + b\mathbf{G}$ to b, the ZK decommitment $\mathbf{X}$ is exactly a short-basis of $\Lambda_q^{\perp}(\mathbf{D}^{(b)})$. Therefore, by the first lattice property, given $\mathbf{X}$, the decryptor can efficiently decode $\boldsymbol{w}_b$ to obtain $\boldsymbol{s}_b$ and then recover ℓ_b.

- *Hiding for No Instances:* For a well-formed $\mathbf{A}$ and a valid commitment $\mathbf{C} = \mathbf{A}\mathbf{R} + (1-b)\mathbf{G}$ to $1-b$, $\mathbf{D}^{(b)} = [\mathbf{A} \mid \mathbf{A}\mathbf{R}]$ and hence the lattice $\Lambda_q(\mathbf{D}^{(b)})$ contains at least k short vectors. This is because, by the structure of a well-formed $\mathbf{A}$, for every $l \in [k]$, $(-\boldsymbol{t}_l||1)\mathbf{D}^{(b)} = (\boldsymbol{e}_l||\boldsymbol{e}_l\mathbf{R})$ is short as $\boldsymbol{e}_l$ and $\mathbf{R}$ are. Moreover, these vectors are independent as long as $\boldsymbol{e}_l$'s are (and $k < \dim(\boldsymbol{e}_l) = M$), where the latter is guaranteed by the (second requirement of) well-formedness of $\mathbf{A}$. Therefore, by the second lattice property, $\boldsymbol{s}_b$ has n bits of entropy conditioned on $\boldsymbol{w}_b$ and the output of the extractor information theoretically hides ℓ_b.

PUTTING PIECES TOGETHER. Combining the homomorphic commitment scheme with ZK decommitments and the witness encryption, we obtain 2rNISC for computing functional OT with semi-malicious security. Let's now examine the magnitude of the modulus, which we wish to be polynomial. Based on LWE, to support homomorphic evaluation of a circuit g_2 of depth d requires the modulus to grow exponentially in d . Therefore, only when d is a fixed logarithmic function in the security parameter λ, would the modulus be a fixed polynomial in λ as desired.

Following a technique used in [BL20], we can generically bootstrap to 2rNISC supporting circuits g_2 with unbounded polynomial depth, with the help of a PRF in NC^1 and Yao's garbled circuits. At a high-level, P_1 is going to hide the sender's string ℓ_b in a garbled circuit $\widehat{G}_{\ell_b}$ for a function $G_{\ell_b}(\Lambda)$ that on input a randomized encoding Λ, outputs ℓ_b iff Λ evaluates to b. At evaluation time, the evaluator will obtain the set of labels $\{\bar{\ell}_j\}$ of $\widehat{G}_{\ell_b}$ corresponding exactly to a randomized encoding Λ of $(g_2, \boldsymbol{x}_2)$ generated using pseudorandom coins expanded via PRF on a key k_2 belong to the receiver P_2. Then the evaluator can recover ℓ_b iff $g_2(\boldsymbol{x}_2) = b$. Crucially, the task for revealing the labels corresponding to Λ can exactly be accomplished using 2rNISC for logarithmic-depth receiver's circuits, as every bit of Λ can be computed by a logarithmic-depth circuit evaluated on $(\boldsymbol{x}_2, k_2)$ if $\mathsf{PRF} \in \mathrm{NC}^1$. Correspondingly, every party now needs to commit to their private input $\boldsymbol{x}$ and a PRF key k. This yields our final 2rNISC for functional OT from LWE with polynomial modulus and PRF in NC^1.

2.3 Step 2: 2rNISC for Functional OT to General mrNISC for P

We construct general mrNISC for polynomial-sized circuits from 2rNISC for functional OT following a similar approach as [BL20], which in turn is based on the round collapsing approach for constructing 2-round MPC protocols started in [GGHR14, GLS15]. The *round-collapsing* approach collapses an inner MPC protocol with a polynomial L number of rounds into a 2-round outer MPC protocol, essentially by letting every party garble its next-step message function for computing the inner MPC messages. The challenge lies in how to enable the garbled circuits generated independently by different parties "talk" to each other: the output of one party's garbled circuit is the input of another party's garbled circuit. What is new in this work is that we use 2rNISC for functional OT to enable this, which is weaker than the tools used in previous works. Specifically, the work of [BL20] proposed and constructed a primitive called Witness Encryption for NIZK of commitments, which is a witness encryption scheme for verifying NIZK proof of the correctness of deterministic computation over committed values. In comparison, 2rNISC is weaker (in particular, is implied by WE for NIZK of commitments) and has a simpler definition, thanks to which we manage to instantiate it from LWE and PRF in NC^1. Next, we give an overview of our mrNISC from 2rNISC for functional OT.

Round Collapsing via 2rNISC for Functional OT. In mrNISC, each party P_i uses 2rNISC for functional OT to encode its private input x_i and a PRF key fk_i, $((\widehat{x}_i, \mathrm{fk}\rho_i), s_i) \leftarrow \mathsf{Com}(x_i, \rho_i)$. The PRF key will be used to expand pseudo-random coins for running the inner MPC protocol and generating garbled circuits described below.

A subset I of parties $\{P_i\}_{i \in I}$ wishes to compute $f(z, \{x_i\}_{i \in I})$. Assume that each party P_1 in the inner MPC broadcasts one message m_i^ℓ in each round ℓ; but we now want to carry out this multi-round interaction non-interactively. To do so, each P_i sends one garbled circuit $\widehat{\mathsf{F}}_i^\ell$ per round $\ell \in [L]$ of the inner MPC protocol corresponding to the next message function F_i^ℓ of P_i. This garbled circuit takes as input all the messages $\boldsymbol{m}^{<\ell} = \{m_j^l\}_{l<\ell, j\in[n]}$ sent in previous rounds, and outputs the next message m_i^ℓ of P_i of the inner MPC (or the output for the last round $\ell = L$).

For an evaluator to compute the output from these garbled circuits $\{\widehat{\mathsf{F}}_i^\ell\}_{\ell\in[L], i\in[n]}$, we need a mechanism to reveal the labels of P_i's garbled circuits $\widehat{\mathsf{F}}_i^\ell$ that correspond to the correct messages of the inner MPC. More specifically, let k_0, k_1 be two labels of P_i's garbled circuit $\widehat{\mathsf{F}}_i^\ell$ for an input wire that takes in the t'th bit $y = m_{j,t}^l$ of a message from P_j. The goal is revealing only k_y, which can be accomplished using exactly 2rNISC for functional OT.

First, we let k_0, k_1 be expanded from P_i's PRF key ρ_i, that is $(k_0, k_1) = g_1(x_i, \mathrm{fk}_i)$ for some well-chosen g_1. Second, $y = m_j^1$ is P_j's inner MPC message computed from its input x_j and randomness expanded from ρ_j; hence, $y = g_2(x_i, \rho)$ for some g_2. Therefore, to reveal k_y, we can modify garbled circuits of P_i and P_j to additionally output the right 2rNISC computation encodings:

- $\widehat{\mathsf{F}}_i^{\ell-1}$ for round $\ell - 1$ additionally outputs $\alpha_i \leftarrow \mathsf{Encode}((g_1, g_2), (\widehat{x}_i, \widehat{\mathrm{fk}}_i), (\widehat{x}_j, \widehat{\mathrm{fk}}_j), s_i)$.
- $\widehat{\mathsf{F}}_j^{l}$ for round l where P_j outputs m_j^l additionally outputs $\alpha_j \leftarrow \mathsf{Encode}((g_1, g_2), (\widehat{x}_i, \widehat{\mathrm{fk}}_i), (\widehat{x}_j, \widehat{\mathrm{fk}}_j), s_j)$.

By the correctness and security of 2rNISC, the evaluator can recover only k_y as desired.

We do not know however how to prove the above construction secure. The issue is that the PRF key fk_i is used to generate the labels of all the garbled circuits and our security hybrids switch garbled circuits to simulated ones, one by one. Concretely, to switch the garbled circuit for round ℓ into a simulated one, its input labels must first be switched to uniformly random ones (instead of being PRF outputs). The usual solution for that is to use the pseudorandom property of the PRF. Unfortunately, we cannot do that, because the secret key fk_i of the PRF is an input of the 2rNISC for functional OT for the rounds after round ℓ. To solve this issue, our final scheme actually uses $L + 1$ PRF keys, one for the randomness of the inner MPC and one for the labels of the garbled circuit for each of the L rounds. To make sure that the input encodings do not depend on the parameters of computations later, we employ a constant round inner MPC protocol, that is, $L = O(1)$.

3 Preliminaries

We denote the security parameter by λ. Let $\mathbb{N}$ be the set of non-negative integers. A function $\mathsf{negl}\colon \mathbb{N} \to \mathbb{N}$ is negligible if for any polynomial $p\colon \mathbb{N} \to \mathbb{N}$, for any large enough $\lambda \in \mathbb{N}$, $\mathsf{negl}(\lambda) < 1/p(\lambda)$.

We recall the notion of statistical and computational indistinguishability in Appendix A.

We make use of garbled circuits, collision-resistant hash functions, and pseudorandom functions. A *garbled circuit* scheme GC is defined as a tuple of four polynomial-time algorithms $\mathsf{GC} = (\mathsf{GC.Gen}, \mathsf{GC.Garble}, \mathsf{GC.Eval}, \mathsf{GC.Sim})$: i) $\mathsf{key} \leftarrow_{\mathrm{R}} \mathsf{GC.Gen}(1^\lambda)$ generates labels or keys $\mathsf{key} = \{\mathsf{key}[i, b]\}_{i, b \in \{0,1\}}$, ii) $\widehat{C} \leftarrow_{\mathrm{R}} \mathsf{GC.Garble}(\mathsf{key}, C)$ garbles the circuit, iii) $y = \mathsf{GC.Eval}(\widehat{C}, \mathsf{key}')$ evaluates the garbled circuit on the input x corresponding to the selected labels $\mathsf{key}' = \{\mathsf{key}[i, x_i]\}_i$, iv) $(\mathsf{key}', \widetilde{C}) \leftarrow_{\mathrm{R}} \mathsf{GC.Sim}(1^\lambda, y)$ simulates a garbled circuit and the corresponding input labels from the output.

3.1 General Lattice Preliminaries

Lattices. An m-dimensional lattice $\mathcal{L}$ is a discrete additive subgroup of $\mathbb{R}^m$. Given positive integers n, m, q and a matrix $\mathbf{A} \in \mathbb{Z}_q^{n \times m}$, we let $\Lambda_q^{\perp}(\mathbf{A})$ denote the lattice $\{\boldsymbol{x} \in \mathbb{Z}^m \mid \mathbf{A}\boldsymbol{x}^\top = \mathbf{0}^\top \bmod q\}$.

Discrete Gaussians. Let σ be any positive real number. The Gaussian distribution $\mathcal{D}_\sigma$ with parameter σ is defined by the probability distribution function $\rho_\sigma(\boldsymbol{x}) = \exp(-\pi \|x\|^2/\sigma^2)$. For any discrete set $\mathcal{L} \subseteq \mathcal{R}^m$, define $\rho_\sigma(\mathcal{L}) = \sum_{\boldsymbol{x} \in \mathcal{L}} \rho_\sigma(\boldsymbol{x})$. The discrete Gaussian distribution $\mathcal{D}_{\mathcal{L},\sigma}$ over $\mathcal{L}$ with parameter σ is defined by the probability distribution function $\rho_{\mathcal{L},\sigma}(\boldsymbol{x}) = \rho_\sigma(\boldsymbol{x})/\rho_\sigma(\mathcal{L})$.

The following lemma (e.g., [MR04, Lemma 4.4]) shows that if the parameter σ of a discrete Gaussian distribution is small, then any vector drawn from this distribution will be short (with high probability).

Lemma 3.1. *Let m, n, q be positive integers with $m > n$, $q > 2$. Let $\mathbf{A} \in \mathbb{Z}_q^{n \times m}$ be a matrix of dimensions $n \times m$, $\sigma \in \widetilde{\Omega}(n)$, and $\mathcal{L} = \Lambda_q^{\perp}(\mathbf{A})$. Then, there is a negligible function $\mathsf{negl}(\cdot)$ such that*

$$\Pr_{\boldsymbol{x} \leftarrow \mathcal{D}_{\mathcal{L},\sigma}} [\|\boldsymbol{x}\| > \sqrt{m}\sigma] \leq \mathsf{negl}(n),$$

where $\|\boldsymbol{x}\|$ denotes the ℓ_2 norm of $\boldsymbol{x}$.

Truncated Discrete Gaussians. The truncated discrete Gaussian distribution over $\mathbb{Z}^m$ with parameter σ, denoted by $\widetilde{\mathcal{D}}_{\mathbb{Z}^m,\sigma}$, is the same as the discrete Gaussian distribution $\mathcal{D}_{\mathbb{Z}^m,\sigma}$ except that it outputs 0 whenever the ℓ_∞ norm exceeds $\sqrt{m}\sigma$. By definition, we can say that $\widetilde{\mathcal{D}}_{\mathbb{Z}^m,\sigma}$ is $\sqrt{m}\sigma$-bounded, where a family of distributions $\mathcal{D} = \{\mathcal{D}_\lambda\}_{\lambda \in \mathbb{N}}$ over the integers is *B-bounded* (for $B = B(\lambda) > 0$) if for every $\lambda \in \mathbb{N}$ it holds that $\Pr_{x \leftarrow \mathcal{D}_\lambda}[|x| \leq B(\lambda)] = 1$.

Also, by Lemma 3.1 we get that $\widetilde{\mathcal{D}}_{\mathbb{Z}^m,\sigma}$ and $\mathcal{D}_{\mathbb{Z}^m,\sigma}$ are statistically indistinguishable. Therefore, in the preliminaries below, unless specified, the lemata will apply in the setting where by sampling from discrete Gaussian we mean sampling from truncated discrete Gaussian distribution.

3.2 Learning With Errors

The learning with errors (LWE) problem was defined by Regev [Reg05]. The $\mathrm{LWE}_{n,m,q,\chi}$ problem for parameters $n, m, q \in \mathbb{N}$ and for a distribution χ supported over $\mathbb{Z}$ is to distinguish between the following pair of distributions

$$(\mathbf{A}, \boldsymbol{s}\mathbf{A} + \boldsymbol{e} \bmod q) \quad \text{and} \quad (\mathbf{A}, \boldsymbol{u}),$$

where $\mathbf{A} \leftarrow \mathbb{Z}_q^{n \times m}$, $\boldsymbol{s} \leftarrow \mathbb{Z}_q^{1 \times n}$, $\boldsymbol{e} \leftarrow \chi^{1 \times n}$ and $\boldsymbol{u} \leftarrow \mathbb{Z}_q^{1 \times m}$. Similarly, we can define the matrix version of the problem, which is known to be hard, if the version above is hard. Specifically, let $k \in \mathsf{poly}(n, m)$, then in the matrix the task is to distinguish between the following two distributions

$$(\mathbf{A}, \mathbf{S}\mathbf{A} + \mathbf{E} \bmod q) \quad \text{and} \quad (\mathbf{A}, \mathbf{U}),$$

where $\mathbf{A} \leftarrow \mathbb{Z}_q^{n \times m}$, $\mathbf{S} \leftarrow \mathbb{Z}_q^{k \times n}$, $\mathbf{E} \leftarrow \chi^{k \times n}$ and $\mathbf{U} \leftarrow \mathbb{Z}_q^{k \times m}$.

The gadget matrix [MP12]. Fix a dimension n and a modulus q. Define the gadget vector $\boldsymbol{g} = (1, 2, 4, \ldots, 2^{\log q - 1})$ and the gadget function $g^{-1} \colon \mathbb{Z}_q \to \{0,1\}^{\lceil \log q \rceil}$ to be the function that computes the $(\log q)$th bit decomposition of an integer. For some integer z the function is defined as $g^{-1}(z) = \boldsymbol{v} = (v_1, \ldots, v_{\log q})$ where $v_i \in \{0, 1\}$ such that $z = \langle \boldsymbol{g}, \boldsymbol{v} \rangle$. By extension we define the augmented gadget function $G^{-1} \colon \mathbb{Z}_q^{n \times m} \to \{0,1\}^{(n \cdot \lceil \log q \rceil) \times m}$ to be the function that computes the $(\log q)$th bit decomposition of every integer in a matrix $\mathbf{A} \in \mathbb{Z}_q^{n \times m}$, and arranges them as a binary matrix of dimension $(n \cdot \log q) \times k$ which we denote $\boldsymbol{G}^{-1}(\mathbf{A})$. Hence, $\mathbf{G}_n \cdot G^{-1}(\boldsymbol{z}) = \mathbf{Z}$, where the gadget matrix $\mathbf{G}_n$ is $\mathbf{G}_n = \boldsymbol{g} \otimes \mathbf{I}_n \in \mathbb{Z}_q^{n \times (n \cdot \lceil \log q \rceil)}$. When n is clear from context, we denote $\mathbf{G}_n$ simply by $\mathbf{G}$.

3.3 Review of Gentry-Sahai-Waters FHE Scheme

We now recall the Gentry-Sahai-Waters FHE scheme [GSW13]. The scheme has the following overall structure:

GSW.Setup: The public key consists of a matrix $\mathbf{A} \in \mathbb{Z}_q^{n\times m}$. This matrix is typically generated by sampling a matrix $\mathbf{B} \in \mathbb{Z}_q^{n_1\times m}$, a secret $\mathbf{S} \leftarrow \mathbb{Z}_q^{k\times n_1}$, errors $\mathbf{E} \leftarrow \chi^{k\times m}$, and finally setting $\mathbf{A} = [\mathbf{B}^\top|(\mathbf{SB}+\mathbf{E})^\top]^\top \in \mathbb{Z}_q^{n\times m}$ where $n = n_1 + k$ and $m \in \Omega(n \cdot \lceil \log q\rceil))$. The secret key is $\mathbf{S}$.

GSW.Encrypt: To encrypt a message $\mu \in \{0,1\}$, sample randomness $\mathbf{R} \in \{-1,0,1\}^{m\times(n\cdot\lceil\log q\rceil)}$ and finally setting $\mathbf{C} = \mathbf{A}\cdot\mathbf{R} + \mu\cdot\mathbf{G}$. Note that if $\mathbf{A}$ is generated in the manner above, μ is recoverable, and if it is generated at random, then μ is information theoretically lost.

GSW.Eval: Let $f : \{0,1\}^\kappa \to \{0,1\}$ be a depth $d(\kappa)$ boolean circuit, then, given honestly generated ciphertexts $\mathbf{C}_i = \mathbf{A}\cdot\mathbf{R}_i + \mu_i\cdot\mathbf{G}$ for $i \in [\kappa]$. Then $\mathsf{GSW.Eval}(f, \{\mathbf{C}_i\}_{i\in[\kappa]})$ computes the evaluated ciphetext $\widetilde{\mathbf{C}} = \mathbf{A}\cdot\widetilde{\mathbf{R}} + f(\mu_1,\ldots,\mu_\kappa)\cdot\mathbf{G}$. There are two facts about this computation:

Randomness Homomorphism: There is a polynomial time algorithm GSW.RandEval that on input $\mathbf{A}, \{\mathbf{R}_i, \mu_i\}_{i\in[\kappa]}$, and f, computes $\widetilde{\mathbf{R}}$.

Bounds: If $f \in \mathsf{NC}^1$, then $\|\widetilde{\mathbf{R}}\|_\infty \le O(4^d \cdot m)$ as shown in [BV14]. Otherwise, $\|\widetilde{\mathbf{R}}\|_\infty \le O(m^d)$ [GSW13].

3.4 Lattice Trapdoors

Definition 3.1 (Lattice trapdoors [Ajt99, GPV08, AP09, MP12])**.** *There is an efficient randomized algorithm* $\mathsf{TrapGen}(1^n, 1^m, q)$ *that given any integers* $n \ge 1, q \ge 2$ *and* $m \in \Omega(n \log q)$*, outputs a full-rank matrix* $\mathbf{A} \in \mathbb{Z}_q^{n\times m}$ *and a trapdoor matrix* $\mathbf{T_A} \in \mathbb{Z}^{m\times m}$ *such that*

1. $\mathbf{A}\cdot\mathbf{T_A} = \mathbf{0}^{n\times m} \bmod q$.
2. *The distribution of* $\mathbf{A}$ *is* $\mathsf{negl}(n)$*-close to uniform.*
3. $\mathbf{T_A} \in \mathbb{Z}^{m\times m}$ *is a short matrix with linearly independent columns over* $\mathbb{R}$*. More precisely,* $\|\mathbf{T_A}\|_{\mathsf{GS}} = O(\sqrt{n\cdot\log q})$*, where for a matrix* $\mathbf{X}$*,* $\|\mathbf{X}\|_{\mathsf{GS}}$ *is the operator norm of the matrix obtained by performing Gram-Schmidt (GS) orthogonalization of* $\mathbf{X}$*.*

The following lemma is standard and follows from the leftover hash lemma.

Lemma 3.2. *For any* $k \in \mathsf{poly}(n)$ *and* $m \in \Omega(n\log q)$*, the following two distributions are* $\mathsf{negl}(n)$*-close in statistical distance:*

$$\{(\mathbf{A}, \mathbf{T_A}, \mathbf{U}) \,|(\mathbf{A}, \mathbf{T_A}) \leftarrow \mathsf{TrapGen}(1^n, 1^m, q),\ \mathbf{U} \leftarrow \mathbb{Z}_q^{n\times k}\}$$

and

$$\{(\mathbf{A}, \mathbf{T_A}, \mathbf{A}\cdot\mathbf{R}) \,|(\mathbf{A}, \mathbf{T_A}) \leftarrow \mathsf{TrapGen}(1^n, 1^m, q),\ \mathbf{R} \leftarrow \{-1,+1\}_q^{m\times k}\}.$$

We will use the following algorithms for sampling trapdoor matrices.

Algorithm $\mathsf{SampleLeft}(\mathbf{A}, \mathbf{B}, \mathbf{T_A}, \alpha) \mapsto \mathbf{T}_{[\mathbf{A}|\mathbf{B}]}$**:** The sample left algorithm takes as input a full rank matrix $\mathbf{A} \in \mathbb{Z}_q^{n\times m_1}$, a matrix $\mathbf{B} \in \mathbb{Z}_q^{n\times m_2}$, a trapdoor $\mathbf{T_A}$ and it outputs a trapdoor $\mathbf{T}_{[\mathbf{A}|\mathbf{B}]}$ of $[\mathbf{A} \mid \mathbf{B}]$.

Algorithm $\mathsf{SampleRight}(\mathbf{A}, \mathbf{B}, \mathbf{R}, \mathbf{T_B}, \alpha) \mapsto \mathbf{T}_{[\mathbf{A}|\mathbf{AR}+\mathbf{B}]}$**:** The sample right algorithm takes as input a matrix $\mathbf{A} \in \mathbb{Z}_q^{n\times m_1}$, a full rank matrix $\mathbf{B} \in \mathbb{Z}_q^{n\times m_2}$ and its trapdoor $\mathbf{T_B}$, along with $\mathbf{R} \in \mathbb{Z}_q^{m_1\times m_2}$. It outputs a trapdoor $\mathbf{T}_{[\mathbf{A}|\mathbf{A}\cdot\mathbf{R}+\mathbf{B}]}$ of $[\mathbf{A}|\mathbf{A}\cdot\mathbf{R}+\mathbf{B}]$.

The following lemma says that the process of sampling from $\mathsf{SampleLeft}$ and $\mathsf{SampleRight}$ produce indistinguishable outputs, when executed on the appropriate inputs. The lemma follows from [ABB10, CHKP10].

Lemma 3.3 (Indistinguishability of $\mathsf{SampleRight}$, $\mathsf{SampleLeft}$)**.** *Let $\mathbf{A} \in \mathbb{Z}_q^{n\times m_1}$ be a full rank matrix with a trapdoor $\mathbf{T_A}$. Let $\mathbf{B} \in \mathbb{Z}_q^{n\times m_2}$ be a full rank matrix with a trapdoor $\mathbf{T_B}$. Let $\mathbf{R} \in \mathbb{Z}^{m_1\times m_2}$. Let*

$$\alpha > \max\left\{ \|\mathbf{T_A}\|_{\mathsf{GS}} \cdot \omega(\sqrt{\log(m_1+m_2)}), \|\mathbf{T_B}\|_{\mathsf{GS}} \cdot \|\mathbf{R}\| \cdot \omega(\sqrt{\log(m_2)}) \right\}.$$

Then, the following two distributions are statistically close (up to negligible in n distance):

$$\{\mathbf{X} \mid \mathbf{X} \leftarrow \mathsf{SampleLeft}(\mathbf{A}, \mathbf{A}\cdot\mathbf{R}+\mathbf{B}, \mathbf{T_A}, \alpha)\}$$

and

$$\{\mathbf{X} \mid \mathbf{X} \leftarrow \mathsf{SampleRight}(\mathbf{A}, \mathbf{B}, \mathbf{R}, \mathbf{T_B}, \alpha)\}$$

Further, $\|\mathbf{X}\| \in O(\sqrt{m_1+m_2}\cdot\alpha)$.

3.5 Lossy Modes and Unique Decoding

For a given matrix $\mathbf{A} \in \mathbb{Z}_q^{n\times m}$ we consider the function:

$$f_{\mathbf{A}}(\boldsymbol{s}, \boldsymbol{e}) = \boldsymbol{s}\cdot\mathbf{A} + \boldsymbol{e} \bmod q,$$

where $\boldsymbol{s} \in \mathbb{Z}_q^{1\times n}$ and $\boldsymbol{e} \in \mathbb{Z}_q^{1\times m}$. We now consider two settings where in one $f_{\mathbf{A}}$ is invertible and in the other it is lossy.

Invertible mode. When we have a short trapdoor for $\mathbf{A}$, and if $\boldsymbol{e}$ is short, then $\boldsymbol{s}$ is recoverable. This is captured by the following lemma.

Lemma 3.4 ([MP12])**.** *There exist a polynomial time (deterministic) algorithm* $\mathsf{RecoverSecret}$ *such that the following holds. Let $\mathbf{A} \in \mathbb{Z}_q^{n\times m}$ be any full rank matrix and $\mathbf{T_A}$ be a corresponding trapdoor. Let $\boldsymbol{s} \in \mathbb{Z}_q^{1\times n}$ and $\boldsymbol{e} \in \mathbb{Z}^{1\times m}$ be arbitrary. Then,* $\mathsf{RecoverSecret}(\mathbf{A}, \mathbf{T_A}, \boldsymbol{s}\mathbf{A}+\boldsymbol{e} \bmod q) = \boldsymbol{s}$ *whenever $q > \|\mathbf{T_A}\|\cdot\|\boldsymbol{e}\|$.*

Lossy Mode. In the other extreme when the row span of $\mathbf{A}$ has k linearly independent vectors of short norm, $\boldsymbol{s}$ is chosen at random from $\mathbb{Z}_q^n$, and $\boldsymbol{e} \leftarrow \mathcal{D}_{\mathbb{Z}^m,\sigma}$ is sampled from a wide enough discrete Gaussian, then $\boldsymbol{s}\boldsymbol{A}+\boldsymbol{e} \bmod q$ hides $\boldsymbol{s}$ information theoretically. This is captured by the following lemma.

Lemma 3.5 (From Lemma 4.3 and Lemma 3.2 of [BD18])**.** *Let $\mathbf{A} \in \mathbb{Z}_q^{n\times m}$ where $m \in \Omega(n\log q)$. Assume that there exist $k \le n$ linearly independent vectors in the row span of $\mathbf{A}$, each with norm bounded by γ. Then,*

$$\widetilde{H}_\infty\left(\boldsymbol{s} \mid (\mathbf{A}, \boldsymbol{s}\mathbf{A}+\boldsymbol{e} \bmod q\right) \ge k\cdot\log\frac{\sigma}{\gamma} - 1,$$

where $\boldsymbol{s} \leftarrow \mathbb{Z}_q^{1\times n}$ and $\boldsymbol{e} \leftarrow \mathcal{D}_{\mathbb{Z}^m,\sigma}$. ($\widetilde{H}_\infty$ denotes average-conditional min-entropy; see Definition 3.2.)

3.6 Other Preliminaries

Definition 3.2 (Average Conditional Min-Entropy)**.** *Let X be a random-variable supported on a finite set $\mathcal{X}$ and Z be a possibly correlated random-variable supported on a finite set $\mathcal{Z}$. The average conditional min-entropy:*

$$\widetilde{H}_\infty(X|Z) = -\log\left(\mathop{\mathbb{E}}_{z}\left[\max_{x\in\mathcal{X}} \Pr\left[X = x \mid Z = z\right]\right]\right)$$

Definition 3.3 ((k,ϵ)-average case strong seeded extractor)**.** *A function* $\mathsf{Ext}\colon \{0,1\}^{\ell_{\mathsf{Ext}}} \times \mathcal{X} \to \{0,1\}^{\ell}$ *is called a seeded strong average-case extractor, if it holds that for all random variables X and Z defined on some domains with a finite support, if $\widetilde{H}_\infty(X|Z) \geq k$ then it holds that:*

$$(s, \mathsf{Ext}(s, X), Z) \approx_\epsilon (s, U, Z)$$

where $s \leftarrow \{0,1\}^{\ell_{\mathsf{Ext}}}$ and $U \leftarrow \{0,1\}^{\ell}$.

There exists explicit polynomial-time constructions of seeded strong average-case $(\ell+O(\log(1/\epsilon)),\epsilon)$ extractors [DRS04, DORS08].

Lemma 3.6 (Error vectors are linearly independent)**.** *Let $k, m \in \mathbb{N}$ such that $k < m/2$. Let $\boldsymbol{e}_i \leftarrow \mathcal{D}_{\mathbb{Z}^m,\sigma}$ for $i \in [k]$, where $\sigma > m$. Except with* $\mathsf{negl}(m)$ *probability, the vectors $\{\boldsymbol{e}_i\}_{i\in[k]}$ are linearly independent.*

Proof. First observe that the column rank of the matrix $\mathbf{E} = [\boldsymbol{e}_1^\top | \ldots | \boldsymbol{e}_k^\top]$ is at least as much as the column rank of the matrix $\mathbf{E}$ mod 2 (over the field $\mathbb{Z}_2$). Due to the smoothing lemma [MR04], it is known that the statistical distance between $\boldsymbol{e}$ mod 2 and $\mathbb{Z}_2^m$ is at most $2^{-\Omega(m)}$ as $\sigma > m$. Finally, the claim holds since for a matrix $\mathbf{A} \leftarrow \mathbb{Z}_2^{k\times m}$ sampled uniformly at random

$$\Pr[\mathsf{rank}(\mathbf{A}) = k] > 1 - O(k \cdot 2^{k-m}).$$

□

4 Construction of 2rNISC

In this section, we give a construction of 2rNISC for the functionality:

$$\mathcal{U}_{\mathsf{fOT}} = \{\mathcal{U}_{\mathsf{fOT},\lambda}\}_{\lambda\in\mathbb{N}}$$

This functionality takes three inputs. The public input consists of two polynomial sized (in λ) functions $g_1\colon \{0,1\}^{n_1} \to \{0,1\}^\lambda \times \{0,1\}^\lambda$ and $g_2\colon \{0,1\}^{n_2} \to \{0,1\}$. (We assume that functions are given in the form of Boolean circuits). The functionality is evaluated as in the specifications described in Figure 1.

We recall that a 2rNISC is a mrNISC where the functionality to be evaluated is restricted to 2 parties. A 2rNISC allows for an arbitrary number of parties to commit or encode their inputs. The notion of mrNISC was recalled in the overview (Section 2.1). A formal definition can be found in Appendix B.

The main result of this section is a semi-malicious 2rNISC scheme for $\mathcal{U}_{\mathsf{fOT},\lambda}$ assuming LWE and a PRF in NC^1.

Functionality $\mathcal{U}_{\mathsf{fOT},\lambda}$

Public Input: Polynomial-sized functions $g_1\colon \{0,1\}^{n_1} \to \{0,1\}^\lambda \times \{0,1\}^\lambda$ and $g_1\colon \{0,1\}^{n_2} \to \{0,1\}$.

Input of the First Party: $x_1 \in \{0,1\}^{n_1}$.

Input of the Second Party: $x_2 \in \{0,1\}^{n_2}$.

Output to both Parties: Compute $(y_0, y_1) = g_1(x_1)$ where $y_0, y_1 \in \{0,1\}^\lambda$. Output $(g_2(x_2), y_{g_2(x_2)})$.

Figure 1: The functionality $\mathcal{U}_{\mathsf{fOT},\lambda,d}$

Theorem 4.1. *Assume LWE with polynomial modulus and a PRF in* NC^1*. Then, there exists a semi-malicious* $\mathsf{2rNISC}$ *for* $\mathcal{U}_{\mathsf{fOT}}$*.*

The construction that gives Theorem 4.1 is obtained in two modular steps. In the first step (see Section 4.1 and Theorem 4.2), we construct a $\mathsf{2rNISC}$ for a subset of all functions in the functionality $\mathcal{U}_{\mathsf{fOT},\lambda}$. Specifically, we restrict the circuit depth of g_2 to be an a priori fixed $d = d(\lambda)$ and obtain a protocol based solely on LWE. In the next step (see Section 4.2 and Theorem 4.3), using standard bootstrapping techniques using randomized encodings, we obtain our final $\mathsf{2rNISC}$ without any restriction on d. This step relies, in addition to LWE, on a PRF in NC^1.[8]

4.1 $\mathsf{2rNISC}$ for Depth-Bounded Functions

In this section, we give a construction of a semi-malicious $\mathsf{2rNISC}$ for the restricted functionality, where g_2 has a priori bounded depth $d = d(\lambda)$. We denote this functionality by $\{\mathcal{U}_{\mathsf{fOT},\lambda,d}\}_{\lambda,d\in\mathbb{N}}$.

Theorem 4.2. *Assuming LWE with polynomial modulus, there exists a semi-malicious* $\mathsf{2rNISC}$ *for* $\mathcal{U}_{\mathsf{fOT},\lambda,d}$ *for all (a priori) bounded* $d \in O(\log\lambda)$*. Further, assuming LWE assumption holds with modulus-to-noise ratio* 2^{N^ϵ} *for any constant* ϵ*, where* N *is the dimension, the same protocol is a semi-malicious* $\mathsf{2rNISC}$ *protocol for* $\mathcal{U}_{\mathsf{fOT},\lambda,d}$ *for any (a priori) bounded polynomial* $d(\lambda)$*.*

Before presenting the protocol, we list various parameters used in the scheme. We will explain how to set these parameters to achieve correctness and security in Section C.1.

Parameters.

- λ is the security parameter,
- n_i is the length of the input of party i,
- d is the depth parameter,
- N_1 is a lattice dimension involved,

[8]The common definition of a PRF in NC^1 is a PRF whose circuit representation is in NC^1 when viewed as a function of both the input and the seed. We actually need a slightly weaker condition, namely, that the circuit computing $F_x(\cdot) = \mathsf{PRF.Eval}(\cdot, x)$ with the hardwired input x, as a function of the PRF key is in NC^1.

- k is the number of secrets used to generate the commitment key,
- $N := N_1 + k$,
- q is a modulus,
- $M \in \Omega(N \cdot \log q)$ is a dimension involved,
- σ, σ' are discrete Gaussian parameters,
- ρ is a parameter for trapdoor sampling,
- ℓ_{Ext} is the seed length of an average-case strong-seeded extractor (Definition 3.3).

The protocol. We now describe the protocol which consists of three phases. The first phase is a commitment phase where any party can publish a commitment to its input. The second phase is when two parties decide to execute the functionality $\mathcal{U}_{\mathsf{fOT},\lambda,d}$ with their respective commitments from the first phase. In this phase, one message is published from each of these parties. In the third and last phase, each party locally computes their output, given the public transcript. No communication is involved in this phase.

We present the protocol from the point of view of a given party which we call P. This party first commits to its input on a public board. Later, P can engage in a computation phase with some other party P', by each broadcasting just one message. For this phase, we distinguish between two cases: whether P is the "first" or "second" party among P, P', where the ordering is given by the functionality. Lastly, each party can recover the output of the computation just from the public messages.

Commit on input $(1^\lambda, x)$**:** On input $x \in \{0,1\}^*$ perform the following steps:

- Sample a matrix $\mathbf{B} \leftarrow \mathbb{Z}_q^{N_1 \times M}$ uniformly at random.
- Sample secrets $\boldsymbol{t}_l \leftarrow \mathbb{Z}_q^{1 \times N_1}$ for $l \in [k]$.
- For $l \in [k]$, compute $\boldsymbol{b}_l = \boldsymbol{t}_l \cdot \mathbf{B} + \boldsymbol{e}_l$ where $\boldsymbol{e}_l$ is sampled from $\mathcal{D}_\sigma^M$.
- Set $\mathsf{flag} = 0$ if $\{\boldsymbol{e}_l\}_{l \in [k]}$ are not linearly independent. Otherwise set $\mathsf{flag} = 1$. Observe that due to Lemma 3.6, with overwhelming probability $\mathsf{flag} = 1$.
- Denote $\mathbf{A} = [\mathbf{B}^\top | \boldsymbol{b}_1^\top | \ldots | \boldsymbol{b}_k^\top]^\top \in \mathbb{Z}_q^{N \times M}$.
- Compute commitments of input x. Parse $x = (x_1, \ldots, x_n)$, where $n = |x|$. Compute matrices $\mathbf{C}_\ell = \mathbf{A} \cdot \mathbf{R}_\ell + x_\ell \mathbf{G}$ for $\ell \in [n]$. Here $\mathbf{R}_\ell \leftarrow \{-1, +1\}^{M \times (N \lceil \log q \rceil)}$ is chosen uniformly at random and $\mathbf{G} \in \mathbb{Z}_q^{N \times (N \lceil \log q \rceil)}$ is the gadget matrix.
- Output $\widehat{x} = (\mathsf{flag}, \mathbf{A}, \{\mathbf{C}_\ell\}_{\ell \in [n]})$ as a public string and remember $s = (\{\mathbf{R}_\ell\}_{\ell \in [n]}, x)$ as a private string.

Encode: There are two cases, depending on the "order" of the parties involved, denoted P and P'. In both cases, the view of party P (or its query) consists of $\widehat{x}, \widehat{x}', s$ and the view of P' consists of $\widehat{x}, \widehat{x}', s'$. The descriptions of g_1, g_2 are public. In both cases, party P first parses the public message of P' as follows:

- Parse $\widehat{x}' = (\mathsf{flag}', \mathbf{A}', \{\mathbf{C}'_\ell\}_{\ell\in[n']})$, where n' is the input length of party P'. If $\mathsf{flag}' = 0$, output $\perp$. Otherwise, proceed.

Party P proceeds as follows, depending on whether it is the "first" party or the "second".

Case 1: Party P is the "first" party.

- Compute $(y_0, y_1) = g_1(x)$.
- Compute $\widetilde{\mathbf{C}}'_{g_2} = \mathsf{GSW.Eval}(g_2, \{\mathbf{C}'_\ell\}_{\ell\in[n_j]})$.
- Sample two secrets $\boldsymbol{u}_0, \boldsymbol{u}_1 \leftarrow \mathbb{Z}_q^{1\times N}$.
- Compute $\boldsymbol{w}_b = \boldsymbol{u}_b \cdot [\mathbf{A}'|\widetilde{\mathbf{C}}'_{g_2} - (1-b)\cdot\mathbf{G}] + \widetilde{\boldsymbol{e}} \bmod q$ for $b\in\{0,1\}$. Here $\widetilde{\boldsymbol{e}}$ is sampled from $\mathcal{D}_{\sigma'}^{1\times(M+N\lceil\log q\rceil)}$.
- Let $\mathsf{Ext}\colon \{0,1\}^{\ell_{\mathsf{Ext}}}\times\{0,1\}^{N\log q}\to\{0,1\}^\lambda$ be a $(\lambda, 2^{-\lambda})$-strong seeded extractor. Sample a seed sd of the extractor. Output $\alpha = (\mathsf{sd}, \boldsymbol{w}_0, \boldsymbol{w}_1, v_0 = \mathsf{Ext}(\mathsf{sd}, \boldsymbol{u}_0)\oplus y_0, v_1 = \mathsf{Ext}(\mathsf{sd}, \boldsymbol{u}_1)\oplus y_1)$.

Case 2: Party P is the "second" party.

- Compute $\widetilde{\mathbf{C}}_{g_2} = \mathsf{GSW.Eval}(g_2, \{\mathbf{C}_\ell\}_{\ell\in[n]})$.
- Compute $\mathsf{GSW.RandEval}(\mathbf{A}, \{\mathbf{R}_\ell, x_\ell\}_{\ell\in[n]}) \to \widetilde{\mathbf{R}}_{g_2}$ such that $\widetilde{\mathbf{C}}_{g_2} = \mathbf{A}\cdot\widetilde{\mathbf{R}}_{g_2} + g_2(x)\cdot\mathbf{G}$.
- Compute a matrix $\mathbf{X}_{g_2}$ as:

$$\mathbf{X}_{g_2} = \begin{cases} \mathsf{SampleRight}(\mathbf{A}, -\mathbf{G}, \widetilde{\mathbf{R}}_{g_2}, \mathbf{T}_{\mathbf{G}}, \rho) \text{ when } g_2(x) = 0 \\ \mathsf{SampleRight}(\mathbf{A}, \mathbf{G}, \widetilde{\mathbf{R}}_{g_2}, \mathbf{T}_{\mathbf{G}}, \rho) \text{ when } g_2(x) = 1 \end{cases}$$

  Observe that $\mathbf{X}_{g_2}$ is a trapdoor of $[\mathbf{A} \mid \widetilde{\mathbf{C}}_{g_2} - (1-g_2(x))\mathbf{G}]$.
- Output $\alpha = (g_2(x), \mathbf{X}_{g_2})$.

Eval on input $(z = (g_1, g_2), \widehat{x}, \widehat{x}', \alpha, \alpha')$**:** Let P be the first party and P' be the second party.

- Parse $\widehat{x} = (\mathsf{flag}, \mathbf{A}, \{\mathbf{C}_\ell\}_{\ell\in[n]})$ and $\widehat{x}' = (\mathsf{flag}', \mathbf{A}', \{\mathbf{C}'_\ell\}_{\ell\in[n']})$. If $\alpha = \perp$ or $\alpha' = \perp$, then output $\perp$. Otherwise,
- Parse $\alpha = (\mathsf{sd}, \boldsymbol{w}_0, \boldsymbol{w}_1, v_0, v_1)$ and $\alpha' = (\alpha'_1, \mathbf{X})$ where α'_1 is a bit.
- Compute $\boldsymbol{u} = \mathsf{RecoverSecret}([\mathbf{A}'|\widetilde{\mathbf{C}}'_{g_2} - (1-\alpha'_1)\mathbf{G})],\ \mathbf{X}, \boldsymbol{w}_{\alpha'_1})$, where recall that $\widetilde{\mathbf{C}}'_{g_2} = \mathsf{GSW.Eval}(g_2, \{\mathbf{C}'_\ell\}_{\ell\in[n]})$.
- Compute $\mathsf{out}_2 = \mathsf{Ext}(\mathsf{sd}, \boldsymbol{u}) \oplus v_{\alpha'_1}$. Set $\mathsf{out}_1 = \alpha'_1$
- Output $\mathsf{out} = (\mathsf{out}_1, \mathsf{out}_2)$.

In Section C.1, we derive a concrete setting of parameters with which we can instantiate the scheme. Then, in Sections 4.1.1 and 4.1.2 we prove correctness and security of the scheme, respectively.

4.1.1 Correctness

Let P and P' be two parties who follow the protocol with respective inputs $x \in \{0,1\}^n$ and $x' \in \{0,1\}^{n'}$. Let P and P' participate in an execution to compute the functionality with the public input $z = (g_1, g_2)$ where $g_1 \colon \{0,1\}^n \to \{0,1\}^\lambda \times \{0,1\}^\lambda$, $g_2 \colon \{0,1\}^{n'} \to \{0,1\}$, and assume that P is the "first" party and P' is the "second". We claim that with overwhelming probability over the coins of Com and Encode, both parties learn the right output $(g_2(x'), g_1(x)_{g_2(x')})$. First, note that if the parties are honest then $\mathsf{flag} \neq 0$ and $\mathsf{flag}' \neq 0$ with overwhelming probability. In this case, during the encode phase P outputs $(\mathsf{sd}, \boldsymbol{w}_0, \boldsymbol{w}_1, v_0, v_1)$. Here for $b \in \{0,1\}$,

- $\boldsymbol{w}_b = \boldsymbol{u}_b \cdot [\mathbf{A}' | \widetilde{\mathbf{C}}'_{g_2} - (1-b) \cdot \mathbf{G}] + \widetilde{e}$, where the distributions of various elements is described in the algorithm.
- $v_b = \mathsf{Ext}(\mathsf{sd}, \boldsymbol{u}_b) \oplus y_b$ where $(y_0, y_1) = g_1(x)$.

On the other hand, P' outputs a trapdoor $\mathbf{X}$ of $[\mathbf{A}' | \widetilde{\mathbf{C}}'_{g_2} - (1 - g_2(x'))\mathbf{G}]$ along with the output $g_2(x')$. Thus, the Eval procedure lets one recover $\boldsymbol{u}_{g_2(x')}$ by relying on $\boldsymbol{w}_{g_2(x')}$ and the trapdoor $\mathbf{X}$ (using Lemma 3.4). Finally, $y_{g_2(x')}$ can be recovered by computing $\mathsf{Ext}(\mathsf{sd}, \boldsymbol{u}_{g_2(x')}) \oplus v_{g_2(x')} = y_{g_2(x')}$.

4.1.2 Proof of Security

We formally prove the security of the construction in Appendix C.2.

4.2 Bootstrapping 2rNISC for all depths

In this section, we use a PRF in NC^1 to bootstrap a 2rNISC protocol for the functionality $\mathcal{U}_{\mathsf{fOT},\lambda,c\log\lambda}$ for some fixed large enough constant c to a 2rNISC for $\mathcal{U}_{\mathsf{fOT},\lambda}$, as required in Theorem 4.1. Namely, the theorem we prove is:

Theorem 4.3. *Assuming a* 2rNISC *protocol for the functionality* $\mathcal{U}_{\mathsf{fOT},\lambda,c\log\lambda}$ *for a large enough constant* $c > 0$*, a* PRF *in* NC^1*, and a collision resistant hash function, there exist a* 2rNISC *for the functionality* $\mathcal{U}_{\mathsf{fOT},\lambda}$.

By combining Theorems 4.2 and 4.2, and using the fact that LWE (with polynomial modulus) imply collision-resistant hash functions [Ajt96], imply Theorem 4.1.

We prove Theorem 4.3 in Appendix C.3. An overview of the construction is provided at the end of Section 2.2.

5 Construction of mrNISC Schemes

Let us now show our construction of mrNISC schemes. We recall the mrNISC notion from the overview (Section 2.1) and the definition of Functional OT ($\mathcal{U}_{\text{fOT}}$, Fig. 1).

We have the following theorem.

Theorem 5.1. *Assuming the existence of a semi-malicious 2rNISC for Functional OT there exists an mrNISC scheme for any polynomial-time functionality.*

Our construction of mrNISC for a polynomial-time functionality $\mathcal{U}$ uses the following building blocks:

- A 2rNISC $\mathsf{2rNISC} = (\mathsf{Com}', \mathsf{Encode}', \mathsf{Eval}')$ for Functional OT (fOT).

- A semi-malicious output-delayed simulatable L-round MPC protocol $\Pi = (\mathsf{Next}, \mathsf{Output})$ for f. Output-delayed simulatability was introduced in [BL20] and ensures that the transcript excluding the last messages can be simulated for all-but-one honest parties before knowing the output. Formal definitions and constructions from standard semi-malicious MPC are recalled in Appendix A.6. We require the number of rounds L to be constant. 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. Without loss of generality, we will assume that in each round ℓ of Π, each party P_i broadcasts a single message that depends on its input x_i, randomness r_i and on the messages $\mathsf{Msg}^{<\ell} = \{\mathsf{msg}_j^{\ell'}\}_{j\in[n],\ell'<\ell}$ that it received from all parties in all previous rounds such that $\mathsf{msg}_j^{\ell} = \mathsf{Next}_j(z, x_j, r_j, \mathsf{Msg}^{<\ell})$, where z is the public input. In other words, Next_j is the next message function that computes the message broadcast by P_j. In the last round L of Π anybody computes the public output $y = \mathsf{Output}(z, \mathsf{Msg}) = \mathcal{U}(z, \{x_i\})$, from the messages $\mathsf{Msg} = \{\mathsf{msg}_j^{\ell}\}_{j\in[n],\ell\in[L]}$. We denote by ν_r the number of bits of r_i and by ν_m the number of bits of messages msg_i^{ℓ} (without loss of generality, we suppose that these numbers are independent of i and ℓ, but they may depend on z and the security parameter). Next_j and Output implicitly take as input a unary representation of the security parameter 1^λ.

- A garbled circuit scheme $\mathsf{GC} = (\mathsf{GC.Gen}, \mathsf{GC.Garble}, \mathsf{GC.Eval}, \mathsf{GC.Sim})$ for P. The keys (aka labels) of the garbled circuits have κ bits.

- A pseudorandom function PRF. Each party will generate $L+1$ PRF keys $\mathrm{fk}_i^0, \dots, \mathrm{fk}_i^L$. The key fk_i^0 is used to generate the randomness for the internal MPC (via $\mathsf{PRF}(\mathrm{fk}_i, 0\|z\|\dots)$), while the keys $\mathrm{fk}_i^1, \dots, \mathrm{fk}_i^L$ are used to encrypt (via a one-time pad) the labels of the used garbled circuits for rounds $1, \dots, L$ respectively (via $\mathsf{PRF}(\mathrm{fk}_i, 1\|z\|\dots)$).

Our mrNISC scheme is constructed as follows:

- Input: $(\widehat{x}_i, s_i) \leftarrow \mathsf{Com}(1^\lambda, x_i)$ samples $L+1$ PRF key $\mathrm{fk}_i^0, \dots, \mathrm{fk}_i^L \leftarrow_{\mathrm{R}} \{0,1\}^\lambda$.For each $\ell \in L$, Com also generates 2rNISC input encodings and associated secret state for $x_i\|\mathrm{fk}_i^0\|\mathrm{fk}_i^\ell$:

$$(\widehat{x}_i^\ell, s_i^\ell) \leftarrow_{\mathrm{R}} \mathsf{Com}'(x_i\|\mathrm{fk}_i^0\|\mathrm{fk}_i^\ell) \ . \tag{1}$$

  In other words, party P_i make L 2rNISC input encodings. When we need to differentiate these encodings, we say that the ℓ-th such input encoding is made by the virtual party P_i^ℓ. Finally, Com sets $\widehat{x}_i \coloneqq \{\widehat{x}_i^\ell\}_{\ell\in[L]}$ and $s_i \coloneqq (x_i, \{\mathrm{fk}_i^\ell\}_{\ell\in[0,L]}, \{s_i^\ell\}_{\ell\in[L]})$.

- Computation of $\mathcal{U}(z,\star)$: $\alpha_i \leftarrow \mathsf{Encode}(z, \{\widehat{x}_j\}_{j\in[n]}, s_i)$ proceeds as follows:[9]

  - For $\ell \in [L]$, generate input labels that will be used to garble the evaluation circuit F_i^ℓ defined in Fig. 2:

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

    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.

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

- For $\ell \in [L]$, $j \in [n]$, $k \in [\nu_m]$, $b \in \{0,1\}$, compute the following ciphertexts

$$\mathsf{ct}^{\ell}_{i,j,k,b} \leftarrow_{\mathrm{R}} \mathbf{msgKey}^{\ell+1}_{i,j}[k,b] \oplus \mathsf{PRF}(\mathrm{fk}^{\ell}_i, 1\|z\|j\|k\|b\|[\kappa]) \ . \tag{2}$$

  If $\ell = L$, these ciphertexts are set to be empty strings.
- For $\ell \in [L]$, garble the evaluation circuit F^{ℓ}_i:

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

- Set $\alpha_i \coloneqq (\{\widehat{\mathsf{F}}^{\ell}_i\}_{\ell\in[L]},\ \{\mathsf{ct}^{\ell+1}_{i,j,k,b}\}_{j,k,b})$.

• <u>Output:</u> $y = \mathsf{Eval}(z, \{\widehat{x}_i\}_{i\in[n]}, \{\alpha_i\}_{i\in[n]})$ proceeds as follows in L iterations, for $\ell = 1,\dots,L$:
  - Evaluate the garbled circuits for round ℓ, for $i \in [n]$:

$$\begin{aligned}\Big(\mathbf{stateKey}'^{\ell+1}_i,\ \mathsf{msg}^{\ell}_i,\ \{\alpha^{\ell}_{i,j,k,1}\}_{j,k},\ \{\alpha^{\ell}_{j,i,k,2}\}_{j,k}\Big)&\\ \coloneqq \mathsf{GC.Eval}(\widehat{\mathsf{F}}_i,\ (\mathbf{stateKey}'^{\ell}_i,\ \{\mathbf{msgKey}^{\ell}_{i,j}[\mathsf{msg}^{\ell-1}_j]\}_{j\in[n]}))& \ .\end{aligned}$$

    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 $g^{\ell}_{1,j,k}, g^{\ell}_{2,i,k}$ as in Fig. 2 and compute:

$$\begin{aligned}(\_, K^{\ell}_{i,j,k}) &\coloneqq \mathsf{Eval}'((g^{\ell}_{1,j,k}, g^{\ell}_{2,j,k}),\ (\widehat{x}'_i, \widehat{x}'_j),\ (\alpha^{\ell}_{i,j,k,1}, \alpha^{\ell}_{i,j,k,2}))\ ,\\ \mathbf{msgKey}^{\ell+1}_{i,j}[\mathsf{msg}^{\ell}_j] &\coloneqq \{\mathsf{ct}^{\ell}_{i,j,k,}\ \oplus\ K^{\ell}_{i,j,k}\}_{k\in[\nu_m]}\ ,\end{aligned}$$

    where _ just indicates that we ignore the output.

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

The correctness of the mrNISC scheme is follows from the perfect correctness properties of the inner MPC protocol, of the garbled circuit scheme, and the following fact (if everything is generated as specified in the description above):

$$\begin{aligned}&\mathsf{Eval}'((g^{\ell}_{1,j,k}, g^{\ell}_{2,j,k}), (\widehat{x}'_i, \widehat{x}'_j), (\alpha^{\ell}_{i,j,k,1}, \alpha^{\ell}_{i,j,k,2})) = (\beta, y_\beta)\\ &\qquad\text{where } \beta = g^{\ell}_{2,j,k}(x_j\|\mathrm{fk}^0_j\|\mathrm{fk}^{\ell}_j) = \text{the } k\text{-th bit of } \mathsf{msg}^{\ell}_j\\ &\qquad\text{and } (y_0, y_1) = g^{\ell}_{1,j,k}(x_i\|\mathrm{fk}^0_i\|\mathrm{fk}^{\ell}_i)\end{aligned}$$

thus:

$$K^{\ell}_{i,j,k} = y_b = \mathsf{PRF}(\mathrm{fk}_j, 1\|z\|j\|k\|\beta\|[\kappa]) \ .$$

The proof is similar to the security proof of the mrNISC in [BL20] and is formally presented in Appendix D.

Circuit F_i^ℓ

Hardwired Values: 1^λ, ℓ, i, z, $\{\widehat{x}_j = \{\widehat{x}_j^\ell\}_{\ell\in[L]}\}_{j\in[n]}$, $s_i = (x_i, \{\mathrm{fk}_i^\ell\}_{\ell\in[0,L]}, \{s_i^\ell\}_{\ell\in[L]})$, $\mathbf{stateKey}_i^{\ell+1}$, $\{\mathbf{msgKey}_{i,j}^{\ell+1}\}_{j\in[n]}$.

Inputs: $(\mathsf{Msg}^{<\ell-1}, \boldsymbol{\mathsf{msg}}^{\ell-1})$ 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 $\boldsymbol{\mathsf{msg}}^{\ell-1} := \{\mathsf{msg}_j^{\ell-1}\}_{j\in[n]}$ 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 $j \in [n]$, $k \in [\nu_m]$, and $b \in \{0,1\}$, define the functions $g_{1,j,k}^\ell$ and $g_{2,j}^\ell$ by:

$$g_{1,j,k}^\ell(x\|\mathrm{fk}^0\|\mathrm{fk}^\ell) := \{\mathsf{PRF}(\mathrm{fk}^\ell, 1\|z\|j\|k\|b\|[\kappa])\}_{b\in\{0,1\}} \ ,$$
$$g_{2,j}^\ell(x\|\mathrm{fk}^0\|\mathrm{fk}^\ell) := \mathsf{Next}_j(z, x, \mathsf{PRF}(\mathrm{fk}^0, 0\|z\|[\nu_r]), \mathsf{Msg}^{<\ell-1}, \boldsymbol{\mathsf{msg}}^{\ell-1}) \ ,$$

   and define the functions $g_{2,j,k}^\ell$ to output the k'th bit of $g_{2,j}^\ell$, for $k \in [\nu_m]$.

2. Compute the ℓ-th round message $\mathsf{msg}_i^\ell = \mathsf{msg}_{i,1}^\ell\|\cdots\|\mathsf{msg}_{i,\nu_m}^\ell := g_{2,i}^\ell(x_i\|\mathrm{fk}_i)$ of P_i in the inner protocol Π, and associated 2rNISC encodings, for $j \in [n]$, $k \in [\nu_m]$

$$\alpha_{i,j,k,1}^\ell \leftarrow_{\mathrm{R}} \mathsf{Encode}'((g_{1,j,k}^\ell, g_{2,j,k}^\ell), (\widehat{x}_i^\ell, \widehat{x}_j^\ell), s_i^\ell) \tag{3}$$
$$\alpha_{j,i,k,2}^\ell \leftarrow_{\mathrm{R}} \mathsf{Encode}'((g_{1,i,k}^\ell, g_{2,i,k}^\ell), (\widehat{x}_j^\ell, \widehat{x}_i^\ell), s_i^\ell) \tag{4}$$

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

Output: $(\mathbf{stateKey}_i^{\ell+1}[\mathsf{Msg}^{<\ell-1}\|\boldsymbol{\mathsf{msg}}^{\ell-1}]$, msg_i^ℓ, $\{\alpha_{i,j,k,1}^\ell\}_{j,k}$, $\{\alpha_{j,i,k,2}^\ell\}_{j,k})$.

Figure 2: Circuit F_i^ℓ for the construction of mrNISC in Section 5

Acknowledgments

Aayush Jain was supported by a Google PhD fellowship in the area of security and privacy (2018) and in part from DARPA SAFEWARE and SIEVE awards, NTT Research, NSF Frontier Award 1413955, and NSF grant 1619348, BSF grant 2012378, a Xerox Faculty Research Award, a Google Faculty Research Award, an equipment grant from Intel, and an Okawa Foundation Research Grant. This material is based upon work supported by the Defense Advanced Research Projects Agency through Award HR00112020024 and the ARL under Contract W911NF-15-C- 0205.

Ilan Komargodski is supported in part by an Alon Young Faculty Fellowship and by an ISF grant (No. 1774/20).

Huijia Lin was supported by NSF grants CNS-1528178, CNS-1929901, CNS-1936825 (CAREER), CNS-2026774, a Hellman Fellowship, a JP Morgan AI Research Award, a Simons Collaboration grant on the Theory of Algorithmic Fairness, 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.

6 References

[ABB10] Shweta Agrawal, Dan Boneh, and Xavier Boyen. Efficient lattice (H)IBE in the standard model. In Henri Gilbert, editor, *EUROCRYPT 2010*, volume 6110 of *LNCS*, pages 553–572. Springer, Heidelberg, May / June 2010. 8, 14

[ABJ$^+$19] Prabhanjan Ananth, Saikrishna Badrinarayanan, Aayush Jain, Nathan Manohar, and Amit Sahai. From FE combiners to secure MPC and back. In Dennis Hofheinz and Alon Rosen, editors, *TCC 2019, Part I*, volume 11891 of *LNCS*, pages 199–228. Springer, Heidelberg, December 2019. 2

[ACGJ18] Prabhanjan Ananth, Arka Rai Choudhuri, Aarushi Goel, and Abhishek Jain. Round-optimal secure multiparty computation with honest majority. In Hovav Shacham and Alexandra Boldyreva, editors, *CRYPTO 2018, Part II*, volume 10992 of *LNCS*, pages 395–424. Springer, Heidelberg, August 2018. 2

[AIK06] Benny Applebaum, Yuval Ishai, and Eyal Kushilevitz. Computationally private randomizing polynomials and their applications. *computational complexity*, 15(2):115–162, Jun 2006. 30, 31

[AJJ20] Prabhanjan Ananth, Abhishek Jain, and Zhengzhong Jin. Multiparty homomorphic encryption (or: On removing setup in multi-key FHE). *IACR Cryptol. ePrint Arch.*, 2020:169, 2020. 2, 4

[AJJM20] Prabhanjan Ananth, Abhishek Jain, Zhengzhong Jin, and Giulio Malavolta. Multi-key fully-homomorphic encryption in the plain model. In *TCC*, pages 28–57, 2020. 3

[AJL$^+$12] Gilad Asharov, Abhishek Jain, Adriana López-Alt, Eran Tromer, Vinod Vaikuntanathan, and Daniel Wichs. Multiparty computation with low communication, computation and interaction via threshold FHE. In David Pointcheval and Thomas Johansson, editors, *EUROCRYPT 2012*, volume 7237 of *LNCS*, pages 483–501. Springer, Heidelberg, April 2012. 1, 2, 4, 5

[Ajt96] Miklós Ajtai. Generating hard instances of lattice problems (extended abstract). In *28th ACM STOC*, pages 99–108. ACM Press, May 1996. 19

[Ajt99] Miklós Ajtai. Generating hard instances of the short basis problem. In Jirí Wiedermann, Peter van Emde Boas, and Mogens Nielsen, editors, *ICALP 99*, volume 1644 of *LNCS*, pages 1–9. Springer, Heidelberg, July 1999. 13

[AMPR14] Arash Afshar, Payman Mohassel, Benny Pinkas, and Ben Riva. Non-interactive secure computation based on cut-and-choose. In Phong Q. Nguyen and Elisabeth Oswald, editors, *EUROCRYPT 2014*, volume 8441 of *LNCS*, pages 387–404. Springer, Heidelberg, May 2014. 4

[AP09] Joël Alwen and Chris Peikert. Generating shorter bases for hard random lattices. In Susanne Albers and Jean-Yves Marion, editors, *26th International Symposium on Theoretical Aspects of Computer Science, STACS 2009, February 26-28, 2009, Freiburg, Germany, Proceedings*, volume 3 of *LIPIcs*, pages 75–86. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, Germany, 2009. 13

[BD18] Zvika Brakerski and Nico Döttling. Two-message statistically sender-private OT from LWE. In Amos Beimel and Stefan Dziembowski, editors, *TCC 2018, Part II*, volume 11240 of *LNCS*, pages 370–390. Springer, Heidelberg, November 2018. 3, 7, 9, 14

[BGI$^+$14] Amos Beimel, Ariel Gabizon, Yuval Ishai, Eyal Kushilevitz, Sigurd Meldgaard, and Anat Paskin-Cherniavsky. Non-interactive secure multiparty computation. In Juan A. Garay and Rosario Gennaro, editors, *CRYPTO 2014, Part II*, volume 8617 of *LNCS*, pages 387–404. Springer, Heidelberg, August 2014. 4

[BGI$^+$17] Saikrishna Badrinarayanan, Sanjam Garg, Yuval Ishai, Amit Sahai, and Akshay Wadia. Two-message witness indistinguishability and secure computation in the plain model from new assumptions. In Tsuyoshi Takagi and Thomas Peyrin, editors, *ASIACRYPT 2017, Part III*, volume 10626 of *LNCS*, pages 275–303. Springer, Heidelberg, December 2017. 4

[BGMM20] James Bartusek, Sanjam Garg, Daniel Masny, and Pratyay Mukherjee. Reusable two-round MPC from DDH. In *Theory of Cryptography - TCC*, pages 320–348, 2020. 2, 4

[BGW88] Michael Ben-Or, Shafi Goldwasser, and Avi Wigderson. Completeness theorems for non-cryptographic fault-tolerant distributed computation (extended abstract). In *20th ACM STOC*, pages 1–10. ACM Press, May 1988. 1

[BHP17] Zvika Brakerski, Shai Halevi, and Antigoni Polychroniadou. Four round secure computation without setup. In Yael Kalai and Leonid Reyzin, editors, *TCC 2017, Part I*, volume 10677 of *LNCS*, pages 645–677. Springer, Heidelberg, November 2017. 2

[BJMS18] Saikrishna Badrinarayanan, Aayush Jain, Nathan Manohar, and Amit Sahai. Secure MPC: laziness leads to GOD. *IACR Cryptol. ePrint Arch.*, 2018:580, 2018. 2

[BJMS20] Saikrishna Badrinarayanan, Aayush Jain, Nathan Manohar, and Amit Sahai. Secure MPC: laziness leads to GOD. In Shiho Moriai and Huaxiong Wang, editors, *Advances in Cryptology - ASIACRYPT 2020 - 26th International Conference on the Theory and Application of Cryptology and Information Security, Daejeon, South Korea, December 7-11, 2020, Proceedings, Part III*, Lecture Notes in Computer Science, 2020. 4

[BJOV18] Saikrishna Badrinarayanan, Abhishek Jain, Rafail Ostrovsky, and Ivan Visconti. Non-interactive secure computation from one-way functions. In Thomas Peyrin and Steven Galbraith, editors, *ASIACRYPT 2018, Part III*, volume 11274 of *LNCS*, pages 118–138. Springer, Heidelberg, December 2018. 4

[BL18] Fabrice Benhamouda and Huijia Lin. k-round multiparty computation from k-round oblivious transfer via garbled interactive circuits. In Jesper Buus Nielsen and Vincent Rijmen, editors, *EUROCRYPT 2018, Part II*, volume 10821 of *LNCS*, pages 500–532. Springer, Heidelberg, April / May 2018. 2, 4

[BL20] Fabrice Benhamouda and Huijia Lin. Mr NISC: multiparty reusable non-interactive secure computation. In *Theory of Cryptography - TCC*, pages 349–378, 2020. 1, 3, 4, 5, 9, 10, 20, 21, 32, 42

[BLMR13] Dan Boneh, Kevin Lewi, Hart William Montgomery, and Ananth Raghunathan. Key homomorphic PRFs and their applications. In Ran Canetti and Juan A. Garay, editors, *CRYPTO 2013, Part I*, volume 8042 of *LNCS*, pages 410–428. Springer, Heidelberg, August 2013. 2

[BLP$^+$13] Zvika Brakerski, Adeline Langlois, Chris Peikert, Oded Regev, and Damien Stehlé. Classical hardness of learning with errors. In Dan Boneh, Tim Roughgarden, and Joan Feigenbaum, editors, *45th ACM STOC*, pages 575–584. ACM Press, June 2013. 2

[BP14] Abhishek Banerjee and Chris Peikert. New and improved key-homomorphic pseudorandom functions. In *Advances in Cryptology - CRYPTO*, pages 353–370, 2014. 2

[BP16] Zvika Brakerski and Renen Perlman. Lattice-based fully dynamic multi-key FHE with short ciphertexts. In Matthew Robshaw and Jonathan Katz, editors, *CRYPTO 2016, Part I*, volume 9814 of *LNCS*, pages 190–213. Springer, Heidelberg, August 2016. 2

[BPR12] Abhishek Banerjee, Chris Peikert, and Alon Rosen. Pseudorandom functions and lattices. In *Advances in Cryptology–EUROCRYPT 2012*, pages 719–737. Springer, 2012. 2

[BV14] Zvika Brakerski and Vinod Vaikuntanathan. Lattice-based FHE as secure as PKE. In Moni Naor, editor, *ITCS 2014*, pages 1–12. ACM, January 2014. 13, 35

[CCD88] David Chaum, Claude Crépeau, and Ivan Damgård. Multiparty unconditionally secure protocols (extended abstract). In *20th ACM STOC*, pages 11–19. ACM Press, May 1988. 1

[CDI$^+$19] Melissa Chase, Yevgeniy Dodis, Yuval Ishai, Daniel Kraschewski, Tianren Liu, Rafail Ostrovsky, and Vinod Vaikuntanathan. Reusable non-interactive secure computation. In Alexandra Boldyreva and Daniele Micciancio, editors, *CRYPTO 2019, Part III*, volume 11694 of *LNCS*, pages 462–488. Springer, Heidelberg, August 2019. 4

[CGP15] Ran Canetti, Shafi Goldwasser, and Oxana Poburinnaya. Adaptively secure two-party computation from indistinguishability obfuscation. In Yevgeniy Dodis and Jesper Buus Nielsen, editors, *TCC 2015, Part II*, volume 9015 of *LNCS*, pages 557–585. Springer, Heidelberg, March 2015. 2

[CHKP10] David Cash, Dennis Hofheinz, Eike Kiltz, and Chris Peikert. Bonsai trees, or how to delegate a lattice basis. In Henri Gilbert, editor, *EUROCRYPT 2010*, volume 6110 of *LNCS*, pages 523–552. Springer, Heidelberg, May / June 2010. 8, 14

[CJS14] Ran Canetti, Abhishek Jain, and Alessandra Scafuro. Practical UC security with a global random oracle. In Gail-Joon Ahn, Moti Yung, and Ninghui Li, editors, *ACM CCS 2014*, pages 597–608. ACM Press, November 2014. 4

[CM15] Michael Clear and Ciaran McGoldrick. Multi-identity and multi-key leveled FHE from learning with errors. In Rosario Gennaro and Matthew J. B. Robshaw, editors, *CRYPTO 2015, Part II*, volume 9216 of *LNCS*, pages 630–656. Springer, Heidelberg, August 2015. 2, 4, 47

[DKR15] Dana Dachman-Soled, Jonathan Katz, and Vanishree Rao. Adaptively secure, universally composable, multiparty computation in constant rounds. In Yevgeniy Dodis and Jesper Buus Nielsen, editors, *TCC 2015, Part II*, volume 9015 of *LNCS*, pages 586–613. Springer, Heidelberg, March 2015. 2

[DORS08] Yevgeniy Dodis, Rafail Ostrovsky, Leonid Reyzin, and Adam D. Smith. Fuzzy extractors: How to generate strong keys from biometrics and other noisy data. *SIAM J. Comput.*, 38(1):97–139, 2008. 15

[DRS04] Yevgeniy Dodis, Leonid Reyzin, and Adam Smith. Fuzzy extractors: How to generate strong keys from biometrics and other noisy data. In Christian Cachin and Jan Camenisch, editors, *EUROCRYPT 2004*, volume 3027 of *LNCS*, pages 523–540. Springer, Heidelberg, May 2004. 15

[FKN94] Uriel Feige, Joe Kilian, and Moni Naor. A minimal model for secure computation (extended abstract). In *26th ACM STOC*, pages 554–563. ACM Press, May 1994. 4

[GGHR14] Sanjam Garg, Craig Gentry, Shai Halevi, and Mariana Raykova. Two-round secure MPC from indistinguishability obfuscation. In Yehuda Lindell, editor, *TCC 2014*, volume 8349 of *LNCS*, pages 74–94. Springer, Heidelberg, February 2014. 2, 4, 10

[GIKM98] Yael Gertner, Yuval Ishai, Eyal Kushilevitz, and Tal Malkin. Protecting data privacy in private information retrieval schemes. In *30th ACM STOC*, pages 151–160. ACM Press, May 1998. 4

[GIS18] Sanjam Garg, Yuval Ishai, and Akshayaram Srinivasan. Two-round MPC: Information-theoretic and black-box. In Amos Beimel and Stefan Dziembowski, editors, *TCC 2018, Part I*, volume 11239 of *LNCS*, pages 123–151. Springer, Heidelberg, November 2018. 2

[GLS15] S. Dov Gordon, Feng-Hao Liu, and Elaine Shi. Constant-round MPC with fairness and guarantee of output delivery. In Rosario Gennaro and Matthew J. B. Robshaw, editors, *CRYPTO 2015, Part II*, volume 9216 of *LNCS*, pages 63–82. Springer, Heidelberg, August 2015. 2, 4, 10

[GMS18] Sanjam Garg, Peihan Miao, and Akshayaram Srinivasan. Two-round multiparty secure computation minimizing public key operations. In Hovav Shacham and Alexandra Boldyreva, editors, *CRYPTO 2018, Part III*, volume 10993 of *LNCS*, pages 273–301. Springer, Heidelberg, August 2018. 2

[GMW87] Oded Goldreich, Silvio Micali, and Avi Wigderson. How to play any mental game or A completeness theorem for protocols with honest majority. In Alfred Aho, editor, *19th ACM STOC*, pages 218–229. ACM Press, May 1987. 1

[GP15] Sanjam Garg and Antigoni Polychroniadou. Two-round adaptively secure MPC from indistinguishability obfuscation. In Yevgeniy Dodis and Jesper Buus Nielsen, editors, *TCC 2015, Part II*, volume 9015 of *LNCS*, pages 614–637. Springer, Heidelberg, March 2015. 2

[GPV08] Craig Gentry, Chris Peikert, and Vinod Vaikuntanathan. Trapdoors for hard lattices and new cryptographic constructions. In Richard E. Ladner and Cynthia Dwork, editors, *40th ACM STOC*, pages 197–206. ACM Press, May 2008. 13

[GS17] Sanjam Garg and Akshayaram Srinivasan. Garbled protocols and two-round MPC from bilinear maps. In Chris Umans, editor, *58th FOCS*, pages 588–599. IEEE Computer Society Press, October 2017. 2

[GS18] Sanjam Garg and Akshayaram Srinivasan. Two-round multiparty secure computation from minimal assumptions. In Jesper Buus Nielsen and Vincent Rijmen, editors, *EUROCRYPT 2018, Part II*, volume 10821 of *LNCS*, pages 468–499. Springer, Heidelberg, April / May 2018. 2, 4

[GSW13] Craig Gentry, Amit Sahai, and Brent Waters. Homomorphic encryption from learning with errors: Conceptually-simpler, asymptotically-faster, attribute-based. In Ran Canetti and Juan A. Garay, editors, *CRYPTO 2013, Part I*, volume 8042 of *LNCS*, pages 75–92. Springer, Heidelberg, August 2013. 6, 12, 13, 35, 47

[GVW15] Sergey Gorbunov, Vinod Vaikuntanathan, and Daniel Wichs. Leveled fully homomorphic signatures from standard lattices. In Rocco A. Servedio and Ronitt Rubinfeld, editors, *47th ACM STOC*, pages 469–477. ACM Press, June 2015. 3, 6, 7, 8

[HIJ$^+$16] Shai Halevi, Yuval Ishai, Abhishek Jain, Eyal Kushilevitz, and Tal Rabin. Secure multiparty computation with general interaction patterns. In Madhu Sudan, editor, *ITCS 2016*, pages 157–168. ACM, January 2016. 4

[HIJ$^+$17] Shai Halevi, Yuval Ishai, Abhishek Jain, Ilan Komargodski, Amit Sahai, and Eylon Yogev. Non-interactive multiparty computation without correlated randomness. In Tsuyoshi Takagi and Thomas Peyrin, editors, *ASIACRYPT 2017, Part III*, volume 10626 of *LNCS*, pages 181–211. Springer, Heidelberg, December 2017. 4

[HLP11] Shai Halevi, Yehuda Lindell, and Benny Pinkas. Secure computation on the web: Computing without simultaneous interaction. In Phillip Rogaway, editor, *CRYPTO 2011*, volume 6841 of *LNCS*, pages 132–150. Springer, Heidelberg, August 2011. 4

[IK00] Yuval Ishai and Eyal Kushilevitz. Randomizing polynomials: A new representation with applications to round-efficient secure computation. In *41st FOCS*, pages 294–304. IEEE Computer Society Press, November 2000. 30

[IKO$^+$11] Yuval Ishai, Eyal Kushilevitz, Rafail Ostrovsky, Manoj Prabhakaran, and Amit Sahai. Efficient non-interactive secure computation. In Kenneth G. Paterson, editor, *EUROCRYPT 2011*, volume 6632 of *LNCS*, pages 406–425. Springer, Heidelberg, May 2011. 4

[LTV12] Adriana López-Alt, Eran Tromer, and Vinod Vaikuntanathan. On-the-fly multiparty computation on the cloud via multikey fully homomorphic encryption. In Howard J. Karloff and Toniann Pitassi, editors, *44th ACM STOC*, pages 1219–1234. ACM Press, May 2012. 3, 47

[MM11] Daniele Micciancio and Petros Mol. Pseudorandom knapsacks and the sample complexity of LWE search-to-decision reductions. In Phillip Rogaway, editor, *CRYPTO 2011*, volume 6841 of *LNCS*, pages 465–484. Springer, Heidelberg, August 2011. 2

[MP12] Daniele Micciancio and Chris Peikert. Trapdoors for lattices: Simpler, tighter, faster, smaller. In David Pointcheval and Thomas Johansson, editors, *EUROCRYPT 2012*, volume 7237 of *LNCS*, pages 700–718. Springer, Heidelberg, April 2012. 2, 8, 12, 13, 14

[MR04] Daniele Micciancio and Oded Regev. Worst-case to average-case reductions based on Gaussian measures. In *45th FOCS*, pages 372–381. IEEE Computer Society Press, October 2004. 11, 15

[MW16] Pratyay Mukherjee and Daniel Wichs. Two round multiparty computation via multi-key FHE. In Marc Fischlin and Jean-Sébastien Coron, editors, *EUROCRYPT 2016, Part II*, volume 9666 of *LNCS*, pages 735–763. Springer, Heidelberg, May 2016. 2, 4, 47

[Pei09] Chris Peikert. Public-key cryptosystems from the worst-case shortest vector problem: extended abstract. In Michael Mitzenmacher, editor, *Proceedings of the 41st Annual ACM Symposium on Theory of Computing, STOC 2009, Bethesda, MD, USA, May 31 - June 2, 2009*, pages 333–342. ACM, 2009. 2

[PS16] Chris Peikert and Sina Shiehian. Multi-key FHE from LWE, revisited. In Martin Hirt and Adam D. Smith, editors, *TCC 2016-B, Part II*, volume 9986 of *LNCS*, pages 217–238. Springer, Heidelberg, October / November 2016. 2

[PS19] Chris Peikert and Sina Shiehian. Noninteractive zero knowledge for NP from (plain) learning with errors. In Alexandra Boldyreva and Daniele Micciancio, editors, *CRYPTO 2019, Part I*, volume 11692 of *LNCS*, pages 89–114. Springer, Heidelberg, August 2019. 2

[QWW18] Willy Quach, Hoeteck Wee, and Daniel Wichs. Laconic function evaluation and applications. In Mikkel Thorup, editor, *59th FOCS*, pages 859–870. IEEE Computer Society Press, October 2018. 2

[Reg05] Oded Regev. On lattices, learning with errors, random linear codes, and cryptography. In *STOC*, pages 84–93, 2005. 2, 12

A Additional Preliminaries

A.1 Statistical and Computational Indistinguishability

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

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 $\mathsf{poly}(\lambda)$. 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 $\mathcal{D} = \{\mathcal{D}_\lambda\}_{\lambda\in\mathbb{N}}$ 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 $\lambda \in \mathbb{N}$,*

$$\left|\Pr\left[\mathcal{D}_\lambda(w_\lambda, X_{\lambda,w_\lambda}) = 1\right] - \Pr\left[\mathcal{D}_\lambda(w_\lambda, Y_{\lambda,w_\lambda}) = 1\right]\right| \leq \mathsf{negl}(\lambda) \ .$$

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:* $\mathsf{key} \leftarrow_R \mathsf{GC.Gen}(1^\lambda)$ *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:* $\widehat{C} \leftarrow_R \mathsf{GC.Garble}(\mathsf{key}, C)$ *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:* $(\mathsf{key}', \widetilde{C}) \leftarrow_R \mathsf{GC.Sim}(1^\lambda, y)$ *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:*

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

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* $\lambda \in \mathbb{N}$*:*

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

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* $x \in \{0,1\}^{\mathsf{poly}(\lambda)}$ *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* $\lambda \in \mathbb{N}$*:*

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

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

Remark A.1. For simplicity of notation, we often write $\mathsf{PRF}(\mathrm{fk}, z' \| \{z_1, \dots, z_\ell\} \| \mathbf{0}) = r$ for $z' \in \{0,1\}^{\mathsf{poly}(\lambda)}$ and $z_i \in \{0,1\}^{\mathsf{poly}(\lambda)}$ 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 Preliminary: Computational Randomized Encodings.

We recall the definition of computational randomized encodings from [IK00, AIK06].

Definition A.5 (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:* $\widehat{G} \coloneqq \mathsf{CRE.Enc}(1^\lambda, G)$ *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 $\widehat{G} \colon \{0,1\}^n \times \{0,1\}^m \to \{0,1\}^s$*.* $\mathsf{CRE.Enc}$ *is deterministic.*
- *Decoding:* $y \coloneqq \mathsf{CRE.Dec}(1^\lambda, G, \widehat{y})$ *on input the security parameter, a circuit $G \in \mathcal{G}$ ($G \colon \{0,1\}^n \to \{0,1\}^\ell$), and the output of $\widehat{y}$ of the randomized encoding $\widehat{G}$, outputs y the output of G.* $\mathsf{CRE.Dec}$ *is deterministic.*
- *Simulation:* $\widehat{G} \leftarrow \mathsf{CRE.Sim}(1^\lambda, G, y)$ *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 $\widehat{G}$.*

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 $v \in \{0,1\}^n$, for every bit string $r \in \{0,1\}^m$, if $\widehat{G} = \mathsf{CRE.Enc}(1^\lambda, G)$, then $\mathsf{CRE.Dec}(1^\lambda, G, \widehat{G}(v,r)) = G(v)$.*

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:*

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

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 $\widehat{G}$ is in NC^1.

Our definition slightly diverges from [IK00, AIK06]: we introduce an explicit computational randomized encoder, as to make explicit the fact that computational randomized encoding must be efficiently computable.

Remark A.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 [AIK06], we have the following theorem

Theorem A.1 (Corollary of [AIK06])**.** *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*.*

A.6 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.6 (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:* $\mathsf{msg}_i^\ell := \mathsf{Next}_i(1^\lambda, 1^n, z, x_i, r_i, \mathbf{msg}^{<\ell})$ *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* $\mathsf{Msg}^{<\ell} = \{\mathsf{msg}_j^{\ell'}\}_{j\in[n],\ell'<\ell}$*, where* $\mathsf{msg}_j^{\ell'}$ *is the message broadcasted by party P_j on round $\ell' \in [\ell-1]$.*
- *Output:* $y := \mathsf{Output}(1^\lambda, 1^n, z, \mathsf{Msg})$ *is the public output of the MPC protocol, for the public input z, and the transcript* $\mathsf{Msg} = \{\mathsf{msg}_j^\ell\}_{j\in[n],\ell\in[L]}$.

satisfying the perfect correctness property (formally recalled in Definition A.7).

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

Security properties are defined below.

Definition A.7 (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 $(x_1, \dots, x_n)$,*

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

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

Definition A.8 (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:* $(\mathsf{Msg}^{<L}, \{\mathsf{msg}_i^L\}_{i\in\bar{H}})$.

3. *The adversary $\mathcal{A}$ interacts with the challenger by sending queries ($\textsf{compute}, P_i$) for $i \in H$. Upon receiving each query, the challenger sends the last message $\textsf{msg}_i^L$ of P_i.*

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

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 $\textsf{Sim}$. The simulator then outputs a transcript without the last message of the honest parties: ($\textsf{Msg}^{<L}, \{\textsf{msg}_i^L\}_{i\in\bar{H}}$), that the challenger sends back to $\mathcal{A}$.*

3. *The adversary $\mathcal{A}$ interacts with the challenger by sending queries ($\textsf{compute}, P_i$) for $i \in H$. Upon receiving each query, if all the honest parties have not been queried yet (after this query), the challenger sends ($\textsf{compute}, P_i$) to $\textsf{Sim}$ which answers with $\textsf{msg}_i^L$ that the challenger forwards to $\mathcal{A}$. If all the honest parties have been queried, the challenger sends ($\textsf{compute}, P_i, f(x_1, \ldots, x_n)$) to $\textsf{Sim}$, which answers with $\textsf{msg}_i^L$ 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':

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

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 Definition of mrNISC Schemes

We formally recall the definition of mrNISC from [BL20].

Definition B.1 (mrNISC Schemes)**.** *An mrNISC scheme for a functionality $\mathcal{U}$ 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 $\textsf{mrNISC} = (\textsf{Com}, \textsf{Encode}, \textsf{Eval})$ with the following syntax.*

- *Input:* $(\widehat{x}_i, s_i) \leftarrow \textsf{Com}(1^\lambda, x_i)$ *on input the security parameter and an input $x_i \in \{0,1\}^\nu$, generates an input encoding $\widehat{x}_i$ and a secret state s_i, where the input length ν is polynomial in the security parameter λ.*

- *<u>Computation of $\mathcal{U}(z,\star)$:</u> $\alpha_i \leftarrow \mathsf{Encode}(z, \{\widehat{x}_j\}_{j\in I}, s_i)$, on input a public input $z \in \{0,1\}^\nu$, a set of input encodings $\{\widehat{x}_j\}_{j\in I}$, and the secret state s_i generated together with the i'th input encoding $\widehat{x}_i$, generates the* computation encoding *α_i of $\mathcal{U}(z, \{x_j\}_{j\in I})$.*

- *<u>Output:</u> $y = \mathsf{Eval}(z, \{\widehat{x}_i\}_{i\in I}, \{\alpha_i\}_{i\in I})$, on input a public input $z \in \{0,1\}^\nu$, a set of input encodings $\{\widehat{x}_i\}_{i\in I}$ and the corresponding set of computation encodings $\{\widehat{x}_i\}_{i\in I}$, produces an output $y \in \{0,1\}^* \cup \{\perp\}$.*

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

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

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 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 $\mathcal{U}$ 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 $\mathsf{Exp}_{\mathcal{A},\mathcal{S}}(\mathsf{Real}, \lambda, \mathcal{U})$ and $\mathsf{Exp}_{\mathcal{A},\mathcal{S}}(\mathsf{Ideal}, \lambda, \mathcal{U})$ are indistinguishable.*

Experiment $\mathsf{Exp}_{\mathcal{A},\mathcal{S}}(\mathsf{Real}, \lambda, \mathcal{U})$**:** *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 $(\mathit{input}, P_i, x_i, \rho_i)$ for a corrupt party $i \in \bar{H}$, record the input encoding $\widehat{x}_i$ generated as $(\widehat{x}_i, s_i) = \mathsf{Com}(1^\lambda, x_i; \rho_i)$, 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 $(\mathit{input}, P_i, x_i)$ of an honest party $i \in H$, generate $(\widehat{x}_i, s_i) \leftarrow \mathsf{Com}(1^\lambda, x_i)$ and send $\widehat{x}_i$ to $\mathcal{A}$.*

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

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

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

Table 1: Parameter examples (one for $q/\sigma = \lambda^\tau$ and one for $q/\sigma = 2^{\lambda^\epsilon}$)

	$q/\sigma = \lambda^\tau$	$q/\sigma = 2^{\lambda^\epsilon}$
N	4λ	4λ
k	3λ	3λ
q	$\Theta(\lambda^{\tau+2})$	$\Theta(\lambda^2 \cdot 2^{\lambda^\epsilon})$
M	λ^2	λ^2

	$q/\sigma = \lambda^\tau$	$q/\sigma = 2^{\lambda^\epsilon}$
σ	λ^2	λ^2
d	$O(\frac{\tau-22}{4}\log\lambda)$	$O(\frac{\lambda^\epsilon}{\log\lambda})$
σ'	$\lambda^{12} \cdot 4^d$	$\lambda^{8+2\cdot d}$
ρ	$\lambda^9 \cdot 4^d$	λ^{5+2d}

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 $\widetilde{x}_i$ 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 $\mathcal{U}(z, \star)$ 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 $y = \mathcal{U}(z, \{x_t\}_{t\in I})$; otherwise, send $\mathcal{S}$ the query $(\mathsf{compute}, P_i, z, I)$ without y. Forward to $\mathcal{A}$ the simulated computation encoding $\widetilde{\alpha}_i$ 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.

C Additional Material for the 2rNISC Construction

C.1 Setting of Parameters

We describe two example settings of parameters in Table 1. The first setting gives us a scheme which handles (bounded) $d \in O(\log\lambda)$. The second one gives us a scheme that handles d which is any a priori bounded polynomial in λ, but requires subexponential modulus-to-noise ratio. Let $\tau > 22$ and $\epsilon \in (0, 1)$ be constants. We now describe all the criteria that is used to compute these parameters. Let us fix a desired modulus-to-noise ratio δ. Roughly, the requirements can be summarized as follows:

1. Set $N_1 = \lambda$. Set $N = N_1 + k$. Set $k = 3\lambda$ (This is so that, we can extract λ bits upto a statistical distance of $2^{-\lambda}$).
2. Set $\sigma = \lambda^2$, some polynomial in the security of parameter.
3. Set q so that noise-to-modulus ratio is satisfied. Namely, for the first column $q = \Theta(\lambda^{\tau+2})$. For the second column, $\Theta(\lambda^2 \cdot 2^{\lambda^\epsilon})$.
4. Set $M = \omega(N \cdot \log q)$. We set $M = \lambda^2$ for both the settings.
5. We now recall how the norm of $\widetilde{\mathbf{R}}_{g_2}$ grows as d grows. This will also help us to set σ' and ρ later. Consider the first column. We use $\mathsf{GSW.Eval}$ to compute $\widetilde{\mathbf{C}}_{g_2} = \mathbf{A}\widetilde{\mathbf{R}}_{g_2} + g_2(x)\mathbf{G}$ on

input $\{\mathbf{C}_\ell = \mathbf{A}\mathbf{R}_\ell + x_\ell \mathbf{G}\}_{\ell \in [n]}$ where $\mathbf{R}_\ell \leftarrow \{-1,+1\}^{M \times N\lceil \log q \rceil}$ using the evaluation algorithm proposed by [BV14]. To evaluate a depth d circuit, it takes time $O(4^d \cdot \mathsf{poly}(M))$, satisfying:

$$\|\widetilde{\mathbf{R}}_{g_2}\|_\infty = O(4^d \cdot M^2).$$

This let's us evaluate all circuits in NC^1 with a polynomial modulus-to-noise ratio. Further, the modulus is also polynomial.

For the second column, we use the regular evaluation procedure specified in [GSW13]. To evaluate a depth d circuit,

$$\|\widetilde{\mathbf{R}}_{i,g_2}\|_\infty = O(M^d).$$

6. We now set σ' so that conditional entropy in $\boldsymbol{u}_{1-g_2(x)}$ given $\boldsymbol{w}_{1-g_2(x)}$ is at least k bits. We set σ' so that it is λ times larger than the radius of the ball containing k small linearly independent vectors in row span of $[\mathbf{A}|\ \mathbf{A}\widetilde{\mathbf{R}}_{g_2}]$. This will ensure that constraints of Lemma 3.5 are satisfied and λ bits can be extracted from $\boldsymbol{u}_{1-g_2(x)}$. Thus, we set σ' loosely so that:

$$\sigma' > \lambda \cdot M \cdot (M + N\lceil \log q \rceil) \cdot \|\boldsymbol{e}_l\|_\infty \cdot \|\widetilde{\mathbf{R}}_{g_2}\|_\infty.$$

Substituting the bound on the error vectors, we get even more loose bound below:

$$\sigma' > \lambda \cdot M^2 \cdot (\lambda^2 \cdot \sqrt{M}) \cdot \|\widetilde{\mathbf{R}}_{g_2}\|_\infty.$$

Thus in first column, we set:

$$\sigma' > \lambda^3 \cdot M^{4.5} \cdot 4^d = \lambda^{12} \cdot 4^d.$$

For the second column, we set:

$$\sigma' > \lambda^3 \cdot M^{d+2.5}.$$

7. Finally, we set ρ, so that one can decode uniquely $\boldsymbol{u}_{g_2(x)}$ given $\boldsymbol{w}_{g_2(x)}$. This can be done if the conditions of Lemmas 3.3 and 3.4 are met. From Lemma 3.3, we can set:

$$\rho > \max\{\lambda \cdot \|\mathbf{T}_{\mathbf{A}}\|_{\mathsf{GS}},\ \lambda \cdot \|\mathbf{T}_{\mathbf{G}}\|_{\mathsf{GS}} \cdot \|\widetilde{\mathbf{R}}_{g_2}\|\}).$$

Substituting the bound from Definition 3.1 we obtain a more loose upper bound.

$$\sigma' > \max\{\lambda \cdot \sqrt{M},\ \lambda \cdot M \cdot \|\widetilde{\mathbf{R}}_{g_2}\|\}.$$

Finally, we substitute a bound on the operator norm as a function of the infinity norm to get the following loose upper bound.

$$\sigma' > \max\{\lambda \cdot \sqrt{M},\ \lambda \cdot M^2 \cdot \|\widetilde{\mathbf{R}}_{g_2}\|_\infty\},$$

Thus, we will set $\rho > \lambda \cdot M^4 \cdot 4^d$ for the first column and $\rho > \lambda \cdot M^{2+d}$ in the second column. Secondly, we also relate ρ, σ' and q by requiring the constraints of Lemma 3.4 are met. Namely:

$$q > \|\widetilde{\boldsymbol{e}}\| \cdot \|\mathbf{X}_{g_2}\|.$$

Substituting bounds on the operator norms we can set q loosely as follows:

$$\begin{aligned} & q > (\sigma' \cdot M) \cdot (\sqrt{M} \cdot \rho), \\ \implies & q > M^{1.5} \cdot \rho \cdot \sigma'. \end{aligned}$$

For the first case, this can be done by setting:

$$\begin{aligned} & q > M^{1.5} \cdot \lambda \cdot M^4 \cdot 4^d \cdot \lambda^{12} \cdot (4)^d, \\ & q > \lambda^{13} \cdot M^{5.5} \cdot (16)^d = \lambda^{24} \cdot (16)^d. \text{ (as } M = \lambda^2\text{)}. \end{aligned}$$

For the second case, this can be done by setting:

$$\begin{aligned} & q > M^{1.5} \cdot \lambda \cdot M^{2+d} \cdot \lambda^3 \cdot M^{d+2.5} \\ & q > \lambda^4 \cdot M^{2d+6}. \end{aligned}$$

The parameters we give for both the cases above satisfy all these constraints.

C.2 Security Proof of the 2rNISC for Depth-Bounded Functions

We give a sequence of hybrid experiments used to argue that the scheme described in Section 4.1 is secure. We shall prove that each two consecutive hybrids are indistinguishable. The first hybrid $\textbf{Hybrid}_0$ corresponds to the real world, where as the last hybrid corresponds to the ideal world, where the view is simulated without the input. A simulator that generates the view of the last hybrid is given in Figure 3.

$\textbf{Hybrid}_0$: This hybrid is exactly the experiment defined by $\mathsf{Exp}_{\mathcal{A},\mathcal{S}}(\mathsf{Real}, \lambda, \mathcal{U}_{\mathsf{fOT},\lambda,d})$:
On input 1^λ and d, the adversary outputs the set of parties $[\kappa]$ along with a set $H \subseteq [\kappa]$ of the honest parties. Let $\bar{H} = [\kappa] \setminus H$. Then it can adaptively make queries of the following forms:

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 $\widehat{x}_i$ generated as $(\widehat{x}_i, s_i) = \mathsf{Com}(1^\lambda, x_i; \rho_i)$, using input x_i and randomness ρ_i. (If we were to prove semi-honest security, ρ_i would be sampled randomly, but since we prove semi-malicious security, we let $\mathcal{A}$ choose ρ_i.)

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

Honest Computation Encoding: $\mathcal{A}$ makes upto Q number of queries, where Q is a polynomial of its own choice. For every query $\eta \in [Q]$, it submits a tuple of the form $(\mathsf{compute}, P_i, (g_{1,\eta}, g_{2,\eta}), I_\eta)$ for an honest party $i \in H \cap I_\eta$. Here, I_η is an ordered set of size 2. If the input encodings $\{\widehat{x}_j\}_{j \in I_\eta}$ of all participants have been generated, send $\mathcal{A}$ the computation encoding $\alpha_{i,\eta} \leftarrow \mathsf{Encode}(z_\eta = (g_{1,\eta}, g_{2,\eta}), s_i, \{\widehat{x}_j\}_{j \in I_\eta})$. Otherwise send $\perp$.

$\textbf{Hybrid}_{1,i}$: For $i \in [\kappa]$, this hybrid is the same as the previous one, except that for an honest party P_i, the queries of the form $(\mathsf{compute}, P_i, (g_{1,\eta}, g_{2,\eta}), I_\eta = \{i, j\})$ (where P_i's input is the first input). In this case, we generate the output of P_i differently. If $\mathsf{flag}_j = \perp$, then we output $\perp$ as before, but, otherwise let $\alpha_{i,\eta} = (\mathsf{sd}, \boldsymbol{w}_{0,\eta}, \boldsymbol{w}_{1,\eta}, v_{0,\eta}, v_{1,\eta})$. In this case, we change $v_{1-g_2(x_j),\eta}$ to be a randomly sampled bit string. Everything else, remains the same.

Denote $\mathbf{Hybrid}_0 = \mathbf{Hybrid}_{1,0}$, then for $i \in [0, \kappa-1]$, $\mathbf{Hybrid}_{1,i}$ is indistinguishable to $\mathbf{Hybrid}_{1,i+1}$ due to Lemma 3.5. For any query of the form: $(\mathsf{compute}, P_i, (g_{1,\eta}, g_{2,\eta}), I_\eta = \{i, j\})$, we compute $\boldsymbol{w}_{b,\eta} = \boldsymbol{u}_{b,\eta} \cdot [\mathbf{A}_j | \widetilde{\mathbf{C}}_{j,g_{2,\eta}} - (1-b) \cdot \mathbf{G}] + \widetilde{\boldsymbol{e}}_{b,\eta} \bmod q$ for $b \in \{0,1\}$. Here $\widetilde{\boldsymbol{e}}_{b,\eta}$ is sampled from $\mathcal{D}_{\sigma'}^{1\times(M+N\lceil\log q\rceil)}$. Now, $\mathbf{A}_j$ is always sampled to have k short linearly independent vectors. Since: $\widetilde{\mathbf{C}}_{j,g_{2,\eta}} = \mathbf{A}_j \cdot \widetilde{\mathbf{R}}_{j,g_{2,\eta}} + g_{2,\eta}(x_j) \cdot \mathbf{G}$,

$$\boldsymbol{w}_{1-g_{2,\eta}(x_j),\eta} = \boldsymbol{u}_{1-g_{2,\eta}(x_j),\eta} \cdot [\mathbf{A}_j | \mathbf{A}_j \widetilde{\mathbf{R}}_{j,g_{2,\eta}}] + \widetilde{\boldsymbol{e}}_{1-g_{2,\eta}(x_j),\eta} \bmod q$$

This means that $[\mathbf{A}_j | \mathbf{A}_j \widetilde{\mathbf{R}}_{j,g_{2,\eta}}]$ also has short k linearly independent vectors as $\widetilde{\mathbf{R}}_{j,g_{2,\eta}}$ is also short. By Lemma 3.5, $\boldsymbol{u}_{1-g_{2,\eta}(x_j),\eta}$ has at least $3 \cdot \lambda$ bits of entropy due to the set parameters. Thus, replacing $v_{1-g_{2,\eta}(x_j),\eta}$ by a random λ bit string is statistically indistinguishable due to the security of Ext.

$\mathbf{Hybrid}_{2,i}$: For $i \in [\kappa]$, the hybrid is the same as the previous one, except that if party P_i is honest then the input commitment is done differently. It it is corrupt then there is no change. For an honest party P_i, the matrix $\mathbf{A}_i \leftarrow \mathbb{Z}_q^{N \times M}$ is sampled at random.

Denote $\mathbf{Hybrid}_{1,\kappa} = \mathbf{Hybrid}_{2,0}$, then for $i \in [0, \kappa-1]$, $\mathbf{Hybrid}_{2,i}$ is indistinguishable to $\mathbf{Hybrid}_{2,i+1}$ due to the hardness of LWE. Note that if P_{i+1} is honest then $\mathbf{A}_{i+1}$ is generated using LWE distribution, and both the secrets and the error vectors are hidden thus these two distributions are indistinguishable. Further, if P_i is corrupt, then there is no change.

$\mathbf{Hybrid}_{3,i}$: For $i \in [\kappa]$, this hybrid is the same as the previous one, except that for an honest party P_i, the input commitment is done differently. The matrix $\mathbf{A}_i$ are sampled by running $\mathsf{TrapGen}(1^N, 1^M, q) \to (\mathbf{A}_i, \mathbf{T}_{\mathbf{A}_i})$.

Denote $\mathbf{Hybrid}_{2,\kappa} = \mathbf{Hybrid}_{3,0}$, then for $i \in [0, \kappa-1]$, $\mathbf{Hybrid}_{3,i}$ is indistinguishable to $\mathbf{Hybrid}_{3,i+1}$ due to the security property of the $\mathsf{TrapGen}$ algorithm. Note that the matrices generated by $\mathsf{TrapGen}$ algorithm are statistically indistinguishable to random matrices.

$\mathbf{Hybrid}_{4,i}$: For $i \in [\kappa]$, this hybrid is the same as the previous one, except that for an honest party P_i, the queries of the form $(\mathsf{compute}, P_i, (g_{1,\eta}, g_{2,\eta}), I_\eta = \{j, i\})$ (where P_i's input is the second input). In this case, $\mathbf{X}_{i,g_{2,\eta}}$ is generated using $\mathsf{SampleLeft}$ algorithm using the trapdoor $\mathbf{T}_{\mathbf{A}_i}$ for $\mathbf{A}_i$. Namely,

$$\mathbf{X}_{i,g_{2,\eta}} = \begin{cases} \mathsf{SampleLeft}(\mathbf{A}_i, \widetilde{\mathbf{C}}_{i,g_{2,\eta}} - \mathbf{G}, \mathbf{T}_{\mathbf{A}_i}, \rho) \text{ when } g_{2,\eta}(x_i) = 0 \\ \mathsf{SampleLeft}(\mathbf{A}_i, \widetilde{\mathbf{C}}_{i,g_{2,\eta}}, \mathbf{T}_{\mathbf{A}_i}, \rho) \text{ when } g_{2,i}(x_i) = 1 \end{cases}$$

Denote $\mathbf{Hybrid}_{3,\kappa} = \mathbf{Hybrid}_{4,0}$, then for $i \in [0, \kappa-1]$ where P_{i+1} is honest, $\mathbf{Hybrid}_{4,i}$ is statistically indistinguishable to $\mathbf{Hybrid}_{4,i+1}$ due to the left-right indistinguishability property of trapdoor sampling algorithms $\mathsf{SampleLeft}$ and $\mathsf{SampleRight}$ (Refer Lemma 3.3). If P_{i+1} is corrupt, then these hybrids is identical.

Hybrid$_{5,i}$: For $i \in [\kappa]$, this hybrid is the same as the previous one, except that for an honest party P_i, the input commitment is done differently. Namely, sample $\mathbf{C}_{i,\ell}$ for $\ell \in [n_i]$ uniformly at random from $\mathbb{Z}_q^{N \times N \cdot \lceil \log q \rceil}$.

Denote $\mathbf{Hybrid}_{4,\kappa} = \mathbf{Hybrid}_{5,0}$, then for $i \in [0, \kappa - 1]$ where P_{i+1} is honest, $\mathbf{Hybrid}_{5,i}$ is statistically indistinguishable to $\mathbf{Hybrid}_{5,i+1}$ due to Lemma 3.2. Each ciphertext $\mathbf{C}_{i+1,\ell} = \mathbf{A}_{i+1} \cdot \mathbf{R}_{i+1,\ell} + x_{i+1,\ell}\mathbf{G}$. By Lemma 3.2, $\mathbf{A}_{i+1} \cdot \mathbf{R}_{i+1,\ell}$ is indistinguishable to a random matrix in $\mathbb{Z}_q^{N \times N \lceil \log q \rceil}$ even given $\mathbf{T}_{\mathbf{A}_{i+1}}$. Thus, $\mathbf{C}_{i+1,\ell}$ is statistically indistinguishable to random. Note that $\mathbf{R}_{i+1,\ell}$ is not used anywhere else. Therefore, the claim follows.

$\mathbf{Hybrid}_{5,\kappa}$ gives us our simulator, as this is completely simulatable using the number of parties κ, set of honest parties H, corrupted inputs $\{x_i\}_{i \in \bar{H}}$, corrupted randomness $\{\rho\}_{i \in \bar{H}}$ and the outputs learnt for each of the queries $\{g_{1,\eta}(x_i)_{g_{2,\eta}(x_j)}, g_{2,\eta}(x_j)\}_{(i,j) \in I_\eta \forall \eta \in [Q]}$. This hybrid corresponds to $\mathsf{Exp}_{\mathcal{A},\mathcal{S}}(\mathsf{Ideal}, \lambda, \mathcal{U}_{\mathsf{fOT},\lambda,d})$

We describe our simulator $\mathcal{S}$ in Figure 3.

C.3 Bootstrapping 2rNISC for All Functions

In this section, we formally prove Theorem 4.3. The ingredients that our transformation uses are listed next:

- A 2rNISC protocol for $\mathcal{U}_{\mathsf{fOT},\lambda,c\log\lambda}$, denoted as 2rNISC$'$, for some large enough constant $c > 0$.
- A collision-resistant hash function family $\{\mathcal{HF}_\lambda\}_{\lambda \in \mathbb{N}}$, where each function in the family $\mathcal{H} : \{0,1\}^* \to \{0,1\}^{2\lambda}$.
- A pseudorandom function $\mathsf{PRF} : \{0,1\}^\lambda \times \{0,1\}^{5\lambda} \to \{0,1\}$ in NC^1. Assume that the depth of PRF is bounded by $c_1 \cdot \log_2 \lambda$ for some constant $c_1 > 0$.
- A circuit garbling scheme (see Definition A.2) $\mathsf{GC} = (\mathsf{GC.Gen}, \mathsf{GC.Garble}, \mathsf{GC.Eval}, \mathsf{GC.Sim})$ for all circuits with input label length of λ. Such a scheme can be built from any one-way function.
- A randomized encoding scheme $\mathsf{CRE} = (\mathsf{CRE.Enc}, \mathsf{CRE.Dec}, \mathsf{CRE.Sim})$ for all circuits (see Definition A.5), where $\mathsf{CRE.Enc}(\cdot)$ is a depth $c_2 \log \lambda$ circuit for some constant $c_2 > 0$. This can be instantiated using a PRF in NC^1.

We now describe our construction. The constructed protocol 2rNISC is described as follows:

Commit on input $(1^\lambda, x)$**:** On input $x \in \{0,1\}^*$ perform the following steps:

- Sample a function $\mathcal{H}$ from the collision-resistant hash function family.
- Sample a PRF key $K \leftarrow \{0,1\}^\lambda$.
- Compute $\mathsf{2rNISC}'.\mathsf{Com}(1^\lambda, (x, K)) \to (\mathsf{2rNISC}'.\widehat{x}, \mathsf{2rNISC}'.s)$.
- Output $\widehat{x} = (\mathsf{2rNISC}'.\widehat{x}, \mathcal{H})$ and $s = (\mathsf{2rNISC}'.s, K)$.

Simulator $\mathcal{S}$

Corrupt Input Encoding: Upon $\mathcal{A}$ sending a query $(\mathsf{input}, P_i, x_i, \rho_i)$ for a corrupt party $i \in \bar{H}$ for the first time, record the input encoding $\widehat{x}_i$ generated as $(\widehat{x}_i, s_i) = \mathsf{Com}(1^\lambda, x_i; \rho_i)$, using input x_i and randomness ρ_i. Parse $\widehat{x}_i = (\mathsf{flag}_i, \mathbf{A}_i, \{\mathbf{C}_{i,\ell}\}_{\ell \in [n_i]})$.

Honest Input Encoding: Upon $\mathcal{A}$ querying $(\mathsf{input}, P_i, n_i)$ for an honest party $i \in H$ for the first time, generate $(\widehat{x}_i, s_i)$ as follows. Set $\mathsf{flag}_i = 1$. Sample $(\mathbf{A}_i, \mathbf{T}_{\mathbf{A}_i}) \leftarrow \mathsf{TrapGen}(1^N, 1^M, q)$. Sample $\mathbf{C}_{i,\ell} \leftarrow \mathbb{Z}_q^{N \times N\lceil \log q \rceil}$. Output $\widehat{x}_i = (\mathsf{flag}_i, \mathbf{A}_i, \{\mathbf{C}_{i,\ell}\}_{\ell \in [n_i]})$ and set and store $s_i = \mathbf{T}_{\mathbf{A}_i}$.

Honest Computation Encoding: $\mathcal{A}$ makes upto Q number of queries, where Q is a polynomial of its own choice. For every query $\eta \in [Q]$, it submits a tuple of the form $(\mathsf{compute}, P_i, (g_{1,\eta}, g_{2,\eta}), I_\eta)$ for an honest party $i \in H \cap I_\eta$. If the input encodings of each party in I_η is generated then proceed as follows. If $I_\eta = (i, j)$ (Party i is the party with the first input.) then,

- Parse $\widehat{x}_j = (\mathsf{flag}_j, \mathbf{A}_j, \{\mathbf{C}_{j,\ell}\}_{\ell \in [n_j]})$. If P_j is corrupted and $\mathsf{flag}_j = 1$, output $\perp$.
- Let (β_η, y_η) be the output, where $\beta_\eta \in \{0,1\}$ and $y_\eta \in \{0,1\}^\lambda$.
- Sample sd as a seed of the extractor Ext. Sample $\boldsymbol{u}_{0,\eta}, \boldsymbol{u}_{1,\eta} \leftarrow \mathbb{Z}_q^{1 \times N}$. Compute $\widetilde{\mathbf{C}}_{j,g_{2,\eta}} = \mathsf{GSW.Eval}(g_{2,\eta}, \{\mathbf{C}_{j,\ell}\}_{\ell \in [n_j]})$.
- For $b \in \{0,1\}$, compute $\boldsymbol{w}_{b,\eta} = \boldsymbol{u}_{b,\eta}[\mathbf{A}_j | \widetilde{\mathbf{C}}_{j,g_{2,\eta}} - (1-b)\mathbf{G}] + \widetilde{\boldsymbol{e}}_{b,\eta}$ where $\widetilde{\boldsymbol{e}}_{b,\eta} \leftarrow \mathcal{D}_{\mathbb{Z}^{M+N\lceil \log q \rceil}, \sigma'}$.
- Set $v_{\beta_\eta,\eta} = \mathsf{Ext}(\mathsf{sd}, \boldsymbol{u}_{\beta_\eta,\eta}) \oplus y_\eta$ and $v_{1-\beta_\eta,\eta} \leftarrow \{0,1\}^\lambda$.
- Output $(\mathsf{sd}, \boldsymbol{w}_{0,\eta}, \boldsymbol{w}_{1,\eta}, v_{0,\eta}, v_{1,\eta})$.

If $I_\eta = (j, i)$ (Party i is the party with the second input.) then,

- Parse $\widehat{x}_i = (\mathsf{flag}_i, \mathbf{A}_i, \{\mathbf{C}_{i,\ell}\}_{\ell \in [n_i]})$ and $s_i = \mathbf{T}_{\mathbf{A}_i}$. Parse $\widehat{x}_j = (\mathsf{flag}_j, \mathbf{A}_j, \{\mathbf{C}_{j,\ell}\}_{\ell \in [n_j]})$. If P_j is corrupted and $\mathsf{flag}_j = 1$, output $\perp$.
- Let (β_η, y_η) be the output, where $\beta_\eta \in \{0,1\}$ and $y_\eta \in \{0,1\}^\lambda$.
- Compute $\mathsf{GSW.Eval}(g_{2,\eta}, \{\mathbf{C}_{i,\ell}\}_{\ell \in [n_i]}) = \widetilde{\mathbf{C}}_{i,g_{2,\eta}}$.
- Compute $\mathbf{X}_\eta = \mathsf{SampleLeft}(\mathbf{A}_i, \widetilde{\mathbf{C}}_{i,g_{2,\eta}} - (1-\beta_\eta)\mathbf{G}, \mathbf{T}_{\mathbf{A}_i}, \rho)$.
- Output $(\beta_\eta, \mathbf{X}_\eta)$.

Figure 3: Description of Simulator $\mathcal{S}$ for the proof Theorem 4.2.

Encode: There are two cases, depending on the "order" of the parties involved, denoted P and P'. In both cases, the view of party P (or its query) consists of $\widehat{x}, \widehat{x}', s$ and the view of P' consists of $\widehat{x}, \widehat{x}', s'$. The descriptions of g_1, g_2 are public. Remember both g_1 and g_2 are arbitrary depth, polynomial sized circuits. In both cases, party P parses the public message of itself as well as the public message of P' as follows:

- Parse $\widehat{x}' = (\mathsf{2rNISC}'.\widehat{x'}, \mathcal{H}')$.
- Parse $\widehat{x} = (\mathsf{2rNISC}'.\widehat{x}, \mathcal{H})$.

First, we define three circuits that will be used in the descriptions below. First function is denoted as $D_{g,z}$. It is parameterized using a function $g : \{0,1\}^n \to \{0,1\}$ and a value $t \in \{0,1\}^{4\lambda}$. The circuit is defined as follows:

$$\begin{aligned}&D_{g,t}(x, K) = \mathsf{CRE.Enc}(g, x; \rho),\\ &\text{where } \rho = (\mathsf{PRF.Eval}(K, t|1), \ldots, \mathsf{PRF.Eval}(K, t|\ell_{\mathsf{CRE},g})),\end{aligned}$$

Where ℓ_ρ is the length of the randomness required to compute the randomized encoding. Let $\ell_{\mathsf{CRE},g}$ denote the length of the output of the circuit and then for every $i \in [\ell_{\mathsf{CRE},g}]$ $D_{i,g,t}$ denote the circuit that outputs the i^{th} bit. Observe that $D_{i,g,t}$ has a depth of $c \log \lambda$ for some fixed c, because depth of the PRF on input $t|j$ for $j \in [\ell_\rho]$ is $c_1 \log \lambda$ (as the input length to the PRF is bounded by 5λ bits) and also the depth to compute the i^{th} bit of $\mathsf{CRE.Enc}$ is $c_2 \log \lambda$. Thus the total depth is bounded by $(c_1 + c_2) \log \lambda$. Therefore, we need to start-off with $c \geq c_1 + c_2$.

Now, define the second circuit $C_{v_0,v_1,\ell} : \{0,1\}^\ell \to \{0,1\}^\lambda$, which is parameterized by two values v_0 and v_1 in $\{0,1\}^\lambda$ and a length ℓ.

$$\begin{aligned}&C_{v_0,v_1,\ell}(a) = (\beta, v_\beta),\\ &\text{where } \beta = \mathsf{CRE.Dec}(a)\end{aligned}$$

Finally, we define the third circuit $\phi_{t,\ell}$. The circuit ϕ is parameterized with a value $t \in \{0,1\}^{4\lambda}$ and ℓ is a length parameter.

$$\begin{aligned}&\phi_{t,\ell}(x, K) = \mathsf{GC.Gen}(1^\lambda; \rho') = \mathsf{key} \in \{0,1\}^{2\ell\lambda},\\ &\text{where } \rho' = (\mathsf{PRF.Eval}(K, t|1), \ldots, \mathsf{PRF.Eval}(K, t|\ell'))\end{aligned}$$

Above ℓ' is the length of randomness used to sample garbling key to garble circuit $C_{v_0,v_1,\ell}$. For $i \in [\ell]$, let $\phi_{i,t,\ell}$ denote the circuit that outputs $(\mathsf{key}[i,0], \mathsf{key}[i,1])$. Party P proceeds as follows, depending on whether it is the "first" party or the "second". With this notation, we can finally describe the two cases.

Case 1: Party P is the "first" party.

- Compute $(v_0, v_1) = g_1(x)$.
- Compute $t_1 = \mathcal{H}(g_1, g_2, \widehat{x}, \widehat{x}')$ and $t_2 = \mathcal{H}'(g_1, g_2, \widehat{x}, \widehat{x}')$. Define $t = (t_1|t_2)$.
- Let ℓ_{CRE,g_2} be the length of the output of $D_{g_2,t}$. Compute $\mathsf{key} = \mathsf{GC.Gen}(1^\lambda; \rho)$ where $(\rho = (\mathsf{PRF.Eval}(K, t|1), \ldots, \mathsf{PRF.Eval}(K, t|\ell_\rho))$ is used to sample garbling key to garble $C_{v_0,v_1,\ell_{\mathsf{CRE},g_2}}$). Compute $\mathsf{GC.Garble}(\mathsf{key}, C_{v_0,v_1,\ell_{\mathsf{CRE},g_2}}) = \widetilde{C}_{v_0,v_1,\ell_{\mathsf{CRE},g_2}}$. Set $\alpha_0 = \widetilde{C}_{v_0,v_1,\ell_{\mathsf{CRE},g_2}}$.

- Compute for each $i \in [\ell_{\mathsf{CRE},g_2}]$, $\alpha_i = \mathsf{2rNISC}'.\mathsf{Encode}((\phi_{i,t,\ell_{\mathsf{CRE},g_2}}, D_{i,g_2,t}), \mathsf{2rNISC}'.\widehat{x}, \mathsf{2rNISC}'.\widehat{x}', \mathsf{2rNISC}'.s)$. Note that this okay to do because the depth of $D_{i,g_2,t}$ is bounded by $(c_1+c_2)\log\lambda \le c\log\lambda$ as pointed above.
- Output $\alpha = (\alpha_0, \alpha_1, \ldots, \alpha_{\ell_{\mathsf{CRE},g_2}})$.

Case 2: Party P is the "second" party.

- Compute $t_1 = \mathcal{H}'(g_1, g_2, \widehat{x}', \widehat{x})$ and $t_2 = \mathcal{H}'(g_1, g_2, \widehat{x}', \widehat{x})$. Set $t = (t_1|t_2)$.
- For every $i \in [\ell_{\mathsf{CRE},g_2}]$, compute $\alpha_i = \mathsf{2rNISC}'.\mathsf{Encode}((\phi_{i,t,\ell_{\mathsf{CRE},g_2}}, D_{i,g_2,t}), \mathsf{2rNISC}'.\widehat{x}', \mathsf{2rNISC}'.\widehat{x}, \mathsf{2rNISC}'.s)$. Note that this okay to do because the depth of $D_{i,g_2,t}$ is bounded by $c\log\lambda$ as pointed above.
- Output $\alpha = (\alpha_1, \ldots, \alpha_{\ell_{\mathsf{CRE},g_2}})$.

Eval on input $(z = (g_1, g_2), \widehat{x}, \widehat{x}', \alpha, \alpha')$**:** Let P be the first party and P' be the second party.

- Parse $\widehat{x} = (\mathsf{2rNISC}'.\widehat{x}, \mathcal{H})$ and $\widehat{x'} = (\mathsf{2rNISC}'.\widehat{x}', \mathcal{H}')$.
- Compute $t_1 = \mathcal{H}(g_1, g_2, \widehat{x}, \widehat{x}')$ and compute $t_2 = \mathcal{H}'(g_1, g_2, \widehat{x}, \widehat{x}')$. Set $t = (t_1|t_2)$.
- Parse $\alpha = (\alpha_0, \alpha_1, \ldots, \alpha_{\ell_{\mathsf{CRE},g_2}})$.
- Parse $\alpha' = (\alpha'_1, \ldots, \alpha'_{\ell_{\mathsf{CRE},g_2}})$.
- Compute $\beta_i = \mathsf{2rNISC}'.\mathsf{Eval}((\phi_{i,t,\ell_{\mathsf{CRE},g_2}}, D_{i,g_2,t}), \mathsf{2rNISC}'.\widehat{x}, \mathsf{2rNISC}'.\widehat{x}', \alpha_i, \alpha'_i)$ for $i \in [\ell_{\mathsf{CRE},g_2}]$.
- Compute and output $\mathsf{out} = \mathsf{GC}.\mathsf{Eval}(\alpha_0, \{\beta_i\}_{i\in[\ell_{\mathsf{CRE},g_2}]})$.

We proceed with the proof of correctness and security of the construction.

Correctness. The correctness can be observed by inspection and follows due to the correctness of $\mathsf{2rNISC}'$, GC and CRE. Denote $w = D_{t,g_2}(x', K')$ and $w_i = D_{i,t,g_2}(x', K')$ be its ith bit. Note that during the evaluation β_i is equal to $\mathsf{key}[i, w_i]$ and α_0 is a garbling of $C_{v_0,v_1,\ell_{\mathsf{CRE},g_2}}$ using key. Observe that $w = \mathsf{CRE}.\mathsf{Enc}(g_2, x'; \rho')$ using the randomness ρ' derived from the PRF key K'. Thus, the final step produces $C_{v_0,v_1,\ell_{\mathsf{CRE},g_2}}(w) = (\mathsf{CRE}.\mathsf{Dec}(w), v_{\mathsf{CRE}.\mathsf{Dec}(w)})$. Note that $\mathsf{CRE}.\mathsf{Dec}(w) = g_2(x')$ due to correctness of CRE, and $(v_0, v_1) = g_1(x)$ as described in the protocol. Thus, correctness holds.

Security. For security we present a sequence of hybrid experiments, where each two consecutive ones are proven to be indistinguishable, and the last one is independent of the parties' inputs. The hybrids are as follows:

- **Hybrid**$_0$: This hybrid corresponds to the real experiment in the original security game with the above scheme.

- **Hybrid**$_1$: This experiment is the same as the previous one, except that instead of using $\mathsf{2rNISC}'$ to generate the honest input encoding queries as well as the honest computation encodings, we use the simulator of that protocol.

  The two hybrids above are indistinguishable due to the security of $\mathsf{2rNISC}'$ protocol.

- **Hybrid**$_2$: This experiment is the same as the previous one, except that we abort if the adversary ever makes two distinct computation queries of the form $(\mathsf{compute}, P_i, (g_{1,\eta_1}, g_{2,\eta_1}, I_{\eta_2}))$ and $(\mathsf{compute}, P_i, (g_{1,\eta_2}, g_{2,\eta_2}, I_{\eta_2}))$ for an honest party P_i such that the hash values $\mathcal{H}_i(g_{1,\eta_1}, g_{2,\eta_1}, \{\widehat{x}_j\}_{j\in I_{\eta_1}}) =$

$\mathcal{H}_i(g_{1,\eta_2}, g_{2,\eta_2}, \{\widehat{x}_j\}_{j \in I_{\eta_2}})$.

The two hybrids above are indistinguishable due to the security of the collision resistant hash function.

- **Hybrid**$_3$: This experiment is the same as the previous one, except that instead of using a PRF to generate hardwired outputs (which consists of various encodings) of honest parties to be used by $\mathsf{2rNISC}'$ simulator, we use a truly random function.

  The two hybrids above are indistinguishable due to the security of the PRF.

- **Hybrid**$_4$: This experiment is the same as the previous one, except that for generating outputs of an honest party (used for hardwiring) in executions where it is the second party we use the simulator of the randomized encoding, instead of honest randomized encodings.

  The two hybrids above are indistinguishable due to the security of the CRE scheme.

- **Hybrid**$_5$: This experiment is the same as the previous one, except that we use the simulator of the garbled circuit, to simulate the garbled circuits and labels which are used by the simulator of $\mathsf{2rNISC}'$ scheme, for all honest parties, instead of computing real garbled circuit and labels.

  The two hybrids above are indistinguishable due to the security of the GC scheme.

D Security Proof of the mrNISC Construction

Theorem 5.1. It remains to prove the security of the construction in Section 5.

The proof is similar to the security of the mrNISC in [BL20]. We construct a simulator $\mathcal{S}$ for the mrNISC. We develop the simulator $\mathcal{S}$ via a sequence of hybrids $H_0, H_{1,1,1}, H_{1,1,2}, H_{1,1,3}, H_{1,2,1}, \ldots, H_{1,L,3}$, H_2, H_3 where H_0 is the real experiment $\mathsf{Exp}_{\mathcal{A},\mathcal{S}}(\mathsf{Real}, \lambda, \mathcal{U})$ and H_3 is the ideal experiment $\mathsf{Exp}_{\mathcal{A},\mathcal{S}}(\mathsf{Ideal}, \lambda, \mathcal{U})$, which also contains the description of Sim.

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

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),

1. $\mathcal{S}$ computes the transcript $\mathsf{Msg} = \{\mathsf{msg}_j^\ell\}_{j \in [n], \ell \in [L]}$ 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 $\{r_j := \mathsf{PRF}(\mathrm{fk}_j^0, z \| [\nu_r])\}$, where fk_j can be derived from the randomness ρ_j used to commit to inputs x_j (i.e., $(\widehat{x}_j, s_j) = \mathsf{Com}(1^\lambda, x_j; \rho_j)$).
2. For $\ell < \ell^\star$, $\mathcal{S}$ simulates the input and computation encodings $\widehat{x}_i^\ell$, $\alpha_{i,j,k,1}^\ell$, and $\alpha_{j,i,k,2}^\ell$ (using the simulator for the 2rNISC). Furthermore $\mathcal{S}$ uses a random function instead of $\mathsf{PRF}(\mathrm{fk}_i^\ell, \star)$.
3. For $\ell = \ell^\star$, $\mathcal{S}$ computes honestly the input and computation encodings $\widehat{x}_i^\ell$, $\alpha_{i,j,k,1}^\ell$, and $\alpha_{j,i,k,2}^\ell$ as in Eqs. (1), (3), and (4).

4. For $\ell \leq \ell^\star$, instead of honestly garbling $\mathsf{F}_i^{\ell^\star}$, the simulator $\mathcal{S}$ simulates the garbled circuits:
$$((\mathbf{stateKey}_i^\ell[\mathsf{Msg}^{\leq \ell-2}],\ \{\mathbf{msgKey}_{i,j}^\ell[\mathsf{msg}_j^{\ell-1}]\}_{j\in[n]}),\ \widehat{\mathsf{F}}_i^\ell) \leftarrow_{\mathrm{R}} \mathsf{GC.Sim}(1^\lambda, y_i^\ell)\ ,$$
where
$$y_i^\ell \coloneqq \Big(\mathbf{stateKey}_i^{\ell+1}[\mathsf{Msg}^{\leq \ell-1}],\ \{\mathsf{ct}_{i,j,k,b}^\ell\}_{j,k,b},\ \{\alpha_{i,j,k,1}^\ell\}_{j,k},\ \{\alpha_{j,i,k,2}^\ell\}_{j,k}\Big)\ .$$

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

Hybrid $H_{1,\ell^\star,2}$ ($\ell^\star \in [L]$) is identical to $H_{1,\ell^\star,1}$ except that for $\ell = \ell^\star$, the simulator $\mathcal{S}$ simulates $\widehat{x}_i^\ell$, $\alpha_{i,j,k,1}^\ell$, and $\alpha_{j,i,k,2}^\ell$ as for $\ell < \ell^\star$.

This hybrid is computationally indistinguishable from the previous hybrid $H_{1,\ell^\star,1}$ thanks to the security of the 2rNISC applied to the virtual parties $P_j^{\ell^\star}$.

Hybrid $H_{1,\ell^\star,3}$ ($\ell^\star \in [L]$) is identical to $H_{1,\ell^\star,1}$ except that for $\ell = \ell^\star$, the simulator $\mathcal{S}$ uses a random function instead of $\mathsf{PRF}(\mathrm{fk}_i^\ell, \star)$.

We remark that in $H_{1,\ell^\star,2}$, the key fk_i^ℓ is only used in evaluations of $\mathsf{PRF}(\mathrm{fk}_i^\ell, \star)$ (and is no more included in any garbled circuit). Hence, $H_{1,\ell^\star,3}$ is indistinguishable from $H_{1,\ell^\star,2}$ thanks to pseudorandomness of the PRF.

Furthermore, $\mathsf{PRF}(\mathrm{fk}_i^\ell, \star)$ completely information theoretically hide the labels of the garbled circuit $\mathsf{F}_i^{\ell+1}$ encrypted in $\mathsf{ct}_{i,j,k}^\ell$. Hence, thanks to the security of garbled circuits, $H_{1,\ell^\star,3}$ is indistinguishable from $H_{1,\ell^\star+1,1}$ (if $\ell^\star < L$).

Hybrid H_2 This hybrid is identical to $H_{1,L,3}$, 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_3 This hybrid is identical to H_2, 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 $\{\mathsf{msg}_j^L\}_{j\in H}$ is computed.
- If after this query, all the honest P_j's have been queries for z and I, then msg_i^L is simulated using the output $y = \mathcal{U}(x_1, \dots, x_n)$. Otherwise, msg_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 $\mathsf{Exp}_{\mathcal{A},\mathsf{pSim}}(\mathsf{Ideal}, \lambda, \mathcal{U})$, with the simulator $\mathcal{S}$ implicitly defined by the games above.

This concluses the proof. □

E Applications to Multi-Key FHE

In this section, we show how to construct a Multi-Key FHE with a non-interactive threshold decryption starting from a regular Multi-Key FHE scheme and a mrNISC scheme (which can be built assuming PRF's in NC^1 and LWE with polynomial modulus). The idea behind this is simple. We take in a Multi-Key FHE scheme and decentralize its decryption function using a mrNISC scheme. Below we recall the definition of a multi-key FHE scheme (with and without) non-interactive threshold decryption.

Definition E.1 (Multi-Key FHE Scheme)**.** *A multi-key FHE scheme for $N(\cdot)$ keys consists of the following p.p.t. algorithms* $\mathsf{MFHE} = (\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Eval}, \mathsf{Dec})$ *satisfying the following:*

$\mathsf{Setup}(1^\lambda, 1^d) \to \mathsf{crs}$. *The setup algorithm takes as input the security parameter λ and the depth of the circuits $d(\lambda)$. $d(\lambda)$ is some polynomial in the security parameter. It outputs a string* crs *that is an implicit input to all the other algorithms below.*

$\mathsf{KeyGen}(\mathsf{crs}) \to (\mathsf{pk}, \mathsf{sk})$. *The key generation algorithm takes as input the string* crs *and outputs a key pair* $(\mathsf{pk}, \mathsf{sk})$.

$\mathsf{Enc}(\mathsf{pk}, \mu) \to \mathsf{ct}$. *The encryption algorithm takes as input an encryption key* pk*, along with a message $\mu \in \{0,1\}$, and it outputs a ciphertext* ct*. Without loss of generality the encryption of longer messages is a concatenation of the encryption of individual bits.*

$\mathsf{Eval}(C, \{\mathsf{pk}_i\}_{i\in[n]}, \mathsf{ct}_1, \ldots, \mathsf{ct}_n) \to \widehat{\mathsf{ct}}$. *The deterministic evaluation algorithm takes as input $n \leq N(\lambda)$ public keys $\{\mathsf{pk}_i\}_{i\in[n]}$, n ciphertexts $\{\mathsf{ct}_i\}_{i\in[n]}$ where each ciphertext ct_i encrypts $\ell_i \in \mathbb{N}$ bits under key pk_i. It also takes as input a boolean circuit $C : \{0,1\}^{\sum \ell_i} \to \{0,1\}$ of depth d, and it outputs an evaluated ciphertext $\widehat{ct}$ under $\{\mathsf{pk}_i\}_{i\in[n]}$.*

$\mathsf{Dec}(\{\mathsf{sk}_i\}_{i\in[n]}, \widehat{ct}) \to \widehat{\mu}$. *The decryption algorithm takes as input a possibly evaluated ciphertext $\widehat{\mathsf{ct}}$ under public keys $\{\mathsf{pk}_i\}_{i\in[n]}$ for $n \leq N$. It also takes all secret keys $\{\mathsf{sk}_i\}_{i\in[n]}$ and it outputs the decryption $\widehat{\mu}$.*

Correctness, Compactness and Semantic Security. *We require that for any large enough security parameter λ and polynomials $d(\lambda)$, any $\mathsf{crs} \leftarrow \mathsf{Setup}(1^\lambda, 1^d)$, any $(\mathsf{pk}_i, \mathsf{sk}_i) \leftarrow \mathsf{KeyGen}(\mathsf{crs})$, any plaintexts $\mu_1, \ldots, \mu_n \in \{0,1\}^\ell$ for $\ell_i = \mathsf{poly}(\lambda)$ and $n \leq N$, and any boolean circuit $C\colon \{0,1\}^{\sum \ell_i} \to \{0,1\}$ is a depth d circuit, the following is satisfied:*

Correctness. *Let $\mathsf{ct}_i = \mathsf{Enc}(\mathsf{pk}_i, \mu_i)$ for $i \in [n]$ where each $\mu_i \in \{0,1\}^{\ell_i}$ for $\ell_i = \mathsf{poly}(\lambda)$. Let $\widehat{\mathsf{ct}} = \mathsf{Eval}(C, \mathsf{pk}_1, \ldots, \mathsf{pk}_n, \mathsf{ct}_1, \ldots, \mathsf{ct}_n)$. Then, with all but negligible probability in λ over the coins of* Setup*,* Enc *and* KeyGen*,*

$$\mathsf{Dec}(\{\mathsf{sk}_i\}_{i\in[n]}, \widehat{ct}) = C(\mu_1, \ldots, \mu_n)$$

Compactness of Ciphertexts. *There exists a fixed polynomial, poly_{ct}, such that $|\widehat{\mathsf{ct}}| \leq \mathsf{poly}_{ct}(n, d, \lambda)$ for any ciphertext $\widehat{\mathsf{ct}}$ generated as above.*

Security. MFHE *satisfies semantic security. Namely for any $i \in [n]$ and message μ_0, μ_1 of same length:*

$$(\mathsf{crs}, \mathsf{pk}_i, \mathsf{Enc}(\mathsf{pk}_i, \mu_0)) \approx_c (\mathsf{crs}, \mathsf{pk}_i, \mathsf{Enc}(\mathsf{pk}_i, \mu_1)),$$

Next we define the notion of a multi-key FHE scheme with non-interactive threshold decryption (TMFHE) for short.

Definition E.2 (Multi-Key FHE Scheme with Non-Interactive Threshold Decryption)**.** *A multi-key FHE scheme for N keys with a non-interactive threshold decryption (or TMFHE for short) is a multi-key FHE scheme with following additional algorithms:*

$\mathsf{PartDec}(\mathsf{sk}_i, \widehat{\mathsf{ct}}) \to p_i$. *The partial decryption takes as input an evaluated ciphertext $\widehat{ct}$ encrypted under $(\mathsf{pk}_1, \ldots, \mathsf{pk}_n)$ (for $n \le N$) and a secret key sk_i and it outputs a partial decryption p_i.*

$\mathsf{FinDec}(\{p_i\}_{i\in S}) \to \mu \in \{0, 1, \perp\}$. *The final decryption algorithm takes as input partial decryptions $\{p_i\}_{i\in S}$ for some finite set S and outputs a message in $\{0, 1, \perp\}$.*

There are similar additional properties associated with these algorithms. We require that for any large enough security parameter λ, any polynomial $d(\lambda)$, any $\mathsf{crs} \leftarrow \mathsf{Setup}(1^\lambda, 1^d)$, any positive integer $n \le N(\lambda)$, any $(\mathsf{pk}_i, \mathsf{sk}_i) \leftarrow \mathsf{KeyGen}(\mathsf{crs})$, any plaintexts $\mu_1, \ldots, \mu_n \in \{0,1\}^\ell$ for $\ell_i = \mathsf{poly}(\lambda)$, and any boolean circuit $C\colon \{0,1\}^{\sum \ell_i} \to \{0,1\}$ is a depth d circuit, the following is satisfied:

Correctness. *Let $\mathsf{ct}_i = \mathsf{Enc}(\mathsf{pk}_i, \mu_i)$ for $i \in [n]$ where each $\mu_i \in \{0,1\}^{\ell_i}$ for $\ell_i = \mathsf{poly}(\lambda)$. Let $\widehat{\mathsf{ct}} = \mathsf{Eval}(C, \mathsf{pk}_1, \ldots, \mathsf{pk}_n, \mathsf{ct}_1, \ldots, \mathsf{ct}_n)$. Then, with all but negligible probability in λ over the coins of Setup, Enc and KeyGen. With all but negligible probability in λ over the coins of Setup, Enc and $\mathsf{PartDec}$,*

$$\mathsf{FinDec}(\{p_i\}_{i\in S}) = \begin{cases} C(\mu_1, \ldots, \mu_n), & S = [n] \\ \perp & S \neq [n]. \end{cases}$$

Simulatability of Partial Decryption. *This requires that the following two games are indistinguishable.*

$\mathsf{Expt}^{\mathsf{real}}{}_{\mathcal{A}}(1^\lambda, 1^d)$**:**

1. *On input the security parameter 1^λ and depth d, the adversary $\mathcal{A}$ outputs, number of parties n a non-empty set $H \subseteq [n]$. Let $\bar{H} = [n] \setminus H$.*
2. *Run $\mathsf{Setup}(1^\lambda, 1^d) \to \mathsf{crs}$ and $(\mathsf{pk}_i, \mathsf{sk}_i) \leftarrow \mathsf{KeyGen}(\mathsf{crs}; \rho_i)$ for $i \in [n]$. Here, the randomness ρ_i is chosen by the adversary for $i \in \bar{H}$ and by the challenger otherwise.*
3. *Adversary is given $\mathsf{pk}_1, \ldots, \mathsf{pk}_n$ along with $\{\mathsf{sk}_i\}_{i\in\bar{H}}$.*
4. *The adversary samples messages μ_i for $i \in [n]$. It is given $\mathsf{ct}_i = \mathsf{Enc}(\mathsf{pk}_i, \mu_i; \rho'_i)$ for $i \in [n]$. Here the randomness ρ'_i is chosen by the adversary for $i \in \bar{H}$ and randomly otherwise. Then adversary can make adaptive upto arbitrary polynomial η queries of the following kind.*
5. *Adversary can choose a set $S_j \subseteq [n]$ such that $|S_j| \le N$ for $j \in [\eta]$. Then, it computes $\widehat{\mathsf{ct}}_j = \mathsf{Eval}(C_j, \{\mathsf{pk}_i\}_{i\in S_j}, \{\mathsf{ct}_i\}_{i\in S_j})$ where C_j is a depth d circuit chosen by the adversary. Adversary is given $p_{i,j} = \mathsf{PartDec}(\mathsf{sk}_i, \widehat{\mathsf{ct}}_j)$ for $i \in S_j \cap H$ and $j \in [\eta]$.*
6. *$\mathcal{A}$ outputs out. The output of the experiment is out.*

$\mathsf{Expt}^{\mathsf{ideal}}{}_{\mathcal{A},\mathsf{Sim}}(1^\lambda, 1^d)$**:**

1. *On input the security parameter 1^λ and depth d, the adversary $\mathcal{A}$ outputs, number of parties n a non-empty set $H \subseteq [n]$. Let $\bar{H} = [n] \setminus H$.*

2. Run $\mathsf{Setup}(1^\lambda, 1^d) \rightarrow \mathsf{crs}$ *and* $(\mathsf{pk}_i, \mathsf{sk}_i) \leftarrow \mathsf{KeyGen}(\mathsf{crs}; \rho_i)$ *for* $i \in [n]$*. Here, the randomness* ρ_i *is chosen by the adversary for* $i \in \bar{H}$ *and by the challenger otherwise.*

3. Adversary is given $\mathsf{pk}_1, \ldots, \mathsf{pk}_n$ *along with* $\{\mathsf{sk}_i\}_{i \in \bar{H}}$*.*

4. The adversary samples messages μ_i *for* $i \in [n]$*. It is given* $\mathsf{ct}_i = \mathsf{Enc}(\mathsf{pk}_i, \mu_i; \rho'_i)$ *for* $i \in [n]$*. Here the randomness* ρ'_i *is chosen by the adversary for* $i \in \bar{H}$ *and randomly otherwise. Then adversary can make adaptive upto arbitrary polynomial* η *queries of the following kind.*

5. Adversary can choose a set $S_j \subseteq [n]$ *such that* $|S_j| \leq N$ *for* $j \in [\eta]$*. Then, it computes* $\widehat{\mathsf{ct}}_j = \mathsf{Eval}(C_j, \{\mathsf{pk}_i\}_{i \in S_j}, \{\mathsf{ct}_i\}_{i \in S_j})$ *where* C_j *is a depth* d *circuit chosen by the adversary. Adversary is given* $p_{i,j} = \mathsf{Sim}(\{\mathsf{sk}_i\}_{i \in \bar{H}}, \widehat{\mathsf{ct}}_j, C_j(\{\mu_i\}_{i \in S_j}))$ *for* $i \in S_j \cap H$ *and* $j \in [\eta]$*.*

6. $\mathcal{A}$ *outputs* out*. The output of the experiment is* out*.*

We say that a TMFHE *scheme satisfies simulatability of partial decryption if there exists a (stateful) p.p.t simulator* Sim *such that the two experiments defined above are computationally indistinguishable.*

Remark E.1. We say that a TMFHE is in the plain model if crs consists of only 1^λ and 1^d. Otherwise, the scheme is said to be in the crs model.

E.1 Applications to MFHE

Theorem E.1. *Given an multi-key FHE scheme* MFHE *supporting* $N(\lambda)$ *number of keys and a* mrNISC *scheme for all circuits, there exists a* TMFHE *scheme supporting* $N(\lambda)$ *number of keys. If* MFHE *is in the plain model, then so is the constructed* TMFHE*.*

We now give various algorithms of the TMFHE scheme.
Ingredients. We use two ingredients.

1. A multi-key FHE scheme MFHE supporting upto $N(\lambda)$ keys.

2. An mrNISC scheme for all circuits.

We now describe our construction:

$\mathsf{Setup}(1^\lambda)$: The setup algorithm runs $\mathsf{MFHE.Setup}(1^\lambda, 1^d) \rightarrow \mathsf{crs}$. The algorithm outputs crs.

$\mathsf{KeyGen}(\mathsf{crs})$: The key generation algorithm runs $\mathsf{MFHE.KeyGen}(\mathsf{crs}) \rightarrow (\widetilde{\mathsf{pk}}_i, \widetilde{\mathsf{sk}}_i)$. Then it computes $(C_i, s_i) = \mathsf{Com}(1^\lambda, \widetilde{sk_i})$ using the commitment algorithm of the mrNISC scheme. Set $\mathsf{pk}_i = (\widetilde{\mathsf{pk}}_i, C_i)$ and $\mathsf{sk}_i = (\widetilde{\mathsf{sk}}_i, s_i)$.

$\mathsf{Enc}(\mathsf{pk}_i, \mu)$: The encryption algorithm parses $\mathsf{pk}_i = (\widetilde{\mathsf{pk}}_i, C_i)$. Then it computes and outputs $\mathsf{ct}_i = \mathsf{MFHE.Enc}(\widetilde{\mathsf{pk}}_i, \mu)$.

$\mathsf{Eval}(C, \{\mathsf{pk}_i\}_{i \in [n]}, \{\mathsf{ct}_i\}_{i \in [n]})$: The evaluation algorithm parses $\mathsf{pk}_i = (\widetilde{\mathsf{pk}}_i, C_i)$, then it computes and outputs $\widehat{\mathsf{ct}} = (\widehat{\mathsf{ct}}_1, \widehat{\mathsf{ct}}_2)$. Here, $\widehat{\mathsf{ct}}_1 = (C_1, \ldots, C_n)$ and $\widehat{\mathsf{ct}}_2 = \mathsf{MFHE.Eval}(C, \{\widetilde{\mathsf{pk}}_i\}_{i \in [n]}, \{\mathsf{ct}_i\}_{i \in [n]})$.

$\mathsf{PartDec}(\mathsf{sk}_i, \widehat{\mathsf{ct}})$: The partial decryption algorithm parses $\mathsf{sk}_i = (\widetilde{\mathsf{sk}}_i, s_i)$ and $\widehat{\mathsf{ct}} = (\widehat{\mathsf{ct}}_1, \widehat{\mathsf{ct}}_2)$. Parse $\widehat{\mathsf{ct}}_1 = (C_1, \ldots, C_n)$. Compute $p_i = (C_i, \alpha_i = \mathsf{mrNISC.Encode}(\mathsf{MFHE.Dec}(\cdot, \widehat{\mathsf{ct}}_2), \{C_j\}_{j \in [n]}, s_i))$.

$\mathsf{FinDec}(\{p_i\}_{i \in S})$: Parse $p_i = (C_i, \alpha_i)$. Compute and output $\mathsf{mrNISC.Eval}(\mathsf{MFHE.Dec}(\cdot, \widehat{\mathsf{ct}}_2), \{C_i\}_{i \in [S]}, \{\alpha_i\}_{i \in S})$.

We now argue the associated properties.

Correctness. The correctness of the scheme is immediate and can be verified by a simple inspection. Note that the key generation algorithm samples a key pair as in a MFHE scheme and then it uses mrNISC.Com algorithm of the mrNISC protocol to commit to the sampled secret key. Encryption algorithm just encrypts the message using the underlying public key of the MFHE scheme. The evaluation algorithm evaluates the ciphertext as in the MFHE scheme and then concatenates the result with set of mrNISC commitments of all the underlying secret keys (under whose public keys input ciphertexts were encrypted). The partial decryption then outputs the second message of the underlying mrNISC protocol to compute the decryption function on input the underlying secret keys $\{\widetilde{\mathsf{sk}}_i\}_{i \in [n]}$ hardwired with the evaluated ciphertext. The final decryption then outputs the result of running mrNISC.Eval on the first and the second messages obtained in the partial decryption. The correctness then follows from the correctness of MFHE and mrNISC scheme.

Security. The semantic security is immediate. When one does not learn partial decryptions the commitment C_i in the public key can be replaced with a commitment of 0 due to the security of mrNISC. Then, the security follows from the security of MFHE. Partial decryptions are simulatable due to the security of mrNISC. Given the secret keys $\{\mathsf{sk}_i\}_{i \in [n] \setminus i^*}$, and the outputs $\{\mu_j\}_{j \in Q}$, the partial decryptions can be simulated using the mrNISC.Sim algorithm.

Corollary E.1. *There exists a* TMFHE *scheme in the* crs *model for* NC^1 *circuits in the* crs *model for any polynomial number of keys assuming LWE with polynomial modulus and* PRF*s in* NC^1.

Proof. We recall that the multi-key FHE scheme in [CM15, MW16] satisfies the premise of the theorem. The transformation above gives us the required result. □

Corollary E.2. *Assuming the DSPR assumption, LWE with polynomial modulus and* PRF*'s in* NC^1*, there exist a levelled* TMFHE *scheme for apriori bounded polynomial number of keys in the plain model.*

Proof. To achieve this result, we use the multi-key FHE scheme of [LTV12] in the plain model. □

Corollary E.3. *Assuming LWE with subexponential modulus-to-noise ratio, there exists a (levelled)* TMFHE *scheme for arbitrary constant number of parties.*

Proof. To instantiate MFHE scheme, we observe that the nested (levelled) FHE scheme itself is a multi-key FHE scheme for a constant number of parties. Observe further that a mrNISC scheme can be constructed from LWE with subexponential ratio. Levelled FHE can be instantiated using [GSW13]. □