Towards Building Efficient SCALES Protocols

Anasuya Acharaya[1], Carmit Hazay[2], Vladimir Kolesnikov[3], and Manoj Prabhakaran[4]

[1] Aarhus University, acharya@cs.au.dk
[2] Bar-Ilan University, carmit.hazay@biu.ac.il
[3] Georgia Institute of Technology, kolesnikov@gatech.edu
[4] Indian Institute of Technology Bombay, mp@cse.iitb.ac.in

Abstract. SCALES (Small Clients And Larger Ephemeral Servers) (Acharya et al., TCC 2022, CRYPTO 2024) is a recently proposed model for MPC with several attractive features, including resilience to adaptive corruption. Known SCALES constructions, while offering reasonable asymptotics for large-scale MPC, incur high concrete costs both in computation and communication.

As our primary contribution, we dramatically improve both asymptotic and concrete costs of SCALES for permutation branching programs (PBP), a well-motivated practical model of computation. We achieve linear cost in program length, input size, and the security parameter. Our instantiations of the building blocks may be of independent interest.

Further, we present generic transformations to extend any semi-honestly secure SCALES protocol to achieve (1) guaranteed output delivery in the presence of mixed adversaries (that corrupt servers maliciously and clients semi-honestly) in the all-but-one corruption setting; and (2) protocols for computing general functionalities where each server's computation scales sub-linearly in the function size.

1 Introduction

The SCALES model for outsourced computation with ephemeral servers was introduced by Acharya et al. [AHKP22]. It aims to enable MPC-as-a-Crowd-sourced-Service, exploiting resources like blockchain nodes. The model considers clients with inputs who wish to participate in an MPC protocol and receive function outputs, but do not have the necessary computational resources. They employ the help of (a small number of) servers that are able to perform computations proportional to the function size. These servers are randomly selected from a large pool, a constant (dishonest majority) fraction of whom can be adaptively corrupted.

SCALES is inspired by the YOSO model [GHK+21], but avoids expensive target-anonymous channels between parties: all communication is over a public bulletin-board. On the other hand, it makes a distinction between clients and servers, and requires ephemerality only of the servers. This allows for remarkable performance improvements over the YOSO-model protocols.

SCALES protocol proceeds in *phases*, where in the first round of each phase, each client may post a message on the bulletin-board, followed by a sequence of ephemeral servers posting messages. SCALES guarantees security in the presence of an adversary corrupting all-but-one clients, and *in each phase*, all-but-one servers. [AHKP22] presents a semi-honest protocol with two phases; follow-up work [AHKP24] presents malicious UC-secure protocols in the CRS model with three phases. These works build on *public-key-based* variants of garbling schemes and oblivious transfer; further, the public-key encryption scheme employed [BHHO08] uses bit-by-bit encryption. The recent work of [HvdHJ25] presents an improved implementation of such garbling schemes, resulting in a linear-in-security-parameter reduction in communication complexity for semi-honest protocols compared to [AHKP22]. Nevertheless, all of these protocols incur very significant concrete costs, both in computation and communication.

In sum, SCALES is an attractive model to study because of its guaranteed resilience to adaptive corruptions, its efficiency over the competing models (YOSO and its flavors), and promising asymptotics that are not yet translated into concrete practical efficiency.

Branching Programs (BP) is a model of computation where a program's input is processed bit-by-bit to update the program state repeatedly. Initially designed as a theoretical tool for understanding computation and space complexity, BP effectively expresses many practical tasks. Permutation BPs (PBP) are BPs of a special form, where the transition function between layers is a permutation. PBPs can be easily constructed from BPs [Bar89]. We present PBPs in detail in Section 2.2.

BP/PBP Motivation. Since one of our technical contributions is a (concretely and asymptotically) efficient SCALES for PBP, we briefly discuss the practical usefulness of representing functions as BPs. While, in principle, all function families with $\mathbf{NC}^1$ circuits can be compiled into a PBP [Bar86], this need not be concretely efficient due to the length of the resulting PBP being exponential in the depth of the circuit. But underlying this theoretical result is the fact that many practically useful functions have concretely efficient PBPs, as evidenced by a trend in many practical applications to use BPs. In particular, under the name of Binary Decision Diagrams (BDD), BPs are widely employed in a variety of fields since the influential work of Bryant [Bry86]. They are commonly used in computer-aided design to represent functions that are then implemented in hardware circuits. They are also widely used to represent (transition) functions in the field of Formal Verification, for tasks such as symbolic model checking, verification of combinational logic, and verification of finite-state concurrent systems. Execution of a finite-state machine on bounded-length inputs can naturally be represented as a BP; this covers a large class of applications like substring search and pattern-matching algorithms. Another prominent example of BPs appears in Machine Learning, where models (functions used for classification) are represented as Decision Trees. The effectiveness of BPs as a computational model has also motivated implementing them securely [KJGB06, IP07, Lip09].

1.1 Our Contributions and Outline of Work

We lift theoretical feasibility results to practical regimes for an important class of tasks. Technical overview of our work is in Section 1.3. Our contributions are:

- **Concretely Efficient SCALES protocol for PBPs.** Our primary result is a semi-honest SCALES protocol for computing permutation branching programs (PBP). Examples of computations that have concretely efficient PBPs include parity, iterated integer addition, integer multiplication, and functions implemented by decision diagrams and finite automata (see Section 1). More generally, all functions in $\mathbf{NC}^1$ (i.e., functions computed by log-depth Boolean circuits) have polynomial-length width-5 PBPs [Bar86]. Our protocol incurs communication and computation complexity *linear* in the PBP size and the computational security parameter. Moreover, leveraging the techniques of [AHKP24], our protocol can be upgraded to achieve malicious security, incurring only a multiplicative overhead of κ in both client and server complexities. We achieve up to factor-$O(\kappa^2)$ improvement compared to prior work (see "Concrete Efficiency" below for more details).
  Our construction is given in Section 3. On our way to obtaining this result, we provide constructions for two ingredients introduced in [AHKP24], namely a rerandomizable garbling scheme (RGS) and an updatable oblivious transfer (UOT) protocol that allows the transferred messages to be homomorphically updated.
  - In Section 3.1, we present an RGS construction from the PBP representation of a function with input labels of size $O(\kappa)$ and a garbling size of $O(|f|\kappa)$, where $|f|$ is the size of the PBP. The security of this construction relies on the hardness of the DDH assumption.
  - In Section 3.2, we construct a UOT protocol that is compatible with our RGS construction and achieves $O(\kappa)$ communication complexity per message (improving over previous UOT of [GHV10, AHKP24] by factor κ). Our construction is based on a linearly homomorphic encryption (LHE) scheme (which can in turn be instantiated using DDH over class groups). The protocol supports affine update functions over the sender's inputs, which are group elements, and is more general than those in prior work that support only permutations over bit-strings.
    We further show that the existence of a UOT protocol as we require and an LHE scheme are, in fact, equivalent.

– **Sub-linear Servers:** All SCALES protocols in prior work have the complexity of each ephemeral server growing linearly with the size of the function to be computed. In Section 4.1 we present a transformation from such a SCALES protocol (semi-honest or malicious) to one where each server's computation scales sub-linearly in the function size. This improvement comes at the cost of increasing the client complexity and number of phases in the protocol in proportion to the reduction of the servers' complexity.
– **SCALES with Guaranteed Output Delivery.** SCALES protocols cannot achieve security with guaranteed output delivery (GOD) against a malicious adversary, as there is no honest majority in the SCALES corruption model. In Section 4.2, we present a SCALES protocol that achieves security with GOD in the presence of a mixed adversary that maliciously corrupts all but one of the servers in each phase but only semi-honestly corrupts (any number of) the clients.

Construction	Adversary Type	Server Complexity	Client Complexity	Setup
[AHKP22]	semi-honest	$O(\|\mathbf{C}\|\kappa^3)$	$O(d\kappa^3)$	Bulletin-Board
[AHKP24]	malicious	$O(\|\mathbf{C}\|\kappa^4)$ $O(d\|\mathbf{C}\|\kappa^4)$	$O(\kappa^2)$	CRS, Bulletin-Board
SCALES using RGS in [HvdHJ25]	semi-honest	$O(\|\mathbf{C}\|\kappa^2)$	$O(d\kappa^2)$	Bulletin-Board
this work	semi-honest	$O(\|\mathbf{PBP}\|\kappa)$	$O(\kappa)$	Bulletin-Board
this work	malicious	$O(\|\mathbf{PBP}\|\kappa^2)$	$O(\kappa^2)$	CRS, Bulletin-Board

Table 1 Comparison between SCALES protocols in previous work and a summary of their communication complexities. In the above, κ denotes the computational security parameter, $|\mathbf{C}|$ denotes the number of gates in the Boolean circuit being evaluated, $|\mathbf{PBP}|$ denotes the size (width $\times$ length) of the PBP being evaluated and d denotes the number of servers in each phase.

Concrete Efficiency. Our SCALES protocol for PBP achieves, for the first time, concretely efficient communication and computation costs for all the parties in a SCALES protocol, while keeping each client's complexity linear in the security parameter κ and the size of its private input: in the semi-honest setting, we bring the server's communication complexity down to $2(2w+1)\kappa\ell + \kappa\mathsf{m}$ bits, where m denotes the total input size and w, ℓ are the width and length of the PBP. This results in a factor-$O(\kappa^2)$ improvement over [AHKP22] and a factor-$O(\kappa)$ improvement over [HvdHJ25] (who build RGS on concretely more expensive pairing-based encryption), for the function class of PBPs. We refer the reader to Table 1 for a detailed comparison with related work.

1.2 Additional Related Work

We discuss most relevant prior work in Section 1.1 in explaining our contributions. Here we review additional related work. Over the last few years, the question of practical large scale MPC has led to several approaches [BGG+20, GHK+21, GHM+21, GMPS21, CGG+21, RS22, AHKP22, KRY22, BDO23]. We discuss some of the notable models below.

The YOSO model. [BGG+20, GHK+21, GHM+21] introduced the YOSO model (You Only Speak Once). The model mitigates the threat of adaptive corruptions by having a committee compute the function in a way that it is not known in advance who will be chosen as an active participant (the identities of participating parties are only revealed once they send their message; they never speak again).

YOSO protocols proceed in rounds where in each round a fresh committee of parties (called roles) are designated to communicate. The YOSO model is widely studied; the original works presented an information-theoretic and a computationally secure protocol achieving GOD in the presence of a minority corruption in each committee. Follow-up works [KRY22, BDO23, EMP25] improve upon these to present more efficient computationally secure protocols. Although inspired by the YOSO model, the SCALES model [AHKP22] is incomparable as it allows stateful input providers (clients) to post more than one message, but does not allow the stateless computation servers to send target anonymous messages into the future.

PASSO. Inspired by YOSO and [NRO22], [LMR+24,LMR+25] present the PASSO (PArties Speak Sequentially Once) model. Here parties can speak once, in sequence. [LMR+24, LMR+25] are not for generic MPC; they present protocols for computing randomized functions with no inputs, specifically for distributed randomness generation.

Fluid MPC. Motivated from the need to perform a continuous ongoing computation using MPC (e.g. maintaining a blockchain), [CGG+21] introduce the Fluid MPC model. Like YOSO, protocols in this model proceed in rounds where each round of communication is carried out by a committee of parties. However, parties with limited resources need not stay for too many consecutive rounds. *Maximally* fluid protocols would require each party only contributing to one round of communication. Unlike YOSO, these protocols do not tolerate adaptive corruption and security holds as long as the corruption within each committee does not exceed a pre-determined threshold. In this model, as a feasibility result, [CGG+21] presents maximally fluid protocols for the honest majority setting and follow-up works present protocols in the dishonest majority setting [RS22] and protocols that are communication efficient [BEP23].

Layered MPC. Abstracting the salient features from the models above, [DDG+23] model MPC over layered graphs. In this layered MPC model, parties are grouped into committees and in each ith round of the protocol parties in the ith committee can send messages (over point-to-point and broadcast channels) only to parties in the very next committee. [DDG+23] present an information-theoretically secure protocol in this model that tolerates $t < \frac{n}{3}$ static corruption in each committee. [DKLN24] presents a statistically secure protocol tolerating $t < \frac{n}{2}$ corruption in each committee.

Blockchains enable non-interactive MPC. [GMPS21] presents blockchain-assisted MPC where, roughly, each party garbles all their next-message functions and posts them on the blockchain, achieving speak-once property. They then rely on the liveness of the blockchain protocol to evaluate these and derive the output. Their protocol achieves GOD, with the GOD property of their final protocol being derived from the required GOD property of the underlying malicious secure MPC protocol.

In contrast, we compile a GOD protocol from a semi-honest protocol. Further, unlike [GMPS21], the SCALES client's work is independent of function size.

1.3 Technical Overview

SCALES Review. The SCALES model of computation [AHKP22] involves (1) light-weight input clients with computation complexity independent of the total function and input size, and (2) stateless servers with computation complexity polynomial in function and input size. The model disallows point-to-point communication, and all communication is over a public bulletin-board. A SCALES protocol proceeds in phases, where each phase begins with all clients posting a single message. In subsequent rounds of the phase, an ephemeral server is (randomly) selected in each round to advance the computation by posting one message. This setting ensures security against all-but-one initially corrupt servers in each phase, while tolerating any number of corrupt clients.

Our starting point is the framework of the SCALES protocol from [AHKP24] and we recall a semi-honest SCALES protocol implicit in [AHKP24] [5] with two phases. In the first phase, one server produces a garbled

[5] While the focus of [AHKP24] was to obtain security against malicious corruption, compared to [AHKP22] it has the additional advantage that the complexity of the clients is independent of the number of servers in each phase.

circuit (using an RGS) which is then rerandomized by the remaining servers in that phase; in the second phase, one server evaluates the final garbled circuit. In addition to RGS, it uses an *updatable* OT protocol (UOT). Informally, this is a variant of a 2-round OT protocol augmented with an extra roles of "updaters." An updater has as input a function that can alter the sender's inputs into a different value in the same domain. The SCALES protocol proceeds as follows:

- Each client creates and posts a receiver's message of a UOT protocol using its input bit as the choice bit.
- The first server garbles the function and uses the labels from the input encoding information as the input of the UOT sender.
- Every successive server rerandomizes the latest garbling posted and acts as an updater in the UOT protocol that transforms the sender's message into one that contains labels compatible with the new garbling.
- Starting the second phase, each client decodes the last UOT message corresponding to its inputs and posts the input labels for the final garbling.
- A server evaluates this garbling and posts the function output.

Although the works in [AHKP22, AHKP24] present a modular approach to designing secure SCALES protocols, the concrete instantiations of these protocols are prohibitively expensive for larger functions. These costs stem from the RGS instantiation that uses gate-by-gate garbling of a Boolean circuit using an encryption scheme that is homomorphic in both the key and message spaces. Further, both the key and message spaces need to contain the space of garbling wire labels and support the same homomorphic operations. As a result, known instantiations of this encryption scheme incur $O(\kappa^3)$ (assuming DDH) or $O(\kappa^2)$ (assuming pairings [HvdHJ25]) bits of communication cost per ciphertext.

Semi-Honest Secure SCALES Protocol. As our first contribution, we construct a more efficient semi-honest secure SCALES protocol for PBP computations. In this protocol, the complexity of each server is linear in the PBP size and the security parameter, and that of each client is linear in its own input size and the security parameter. Similarly to [AHKP24], we approach this using RGS and UOT protocols. However, we instantiate these building blocks differently: instead of using an RGS based on a Boolean circuit, we construct an RGS starting from the PBP, instantiating it such that the garbling size is linear in the size of the function representation and the security parameter. The corresponding UOT protocol is instantiated such that each of the receiver's, sender's and updater's message size is linear in the security parameter. Details can be found in Section 3. We explain the constructions of our building blocks next.

Rerandomizable Garbling for PBPs. We present an RGS that can garble any function represented as a PBP. Since all functions in $\mathbf{NC}^1$ admit polynomial-length constant-width PBPs [Bar86], our garbling scheme captures the function class $\mathbf{NC}^1$. Our function garblings have size $O(|f|\kappa)$, where $|f|$ is the size of the PBP. We garble such functions in two stages:

1. Starting from a PBP f, we create an information-theoretically secure RGS with garbling and input labels both of size $O|f|$. When used within the SCALES framework, this results in server complexity $O(|f|)$ for garbling and rerandomizing, and client complexity $O(|f|)$ for decoding and posting the labels, rendering it unsuitable for light-weight clients.
2. We compress this information-theoretic RGS into one with computational security, reducing the input label size to $O(\kappa)$. This transformation increases the server's complexity to $O(|f|\kappa)$ but reduces the client's complexity to $O(\kappa)$, making the protocol efficient even for resource-constrained clients.

Much like prior work in arithmetic garbling [AIK11], our construction effectively decouples the information-theoretic garbling from the computational label compression. Both stages are explained in detail next.

Information-Theoretic (IT) RGS. We refer to an RGS as IT if corresponding random variables defining its privacy properties are identically distributed, and not merely computationally indistinguishable. In Section 3.1, we present a construction of an IT Explainable RGS for any function that can be represented by a PBP.

This representation consists of $|f|$ pairs of permutations, where the evaluation involves selecting one of each pair according to an input bit, and composing them. The function output is zero if the result is the identity permutation and one otherwise.

Kilian [Kil88] pointed out a technique for rerandomizing PBPs information-theoretically: for every consecutive pair of permutations in the PBP, sample a random permutation and compose it and its inverse with those in the PBP in such a way that they cancel each other during the evaluation. Both our garbling and rerandomizing algorithms use this technique and we also show how the randomness is composed – yielding an information-theoretic explainable RGS.

Note that for the class of $\mathbf{NC}^1$ functions, (non-rerandomizable) IT garbling schemes exist, where the garbling algorithm operates on the Boolean formula representation of the function [Kol05]. Since every function in $\mathbf{NC}^1$ admits a polynomial-length PBP representation [Bar86], our result effectively bridges the gap between traditional garbling schemes and rerandomizable garbling schemes (RGS) in terms of the minimal assumptions required for $\mathbf{NC}^1$ function classes. For function classes beyond $\mathbf{NC}^1$, the weakest assumption for standard garbling schemes rely on symmetric-key primitives [Yao86], whereas the best-known constructions for RGS require public-key primitives [GHV10].

In the IT RGS discussed above, the garbling material is empty; all function information is encoded within the input labels (sets of permutations selected according to the input bits). In the following, we describe a transform that yields an RGS where the size of the input labels is independent of the function size.

RGS with Constant Size Input Labels. In the computational setting, we describe a simple yet powerful transformation that converts an RGS—such as the one described earlier—into a scheme with input labels of constant (independent of circuit) size, proportional to the computational security parameter κ. This transformation leverages a strong key and message homomorphic encryption (strong KMHE) scheme [AHKP22], where the message space is compatible with the original RGS input labels. The core idea is to generate a secret key for each input label and encrypt the corresponding permutation (selected based on the input bit's semantic value) under this key. These ciphertexts form the new function garbling, while the set of keys, each of size κ, serves as the compact input labels. To rerandomize the garbling, one applies homomorphic operations to both the ciphertexts (to rerandomize the garbling) and the keys (to refresh the input encodings), ensuring the rerandomized scheme maintains correctness and security while drastically reducing input label size.

The above transform works for converting any RGS into one with short labels. We can further optimize it when the underlying RGS is IT as described above: we show that strong KMHE can be instantiated using a variant of ElGamal encryption, yielding a ciphertext size of $2w + 1$ group elements per plaintext permutation of width w. Informally, we take advantage of the fact that five distinct elements of a DDH-hard group can be used to encode all possible message permutations. This is done by representing each row of the permutation as a distinct group element and encrypting each such element separately. The result is a pair of group elements for encrypting each of the w rows of the permutation, along with one group element to represent the public key. The permutations in the message space function family are then applied by rerandomizing each of the five ciphertexts and subsequently permuting them.

Updatable OT protocol. In the above, we instantiated an RGS in which the input labels are integers in a prime-order group, and the transformations applied to them—produced by the rerandomizing algorithm—are affine functions over this group. To incorporate this RGS into our SCALES protocol, we require a UOT protocol where the sender's inputs are drawn from the same label space, and the updater's inputs support such affine transformations. We construct such a UOT protocol using a linearly homomorphic encryption (LHE) scheme. Notably, this protocol achieves minimal communication: both the sender and each updater send only a single message, and each message consists of just one ciphertext.

The UOT protocol closely follows the structure of the classic OT protocol from [AIR01]. The receiver generates a key pair for the underlying LHE scheme and uses the public key to encrypt its choice bit b. This ciphertext serves as the receiver's message. The sender then homomorphically evaluates this ciphertext using its two input strings s_0 and s_1, resulting in a ciphertext that encrypts s_b.This is the sender's message. Next, an updater with an affine transformation σ can homomorphically apply σ to this ciphertext, thereby

producing an encryption of $\sigma(s_b)$. Finally, the receiver decrypts this ciphertext using its secret key to obtain the UOT output.

Note that all homomorphic operations performed by the sender and updater in the UOT protocol are linear. In fact, we show that this UOT protocol is equivalent to linearly homomorphic encryption (LHE) by demonstrating how to construct an LHE scheme using the next-message functions of the UOT protocol in a fully black-box manner. For our instantiation, we employ the LHE scheme from [CL15], which is based on the hardness of the Decisional Diffie-Hellman (DDH) problem over prime-order class groups.[6] Finally, we note that the current protocol guarantees only semi-honest security, but it can be strengthened to withstand malicious adversaries using non-interactive zero-knowledge (NIZK) proofs, as detailed in Section 3.2.

SCALES with sub-linear server computation. We present general SCALES protocols where each participating server performs computation sub-linear in the function size to be computed. This improves over the state of the art [AHKP22, AHKP24] where each server's computation is linearly proportional to the function size. Our high-level approach follows from re-writing the function f to be computed in terms of sub-function layers: $f = f_n \circ \cdots \circ f_1$. Here f_1 is the first sub-function accepting the input x of f, and f_n is the last sub-function that outputs $f(x)$. Then, each of these layers is computed using SCALES protocols that are composed in a black-box way. Let each $f_i : \{0,1\}^{\ell_{i-1}} \to \{0,1\}^{\ell_i}$, $\ell_{\max} = \max_{i\in[n]} \ell_i$ be the width of f, and $\mathsf{PRF} : \{0,1\}^\kappa \times [n] \to \{0,1\}^{\ell_{\max}}$ be a low-depth pseudo-random function (in the same sub-function family)

- For $\mathsf{m} = \ell_0$, let $F_1 : \{0,1\}^{\ell_0} \times \{0,1\}^{\mathsf{m}\kappa} \to \{0,1\}^{\ell_1}$ be a function that computes:

$$\begin{aligned} \mathsf{input}: &\quad \{x_i \in \{0,1\}, s_i \in \{0,1\}^\kappa\}_{i\in[\mathsf{m}]} \\ \mathsf{output}: &\quad y_1 = f_1(x) \oplus \mathsf{PRF}(s,1) \quad \text{where} \quad s = \bigoplus_{i\in[\mathsf{m}]} s_i \end{aligned}$$

- For each $j \in [n-1]$ where $i > 1$, let $F_j : \{0,1\}^{\mathsf{m}\kappa} \to \{0,1\}^{\ell_j}$ be a function that computes:

$$\begin{aligned} \mathsf{input}: &\quad \{s_i \in \{0,1\}^\kappa\}_{i\in[\mathsf{m}]} \\ \mathsf{output}: &\quad y_j = f_j(y_{j-1} \oplus \mathsf{PRF}(s, j-1)) \oplus \mathsf{PRF}(s,j) \quad \text{where} \quad s = \bigoplus_{i\in[\mathsf{m}]} s_i \end{aligned}$$

- Let $F_n : \{0,1\}^{\mathsf{m}\kappa} \to \{0,1\}^{\ell_n}$ be a function that computes:

$$\begin{aligned} \mathsf{input}: &\quad \{s_i \in \{0,1\}^\kappa\}_{i\in[\mathsf{m}]} \\ \mathsf{output}: &\quad y = f_n(y_{n-1} \oplus \mathsf{PRF}(s, n-1)) \quad \text{where} \quad s = \bigoplus_{i\in[\mathsf{m}]} s_i \end{aligned}$$

At the high-level, the protocol starts by evaluating the first sub-function (where each client C_i has inputs (x_i, s_i)) and publicly posting its masked output. Every successive protocol uses the output of the previous protocol as a hard-coded input and computes a masked output of the next sub-function. The last protocol computes the output in the clear. In both the semi-honest and malicious settings, this protocol can benefit from the following optimizations: (1) all sub-protocols can share a common clients' first message for encoding the function inputs and PRF key shares, and (2) the sub-protocols can be instantiated with client computation independent of the number of servers in each phase. We refer the reader to Section 4.1 for details.

SCALES with GOD for mixed adversaries. We outline a general framework for transforming any semi-honestly secure SCALES protocol into one that achieves guaranteed output delivery (GOD) in the presence of a mixed adversary model. In this setting, all-but-one server in each phase may be maliciously corrupted, while any number of clients may be semi-honestly corrupted. At the heart of our compilation are two key observations: (1) it suffices for the malicious corrupted servers to use general-purpose non-interactive

[6] Alternative LHE instantiations based on LWE, Ring-LWE, and LPN are also viable, though they lose decryption correctness when too many updates are applied.

zero-knowledge (NIZK) proofs to demonstrate that they executed the semi-honest protocol correctly, and (2) since servers in a SCALES protocol are stateless and act sequentially, any misbehaving server can be detected in real-time and its contribution disregarded, allowing the next server to resume from the last valid message. To support this recovery mechanism, it is essential that all parties, including resource-constrained clients, can verify the correctness of the semi-honest computation chain.

In Section 4.2, we present two approaches to enable this. The first uses zk-SNARKs to prove the correctness of server-side computation, allowing lightweight clients to verify the proofs efficiently. The second approach leverages NIZKs by structuring the statement as a conjunction of multiple atomic sub-statements.[7] Each sub-statement is proven independently, and although verification is not succinct, the verifier can identify the index of the first failing sub-proof, enabling even lightweight clients to isolate and verify erroneous behavior.

Our construction uses generic ZK. We remind the reader that [AHKP24] provide a much more efficient tailored construction, which *does not* achieve GOD.

2 Preliminaries

We use κ to represent the security parameter and λ for the empty string.

Definition 1. *Two probability ensembles $X = \{X_n\}_{n\in\mathbb{N}}$ and $Y = \{Y_n\}_{n\in\mathbb{N}}$ defined over a finite domain D are **statistically indistinguishable**, denoted $X \overset{s}{\approx} Y$, if every positive polynomial $p(\cdot)$ and all sufficiently large n's,*

$$\Delta(X_n, Y_n) < \frac{1}{p(n)}$$

where,

$$\Delta(X_n, Y_n) = \frac{1}{2} \cdot \sum_{\alpha\in D} |\Pr[X_n = \alpha] - \Pr[Y_n = \alpha]|$$

Definition 2. *Two probability ensembles $X = \{X_n\}_{n\in\mathbb{N}}$ and $Y = \{Y_n\}_{n\in\mathbb{N}}$ are **computationally indistinguishable**, denoted $X \overset{c}{\approx} Y$, if for every PPT distinguisher D, every positive polynomial $p(\cdot)$ and all sufficiently large n's,*

$$|\Pr[D(X_n, 1^n) = 1] - \Pr[D(Y_n, 1^n) = 1]| < \frac{1}{p(n)}$$

2.1 Rerandomizable Garbling Schemes

We re-state the definition of projective Rerandomizable Garbling Schemes [AHKP22].

Definition 3. *A **Projective Rerandomizable Garbling Scheme** for function family $\mathcal{F}$ with input domain $\{0,1\}^{\mathsf{m}}$, and a leakage function $\phi : \mathcal{F} \to \{0,1\}^*$, is a tuple $\mathsf{GS} = (\mathsf{Gb}, \mathsf{Rerand}, \mathsf{En}, \mathsf{Ev})$ of PPT algorithms such that the following is satisfied. Below R_{Gb} and R_{Rerand} stand for the spaces of random-tapes for Gb and Rerand respectively.*

- ***Correctness**: For every $f \in \mathcal{F}$ and input $x \in \{0,1\}^{\mathsf{m}}$,*

$$\begin{aligned}
&\Pr[y = f(x) \ : (F, e) \leftarrow \mathsf{Gb}(f),\ X = \mathsf{En}(e, x),\ y = \mathsf{Ev}(F, X) &&] = 1\\
&\Pr[y = f(x) \ : (F, e) \leftarrow \mathsf{Gb}(f), \quad (F', \pi_{\mathsf{En}}) \leftarrow \mathsf{Rerand}(F),\\
&\qquad\qquad X' = \mathsf{En}(\pi_{\mathsf{En}}(e), x),\ y = \mathsf{Ev}(F', X') &&] = 1
\end{aligned}$$

- ***Privacy**: $\forall f_0, f_1 \in \mathcal{F}$ s.t. $\phi(f_0) = \phi(f_1)$, and $\forall x_0, x_1 \in \{0,1\}^{\mathsf{m}}$ s.t. $f_0(x_0) = f_1(x_1)$,*

$$\{F_0, X_0\}_{(F_0,e_0)\leftarrow\mathsf{Gb}(f_0), X_0=\mathsf{En}(e_0,x_0)} \overset{c}{\approx} \{F_1, X_1\}_{(F_1,e_1)\leftarrow\mathsf{Gb}(f_1), X_1=\mathsf{En}(e_1,x_1)}$$

[7] Generic NIZK proofs typically decompose the overall statement into a circuit whose gates are individually verified against the witness.

– Rerand-***Privacy***: *For every* $f \in \mathcal{F}$, $x \in \mathcal{X}$, *and* $r \in R_{\mathsf{Gb}}$,

$$\{r, F_0, X_0\}_{\substack{(F,e)\leftarrow \mathsf{Gb}(f;r),\\ (F_0,\pi_{\mathsf{En}})\leftarrow \mathsf{Rerand}(F),\ X_0=\mathsf{En}(\pi_{\mathsf{En}}(e),x)}} \overset{c}{\approx} \{r, F_1, X_1\}_{\substack{(F_1,e_1)\leftarrow \mathsf{Gb}(f),\\ X_1=\mathsf{En}(e_1,x)}}$$

(Note that (F_1, e_1) *is generated using fresh randomness independent of* r*.)*

– ***Randomness Composition***: *There exists a function* $\mathsf{Explain} : R_{\mathsf{Gb}} \times R_{\mathsf{Rerand}} \to R_{\mathsf{Gb}}$ *s.t.* $\forall f \in \mathcal{F}$, $\forall r \in R_{\mathsf{Gb}}, r' \in R_{\mathsf{Rerand}}$, *if* $(F, e) = \mathsf{Gb}(f; r)$, $(F', \pi) = \mathsf{Rerand}(F; r')$, *and* $r^* = \mathsf{Explain}(r, r')$, *then*

$$\mathsf{Gb}(f; r^*) = (F', \pi(e))$$

– ***Projective Encoding***: *The function* $\mathsf{En} : \mathcal{E} \times \{0,1\}^{\mathsf{m}} \to \mathcal{L}^{\mathsf{m}}$, *where* $\mathcal{L}$ *is the input encoding alphabet, is projective. That is:*

- $\forall x = \{x_i\}_{i\in[\mathsf{m}]}, x' = \{x'_i\}_{i\in[\mathsf{m}]} \in \{0,1\}^{\mathsf{m}}$, $\forall e \in \mathcal{E}$, $\{L_i\}_{i\in[\mathsf{m}]} = \mathsf{En}(e, x)$ *and* $\{L_i^{'}\}_{i\in[\mathsf{m}]} = \mathsf{En}(e, x')$ *such that* $\forall i \in [\mathsf{m}]$, *if* $x_i = x'_i$ *then* $L_i = L_i^{'}$.

Further, En *is rerandomizable. That is:*

- $\pi_{\mathsf{En}} = \{\sigma_i\}_{i\in[\mathsf{m}]}$ *produced by* Rerand *is such that* $\forall x \in \{0,1\}^{\mathsf{m}}$, $\forall e \in \mathcal{E}$, $\{L_i\}_{i\in[\mathsf{m}]} = \mathsf{En}(e, x)$ *and* $\{L_i^{'}\}_{i\in[\mathsf{m}]} = \mathsf{En}(\pi_{\mathsf{En}}(e), x)$ *where* $\sigma_i(L_i) = L_i^{'}$.

In order to use garbling schemes in a black-box way in MPC protocols the following equivalent privacy definition makes it easier to reduce the security of an MPC protocol to that of the scheme.

Definition 4. *Let* $\mathsf{GS} = (\mathsf{Gb}, \mathsf{En}, \mathsf{Ev})$ *be a projective GS for function family* $\mathcal{F}$ *with input domain* $\{0,1\}^{\mathsf{m}}$, *and a leakage function* $\phi : \mathcal{F} \to \{0,1\}^*$. *We say that* GS *satisfies* ***privacy by simulation*** *if there exists a PPT algorithm* Sim *such that* $\forall f \in \mathcal{F}, x \in \{0,1\}^{\mathsf{m}}$,

$$\{F, X\}_{(F,e)\leftarrow \mathsf{Gb}(f), X=\mathsf{En}(e,x)} \overset{c}{\approx} \{F, X\}_{(F,X)\leftarrow \mathsf{Sim}(1^\kappa, \phi(f), f(x))}$$

[AHKP22] contains a construction for a projective RGS that is based on garbled circuits and a strong Key and Message Homomorphic Encryption (strong KMHE).

Explainable RGS. A variant of RGS defined in [AHKP24], the definition of an Explainable RGS is as given in Definition 5.

Definition 5. *Let* $\mathsf{RGS} = (\mathsf{Gb}, \mathsf{Rerand}, \mathsf{En}, \mathsf{Ev})$ *be a projective RGS (Definition 3). Let* R *be the space of randomness for garbling and rerandomizing.* RGS *is* ***Explainable*** *if there exists a function* $\mathsf{Explain}(\cdot)$ *s.t. for all* $f \in \mathcal{F}$,

– ***Garbling Randomness Composition***: $\forall r, r' \in R$, *if* $F \leftarrow \mathsf{Gb}(f; r)$ *and* $F' \leftarrow \mathsf{Rerand}(F; r')$, *then* $r^* \leftarrow \mathsf{Explain}(r, r')$ *satisfies,*

$$\Pr[F' \leftarrow \mathsf{Gb}(f; r^*)] = 1$$

– ***Rerandomizing Randomness Composition***: $\forall r, r' \in R$, *such that* $F \leftarrow \mathsf{Rerand}(F^*; r)$ *and* $F' \leftarrow \mathsf{Rerand}(F; r')$, *it must hold that for* $r^* \leftarrow \mathsf{Explain}(r, r')$,

$$\Pr[F' \leftarrow \mathsf{Rerand}(F^*, r^*)] = 1$$

Explainable RGS finds applications in the construction of black-box public coin zero-knowledge proofs for correct garbling and rerandomizing [AHKP24].

2.2 Permutation Branching Programs

For a small constant $w \in \mathbb{N}$, a width-w permutation branching program (PBP) is a function representation for Boolean functions that uses a set of permutations over w elements. For a function with n-bit inputs, a length-k width-w PBP representation contains k pairs of such permutations. Roughly, evaluating this PBP involves choosing one permutation in each of the k positions, according to the values of the input bits, and then composing the resulting set of k permutations in order to derive the permutation that represents the function output. We restate the definition of a PBP from [Kil88]. It is well known that any Boolean function in $\mathbf{NC}^1$ with depth-$c\log(\mathsf{m})$ circuits for input length m can be computed by a width-5 PBP of polynomial length $O(\mathsf{m}^{2c})$ [Bar86]. In the introduction, we discussed possible use cases for the PBP representation; in Appendix A, to give a sense of PBP costs, we discuss a concrete PBP size for representing addition.

Definition 6. *For a constant w, a width-w **permutation branching program (PBP)** M of length k with input $x \in \{0,1\}^n$ is a tuple (Π, V) defined by,*

$$\Pi : [k] \times \{0,1\} \to S_w$$
$$V : [k] \to [n]$$

where S_w is the group of permutations on w elements. An evaluation of this program is defined as,

$$M(x_1, \ldots, x_n) = E(\Pi(1, x_{V(1)}) \circ \ldots \circ \Pi(k, x_{V(k)}))$$

where $E(y \in S_w)$ is equal to 0 if y is the identity permutation, and 1 otherwise.

Rerandomizing a PBP. [Kil88] also points out that a PBP as given above can be rerandomized. Let $\vec{R} = \{r_1, \ldots, r_{k-1}\}$ where each $r_i \in S_w$, and let r_0 and r_k be the identity permutation. Given a PBP $M = (\Pi, V)$, we define $M_R = (\Pi_R, V)$ where,

$$\forall i \in [k], j \in \{0,1\}, \Pi_R(i,j) = r_{i-1}^{-1} \Pi(i,j) r_i$$

Note that this PBP function representation and its rerandomizing does not involve any operations that rely on computational hardness assumptions. Looking ahead, we show in Section 3.1 that one can instantiate an information-theoretically secure explainable RGS using such PBPs. This is an object in which all the privacy properties require identical distributions (as opposed to them being computationally indistinguishable) and security must hold in the presence of computationally unbounded adversaries.

2.3 Linearly Homomorphic Encryption

We state the definition of a linearly homomorphic encryption scheme below.

Definition 7. *A **linearly homomorphic encryption scheme** (LHE) over a ring $\mathcal{M}$ is the set of PPT algorithms $\mathsf{LHE} = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec}, \mathsf{Eval})$ defined on a domain of public and private keys, and ciphertexts $\mathcal{PK}, \mathcal{SK}, \mathcal{C}$, in addition to the message space $\mathcal{M}$ (all indexed by an implicit security parameter κ) such that the following conditions hold:*

- ***Correctness**: $\forall m, b \in \mathcal{M}, \{m_i, a_i\}_{i\in[n]} \in \mathcal{M}^{2n}$,*

$$\Pr[\mathsf{Dec}(\mathsf{sk}, \mathsf{Enc}(\mathsf{pk}, m)) = m;\ (\mathsf{sk}, \mathsf{pk}) \leftarrow \mathsf{Gen}(1^\kappa)] = 1$$

$$\Pr[\mathsf{Dec}(\mathsf{sk}, \mathsf{Eval}(\{c_i\}_{i\in[n]}, \{a_i\}_{i\in[n]}, b)) = b + \Sigma_{i\in[n]} m_i \cdot a_i; \\ (\mathsf{sk}, \mathsf{pk}) \leftarrow \mathsf{Gen}(1^\kappa) \\ \{c_i \leftarrow \mathsf{Enc}(\mathsf{pk}, m_i)\}_{i\in[n]}] = 1$$

and for a negligible function ν,

$$\Pr[\perp = \mathsf{Dec}(\mathsf{sk}', \mathsf{Enc}(\mathsf{pk}, m)); (\mathsf{sk}, \mathsf{pk}) \leftarrow \mathsf{Gen}(1^\kappa), (\mathsf{sk}', \mathsf{pk}') \leftarrow \mathsf{Gen}(1^\kappa)] > 1 - \nu(\kappa)$$

- ***CPA Security***: *$\forall$ PPT adversary $\mathcal{A}$, the advantage $\Pr[b' = b] \leq \frac{1}{2} + \nu(\kappa)$ for a negligible function ν in the following experiment (κ being an implicit input to C and $\mathcal{A}$):*
  1. *C samples a uniform random bit $b \leftarrow \{0,1\}$ and keys $(\mathsf{sk}, \mathsf{pk}) \leftarrow \mathsf{Gen}(1^\kappa)$.*
  2. *$\mathcal{A}$ receives pk and for as many times as $\mathcal{A}$ wants:*
     - *$\mathcal{A}$ produces arbitrary $m_0, m_1 \in \mathcal{M}$ and sends them to C.*
     - *C sends $c_b = \mathsf{Enc}(\mathsf{pk}, m_b)$ to $\mathcal{A}$.*
  3. *$\mathcal{A}$ outputs b'.*
- ***Privacy***: *$\forall (\mathsf{sk}, \mathsf{pk}) \leftarrow \mathsf{Gen}(1^\kappa), b \in \mathcal{M}, \{m_i, a_i\}_{i\in[n]} \in \mathcal{M}^{2n}$,*

$$\{\mathsf{Eval}(\{c_i\}_{i\in[n]}, \{a_i\}_{i\in[n]}, b)\}_{\{c_i \leftarrow \mathsf{Enc}(\mathsf{pk}, m_i)\}_{i\in[n]}} \equiv \{\mathsf{Enc}(\mathsf{pk}, b + \Sigma_{i\in[n]} m_i \cdot a_i)\}$$

for all random tapes in R_{Enc} for Enc and R_{Eval} for Eval.[8]

We use such an LHE scheme to instantiate the updatable OT protocol in Section 3.2, and additionally show that such an updatable OT protocol is equivalent to this notion of LHE.

Instantiating LHE. We outline the linearly homomorphic encryption scheme (LHE) based on the DDH hardness assumption over class groups [CL15].

Construction 1 *Let $\mathbb{G}$ be an abelian group with unknown order that has a cyclic group $\mathbb{F}$ of prime order q with generator f (where computing discrete logarithm is easy and computable by a PPT algorithm $\mathsf{DL}_\mathbb{F}$). The order of $\mathbb{G}$ is qs here s is unknown and co-prime to q. An upper-bound $s' > s$ is known. Let $h = x^q$ for $x \leftarrow \mathbb{G}$. Set $g = fh$ and let $\mathbb{G}^*$ be the cyclic sub-group generated by g. Then for $\mathbb{H} = \{x^q\}_{x\in\mathbb{G}^*}$, h is a generator for $\mathbb{H}$ and $\mathbb{G}^* \simeq \mathbb{F} \times \mathbb{H}$.*

$$\mathsf{Setup}(1^\kappa, q) \rightarrow \mathsf{pp} = (\mathbb{G}, s', g, h, f, \mathsf{DL}_\mathbb{F})$$

Then the algorithms in $\mathsf{KMH} = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec}, \mathsf{Eval})$ are initialized as follows:

- *$\mathcal{SK} = \mathbb{Z}_s$ is the secret key space, and let $\mathcal{SK}' = \mathbb{Z}_{s'}$.*
- *$\mathcal{PK} = \mathbb{H}$ is the public key space.*
- *$\mathcal{C} = \mathbb{H} \times \mathbb{G}^*$ is the ciphertext space.*
- *$\mathcal{M} = \mathbb{Z}_q$ is the message space.*
- *$\mathcal{F}_\mathcal{M} = \{f_{a,b}(m) = am + b \mod q\}_{a,b\in\mathbb{Z}_q}$ is the message space function family.*

These are used to instantiate the following functions:

1. *$(\mathsf{sk}, \mathsf{pk}) \leftarrow \mathsf{Gen}(1^\kappa)$:*
   - *sample $\mathsf{sk} \leftarrow \mathcal{SK}'$ s.t. $\mathsf{sk} \in \mathcal{SK}$ with overwhelming probability*
   - *compute $\mathsf{pk} = h^{\mathsf{sk}} \in \mathbb{H}$*
   - *output - $(\mathsf{sk}, \mathsf{pk})$*
2. *$c \leftarrow \mathsf{Enc}(\mathsf{pk}, m)$:*
   - *input - key $\mathsf{pk} \in \mathbb{H}$ and message $m \in \mathbb{Z}_q$*
   - *sample $r \leftarrow \mathcal{SK}'$ s.t. $r \in \mathcal{SK}$ with overwhelming probability*
   - *compute $c = (h^r, \mathsf{pk}^r \cdot f^m)$*
   - *output - c*
3. *$m = \mathsf{Dec}(\mathsf{sk}, c)$:*

[8] We define perfect privacy for simplicity (and since the instantiation we consider realizes it), but a computational notion would suffice.

- *input - key* $\mathsf{sk} \in \mathbb{Z}_s$ *and ciphertext* $c = (c_1, c_2)$
- *compute* $m = \mathsf{DL}_{\mathbb{F}}(\frac{c_2}{c_1^{\mathsf{sk}}})$
- *output - m*

4. $c' \leftarrow \mathsf{Eval}(\mathsf{pk}, c, a, b)$*:*
   - *input - parse* $f_{a,b} \in \mathcal{F}_{\mathcal{M}}$ *from* $a, b \in \mathbb{Z}_q$
   - *input - ciphertext* $c = (c_1, c_2)$
   - *sample* $r \leftarrow \mathcal{SK}'$ *s.t.* $r \in \mathcal{SK}$ *with overwhelming probability*
   - *compute* $c' = ((c_1 \cdot h^r)^a, (c_2 \cdot \mathsf{pk}^r)^a \cdot f^b)$
   - *output - c'*

2.4 Strong Key and Message Homomorphic Encryption

We re-state the definition of a strong Key and Message Homomorphic Encryption (strong KMHE) scheme as in [AHKP22].

Definition 8. *A* ***strong key-and-message homomorphic encryption scheme*** *(strong KMHE) is the set of PPT algorithms* $\mathsf{KMH} = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec}, \mathsf{Eval})$ *defined on domains of (private) keys, messages and ciphertexts* $\mathcal{K}, \mathcal{M}, \mathcal{C}$*, a key transformation family* $\mathcal{F}_{key}$*, and a message transformation family* $\mathcal{F}_{msg}$ *(all indexed by an implicit security parameter* κ*) such that the following conditions hold:*

- ***Correctness****:* $\forall m \in \mathcal{M}, k \in \mathcal{K}$*,*

$$\Pr[k \leftarrow \mathsf{Gen}(1^\kappa); \mathsf{Dec}(k, \mathsf{Enc}(k, m)) = m] = 1$$

  and for a negligible function ν*,*

$$\Pr[\perp = \mathsf{Dec}(k', \mathsf{Enc}(k, m)) \ ; \ k, k' \leftarrow \mathsf{Gen}(1^\kappa)] > 1 - \nu(\kappa)$$

- ***KMH Correctness****:* $\forall k \leftarrow \mathsf{Gen}(1^\kappa), m \in \mathcal{M}, s \in R_{\mathsf{Enc}}, f \in \mathcal{F}_{key}, g \in \mathcal{F}_{msg}$*,*

$$\{k, m, s, c\}_{c \leftarrow \mathsf{Eval}(\mathsf{Enc}(k,m;s), f, g)} \equiv \{k, m, s, c\}_{c \leftarrow \mathsf{Enc}(f(k), g(m))}$$

  where R_{Enc} *is the space of all random tapes for* Enc*.*
- ***Key Privacy****:* $\forall k, k' \leftarrow \mathsf{Gen}(1^\kappa)$*,* $f \in \mathcal{F}_{key}$*,*

$$\{k, f(k)\} \overset{\text{s}}{\approx} \{k, k'\}$$

- ***CPA Security****:* $\forall$ *PPT adversary* $\mathcal{A}$*, the advantage* $\Pr[b' = b] \leq \frac{1}{2} + \nu(\kappa)$ *for a negligible function* ν *in the following experiment (*κ *being an implicit input to* C *and* $\mathcal{A}$*):*
  1. C *samples a uniform random bit* $b \leftarrow \{0, 1\}$*.*
  2. *For as many times as* $\mathcal{A}$ *wants:*
     - $\mathcal{A}$ *produces arbitrary* $m_0, m_1 \in \mathcal{M}$ *and sends them to* C*.*
     - C *samples a key* $k \leftarrow \mathsf{Gen}(1^\kappa)$ *and sends* $c_b = \mathsf{Enc}(k, m_b)$ *to* $\mathcal{A}$*.*
  3. $\mathcal{A}$ *outputs* b'*.*
- ***KMH Privacy****:* $\forall$ *PPT adversary* $\mathcal{A}$*, the advantage* $\Pr[b' = b] \leq \frac{1}{2} + \nu(\kappa)$ *for a negligible function* ν *in the following experiment (*κ *being an implicit input to* C *and* $\mathcal{A}$*):*
  1. $\mathcal{A}$ *creates* k_0, k_1 *that are in support of* $\mathsf{Gen}(1^\kappa)$ *and sends them to* C*.*
  2. C *samples a uniform random bit* $b \leftarrow \{0, 1\}$*, key* $k' \leftarrow \mathsf{Gen}(1^\kappa)$*, and* $f \leftarrow \mathcal{F}_{key}$*. It sends* $f(k_1)$ *to* $\mathcal{A}$*.*
  3. *For as many times as* $\mathcal{A}$ *wants:*
     - $\mathcal{A}$ *creates* $m, m' \in \mathcal{M}$*,* $g \in \mathcal{F}_{msg}$*, and computes* $c \leftarrow \mathsf{Enc}(k_0, m)$*. It sends* (c, g, m') *to* C*.*
     - C *sends* c_b *to* $\mathcal{A}$*, where* $c_0 \leftarrow \mathsf{Eval}(c, f, g)$ *and* $c_1 \leftarrow \mathsf{Enc}(k', m')$*.*
  4. $\mathcal{A}$ *outputs* b'*.*

An instantiation for strong KMHE based on the [BHHO08] encryption scheme is indicated in [AHKP22]. In our work we use this as a building-block to construct projective RGS with constant-size labels (proportional to the computational security parameter) for application in our SCALES protocols.

2.5 The SCALES Model

The SCALES model and a SCALES protocol family as described in [AHKP24] for a function family $\mathcal{F}$ where the size of each function representation is parameterized by k. The model involves servers, clients and a public append-only bulletin-board. All the communication between the participating parties take place over this bulletin-board.

Definition 9. *Let $\mathcal{F} = \{f_k | k \in \mathbb{N}\}$ be a function family with members $f_k : \{0,1\}^{\mathsf{m}(k)} \to \{0,1\}^{l(k)}$ that have circuit representation $\mathbf{C}_k$. Let $x = \{x_i\}_{i\in[\mathsf{m}]}$ be the function input. For $c \in \mathbb{N}$, a protocol family $\{\Pi_k | k \in \mathbb{N}\}$ is said to be a c-phase **SCALES protocol family** for $\mathcal{F}$ if, for each k, the protocol Π_k has the following structure.*

- ***Participants:***
  - *Clients $\mathcal{C} = \{C_i\}_{i\in[\mathsf{m}]}$ where C_i has input bit x_i.*
  - *Ephemeral server pool $\mathcal{S}$ with constant fraction σ of honest servers.*
  - *Bulletin-board $\mathcal{B}$ to which any party can write. Anything written onto $\mathcal{B}$ is visible to all parties.*
- ***Communication Pattern:*** *For each phase $p \in [c]$, let s_p be the number of servers participating in computation in that phase. Each phase p has $s_p + 1$ rounds where,*
  - *In round 0, all clients $C_i \in \mathcal{C}$ simultaneously post a message m_i^p onto $\mathcal{B}$*
  - *$\forall$ round $j \in [s_p]$, a randomly selected server $S_j^p \in \mathcal{S}$ reads $\mathcal{B}$, silently performs computation, erases its state, and posts a message m_j^p onto $\mathcal{B}$*
- ***Complexity:***
  - *Each client $C_i \in \mathcal{C}$ can perform computation proportional to a security parameter κ and $\max_{p\in[c]}(s_p)$, but independent of $|\mathbf{C}_k|$*
  - *Each server S_j^p participating in Π_k can perform $O(|\mathbf{C}_k|) = \mathrm{poly}(k)$ computation*

In Definition 9 note that all participating servers are stateless and the number of times a (stateful) client posts a message is equal to the number of phases c in the protocol. Note that this definition is a compiler for the circuit representation of any function into a SCALES protocol. Although it remains silent about an additional privacy property of a SCALES protocol, since this would differ depending on the type of adversary being considered, like in both [AHKP22, AHKP24], we consider security in the presence of an adaptive PPT adversary that can corrupt all-but-one server in each phase. Looking ahead, in Section 3 we present a semi-honest secure protocol tolerating arbitrary client corruption, and server computation sub-linear in the circuit size.

Bulletin-Board $\mathcal{F}_\mathcal{B}$

Let $\mathcal{P}$ be the set of all parties. Let publish denote the action of the functionality sending a message to all parties. The bulletin-board $\mathcal{F}_\mathcal{B}$ works as follows:

- $\mathcal{F}_\mathcal{B}$ initializes its transcript to $\tau = \{\}$ and $c = 0$
- Each time $\mathcal{F}_\mathcal{B}$ receives message m from party P_i,
  - $\mathcal{F}_\mathcal{B}$ sets $c = c + 1$ and $\tau[c] = (m, P_i)$
  - $\mathcal{F}_\mathcal{B}$ publish c
- If a party P_i sends read to $\mathcal{F}_\mathcal{B}$, the functionality sends τ to P_i
- For $c' \in [c]$, if a party P_i sends (read, c') to $\mathcal{F}_\mathcal{B}$, the functionality sends $\tau[c']$ to P_i

Fig. 1 Bulletin-Board

Common Reference String $\mathcal{F}_{\mathsf{CRS}}^{\mathsf{D}}$

Let $\mathcal{P}$ be the set of all parties. Let D denote a distribution over strings $\{0,1\}^*$ (implicitly parameterized by a security parameter κ). The functionality $\mathcal{F}_{\mathsf{CRS}}^{\mathsf{D}}$ works as follows:

- Initialization:
  - Sample $s \leftarrow \mathsf{D}$
- If a party P_i sends read to $\mathcal{F}_{\mathsf{CRS}}^{\mathsf{D}}$, the functionality sends s to P_i

Fig. 2 Common Reference String

Functionality $\mathcal{F}_{\mathsf{SCALES}}^{f}$

Let $f : \{0,1\}^{\mathsf{m}} \to \{0,1\}^{l}$ be a function. Let $\mathcal{A}$ be a PPT ideal-world adversary. Let $\mathcal{C} = \{C_i\}_{i\in[\mathsf{m}]}$ be the clients. The functionality $\mathcal{F}_{\mathsf{SCALES}}^{f}$ works as follows:

- $\mathcal{F}_{\mathsf{SCALES}}^{f}$ accepts an input bit x_i from each $C_i \in \mathcal{C}$
- $\mathcal{F}_{\mathsf{SCALES}}^{f}$ computes $f(x)$ and it sends $f(x)$ to $\mathcal{C}$

Fig. 3 SCALES Functionality

Definition 10. *Let $\mathcal{F}_{\mathcal{B}}$ be the Bulletin-Board functionality (Figure 1). A SCALES protocol family (Definition 9) $\{\Pi_k | k \in \mathbb{N}\}$ for a function family $\{f_k | k \in \mathbb{N}\}$ is said to be a **semi-honestly secure SCALES protocol family** if, for each $k \in \mathbb{N}$, Π_k is a UC-secure protocol for the functionality $\mathcal{F}_{\mathsf{SCALES}}^{f_k}$ (Figure 3) in the $\mathcal{F}_{\mathcal{B}}$-hybrid against a semi-honest PPT adversary that is allowed arbitrary adaptive corruption of clients, and, for each phase, all-but-one adaptive corruption with erasures of participating servers.*

Functionality $\mathcal{F}_{\mathsf{SCALES}}^{f}$

Let $f : \{0,1\}^{\mathsf{m}} \to \{0,1\}^{l}$ be a function. Let $\mathcal{A}$ be a PPT ideal-world adversary. Let $\mathcal{C} = \{C_i\}_{i\in[\mathsf{m}]}$ be the clients. The functionality $\mathcal{F}_{\mathsf{SCALES}}^{f}$ works as follows:

- $\mathcal{F}_{\mathsf{SCALES}}^{f}$ accepts an input bit x_i from each $C_i \in \mathcal{C}$
- $\mathcal{F}_{\mathsf{SCALES}}^{f}$ computes $f(x)$ and gives $f(x)$ to $\mathcal{A}$
- if $\mathcal{A}$ sends OK to $\mathcal{F}_{\mathsf{SCALES}}^{f}$, it sends $f(x)$ to $\mathcal{C}$

Fig. 4 SCALES Functionality (Security with Abort)

Definition 11. *Let $\mathcal{F}_{\mathcal{B}}$ be the Bulletin-Board functionality (Figure 1) and $\mathcal{F}_{\mathsf{CRS}}$ be a Common Reference String functionality (Figure 2). A SCALES protocol family (Definition 9) $\{\Pi_k | k \in \mathbb{N}\}$ for a function family $\{f_k | k \in \mathbb{N}\}$ is said to be a **maliciously secure SCALES protocol family** if, for each $k \in \mathbb{N}$, Π_k is a UC-secure protocol for the functionality $\mathcal{F}_{\mathsf{SCALES}}^{f_k}$ (Figure 4) in the $(\mathcal{F}_{\mathcal{B}}, \mathcal{F}_{\mathsf{CRS}})$-hybrid against a malicious PPT adversary that is allowed all-but-one adaptive corruption of clients, and, for each phase, all-but-one adaptive corruption with erasures of participating servers.*

2.6 Oblivious Transfer

Oblivious Transfer (OT) is a 2-party functionality $\mathcal{F}_{\mathsf{OT}}$ between a sender S and a receiver R. The sender S has inputs $a_0, a_1 \in \{0,1\}$, and the receiver R has $b \in \{0,1\}$. $\mathcal{F}_{\mathsf{OT}}$ takes these inputs and gives a_b to R.

Updatable OT. [AHKP24] define a functionality and SCALES protocol for updatable oblivious transfer (UOT) that we re-state below. $\mathcal{F}_{\mathsf{UOT}}$ involves a chooser C with a bit b, a sender S with input strings $s_0, s_1 \in \mathcal{M}$ and multiple updaters $\{U_j\}_{j\in[d]}$ where each U_j has a function $\sigma_j \in \Sigma$. In the 'OT phase' $\mathcal{F}_{\mathsf{UOT}}$ accepts the chooser's and sender's inputs and returns the string s_b to the chooser. In every 'Update Phase', each updater U_j sends σ_j to $\mathcal{F}_{\mathsf{UOT}}$ which gives to C the output of σ_j applied to the output of the previous phase. Finally, in the 'Reveal Phase' $\mathcal{F}_{\mathsf{UOT}}$ makes the final output to C available to all parties.

Updatable Oblivious Transfer $\mathcal{F}_{\mathsf{UOT}}^{\mathcal{M},\Sigma}$

For a message domain $\mathcal{M}$, let S be a sender with input strings $s_0, s_1 \in \mathcal{M}$. Let C be a chooser with input bit $b \in \{0,1\}$. For function domain Σ, let $\{U_j\}_{j\in[d]}$ be updaters with $\sigma_j \in \Sigma$ s.t. $\sigma_j : \mathcal{M} \to \mathcal{M}$. Let $\mathsf{publish}$ be the action of making a message visible to all parties. The functionality $\mathcal{F}_{\mathsf{UOT}}^{\mathcal{M},\Sigma}$ works as follows:

1. OT Phase:
   - receive input b from C and (s_0, s_1) from S
   - send $s_{b,0} = s_b$ to C
2. Update Phase, for each $j \in [d]$:
   - receive input σ_j from U_j and send $s_{b,j} = \sigma_j(s_{b,j-1})$ to C
3. Reveal Phase:
   - $\mathsf{publish}$ $s_{b,d}$

Fig. 5 Updatable Oblivious Transfer

[AHKP24] contains an instantiation of this functionality $\mathcal{F}_{\mathsf{UOT}}$ in a way that requires the sender and each updater to send only one message each. The chooser sends one message at the beginning and one at the end of the protocol and the resulting protocol is secure in the presence of a PPT adversary adaptively corrupting the chooser, and all-but-one of the sender and updaters. This is abstracted in Definition 12. We make an effort to unify the definition of an updatable OT protocol for both the semi-honest and malicious security settings below, where the text marked in gray is only relevant when considering malicious security.

Definition 12. *Let $\mathcal{M}$ be the message space for the sender S and Σ be the function space for each updater in $\{U_j\}_{j\in[d]}$ such that each $\sigma \in \Sigma$ is of the form $\sigma : \mathcal{M} \to \mathcal{M}$. The tuple $\mathsf{UOT} = (\mathsf{CRSgen}, \mathsf{OT}_1, \mathsf{OT}_2, \mathsf{OT}_{\mathsf{updt}}, \mathsf{OT}_{\mathsf{fin}}, \mathsf{OT}_{\mathsf{vrfy}})$ is a **SCALES Protocol for UOT** if the following protocol UC-securely realizes $\mathcal{F}_{\mathsf{UOT}}^{\mathcal{M},\Sigma}$ (Figure 5) in the $(\mathcal{F}_{\mathsf{CRS}}^{\mathsf{CRSgen}}, \mathcal{F}_{\mathcal{B}})$-hybrid in the presence of any PPT adversary $\mathcal{A}$ that can adaptively corrupt C and statically corrupt all-but-one of $\{S\} \cup \{U_j\}_{j\in[d]}$:*

Inputs: $b \in \{0,1\}$ *for receiver* C; $(s_0, s_1) \in \mathcal{M}^2$ *for* S; σ_i *for each updater* U_i.

$$
\begin{aligned}
\mathcal{F}_{\mathsf{CRS}}^{\mathsf{CRSgen}} : \quad & \mathsf{CRS} \leftarrow \mathsf{CRSgen}(1^\kappa) & \\
C : \quad & \mathsf{aux}, \mathbf{m_1} \leftarrow \mathsf{OT}_1(\mathsf{CRS}; b) & \mathbf{m_1} \rightarrow \mathcal{F}_{\mathcal{B}} \\
S : \quad & \mathbf{m_2^0} \leftarrow \mathsf{OT}_2(\mathsf{CRS}; \mathbf{m_1}, (s_0, s_1)) & \mathbf{m_2^0} \rightarrow \mathcal{F}_{\mathcal{B}} \\
C : \quad & M \leftarrow \mathsf{OT}_{\mathsf{fin}}(\mathsf{CRS}; \mathsf{aux}, \mathbf{m_2^0}) & \\
& \{s_b, \perp\} \leftarrow \mathsf{OT}_{\mathsf{vrfy}}(\mathsf{CRS}; M) & \textit{output } s_b \\
\forall j \in [d]\ U_j : \quad & \mathbf{m_2^j} \leftarrow \mathsf{OT}_{\mathsf{updt}}(\mathsf{CRS}; \mathbf{m_2^{j-1}}, \sigma_j) & \mathbf{m_2^j} \rightarrow \mathcal{F}_{\mathcal{B}} \\
C : \quad & M_j \leftarrow \mathsf{OT}_{\mathsf{fin}}(\mathsf{CRS}; \mathsf{aux}, \mathbf{m_2^j}) & M_d \rightarrow \mathcal{F}_{\mathcal{B}} \\
& \{s_{b,j}, \perp\} \leftarrow \mathsf{OT}_{\mathsf{vrfy}}(\mathsf{CRS}; M_j) & \textit{output } s_{b,j} \\
\textit{public} : \quad & \{s_{b,d}, \perp\} \leftarrow \mathsf{OT}_{\mathsf{vrfy}}(\mathsf{CRS}; M_d) & \textit{output } s_{b,d}
\end{aligned}
$$

where $\mathcal{F}_{\mathsf{CRS}}^{\mathsf{CRSgen}}$ *(Figure 2) is the Common Reference String functionality and* $\mathcal{F}_{\mathcal{B}}$ *(Figure 1) is the Bulletin-Board functionality.*

[AHKP24] realizes this functionality using an NIZK protocol and a two-round bit-OT with special properties that they instantiate using the [PVW08] protocol in the CRS-model. In the semi-honest setting, a similar construction can be used with only a semi-honestly secure two-round bit-OT (with similar properties) in the plain model [GHV10]. When plugged into the final protocol in the SCALES model, these updatable OT protocols share the feature that the last client message (C's message) requires computation independent of the function size and the number of servers involved in the protocol.

Distributed Committed-Index OT. [AHKP24] define and build a 'Distributed Committed-Index OT' (DCOT) $\mathcal{F}_{\mathsf{DCOT}}$. This is a functionality between d senders and m choosers. First, each chooser C_i samples a random bit b_i and inputs to the functionality. Each sender S_j gives 2 strings s_0^j, s_1^j as its input and the goal is for the functionality to make publicly visible: s_b^j, for a choice bit $b = \bigoplus_{i=0}^{\mathsf{m}} b_i$. Figure 6 describes this functionality $\mathcal{F}_{\mathsf{DCOT}}$.

Distributed Committed-Index OT $\mathcal{F}_{\mathsf{DCOT}}$

Let $\{S_j\}_{j\in[d]}$ be the senders where each S_j has strings (s_0^j, s_1^j). Let $\{C_i\}_{i\in[\mathsf{m}]}$ be choosers where each C_i has bit b_i. Let $\mathsf{publish}$ denote the action of $\mathcal{F}_{\mathsf{DCOT}}$ making a message visible to all parties. $\mathcal{F}_{\mathsf{DCOT}}$ works as follows:

1. Choosers' Commit Phase:
   - $\forall i \in [\mathsf{m}]$, receive input b_i from C_i
2. $\forall j \in [d]$, Senders' Commit Phase:
   - receive input (s_0^j, s_1^j) from S_j
3. Reveal Phase:
   - $\mathsf{publish}$ $(\{s_b^j\}_{j\in[d]}, \{b_i\}_{i\in[\mathsf{m}]})$, where $b = \bigoplus_{i=0}^{\mathsf{m}} b_i$

Fig. 6 Distributed Committed-Index OT

[AHKP24] also define a protocol for this where each server, acting as a sender, speaks once and the clients, acting as choosers, post at most one message before and after, followed by public verification and decoding. We re-state the definition below with the modification that *the client second message is not a function of the sender messages.*

Definition 13. *The tuple* $\mathsf{DCOT} = (\mathsf{CRSgen}, \mathsf{OT}_1, \mathsf{OT}_2, \mathsf{OT}_{\mathsf{fin}}, \mathsf{OT}_{\mathsf{dec}})$ *is a* ***Maliciously Secure SCALES Protocol for DCOT*** *if the following protocol UC-securely realizes* $\mathcal{F}_{\mathsf{DCOT}}$ *(Figure 6) in the* $(\mathcal{F}_{\mathsf{CRS}}^{\mathsf{CRSgen}}, \mathcal{F}_{\mathcal{B}})$*-hybrid in the presence of any PPT malicious adversary* $\mathcal{A}$ *that can corrupt all-but-one client* $\{C_i\}_{i\in[\mathsf{m}]}$ *adaptively and all-but-one of* $\{S_j\}_{j\in[d]}$ *statically.*

Inputs: $b_i \in \{0,1\}$ *for each chooser* C_i*,* $(s_0^j, s_1^j) \in \mathcal{M}^2$ *for each* S_j

$$
\begin{aligned}
\mathcal{F}_{\mathsf{CRS}}^{\mathsf{CRSgen}}: &\quad \mathsf{CRS} \leftarrow \mathsf{CRSgen}(1^\kappa) \\
\forall i \in [\mathsf{m}], C_i: &\quad \mathsf{aux}_i, \mathsf{m}_1^{\mathsf{i}} \leftarrow \mathsf{OT}_1(\mathsf{CRS}; b_i) && \mathsf{m}_1^{\mathsf{i}} \to \mathcal{F}_{\mathcal{B}} \\
\forall j \in [d], S_j: &\quad \mathsf{m}_2^{\mathsf{j}} \leftarrow \mathsf{OT}_2(\mathsf{CRS}; \{\mathsf{m}_1^{\mathsf{i}}\}_{i\in[\mathsf{m}]}, (s_0^j, s_1^j)) && \mathsf{m}_2^{\mathsf{j}} \to \mathcal{F}_{\mathcal{B}} \\
\forall i \in [\mathsf{m}], C_i: &\quad \mathsf{m}_3^{\mathsf{i}} \leftarrow \mathsf{OT}_{\mathsf{fin}}(\mathsf{CRS}; \mathsf{aux}_i) && \mathsf{m}_3^{\mathsf{i}} \to \mathcal{F}_{\mathcal{B}} \\
\textit{public}: &\quad \{s_b^j\}_{j\in[d]}, \{b_i\}_{i\in[\mathsf{m}]} \leftarrow \mathsf{OT}_{\mathsf{dec}}(\mathsf{CRS}; \{\mathsf{m}_1^{\mathsf{i}}\}_{i\in[\mathsf{m}]}, \{\mathsf{m}_2^{\mathsf{j}}\}_{j\in[d]}, \{\mathsf{m}_3^{\mathsf{i}}\}_{i\in[\mathsf{m}]})
\end{aligned}
$$

where $\mathcal{F}_{\mathsf{CRS}}^{\mathsf{CRSgen}}$ *(Figure 2) is the Common Reference String functionality and* $\mathcal{F}_{\mathcal{B}}$ *(Figure 1) is the Bulletin-Board functionality.*

Note that [AHKP24] instantiates DCOT using a two-round OT protocol in a way that already satisfies the definition above. As a result, it already holds that in the maliciously secure SCALES protocols presented in [AHKP24], the client computation is always independent of the number of participating servers.

3 Semi-Honestly Secure SCALES Protocol

In this section we present a semi-honestly secure SCALES protocol for any bounded function family $\mathcal{F}$ consisting of functions that can be represented by a constant-width PBP of polynomial-length. This includes a large class of functions and, in particular any Boolean function in $\mathbf{NC}^1$ can be represented using a polynomial length width-5 PBP [Bar86]. The protocol has two phases and is secure in the presence of a PPT adversary corrupting up to all clients, all-but-one servers in the first phase and the server in the last phase. While prior work in [AHKP22] present such a protocol for computing general functions, our protocol has the added advantage that the client computation does not grow linearly with the number of servers in the first phase. Further, in our protocol the communication and computation complexity of all parties grows linearly with the computational security parameter. This is in contrast with prior work where the complexity of each server grows cubically and that of the clients grow quadratically in the security parameter.

We construct our SCALES protocol using an outline implicit in previous work [AHKP24] that constructs a maliciously secure protocol. However we restrict ourselves to the semi-honest setting for now and use two key building-blocks: a rerandomizable garbling scheme (RGS – Definition 3) and an updatable OT protocol (UOT – Definition 12). We discuss these in greater detail and present efficient constructions in Section 3.1 and 3.2 respectively. Note here that prior work [AHKP22] also constructs a semi-honestly secure SCALES protocol from RGS but uses OT in a different way that results in their client complexity to grow with the number of computation servers.

Informally, our SCALES protocol proceeds as follows. First, the clients create the UOT chooser's message using its input bit as the choice bit and posts it. The first server garbles the function and for each client's input bit, it uses the pair of input labels to create the UOT senders message for the respective clients choosers message. The function garbling and all the UOT senders messages are posted onto the bulletin board. Next, each successive computation server rerandomizes the garbling and then uses the corresponding transformations as the updaters inputs to the UOT protocols. This server then posts the rerandomized garbling and the UOT messages. At the end of this phase, the clients use the UOT choosers message to decode the the final servers UOT message to reveal one label each. In the next phase, servers use these to evaluate the final garbling. This protocol is formalized below.

Protocol Semi-Honestly Secure SCALES

Let κ be a computational security parameter. Let UOT be a semi-honestly secure UOT protocol that is compatible with the projective RGS RGS. We denote by $\{\{\mathsf{C}_{\mathsf{msg}_p}^{\ i}\}_{i\in[\mathsf{m}]}, \{\mathsf{S}_{\mathsf{msg}_p}^{\ j}\}_{j\in[d]}\}_{p\in[P]}$ a P-phase secure SCALES protocol family in terms of its next-message functions. Within it, $\mathsf{C}_{\mathsf{msg}_p}^{\ i}$ denotes the next-message function of the ith client for the pth phase. $\mathsf{S}_{\mathsf{msg}_p}^{\ j}$ denotes the computation of the jth server in the pth phase. The 2-phase semi-honestly secure SCALES protocol operates as follows:

– $\forall i \in [\mathsf{m}]$, $(\mathsf{aux}_i, m_i^1) \leftarrow \mathsf{C}_{\mathsf{msg}_1}^{\ i}(x_i)$

$$\mathsf{aux}, \mathbf{m_1^i} \leftarrow \mathsf{UOT.OT}_1(x_i)$$
$$m_i^1 = \mathbf{m_1^i} \qquad \mathsf{aux}_i = \mathsf{aux}$$
$$\mathsf{publish} \quad m_i^1$$

– $\forall j \in [d]$, $m_j^1 \leftarrow \mathsf{S}_{\mathsf{msg}_1}^{\ j}(\{m_i^1\}_{i\in[\mathsf{m}]}, \{m_{j'}^1\}_{j'\in[j-1]})$
if $j = 1$,

$$F_1, e = \{\mathsf{L}_i^0, \mathsf{L}_i^1\}_{i\in[\mathsf{m}]} \leftarrow \mathsf{RGS.Gb}(f)$$
$$\forall i \in [\mathsf{m}], \mathbf{m_2^{j,i}} \leftarrow \mathsf{UOT.OT}_2(m_i^1, (\mathsf{L}_i^0, \mathsf{L}_i^1))$$

otherwise,

$$F_j, \sigma = \{\sigma_i\}_{i\in[\mathsf{m}]} \leftarrow \mathsf{RGS.Rerand}(F_{j-1})$$
$$\forall i \in [\mathsf{m}], \mathbf{m_2^{j,i}} \leftarrow \mathsf{UOT.OT}_{\mathsf{updt}}(\mathbf{m_2^{j-1,i}}, \sigma_i)$$
$$\mathsf{publish} \quad m_j^1 = (F_j, \{\mathbf{m_2^{j,i}}\}_{i\in[\mathsf{m}]})$$

– $\forall i \in [\mathsf{m}]$, $m_i^2 \leftarrow \mathsf{C}_{\mathsf{msg}_2}^{\ i}(\mathsf{aux}_i, m_d^1)$

$$\mathsf{L}_i^{x_i} \leftarrow \mathsf{UOT.OT}_{\mathsf{fin}}(\mathsf{aux}_i, \mathbf{m_2^{d,i}}) \qquad \mathsf{publish} \quad m_i^2 = k_i^{x_i}$$

– $\forall j \in [d]$, $m_j^2 \leftarrow \mathsf{S}_{\mathsf{msg}_2}^{\ j}(\{m_i^1, m_i^2\}_{i\in[\mathsf{m}]}, \{m_j^1\}_{j\in[d]})$

$$f(x) = \mathsf{RGS.Ev}(F_d, X = \{\mathsf{L}_i^{x_i}\}_{i\in[\mathsf{m}]})$$
$$\mathsf{publish} \quad m_j^2 = f(x)$$

Fig. 7 Semi-Honestly Secure SCALES

Lemma 1. *Let* RGS *be an RGS (Definition 3) for function family* $\mathcal{F}$ *consisting of functions that can be represented by a width-w PBP of polynomial-length, and* UOT *be a semi-honestly secure UOT protocol (Definition 12) that are compatible (Definition 15). Then Figure 7 is a two-phase semi-honestly secure SCALES protocol for functions in* $\mathcal{F}$ *that is secure as long as one server in the first phase is honest.*

Proof. In order to prove that the protocol in Figure 7 is a semi-honest secure SCALES protocol realizing the functionality $\mathcal{F}_f$ (Figure 3), we first re-write the protocol in the $\mathcal{F}_{\mathsf{UOT}}$-hybrid as follows:

– For $i \in [\mathsf{m}]$, letting $\mathcal{F}_{\mathsf{UOT}}^i$ be the ith independent execution of $\mathcal{F}_{\mathsf{UOT}}$, each client C_i first sends its input bit x_i as the chooser's input to $\mathcal{F}_{\mathsf{UOT}}^i$.
– The first server garbles f, producing,

$$F_1, e = \{\mathsf{L}_i^0, \mathsf{L}_i^1\}_{i\in[\mathsf{m}]} \leftarrow \mathsf{RGS.Gb}(f)$$

For $i \in [\mathsf{m}]$, it sends $(\mathsf{L}_i^0, \mathsf{L}_i^1)$ as the sender's inputs to $\mathcal{F}_{\mathsf{UOT}}^i$. This server then posts F_1.
– Every subsequent server rerandomizes the previous circuit, producing,

$$F_j, \sigma = \{\sigma_i\}_{i\in[\mathsf{m}]} \leftarrow \mathsf{RGS.Rerand}(F_{j-1})$$

For $i \in [\mathsf{m}]$, it sends σ_i as an updater's input to $\mathcal{F}_{\mathsf{UOT}}^i$. Then it posts F_j.

– Finally, for $i \in [\mathsf{m}]$, each $\mathcal{F}^i_{\mathsf{UOT}}$ publishes $\mathsf{L}_i^{x_i}$. Then a server can evaluate the final garbling to post the function output:

$$f(x) = \mathsf{RGS.Ev}(F_d, X = \{\mathsf{L}_i^{x_i}\}_{i\in[\mathsf{m}]})$$

We can now describe a PPT simulator Sim that can simulate the view of a semi-honest adversary (corrupting all-but-one server in the first phase, the evaluator and possibly all clients) in the above protocol as follows:

– Sim has access to all the inputs $\{x_i\}_{C_i\in\mathcal{A}}$ and randomness $\{r_j\}_{S_j\in\mathcal{A}}$ of all the corrupt parties.
– Sim enters $\{x_i\}_{C_i\in\mathcal{A}}$ into the functionality and receives $f(x)$.
– For each $j \in [d]$, if the server is corrupt, it uses its randomness r_j to compute either,

$$F_1, e = \{\mathsf{L}_i^0, \mathsf{L}_i^1\}_{i\in[\mathsf{m}]} \leftarrow \mathsf{RGS.Gb}(f)$$

or,

$$F_j, \sigma = \{\sigma_i\}_{i\in[\mathsf{m}]} \leftarrow \mathsf{RGS.Rerand}(F_{j-1})$$

and the inputs to each $\mathcal{F}^i_{\mathsf{UOT}}$ just like in the real execution.
If the server is honest, it makes black-box use of the RGS simulator and computes,

$$F_j, X_j = \{\mathsf{L}_i^{x_i}\}_{i\in[\mathsf{m}]} \leftarrow \mathsf{Sim}(1^\kappa, \phi(f), f(x))$$

and sends F_j to the adversary $\mathcal{A}$, along with the subset $\{\mathsf{L}_i^{x_i}\}_{C_i\in\mathcal{A}}$, where each $\mathsf{L}_i^{x_i}$ is the output of $\mathcal{F}^i_{\mathsf{UOT}}$ to the chooser at the jth round.
– Finally, for $i \in [\mathsf{m}]$, each $\mathcal{F}^i_{\mathsf{UOT}}$ publishes $\mathsf{L}_i^{x_i}$ (computed by applying the updates derived from the randomness of the corrupt servers' that came after the honest party, on the simulated label). Then the view is completed by posting $f(x)$.

In order to show that the view produced by the above simulator is computationally indistinguishable from the real view, let us consider the following set of hybrids:

– $\mathsf{H}_0 = \{\{x_i\}_{C_i\in\mathcal{A}}, \{r_j\}_{S_j\in\mathcal{A}}, \{F_j, X_{j,\mathcal{A}} = \{\mathsf{L}_i^{x_i}\}_{C_i\in\mathcal{A}}\}_{j\in[d]}, X_d = \{\mathsf{L}_i^{x_i}\}_{i\in[\mathsf{m}]}, f(x)\} \leftarrow \mathsf{Sim}$: This hybrid experiment produces the view that is identically distributed as the simulated view above.
– $\mathsf{H}_1 = \{\{x_i\}_{C_i\in\mathcal{A}}, \{r_j\}_{S_j\in\mathcal{A}}, \{F_j, X_{j,\mathcal{A}} = \{\mathsf{L}_i^{x_i}\}_{C_i\in\mathcal{A}}\}_{j\in[d]}, X_d = \{\mathsf{L}_i^{x_i}\}_{i\in[\mathsf{m}]}, f(x)\}$: This hybrid experiment differs from the simulation in that, on behalf of the honest server, instead of creating a simulated garbling and set of active labels only, an honest fresh garbling of the function is generated with its corresponding input encoding information. Further the function input bits of the honest parties are used to select the input labels, that are updated by the subsequent servers to derive $\mathsf{L}_i^{x_i} \in X_d$ internally in each $\mathcal{F}^i_{\mathsf{UOT}}$.
– $\mathsf{H}_2 = \{\{x_i\}_{C_i\in\mathcal{A}}, \{r_j\}_{S_j\in\mathcal{A}}, \{F_j, X_{j,\mathcal{A}} = \{\mathsf{L}_i^{x_i}\}_{C_i\in\mathcal{A}}\}_{j\in[d]}, X_d = \{\mathsf{L}_i^{x_i}\}_{i\in[\mathsf{m}]}, f(x)\}$: This hybrid experiment is the same as the real execution. It differs from the previous experiment in that, on behalf of the honest server, instead of creating a fresh garbling and labels, now the server rerandomizes the previous garbling and updates the labels within the UOT functionality.

If there existed a PPT distinguisher D that could distinguish between hybrids H_0 and H_2 with non-negligible advantage ϵ, it must follow that D should be able to distinguish between either H_0 and H_1, or H_1 and H_2 with advantage $> \frac{\epsilon}{2}$, which is non-negligible.

We first show that if D can distinguish between H_0 and H_1 with non-negligible advantage, then it can be used in a black-box way by a PPT adversary $\mathcal{A}_{\mathsf{RGS}}$ to violate 'privacy-by-simulation' (Definition 4) of the projective RGS. $\mathcal{A}_{\mathsf{RGS}}$ works as follows:

– $\mathcal{A}_{\mathsf{RGS}}$ receives from the challenger the tuple (F, X) that is either a simulated or a real garbling and the corresponding active labels set. It has the inputs $\{x_i\}_{i\in[\mathsf{m}]}$ of all the parties and randomness $\{r_j\}_{j\in\mathcal{A}}$ of the corrupt parties.
– Let j^* be the index of the honest computation server. $\mathcal{A}_{\mathsf{RGS}}$ generates $\{F_j, X_{j,\mathcal{A}} = \{\mathsf{L}_i^{x_i}\}_{C_i\in\mathcal{A}}\}_{j<j^*}$ just like in the simulation (that is identical for both hybrid experiments). It then sets $F_{j^*} = F$ and $X_{j^*,\mathcal{A}} = \{\mathsf{L}_i^{x_i}\}_{C_i\in\mathcal{A}} \subseteq X$ from the challenge tuple received. It then completes the view of the adversary by using each r_j for $j > j^*$ to rerandomize this challenge garbling and update the corresponding labels. Note that if (F, X) were outputs of the garbling simulation then this is distributed identically as the output of H_0. Otherwise, this is distributed as in the output of H_1.

– Finally, $\mathcal{A}_{\mathsf{RGS}}$ sends to D the tuple $(\{x_i\}_{C_i\in\mathcal{A}}, \{r_j\}_{S_j\in\mathcal{A}}, \{F_j, X_{j,\mathcal{A}}\}_{j\in[d]}, X_d = \{\mathsf{L}_i^{x_i}\}_{i\in[\mathsf{m}]}, f(x))$ generated and outputs whatever it outputs.

In the above execution, the advantage of $\mathcal{A}_{\mathsf{RGS}}$ is the same as that of D, which is non-negligible. However, since RGS satisfies privacy-by-simulation, it follows that no such $\mathcal{A}_{\mathsf{RGS}}$ can exist and therefore D cannot distinguish between the two hybrids.

Next, we show that if D can distinguish between H_1 and H_2 with non-negligible advantage, then it can be used in a black-box way by a PPT adversary $\mathcal{A}_{\mathsf{RGS}}$ to violate 'Rerand-privacy' (Definition 3) of the projective RGS. $\mathcal{A}_{\mathsf{RGS}}$ works as follows:

– $\mathcal{A}_{\mathsf{RGS}}$ has the inputs $\{x_i\}_{i\in[\mathsf{m}]}$ of all the parties and randomness $\{r_j\}_{j\in\mathcal{A}}$ of the corrupt parties.
– Let j^* be the index of the honest computation server. $\mathcal{A}_{\mathsf{RGS}}$ generates $\{F_j, X_{j,\mathcal{A}} = \{\mathsf{L}_i^{x_i}\}_{C_i\in\mathcal{A}}\}_{j<j^*}$ just like in the simulation (that is identical for both hybrid experiments). Note that by 'randomness composition', we have that the randomness $r = \mathsf{Explain}(\{r_j\}_{j<j^*})$, that is the composition of all the randomness of the colluding garbler and rerandomizer before the honest party, is a valid garbling randomness for the garbling algorithm in RGS that will produce the rerandomized garbling.
– Then $\mathcal{A}_{\mathsf{RGS}}$ sends r to the challenger along with f and the complete input x. It receives the tuple (F, X) that is either a fresh, unrelated garbling or a rerandomized garbling from F_{j^*-1}, and the corresponding set of active labels.
– $\mathcal{A}_{\mathsf{RGS}}$ sets $F_{j^*} = F$ and $X_{j^*,\mathcal{A}} = \{\mathsf{L}_i^{x_i}\}_{C_i\in\mathcal{A}} \subseteq X$ from the challenge tuple received. It then completes the view of the adversary by using each r_j for $j > j^*$ to rerandomize this challenge garbling and update the corresponding labels. Note that if (F, X) were fresh garbling and labels then this is distributed identically as the output of H_1. Otherwise, this is distributed as in the output of H_2.
– Finally, $\mathcal{A}_{\mathsf{RGS}}$ sends to D the tuple $(\{x_i\}_{C_i\in\mathcal{A}}, \{r_j\}_{S_j\in\mathcal{A}}, \{F_j, X_{j,\mathcal{A}}\}_{j\in[d]}, X_d = \{\mathsf{L}_i^{x_i}\}_{i\in[\mathsf{m}]}, f(x))$ generated and outputs whatever it outputs.

In the above execution, the advantage of $\mathcal{A}_{\mathsf{RGS}}$ is the same as that of D, which is non-negligible. However, since RGS satisfies Rerand-privacy, it follows that no such $\mathcal{A}_{\mathsf{RGS}}$ can exist and therefore D cannot distinguish between the two hybrids. This concludes the proof.

3.1 Instantiating RGS for PBP

In this section we construct an RGS for any function family $\mathcal{F}$ whose members can be represented using a constant-width polynomial length permutation branching program (PBPs, Definition 6 – Section 2.2). We first point out that any such PBP can be converted into an RGS wherein the privacy and Rerand-privacy properties hold unconditionally. Such an information-theoretic projective RGS, although inexpensive to construct, has the drawback that the length of each of its projective input encoding label grows with the function size. In order to bypass this, we next introduce a generic transform that can convert a projective RGS with arbitrary length labels into one where the input label size is $O(\kappa)$, where κ is a computational security parameter. This transformation makes black-box use of a strong Key and Message Homomorphic Encryption Scheme (strong KMHE – Definition 8) and down-grades the security of the RGS to only hold computationally. Finally, we present an efficient instantiation of the strong KMHE scheme that can be used in compatibility with the information-theoretic RGS.

Lemma 2. *Assuming the hardness of the DDH problem, there exists an explainable projective RGS (Definition 3) for any function family $\mathcal{F}$ consisting of functions that can be represented by a width-w PBP of polynomial-length $\ell = O(\mathrm{poly}(\mathsf{m}))$, for input length m, such that the size of the garbling is $O(\ell w\kappa)$ bits and the size of each input label is $O(\kappa)$ bits, where κ is a computational security parameter.*

The proof for this follows directly from Lemmas 3, 5 and 6 in the ensuing text.

Efficient Information Theoretic RGS. We first point out the existence of an information-theoretic RGS, a projective RGS (Definition 3) wherein the privacy and Rerand-privacy properties hold unconditionally. We further show that this RGS is explainable (Definition 5) meaning garbling/rerandomizing explainability privacy also holds in the presence of unbounded adversaries. All correctness properties hold as usual. This is constructed using a constant-width PBP, an object that can represent any function and, in particular, any Boolean function in $\mathbf{NC}^1$ can be represented using a polynomial length width-5 PBP [Bar86]. More precisely, any function with a $c\log(\mathsf{m})$-depth circuit representation (where m is the input length) can be represented using a $O(\mathsf{m}^{2c})$-length PBP. Moreover, previous work [Kil88] shows how a PBP can be randomized and this process forms the basis of the construction that we present next.

Construction 2 *For a function f with input $x \in \{0,1\}^{\mathsf{m}}$, let $M = (\Pi, V)$ be a length-k width-w PBP (Definition 6). Let S_w be the group of permutations on w elements. Then the algorithms in* $\mathsf{RGS} = (\mathsf{Gb}, \mathsf{Rerand}, \mathsf{En}, \mathsf{Ev}, \mathsf{Explain})$ *are initialized as follows:*

1. $F, e \leftarrow \mathsf{Gb}(f; r)$
   - *let $M = (\Pi, V)$ be a length-k width-w PBP for f*
   - *letting r_0 and r_k be the identity permutation, parse the randomness $r = R$ where $R = \{r_i\}_{i\in[k-1]}$ where each $r_i \leftarrow S_w$*
   - *set $M_r = (\Pi_r, V)$ where,*

$$\forall i \in [k], j \in \{0,1\}, \Pi_r(i,j) = r_{i-1}^{-1}\Pi(i,j)r_i$$

   - *return $F = \perp$; $e = M_r$*
2. $F', \pi_{\mathsf{En}} \leftarrow \mathsf{Rerand}(F; r')$
   - *parse $F = \perp$ and letting r_0 and r_k be the identity permutation, parse $r' = R$ where $R = \{r_i\}_{i\in[k-1]}$ where each $r_i \leftarrow S_w$*
   - *return $F' = \perp$; $\pi_{\mathsf{En}} = r'$*

   compute $\pi_{\mathsf{En}}(e) = M_{r\circ r'}$ by parsing $e = M_r = (\Pi_r, V)$ and setting $M_{r\circ r'} = (\Pi_{r\circ r'}, V)$ where,

$$\forall i \in [k], j \in \{0,1\}, \Pi_{r\circ r'}(i,j) = r_{i-1}^{-1}\Pi(i,j)r_i$$

3. $r^* \leftarrow \mathsf{Explain}(r, r')$
   - *parse $r = R$ where $R = \{r_i\}_{i\in[k-1]}$ where each $r_i \leftarrow S_w$*
   - *parse $r' = R'$ where $R' = \{r'_i\}_{i\in[k-1]}$ where each $r'_i \leftarrow S_w$*
   - *return $r^* = R^*$ where*

$$R^* = \{r^*_i = r_i \circ r'_i\}_{i\in[k-1]}$$

4. $X = \mathsf{En}(e, x)$
   - *parse $e = M_r = (\Pi_r, V)$*
   - *return $X = \{\Pi_r(i, x_{V(i)})\}_{i\in[k]}$*
5. $y = \mathsf{Ev}(F, X)$
   - *parse $F = \perp$ and $X = \{\Pi_r(i, x_{V(i)})\}_{i\in[k]}$*
   - *return $E(\Pi(1, x_{V(1)}) \circ \ldots \circ \Pi(k, x_{V(k)}))$*

In the construction above, the garbling algorithm works by simply taking the canonical PBP representation of a function and randomizing it using the process indicated in [Kil88]. The rerandomizing process does the same thing, now to an already randomized PBP. The randomness composition algorithm Explain simply works by position-wise composing the permutations used in both the garbling and rerandomizing process. Finally, the encoding and evaluation algorithms simply follow the process of PBP evaluation (Definition 6). Note that the entire construction never uses any building-blocks requiring computational assumptions, only relying on compositions of permutations that act as one-time-pads to mask the function representation.

Lemma 3. *For any function family $\mathcal{F}$ consisting of functions that can be represented by a width-w PBP of polynomial-length $O(\mathrm{poly}(\mathsf{m}))$, for input length m, there exists an information-theoretic explainable RGS (Definition 5) with garbling and projective encodings of size $O(w\mathrm{poly}(\mathsf{m}))$.*

The proof of this follows from the proof of the following lemma.

Lemma 4. *For any function family $\mathcal{F}$ where any $f \in \mathcal{F}$ can be represented by a PBP (Definition 6) M of length k, Construction 2 is an information-theoretic explainable RGS (Definition 5) where the garbling and projective encoding sizes are $O(k)$.*

Proof. The RGS in Construction 2 information-theoretically satisfies all the properties of an explainable RGS. Correctness, projective encoding (Definition 3) and randomness composition (Definition 5) are implied from the properties of a length-k width-w PBP (Definition 6). It only remains to argue that privacy and Rerand-privacy are additionally satisfied information-theoretically.

Privacy: This property requires that for two functions $f_0, f_1 \in \mathcal{F}$ and every input x_0, x_1 such that $f_0(x_0) = f_1(x_1)$,

$$\{F_0, X_0\}_{(F_0,e_0)\leftarrow\mathsf{Gb}(f_0),X_0=\mathsf{En}(e_0,x_0)} \equiv \{F_1, X_1\}_{(F_1,e_1)\leftarrow\mathsf{Gb}(f_1),X_1=\mathsf{En}(e_1,x_1)}$$

Note that for our construction, in both distributions the garbled circuit is the empty symbol $\perp$. So it remains only to argue that the output of the encoding function are identically distributed. Both input encoding sets would evaluate to the same output and evaluation requires simply composing all active permutations. Letting $\mathcal{R}$ be the space of random tapes, the above distributions are identical since for each $r_0 \leftarrow \mathcal{R}$ producing a set X_0 from input x_0 in the first distribution, there exists a unique $r \in \mathcal{R}$ producing the same set from x_1. In other words, the pre-image of the evaluation algorithm ev is identically distributed over $\mathcal{R}$ for both distributions. Hence it follows that the privacy property holds information-theoretically.

Rerand-Privacy: This property requires that a tuple containing a rerandomized garbling with an input encoding be distributed identically to a one containing a fresh garbling with it corresponding input encoding. This should hold even given all the randomness used in the prior garbling from with the RGS in the former distribution has been rerandomized.

$$\{r, F_0, X_0\}_{\substack{r\leftarrow\mathcal{R},\ (F,e)\leftarrow\mathsf{Gb}(f;r),\\ (F_0,\pi_{\mathsf{En}})\leftarrow\mathsf{Rerand}(F),\ X_0=\mathsf{En}(\pi_{\mathsf{En}}(e),x)}} \equiv \{r, F_1, X_1\}_{\substack{r\leftarrow\mathcal{R},(F_1,e_1)\leftarrow\mathsf{Gb}(f),\\ X_1=\mathsf{En}(e_1,x)}}$$

Letting $\mathcal{R}$ be the space of random tapes for the garbling algorithm Gb, in Construction 2 this space contains all vectors of permutations in S_5 of length $k-1$. Note that the randomness used for garbling and rerandomizing come from the same space and the function $\mathsf{Explain}$ works by simply pair-wise composing the permutations in the same indices to derive the new vector of permutations. As such, even on fixing one of its inputs, the garbling randomness, the outputs of this function are uniformly distributed in $\mathcal{R}$. It follows from this that a rerandomized garbling can be explained as a fresh garbling using randomness that is distributed identically as the randomness sampled for constructing a fresh garbling, and this holds even when the randomness of the prior garbling is fixed. Since in both distributions the input encoding set is derived from a deterministic function on this randomness and the input, it follows that the complete distributions of interest are identical.

This concludes the proof.

RGS with Constant-Size Input Labels. We now discuss how *any* projective RGS can be converted into one in which the input label sizes grow only in proportion to the security parameter. This transformation becomes useful for producing succinct input encodings in RGS, as stated above, but comes at the cost of introducing computational hardness assumptions. In order to build such a scheme, we ustlize a projective rerandomizable garbling scheme (RGS – Definition 3) and a strong KMHE scheme (Definition 8), using the latter to encrypt the corresponding projective input encoding. The κ-bit keys of the strong KMHE scheme are used as the new projective input encoding, and the ciphertexts formed become part of the new function garbling. Note that encryption here doesn't shrink the label size; instead, long labels are replaced with short keys that can decrypt public ciphertexts to recover the labels. Looking ahead, in our SCALES protocol, these ciphertexts are prepared by the servers as part of the PBP garbling process and do not involve the client's work.

The Computationally Secure Projective RGS. Constructing an RGS as described involves a garbling phase in which, first, the garbling algorithm of the original RGS is executed. This produces a function garbling and projective input encoding information containing one label per value – 0 or 1 – of each input bit. We then generate a strong KMHE key for each such input label and encrypt the label using the corresponding key. Each pair of ciphertexts is first permuted randomly and then the set of ciphertexts so generated now can be added to the function garbling, while the keys used within them become the new input encoding information. Rerandomizing such a garbling involves first executing the original rerandomizing algorithm on the original garbling. This produces a rerandomized garbling and a set of transformations to apply to the projective inputs, one for each input bit position. We then apply these transformations to the respective ciphertexts using message homomorphism. Next, a transformation from the key space function family is sampled for each input position, and these form the set of transformations to be output by the rerandomizing algorithm. These are applied using a key homomorphism to each of the ciphertexts, and the result set of ciphertexts (freshly permuted), along with the garbling, becomes the output rerandomized garbling.

Definition 14. *Let* GS *be a projective garbling scheme (Definition 3) with the input encoding alphabet* $\mathcal{L}$ *(Definition 3). Then* GS *is* ***compatible*** *with a strong KMHE scheme* $\mathcal{E}$ *(Definition 8) with message space* $\mathcal{M}$ *and message space function family* $\mathcal{F}_{\mathcal{M}}$ *if* $\mathcal{L} = \mathcal{M}$ *and* $\mathcal{F}_{\mathcal{M}}$ *is the space of all transformations* π_{En} *that the rerandomizing algorithm* $\mathsf{GS.Rerand}$ *can output.*

While evaluating such a garbling, given one key for each input position, one can correctly decrypt only one of the pair of the corresponding ciphertexts. As a result one label of the base garbling is derived for each input position and this garbling can then be evaluated. This complete scheme is formalized below.

Protocol PRGS with Constant-Sized Labels

Let κ be a computational security parameter. Let KMH be a strong KMHE scheme that is compatible with the projective RGS RGS. Then the computationally secure RGS operates as follows:

– $F, e \leftarrow \mathsf{Gb}(f; r)$

$$F', e' = \{\mathsf{L}_i^0, \mathsf{L}_i^1\}_{i\in[\mathsf{m}]} \leftarrow \mathsf{RGS.Gb}(f)$$
$$\forall i \in [\mathsf{m}], b \in \{0,1\}, k_i^b \leftarrow \mathsf{KMH.Gen}(1^\kappa)$$
$$c_i^b = \mathsf{KMH.Enc}(k_i^b, \mathsf{L}_i^b)$$
$$F = (F', \{\pi_i[0] : c_i^0, \pi_i[1] : c_i^1\}_{i\in[\mathsf{m}]}) \qquad e = \{k_i^0, k_i^1\}_{i\in[\mathsf{m}]}$$

– $F', \pi_{\mathsf{En}} \leftarrow \mathsf{Rerand}(F; r')$

$$F^*, \sigma^* = \{\sigma_i\}_{i\in[\mathsf{m}]} \leftarrow \mathsf{RGS.Rerand}(F')$$
$$\forall i \in [\mathsf{m}], \text{ sample } \rho_i \leftarrow \mathcal{F}_{\mathcal{K}}$$
$$\forall i \in [\mathsf{m}], b \in \{0,1\}, c_i^{b*} = \mathsf{KMH.Eval}(c_i^b, \rho_i, \sigma_i)$$
$$F' = (F^*, \{\pi_i[0] : c_i^{0*}, \pi_i[1] : c_i^{1*}\}_{i\in[\mathsf{m}]}) \qquad \sigma = \{\rho_i\}_{i\in[\mathsf{m}]}$$

– $X = \mathsf{En}(e, x)$

$$\text{parse } e = \{k_i^0, k_i^1\}_{i\in[\mathsf{m}]} \text{ and return } X = \{k_i^{x_i}\}_{i\in[\mathsf{m}]}$$

– $y = \mathsf{Ev}(F, X)$

$$\forall i \in [\mathsf{m}], \mathsf{L}_i^{x_i} = \mathsf{KMH.Dec}(k_i^{x_i}, c_i^{x_i})$$
$$f(x) = \mathsf{RGS.Ev}(F', X' = \{\mathsf{L}_i^{x_i}\}_{i\in[\mathsf{m}]})$$

Fig. 8 RGS with Constant-Sized Labels

Lemma 5. *Let* RGS *be a projective RGS (Definition 3) and* KMH *be a strong KMHE scheme (Definition 8) that is compatible (Definition 14). Then Figure 8 describes a computationally secure projective RGS (Definition 3) where each element in the projective encoding has size* $O(\kappa)$*, the computational security parameter.*

Proof. We begin with pointing out that the RGS in Figure 8 satisfies 'projective encoding' (Definition 3) as the input labels correspond to one-bit input semantic values. Each such label is a key generated from the strong KMHE scheme KMH whose size is $O(\kappa)$ for computational security parameter κ. Further, the 'correctness' (Definition 3) of Figure 8 follows from the 'correctness' of the base projective RGS and the 'correctness' and 'KMH correctness' of KMH. It therefore only remains to show that Figure 8 satisfies the remaining properties of a projective RGS, as we do below:

– **Privacy:** This means that for any function f and two inputs such that $f(x_0) = f(x_1)$, the following distributions need to be computationally indistinguishable:

$$\{F = (F', \{\pi_i[0] : c_i^0, \pi_i[1] : c_i^1\}_{i\in[\mathsf{m}]}), X = \{k_i^{x_i^0}\}_{i\in[\mathsf{m}]}\}_{(F,e)\leftarrow\mathsf{Gb}(f)\ ;\ X=\mathsf{En}(e,x_0)}$$

$$\{F = (F', \{\pi_i[0] : c_i^0, \pi_i[1] : c_i^1\}_{i\in[\mathsf{m}]}), X = \{k_i^{x_i^0}\}_{i\in[\mathsf{m}]}\}_{(F,e)\leftarrow\mathsf{Gb}(f)\ ;\ X=\mathsf{En}(e,x_1)}$$

In order to prove that the RGS in Figure 8 satisfies 'privacy', let us first consider the following set of hybrid distributions:

- $\mathsf{H}_0 = \{F = (F', \{\pi_i[0] : c_i^0, \pi_i[1] : c_i^1\}_{i\in[\mathsf{m}]}), X = \{k_i^{x_i^0}\}_{i\in[\mathsf{m}]}\}_{\substack{(F,e)\leftarrow\mathsf{Gb}(f)\\ X=\mathsf{En}(e,x_0)}}$: This is identical to the first distribution above that contains a garbling and an input encoding corresponding to x_0.
- $\mathsf{H}_1 = \{F = (F', \{\pi_i[0] : c_i^0, \pi_i[1] : c_i^1\}_{i\in[\mathsf{m}]}), X = \{k_i^{x_i^0}\}_{i\in[\mathsf{m}]}\}$: In this hybrid experiment, first the garbling F' is generated using the base garbling scheme $(F', e') \leftarrow \mathsf{RGS.Gb}(f)$ and then the input x_0 is encoded as $X' = \{\mathsf{L}_i^{x_0^i}\}_{i\in[\mathsf{m}]} = \mathsf{RGS.En}(e, x_0)$. For every input bit position $i \in [\mathsf{m}]$, the ciphertexts are then created as follows. The ciphertext $c_i^{x_0^i} = \mathsf{KMH.Enc}(k_i^{x_0^i}, \mathsf{L}_i^{x_0^i})$ encrypts the active label while the other ciphertext $c_i^{\neg x_0^i} = \mathsf{KMH.Enc}(k_i^{\neg x_0^i}, 0^{|\mathsf{L}_i^{x_0^i}|})$ encrypts a 0-string of the same length as the input label. Finally, the overall input encoding algorithm $X = \mathsf{En}(e, x_0)$ is used to sample the keys to be used for evaluation. Note that this distribution differs from the previous only in that the ciphertexts encrypting the inactive input labels are now replaced by ciphertexts encrypting the all-0-string.
- $\mathsf{H}_2 = \{F = (F', \{\pi_i[0] : c_i^0, \pi_i[1] : c_i^1\}_{i\in[\mathsf{m}]}), X = \{k_i^{x_i^1}\}_{i\in[\mathsf{m}]}\}$: In this hybrid experiment, first the garbling F' is generated using the base garbling scheme $(F', e') \leftarrow \mathsf{RGS.Gb}(f)$ and then the input x_1 is encoded as $X' = \{\mathsf{L}_i^{x_1^i}\}_{i\in[\mathsf{m}]} = \mathsf{RGS.En}(e, x_1)$. For every input bit position $i \in [\mathsf{m}]$, the ciphertexts are then created as follows. The ciphertext $c_i^{x_1^i} = \mathsf{KMH.Enc}(k_i^{x_1^i}, \mathsf{L}_i^{x_1^i})$ encrypts the active label while the other ciphertext $c_i^{\neg x_1^i} = \mathsf{KMH.Enc}(k_i^{\neg x_1^i}, 0^{|\mathsf{L}_i^{x_1^i}|})$ encrypts a 0-string of the same length as the input label. Finally, the overall input encoding algorithm $X = \mathsf{En}(e, x_1)$ is used to sample the keys to be used for evaluation. Note that this distribution differs from the previous only in that the input now encoded is x_1 instead of x_0.
- $\mathsf{H}_3 = \{F = (F', \{\pi_i[0] : c_i^0, \pi_i[1] : c_i^1\}_{i\in[\mathsf{m}]}), X = \{k_i^{x_i^1}\}_{i\in[\mathsf{m}]}\}_{\substack{(F,e)\leftarrow\mathsf{Gb}(f)\\ X=\mathsf{En}(e,x_1)}}$: This is identical to the second distribution given above that contains a garbling and an input encoding corresponding to x_1. This differs from the previous hybrid in that the ciphertexts encrypting the all-0-string are replaced with encryptions of the real inactive input labels of the base garbling.

Consider for the sake of contradiction that the RGS in Figure 8 does not satisfy 'privacy' (Definition 3) and there exists a PPT distinguisher D that can, with non-negligible probability ϵ, distinguish H_0 and H_3. Then D should be able to distinguish either H_0 and H_1, H_1 and H_2, or H_2 and H_3 with advantage $> \frac{\epsilon}{3}$, which is non-negligible.
We first show that if D can distinguish between hybrids H_0 and H_1 (resp. H_2 and H_3), then it can be used in a black-box way by a PPT adversary $\mathcal{A}_{\mathsf{KMH}}$ to violate the 'CPA security' of KMH. In order to show this, let us further consider the following set of hybrids:

- $\mathsf{H}'_0 = \{F = (F', \{\pi_i[0] : c_i^0, \pi_i[1] : c_i^1\}_{i\in[\mathsf{m}]}), X = \{k_i^{x_i^0}\}_{i\in[\mathsf{m}]}\}_{\substack{(F,e)\leftarrow\mathsf{Gb}(f)\\ X=\mathsf{En}(e,x_0)}}$: This is the same as the hybrid distribution H_0 given above.

- $\forall j \in [\mathsf{m}], \mathsf{H}'_j = \{F = (F', \{\pi_i[0] : c_i^0, \pi_i[1] : c_i^1\}_{i\in[\mathsf{m}]}), X = \{k_i^{x_i^0}\}_{i\in[\mathsf{m}]}\}$: In this hybrid experiment, all the ciphertexts encrypting the inactive labels are created as in hybrid H_1, up to the jth ciphertext and the rest are created as in H_0. The rest of this distribution (the garbling and active keys) is created identically as both H_0 and H_1.

Note that the last hybrid H'_{m} is the same as H_1. If D can distinguish between hybrids H_0 and H_1 with some non-negligible probability ϵ', then it follows that there must exist an$j \in [\mathsf{m}]$ such that D can distinguish H'_{j-1} and H'_j with advantage $> \frac{\epsilon'}{\mathsf{m}}$. which is non-negligible. We show that such a D can be used in a black-box way by a PPT adversary $\mathcal{A}_{\mathsf{KMH}}$ to violate the 'CPA security' of KMH as follows:

- $\mathcal{A}_{\mathsf{KMH}}$ works by first using the function f to create the garbling $(F', e') \leftarrow \mathsf{RGS.Gb}(f)$ and then the input x_0 to create the input encoding $X' = \{\mathsf{L}_i^{x_0^i}\}_{i\in[\mathsf{m}]} = \mathsf{RGS.En}(e, x_0)$. For each $i \in [\mathsf{m}]$, it samples a key $k_i^{x_0^i} \leftarrow \mathsf{KMH.Gen}(1^\kappa)$ and creates the ciphertext $c_i^{x_0^i} = \mathsf{KMH.Enc}(k_i^{x_0^i}, \mathsf{L}_i^{x_0^i})$. It sets $X = \{k_i^{x_i^0}\}_{i\in[\mathsf{m}]}$.
- For each $i < j$, $\mathcal{A}_{\mathsf{KMH}}$ samples a key $k_i^{\neg x_0^i} \leftarrow \mathsf{KMH.Gen}(1^\kappa)$ and creates the ciphertext $c_i^{\neg x_0^i} = \mathsf{KMH.Enc}(k_i^{\neg x_0^i}, 0^{|\mathsf{L}_i^{\neg x_0^i}|})$.
- For each $i > j$, $\mathcal{A}_{\mathsf{KMH}}$ samples a key $k_i^{\neg x_0^i} \leftarrow \mathsf{KMH.Gen}(1^\kappa)$ and creates the ciphertext $c_i^{\neg x_0^i} = \mathsf{KMH.Enc}(k_i^{\neg x_0^i}, \mathsf{L}_i^{\neg x_0^i})$.
- For $i = j$, $\mathcal{A}_{\mathsf{KMH}}$ prepares messages $m_0 = \mathsf{L}_i^{\neg x_0^i}$ and $m_1 = 0^{|\mathsf{L}_i^{\neg x_0^i}|}$ and sends this to the challenger. It receives a ciphertext c that is the encryption of either message under an unknown key, and sets $c_i^{\neg x_0^i} = c$. Note that when the challenger encrypts m_0, the complete distribution would correspond to that of H'_j and otherwise to would correspond to that of H'_{j-1}.
- Finally, $\mathcal{A}_{\mathsf{KMH}}$ sends the complete tuple $(F = (F', \{\pi_i[0] : c_i^0, \pi_i[1] : c_i^1\}_{i\in[\mathsf{m}]}), X = \{k_i^{x_i^0}\}_{i\in[\mathsf{m}]})$ to the distinguisher D and outputs whatever it outputs.

Note that $\mathcal{A}_{\mathsf{KMH}}$ has the same advantage as D, which is non-negligible. However, since KMH satisfies CPA security, no such $\mathcal{A}_{\mathsf{KMH}}$ can exist and it follows that D cannot distinguish any such set of adjacent distributions either.

Next, we show that if D can distinguish between hybrids H_1 and H_2, then it can be used in a black-box way by a PPT adversary $\mathcal{A}_{\mathsf{RGS}}$ to violate the 'privacy' of RGS as follows:

- $\mathcal{A}_{\mathsf{RGS}}$ receives from the challenger, a tuple (F', X') where F' is a garbling of f under RGS and $X' = \{\mathsf{L}_i\}_{i\in[\mathsf{m}]}$ either encodes the input x_0 or x_1.
- For each position $i \in [\mathsf{m}]$, $\mathcal{A}_{\mathsf{RGS}}$ samples a key $k_i \leftarrow \mathsf{KMH.Gen}(1^\kappa)$ and creates the ciphertext $c_i = \mathsf{KMH.Enc}(k_i, \mathsf{L}_i)$. It sets $X = \{k_i\}_{i\in[\mathsf{m}]}$.
- For each position $i \in [\mathsf{m}]$, $\mathcal{A}_{\mathsf{RGS}}$ samples a key $k'_i \leftarrow \mathsf{KMH.Gen}(1^\kappa)$ and creates the ciphertext $c'_i = \mathsf{KMH.Enc}(k'_i, 0^{|\mathsf{L}_i|})$. Note that if x_0 was encoded, this results in a distribution identical to that in H_1 and otherwise, this yields H_2.
- Finally, $\mathcal{A}_{\mathsf{RGS}}$ sends the complete tuple $(F = (F', \{\pi_i[0] : c_i, \pi_i[1] : c'_i\}_{i\in[\mathsf{m}]}), X = \{k_i\}_{i\in[\mathsf{m}]})$ to the distinguisher D and outputs whatever it outputs.

Note that $\mathcal{A}_{\mathsf{RGS}}$ has the same advantage as D, which is non-negligible. However, since RGS satisfies privacy, no such $\mathcal{A}_{\mathsf{RGS}}$ can exist and it follows that D cannot distinguish these distributions either.

– Rerand-**Privacy:** This means that for any function f and input x, the following distributions need to be computationally indistinguishable:

$$\{r, F_0, X_0\}_{(F,e)\leftarrow\mathsf{Gb}(f;r),\ (F_0,\pi_{\mathsf{En}})\leftarrow\mathsf{Rerand}(F),\ X_0=\mathsf{En}(\pi_{\mathsf{En}}(e),x)}$$

$$\{r, F_1, X_1\}_{(F_1,e_1)\leftarrow\mathsf{Gb}(f),\ X_1=\mathsf{En}(e_1,x)}$$

In order to prove that the RGS in Figure 8 satisfies Rerand-privacy, let us first consider the following set of hybrid distributions:

- $\mathsf{H}_0 = \{r, F, X\}_{\substack{(F,e)\leftarrow\mathsf{Gb}(f),\\ X=\mathsf{En}(e,x)}}$: This hybrid is identically distributed to the second distribution above. The garbling F and set of active labels X in the tuple is freshly generated and unrelated to the randomness r.

- $\mathsf{H}_1 = \{r, F, X\}$: This is an intermediate hybrid distribution where $F = (F', \{\pi_i[0] : c_i^0, \pi_i[1] : c_i^1\}_{i\in[\mathsf{m}]})$ is generated by first creating $(F', e') \leftarrow \mathsf{RGS.Gb}(f)$ as a garbling that is fresh, unrelated to the randomness in r. Then, r is parsed to derive the keys $\{k_i^0, k_i^1\}_{i\in[\mathsf{m}]}$ used in the prior garbling and for each $i \in [\mathsf{m}]$, a function $f_i \leftarrow \mathcal{F}_\mathcal{K}$ is sampled. For the input $x = \{x_i\}_{i\in[\mathsf{m}]}$, the new keys are set to be $k_i^{x_i'} = f_i(k_i^{x_i})$ and $k_i^{\neg x_i'} \leftarrow \mathsf{KMH.Gen}(1^\kappa)$ respectively and these are used to create the set of ciphertexts (as fresh encryptions of the labels in e') within F. Finally, X contains of all the active keys chosen according to the input x.
  This distribution differs from the previous hybrid experiment only in how the active encryption keys are derived (as opposed to freshly sampled).
- $\mathsf{H}_2 = \{r, F, X\}$: This is an intermediate hybrid distribution where $F = (F', \{\pi_i[0] : c_i^0, \pi_i[1] : c_i^1\}_{i\in[\mathsf{m}]})$ is generated by first creating $(F', e' = \{\mathsf{L}_i^{0'}, \mathsf{L}_i^{1'}\}_{i\in[\mathsf{m}]}) \leftarrow \mathsf{RGS.Gb}(f)$ as a garbling that is fresh, unrelated to the randomness in r. Then, r is parsed to derive the keys $\{k_i^0, k_i^1\}_{i\in[\mathsf{m}]}$ used in the prior garbling and for each $i \in [\mathsf{m}]$, a function $f_i \leftarrow \mathcal{F}_\mathcal{K}$ is sampled. Then the new set of keys are determined exactly as in the previous hybrid, as are the ciphertexts encrypting the active input labels corresponding to input x. For each $i \in [\mathsf{m}]$, the ciphertexts $c_i^{\neg x_i}$ are generated using a key $k_i^{\neg x_i'} \leftarrow \mathsf{KMH.Gen}(1^\kappa)$ to encrypt a message $0^{|\mathsf{L}_i^{x_i}|}$. Finally, X contains of all the active keys chosen according to the input x.
  This distribution differs from the previous hybrid experiment only in how the ciphertext of the inactive labels are created (as the underlying message is replaced by the all-0-string).
- $\mathsf{H}_3 = \{r, F, X\}$: This is an intermediate hybrid distribution where the base garbling $F' \in F$ is a rerandomized garbling. This is generated by first using the garbling randomness $r^* \in r$ to create $(F^*, e^*) \leftarrow \mathsf{RGS.Gb}(f; r^*)$ and then rerandomizing to derive $(F', \pi) \leftarrow \mathsf{RGS.Rerand}(F^*)$. The active input labels are derived as $X' = \mathsf{RGS.En}(\pi(e), x)$. The rest of the hybrid is created just like the previous one.
- $\mathsf{H}_4 = \{r, F, X\}$: This is an intermediate hybrid experiment that is generated exactly as the previous hybrid, with the exception that all the inactive ciphertexts are generated by homomorphically transforming those of the prior garbling into rerandomized labels and keys (instead of being fresh ciphertexts under fresh keys encrypting the all-0-string).
- $\mathsf{H}_5 = \{r, F, X\}_{\substack{(F^*,e)\leftarrow\mathsf{Gb}(f;r),\\ (F,\pi_{\mathsf{En}})\leftarrow\mathsf{Rerand}(F^*),\ X=\mathsf{En}(\pi_{\mathsf{En}}(e),x)}}$: This hybrid is identically distributed to the first distribution above, where the garbling and the corresponding input labels are all rerandomized from a prior garbling generated using r.

If there existed a PPT distinguisher D that could distinguish between hybrids H_0 and H_5 with non-negligible advantage ϵ, it must follow that D should be able to distinguish between adjacent pairs of hybrids with advantage $> \frac{\epsilon}{5}$, which is non-negligible.

We first show that hybrids H_0 and H_1 are statistically closely distributed. For this, let us consider the following set of $\mathsf{m}+1$ hybrid distributions:

- $\mathsf{H}'_0 = \{r, F, X\}_{\substack{(F,e)\leftarrow\mathsf{Gb}(f),\\ X=\mathsf{En}(e,x)}}$: This is identically distributed as H_0.
- $\forall j \in [\mathsf{m}], \mathsf{H}'_j = \{r, F, X\}$: This is an intermediate hybrid wherein the active keys $k_i^{x_i}$ up to $i < j$ are computed as in hybrid H_1 and the rest are computed as in H_0.

Note that the hybrid H'_m is identically distributed as H_1. If there existed an unbounded distinguisher D' that can distinguish H_0 and H_1 with non-negligible advantage ϵ, then it must be able to distinguish H'_j and H'_{j-1} for some $j \in [\mathsf{m}]$ with advantage $> \frac{\epsilon}{\mathsf{m}}$ which is non-negligible. However the difference between such identical hybrids originates only from how the key $k_j^{x_j}$ is generated: freshly from Gen in H'_{j-1} and as $f_j(k_j^{x_j'})$ in H'_j, where both distributions contain $k_j^{x_j'}$. If this could be distinguished, then it would follow that D' can violate the 'key privacy' property of KMH, yielding a contradiction.

Next, we show that hybrids H_1 and H_2 are computationally indistinguishable by considering the following set of hybrid distributions:

- $\mathsf{H}'_0 = \{r, F, X\}$: This is identically distributed as H_1.
- $\forall j \in [\mathsf{m}], \mathsf{H}'_j = \{r, F, X\}$: This is an intermediate hybrid wherein the inactive ciphertexts $c_i^{\neg x_i}$ up to $i < j$ are computed as in hybrid H_2 and the rest are computed as in H_1.

Note that the hybrid H'_{m} is identically distributed as H_2. If the PPT distinguisher D could distinguish between H_1 and H_2 with non-negligible advantage ϵ' then it must be able to distinguish H'_j and H'_{j-1} for some $j \in [\mathsf{m}]$ with advantage $> \frac{\epsilon'}{\mathsf{m}}$ which is also non-negligible. We show that such a D can be used in a black-box way by a PPT adversary $\mathcal{A}_{\mathsf{KMH}}$ to violate the CPA security of KMH in the following way:

- $\mathcal{A}_{\mathsf{KMH}}$ works by generating $(F', e' = \{\mathsf{L}_i^{0'}, \mathsf{L}_i^{1'}\}_{i\in[\mathsf{m}]}) \leftarrow \mathsf{RGS.Gb}(f)$ as a garbling that is fresh. Then, a different garbling randomness r is parsed to derive the keys $\{k_i^0, k_i^1\}_{i\in[\mathsf{m}]}$ used in the prior garbling and for each $i \in [\mathsf{m}]$, a function $f_i \leftarrow \mathcal{F}_{\mathcal{K}}$ is sampled.
- For the input $x = \{x_i\}_{i\in[\mathsf{m}]}$, $\mathcal{A}_{\mathsf{KMH}}$ sets the new active keys to be $k_i^{x'_i} = f_i(k_i^{x_i})$. For each $i \in [\mathsf{m}]$, the ciphertexts encrypting the active labels are created as $c_i^{x_i} = \mathsf{KMH.Enc}(k_i^{x'_i}, \mathsf{L}_i^{x'_i})$ and the key $k_i^{x'_i}$ becomes part of X.
- For each $i < j$, $\mathcal{A}_{\mathsf{KMH}}$ samples a fresh key $k_i^{\neg x'_i} \leftarrow \mathsf{Gen}(1^\kappa)$ and creates the ciphertext $c_i^{\neg x_i} = \mathsf{KMH.Enc}(k_i^{\neg x'_i}, 0^{|\mathsf{L}_i^{x'_i}|})$.
- For each $i > j$, $\mathcal{A}_{\mathsf{KMH}}$ samples a fresh key $k_i^{\neg x'_i} \leftarrow \mathsf{Gen}(1^\kappa)$ and creates the ciphertext $c_i^{\neg x_i} = \mathsf{KMH.Enc}(k_i^{\neg x'_i}, \mathsf{L}_i^{\neg x'_i})$.
- For $i = j$, $\mathcal{A}_{\mathsf{KMH}}$ sends to the challenger $m_0 = 0^{|\mathsf{L}_i^{x'_i}|}$ and $m_1 = \mathsf{L}_i^{\neg x'_i}$, and receives a ciphertext that it sets as $c_i^{\neg x_i}$. Note that if this encrypts m_0 then the overall distribution produced is as in H'_j and like H'_{j-1} otherwise.
- Finally, $\mathcal{A}_{\mathsf{KMH}}$ sends $(r, F = (F', \{\pi_i[0] : c_i^0, \pi_i[1] : c_i^1\}_{i\in[\mathsf{m}]}), X = \{k_i^{x'_i}\}_{i\in[\mathsf{m}]})$ to D and outputs whatever it outputs.

Note that $\mathcal{A}_{\mathsf{KMH}}$ has the same advantage as D, which is non-negligible. However, since KMH satisfies CPA security, no such $\mathcal{A}_{\mathsf{KMH}}$ can exist and it follows that D cannot distinguish these distributions either. We show that if D could distinguish between H_2 and H_3 with non-negligible advantage, then it can be used in a black-box way by a PPT adversary $\mathcal{A}_{\mathsf{RGS}}$ to violate Rerand-privacy of RGS as follows:

- $\mathcal{A}_{\mathsf{RGS}}$ receives from the challenger a tuple (r', F', X') and parses $X' = \{\mathsf{L}_i^{x_i}\}_{i\in[\mathsf{m}]}$.
- For each $i \in [\mathsf{m}]$, the keys $\{k_i^0, k_i^1\}_{i\in[\mathsf{m}]}$ used in the prior garbling are derived from randomness r and a function $f_i \leftarrow \mathcal{F}_{\mathcal{K}}$ is sampled. For the input $x = \{x_i\}_{i\in[\mathsf{m}]}$, $\mathcal{A}_{\mathsf{RGS}}$ sets the new active keys to be $k_i^{x'_i} = f_i(k_i^{x_i})$ and the ciphertexts encrypting the active labels are created as $c_i^{x_i} = \mathsf{KMH.Enc}(k_i^{x'_i}, \mathsf{L}_i^{x_i})$. The key $k_i^{x'_i}$ becomes part of the set X.
- For each $i \in [\mathsf{m}]$, $\mathcal{A}_{\mathsf{RGS}}$ samples a fresh key $k_i^{\neg x'_i} \leftarrow \mathsf{Gen}(1^\kappa)$ and creates the ciphertext $c_i^{\neg x_i} = \mathsf{KMH.Enc}(k_i^{\neg x'_i}, 0^{|\mathsf{L}_i^{x'_i}|})$.
- Finally, $\mathcal{A}_{\mathsf{RGS}}$ sends $(r, F = (F', \{\pi_i[0] : c_i^0, \pi_i[1] : c_i^1\}_{i\in[\mathsf{m}]}), X = \{k_i^{x'_i}\}_{i\in[\mathsf{m}]})$ to D and outputs whatever it outputs.

Note that $\mathcal{A}_{\mathsf{RGS}}$ has the same advantage as D, which is non-negligible. However, since RGS satisfies Rerand-privacy, no such $\mathcal{A}_{\mathsf{RGS}}$ can exist and it follows that D cannot distinguish these distributions either. Next, we show that hybrids H_3 and H_4 are computationally indistinguishable by considering the following set of hybrid distributions:

- $\mathsf{H}'_0 = \{r, F, X\}$: This is identically distributed as H_3.
- $\forall j \in [\mathsf{m}], \mathsf{H}'_j = \{r, F, X\}$: This is an intermediate hybrid wherein the inactive ciphertexts $c_i^{\neg x_i}$ up to $i < j$ are computed as in hybrid H_4 and the rest are computed as in H_3.

Note that the hybrid H'_{m} is identically distributed as H_4. If the PPT distinguisher D could distinguish between H_3 and H_4 with non-negligible advantage ϵ' then it must be able to distinguish H'_j and H'_{j-1} for some $j \in [\mathsf{m}]$ with advantage $> \frac{\epsilon'}{\mathsf{m}}$ which is also non-negligible. We show that such a D can be used in a black-box way by a PPT adversary $\mathcal{A}_{\mathsf{KMH}}$ to violate the KMH privacy of KMH in the following way:

- $\mathcal{A}_{\mathsf{KMH}}$ first uses the function f and randomness $r' \in r$ to create the garbling $(F', e' = \{\mathsf{L}_i^0, \mathsf{L}_i^1\}_{i\in[\mathsf{m}]}) \leftarrow \mathsf{RGS.Gb}(f; r')$ and then it is rerandomized to $(F'', e'' = \{\mathsf{L}_i^{0'}, \mathsf{L}_i^{1'}\}_{i\in[\mathsf{m}]})$. Let $\{c_i^0, c_i^1\}_{i\in[\mathsf{m}]}$ be the set of ciphertexts generated using the randomness r and let $\{k_i^0, k_i^1\}_{i\in[\mathsf{m}]}$ be the corresponding keys.

- For all $i \neq j \in [\mathsf{m}]$, $\mathcal{A}_{\mathsf{KMH}}$ samples $f_i \leftarrow \mathcal{F}_{\mathcal{K}}$ and computes $k_i^{x_i'} = f_i(k_i^{x_i})$. It then creates the active ciphertexts $c_i^{x_i'} = \mathsf{KMH.Enc}(k_i^{x_i'}, \mathsf{L}_i^{x_i'})$. If $i < j$ then use f_i and the g_i (used to transform $\mathsf{L}_i^{\neg x_i}$ to $\mathsf{L}_i^{\neg x_i'}$) to derive the ciphertext $c_i^{\neg x_i'} = \mathsf{KMH.Eval}(c_i^{\neg x_i}, f_i, g_i)$. If $i > j$, then sample a new key $k_i^{\neg x_i'} \leftarrow \mathsf{KMH.Gen}(1^\kappa)$ and compute $c_i^{\neg x_i'} = \mathsf{KMH.Enc}(k_i^{\neg x_i'}, 0^{|\mathsf{L}_i^{x_i'}|})$.
- For $i = j$, $\mathcal{A}_{\mathsf{KMH}}$ creates $k_0 = k_j^{x_j}$ and $k_1 = k_j^{\neg x_j}$. After sending these to the challenger, it accepts $f(k_0) = k_j^{x_j'}$ from the challenger and sends it a tuple of the form $(m = 0^{|\mathsf{L}_j^{\neg x_j}|}, g = g_j, c = c_j^{\neg x_j})$. Here g_i is derived from the rerandomizing randomness used to transform $\mathsf{L}_j^{\neg x_j}$ to $\mathsf{L}_j^{\neg x_j'}$.
- The challenger returns to $\mathcal{A}_{\mathsf{KMH}}$ a ciphertext that it sets as $c_j^{\neg x_j'}$. If this is a ciphertext under a fresh key then the resulting distribution is as in H'_{j-1} and otherwise it is as in H'_j.
- Finally, $\mathcal{A}_{\mathsf{KMH}}$ sends the tuple $(r, F = (F', \{\pi_i[0] : c_i^{0'}, \pi_i[1] : c_i^{1'}\}_{i\in[\mathsf{m}]}), X = \{k_i^{x_i'}\}_{i\in[\mathsf{m}]})$ to D and outputs whatever it outputs.

Note that $\mathcal{A}_{\mathsf{KMH}}$ has the same advantage as D, which is non-negligible. However, since KMH satisfies KMH privacy, no such $\mathcal{A}_{\mathsf{KMH}}$ can exist and it follows that D cannot distinguish these distributions either. We finally show that hybrids H_4 and H_5 are statistically closely distributed. For this, let us consider the following set of $\mathsf{m}+1$ hybrid distributions:

- $\mathsf{H}'_0 = \{r, F, X\}$: This is identically distributed as H_4.
- $\forall j \in [\mathsf{m}], \mathsf{H}'_j = \{r, F, X\}$: This is an intermediate hybrid wherein the active ciphertexts $c_i^{x_i}$ up to $i < j$ are computed as in hybrid H_5 and the rest are computed as in H_4.

Note that the hybrid H'_{m} is identically distributed as H_5. If there existed an unbounded distinguisher D' that can distinguish H_4 and H_5 with non-negligible advantage ϵ, then it must be able to distinguish H'_j and H'_{j-1} for some $j \in [\mathsf{m}]$ with advantage $> \frac{\epsilon}{\mathsf{m}}$ which is non-negligible. However the difference between such identical hybrids originates only from how the ciphertext $c_j^{x_j}$ is generated: freshly using Enc in H'_{j-1} and homomorphically using Eval in H'_j, where both distributions contain all the randomness, key and message of the prior ciphertext $c_j^{x_j'}$. If this could be distinguished, then it would follow that D' can violate the '(weak) KMH privacy' property of KMH, yielding a contradiction.

– **Randomness Composition:** In order to show that the RGS in Figure 8 satisfies randomness composition, it suffices to describe the function $\mathsf{Explain}$ that can combine two consecutive garbling and rerandomizing randomness to compute a valid garbling randomness for this RGS. This function would operate as follows:

- Parse the garbling randomness as,
$$(r, \{k_i^0, k_i^1, r_i^0, r_i^1\}_{i\in[\mathsf{m}]})$$
- Parse the rerandomizing randomness as,
$$(r', \{f_i, r_i^{0'}, r_i^{1'}\}_{i\in[\mathsf{m}]})$$
- Compute the composed randomness of RGS :
$$r^* = \mathsf{RGS.Explain}(r, r')$$
This can be done since RGS is a projective RGS satisfying randomness composition.
- For each $i \in [\mathsf{m}]$, compute the new input labels and ciphertext randomnesses as,
$$k_i^{0*} = f_i(k_i^0)$$
$$k_i^{1*} = f_i(k_i^1)$$
$$r_i^{0*} = r_i^0 \circ r_i^{0'}$$
$$r_i^{1*} = r_i^1 \circ r_i^{1'}$$
This can be done due to KMH correctness property of KMH.
- return,
$$(r^*, \{k_i^{0*}, k_i^{1*}, r_i^{0*}, r_i^{1*}\}_{i\in[\mathsf{m}]})$$

This concludes the proof.

Realizing Strong KMHE from El Gamal Encryption. We now present an instantiation of the strong KMHE scheme (Definition 8) that is compatible (Definition 14) with the information-theoretic RGS construction presented in Construction 2. For a small constant $w \in \mathbb{N}$[9], such a strong KMHE scheme would need to support a message space $\mathcal{M}$ that can encode width-w permutation matrices and a function space $\mathcal{F}_{\mathcal{M}}$ containing compositions with width-w permutation matrices. Although [GHV10, AHKP22] already present an instantiation of strong KMHE from [BHHO08], assuming the hardness of the DDH problem, it admits large ciphertext sizes: $O(\kappa^3)$ bits. In contrast, what we require for our application is much simpler and, although still relying on DDH, it admits much smaller ciphertext sizes: $O(\kappa)$ bits. We describe our construction below.

Construction 3 *Let $\mathbb{G}$ be a cyclic group of order q and generator g where DDH is hard. Let $\{\alpha_i\}_{i\in[w]} \in \mathbb{G}^w$. Let S_w be the group of permutations on a vector of w elements: $[\alpha_1, \cdots, \alpha_w]$. Then the algorithms in $\mathsf{KMH} = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec}, \mathsf{Eval})$ are initialized as follows:*

- *$\mathcal{K} = \mathbb{Z}_q$ is the key space.*
- *$\mathcal{F}_{key} = \{f_{a,b}(k) = ak + b\}_{a,b\in\mathbb{Z}_q}$ is the key space function family.*
- *$\mathcal{C} = \mathbb{G}^{2\times w}$ is the ciphertext space.*
- *$\mathcal{M} = S_w$ is the message space that contains all permutations of the vector $[\alpha_1, \cdots, \alpha_w] \in \mathbb{G}^w$.*
- *$\mathcal{F}_{msg} = S_w$ is the message space function family.*

These are used to instantiate the following functions:

1. $\mathsf{sk} \leftarrow \mathsf{Gen}(1^\kappa)$*:*
   - *sample* $\mathsf{sk} \leftarrow \mathcal{K}$
   - *output* - sk
2. $\mathsf{pk} \leftarrow \mathsf{PKgen}(\mathsf{sk})$*:*
   - *input* - $\mathsf{sk} \in \mathbb{Z}_q$
   - *output* - $\mathsf{pk} = g^{\mathsf{sk}} \in \mathbb{G}$
3. $c \leftarrow \mathsf{Enc}(\mathsf{sk}, m)$*:*
   - *input - key* $\mathsf{sk} \in \mathbb{Z}_q$ *and message* $m = \{m_i\}_{i\in[w]} \in \mathbb{G}^w$
   - *compute* $\mathsf{pk} \leftarrow \mathsf{PKgen}(\mathsf{sk})$
   - $\forall i \in [w]$*, sample* $r_i \leftarrow \mathbb{Z}_q$*, compute* $c_i = (m_i \cdot \mathsf{pk}^{r_i}, g^{r_i})$
   - *output* - $c = (\mathsf{pk}, \{c_i\}_{i\in[w]} \in \mathbb{G}^{2\times w})$
4. $m = \mathsf{Dec}(\mathsf{sk}, c)$*:*
   - *input - key* $\mathsf{sk} \in \mathbb{Z}_q$ *and ciphertext* $c = (\mathsf{pk}, \{c_i\}_{i\in[w]})$
   - $\forall i \in [w]$*, parse* $c_i = (c_1, c_2)$ *and compute* $m_i = \frac{c_1}{c_2^{\mathsf{sk}}}$
   - *output* - $m = \{m_i\}_{i\in[w]} \in \mathbb{G}^w$
5. $c' \leftarrow \mathsf{Eval}(c, f, \pi)$*:*
   - *input* - $f_{a,b} \in \mathcal{F}_{key}$ *and permutation* $\pi : [w] \to [w] \in \mathcal{F}_{msg}$
   - *input - ciphertext* $c = (\mathsf{pk}, \{c_i\}_{i\in[w]} \in \mathbb{G}^{2\times w})$
   - $\forall i \in [w]$*, sample* $r_i \leftarrow \mathbb{Z}_q$*, compute* $c'_i = (c_1 \cdot \mathsf{pk}^{r_i}, c_2 \cdot g^{r_i})$ *using* $c_i = (c_1, c_2)$
   - $\forall i \in [w]$*, compute* $c''_i = (c_1 \cdot c_2^{\frac{b}{a}}, c_2^{\frac{1}{a}})$ *using* $c'_i = (c_1, c_2)$
   - *output* - $c' = (\mathsf{pk}^a \cdot g^b, \{\pi[i] : c''_i\}_{i\in[w]})$

Lemma 6. *Assuming the hardness of the DDH problem, Construction 3 is a strong KMHE scheme (Definition 8) that is compatible (Definition 14) with the RGS in Construction 2.*

Proof. Construction 3 satisfies 'correctness' (Definition 8) by definition. It therefore only remains to show that the remaining properties of a strong KMHE scheme are satisfied, as we do below:

[9] typically $2 \leq w \leq 5$

– **Key Privacy:** This property for Construction 3 would require that,

$$\{\mathsf{sk}, a \cdot \mathsf{sk} + b\}_{\mathsf{sk},a,b \leftarrow \mathcal{SK}} \overset{s}{\approx} \{\mathsf{sk}, \mathsf{sk}'\}_{\mathsf{sk},\mathsf{sk}' \leftarrow \mathcal{SK}}$$

In fact, these are identical distributions as $(a \cdot \mathsf{sk} + b)$ is uniformly distributed over $\mathcal{SK}$ if each of (sk, a, b) are uniformly distributed over $\mathcal{SK}$.

– **KMH correctness:** This property for Construction 3 would require that the following are identically distributed,

$$\{\mathsf{sk}, \{m_i\}_{i\in[w]}, \{r_i\}_{i\in[w]}, (\mathsf{pk}' = g^{a\cdot\mathsf{sk}+b}, \{\pi[i] : (m_i \cdot \mathsf{pk}'^{r_i+r'_i}, g^{r_i+r'_i})\}_{i\in[w]})\}_{\substack{\mathsf{sk},a,b\leftarrow\mathcal{SK},\pi\leftarrow S_w,\\ \{r_i,r'_i\}_{i\in[w]}\leftarrow\mathcal{SK}^{10}}}$$

$$\{\mathsf{sk}, \{m_i\}_{i\in[w]}, \{r_i\}_{i\in[w]}, (\mathsf{pk}' = g^{a\cdot\mathsf{sk}+b}, \{\pi[i] : (m_i \cdot \mathsf{pk}'^{r^*_i}, g^{r^*_i})\}_{i\in[w]})\}_{\substack{\mathsf{sk},a,b\leftarrow\mathcal{SK},\pi\leftarrow S_w,\\ \{r^*_i\}_{i\in[w]}\leftarrow\mathcal{SK}^{w}}}$$

This holds from the construction in how the functions Enc and Eval are instantiated since for each $i \in [w]$, the values $r_i + r'_i$ and r^*_i are identically distributed.

– **CPA Security:** This property for Construction 3 would require that there exists no PPT distinguisher D that can win the CPA security game. That is, for any number of pairs of messages $m^j_0, m^j_1 \in \mathcal{M}$ of its choice,

$$\{\mathsf{Enc}(\mathsf{sk}, m^j_0)\}_{j\in[n]} \overset{c}{\approx} \{\mathsf{Enc}(\mathsf{sk}, m^j_1)\}_{j\in[n]}$$

This property can be reduced to the CPA security of the underlying El Gamal encryption scheme, that holds assuming the hardness of the DDH problem. Consider for the sake of contradiction that there exists a PPT distinguisher D that can distinguish the above distributions with non-negligible advantage ϵ. We show that this can be used in a black-box way by a PPT adversary $\mathcal{A}$ to violate the CPA security of the El Gamal encryption scheme as follows:

- $\mathcal{A}$ receives from the challenger $\mathcal{C}$ a public key pk.
- For polynomially many times, $\mathcal{A}$ receives from D a pair of messages $m^j_0, m^j_1 \in \mathcal{M}$. It parses the messages as $m^j_0 = \{m^j_{0,i}\}_{i\in[w]}$ and $m^j_1 = \{m^j_{1,i}\}_{i\in[w]}$. For $i \in [w]$, it sends the pair $(m^j_{0,i}, m^j_{1,i})$ to $\mathcal{C}$ and receives a ciphertext c_i encrypting $m^j_{b,i}$. $\mathcal{A}$ sends $(\mathsf{pk}, \{c_i\}_{i\in[w]})$ to D.
- Finally, $\mathcal{A}$ outputs whatever D outputs.

In the above execution, $\mathcal{A}$ has the same advantage as D, which is non-negligible. However, since the El Gamal encryption scheme is CPA secure, no such $\mathcal{A}$ can exist and therefore no such D can exist.

– **KMH Privacy:** This property for Construction 3 would require that there exist no PPT distinguisher that can win the KMH privacy game. That is, for any sk_0, sk_1 and $f(\mathsf{sk}_1) = a \cdot \mathsf{sk}_1 + b \mod q$ given by the challenger $\mathcal{C}$, the following holds:

$$\{m, m', g, c \leftarrow \mathsf{Enc}(\mathsf{sk}_0, m), c_0 \leftarrow \mathsf{Eval}(c, f, g)\}_{\substack{\mathsf{sk}_0,\mathsf{sk}_1\in\mathsf{supp}(\mathsf{Gen}(1^\kappa)),\\ m,m'\in\mathcal{M},g\in\mathcal{F}_{msg}}}$$

$$\overset{c}{\approx}\{m, m', g, c \leftarrow \mathsf{Enc}(\mathsf{sk}_0, m), c_1 \leftarrow \mathsf{Enc}(\mathsf{sk}', m')\}_{\substack{\mathsf{sk}_0,\mathsf{sk}_1\in\mathsf{supp}(\mathsf{Gen}(1^\kappa)),\mathsf{sk}'\leftarrow\mathsf{Gen}(1^\kappa),\\ m,m'\in\mathcal{M},g\in\mathcal{F}_{msg}}}$$

We show this by considering the following hybrid distributions:

- $\mathsf{H}_0 = \{\mathsf{sk}_0, \mathsf{sk}_1, f(\mathsf{sk}_1), m, m', g, c \leftarrow \mathsf{Enc}(\mathsf{sk}_0, m), c_0 \leftarrow \mathsf{Eval}(c, f, g)\}_{\substack{\mathsf{sk}_0,\mathsf{sk}_1\in\mathsf{supp}(\mathsf{Gen}(1^\kappa)),\\ m,m'\in\mathcal{M},g\in\mathcal{F}_{msg}}}$

  This is identically distributed as the first distribution above, where the final ciphertext c_0 is a product of rerandomizing the ciphertext c created by the distinguisher.
- $\mathsf{H}_1 = \{\mathsf{sk}_0, \mathsf{sk}_1, f(\mathsf{sk}_1), m, m', g, c \leftarrow \mathsf{Enc}(\mathsf{sk}_0, m), c'_0 \leftarrow \mathsf{Enc}(f(\mathsf{sk}_0), g(m))\}_{\substack{\mathsf{sk}_0,\mathsf{sk}_1\in\mathsf{supp}(\mathsf{Gen}(1^\kappa)),\\ m,m'\in\mathcal{M},g\in\mathcal{F}_{msg}}}$

  This is an intermediate hybrid experiment where the last ciphertext c'_0 is generated freshly, but encrypts the same message using the same key as in the previous hybrid. The rest of the distribution is also the same as that in H_0. Note that, owing to KMH correctness of this encryption scheme, $\mathsf{H}_0 \equiv \mathsf{H}_1$.

- $\mathsf{H}_2 = \{\mathsf{sk}_0, \mathsf{sk}_1, f(\mathsf{sk}_1), m, m', g, c \leftarrow \mathsf{Enc}(\mathsf{sk}_0, m), c'_1 \leftarrow \mathsf{Enc}(\mathsf{sk}', g(m))\}_{\substack{\mathsf{sk}_0,\mathsf{sk}_1 \in \mathsf{supp}(\mathsf{Gen}(1^\kappa)),\\ \mathsf{sk}' \leftarrow \mathsf{Gen}(1^\kappa),\\ m,m' \in \mathcal{M}, g \in \mathcal{F}_{msg}}}$
  This is an intermediate hybrid experiment where the last ciphertext c'_1 is generated freshly, encrypting the same message $g(m)$ but with a freshly sampled key sk' that is unrelated to $\mathsf{sk}_0, \mathsf{sk}_1, f(\mathsf{sk}_1)$ that the adversary receives from the challenger. The rest of the distribution is also the same as that in H_1. This distribution is identical to that of the previous hybrid experiment since,
  $$\{\mathsf{sk}_0, \mathsf{sk}_1, f_{a,b}(\mathsf{sk}_1), f_{a,b}(\mathsf{sk}_0)\} \equiv \{\mathsf{sk}_0, \mathsf{sk}_1, f_{a,b}(\mathsf{sk}_1), \mathsf{sk}'\}$$
  over random choices of $a, b \leftarrow \mathbb{Z}_q$, any $\mathsf{sk}_0, \mathsf{sk}_1$ and random choice of secret key $\mathsf{sk}' \leftarrow \mathbb{Z}_q$. Therefore, we have that $\mathsf{H}_1 \equiv \mathsf{H}_2$.
- $\mathsf{H}_3 = \{\mathsf{sk}_0, \mathsf{sk}_1, f(\mathsf{sk}_1), m, m', g, c \leftarrow \mathsf{Enc}(\mathsf{sk}_0, m), c_1 \leftarrow \mathsf{Enc}(\mathsf{sk}', m')\}_{\substack{\mathsf{sk}_0,\mathsf{sk}_1 \in \mathsf{supp}(\mathsf{Gen}(1^\kappa)),\\ \mathsf{sk}' \leftarrow \mathsf{Gen}(1^\kappa),\\ m,m' \in \mathcal{M}, g \in \mathcal{F}_{msg}}}$
  This hybrid experiment outputs a distribution that is identical to the final distribution. Here the last ciphertext c_1 is generated freshly and encrypts a fresh message m' of the adversary's choice with a fresh secret key sk'. The rest of the distribution is also the same as that in H_2.

We proceed with showing that $\mathsf{H}_3 \overset{c}{\approx} \mathsf{H}_2$ using a reduction to the CPA security of the El Gamal encryption scheme, that holds assuming the hardness of the DDH problem. Consider for the sake of contradiction that there exists a PPT distinguisher D that can distinguish the above hybrid distributions H_2 and H_3 with non-negligible advantage ϵ. We show that this can be used in a black-box way by a PPT adversary $\mathcal{A}$ to violate the CPA security of the El Gamal encryption scheme as follows:

- $\mathcal{A}$ receives from the challenger $\mathcal{C}$ a public key pk.
- $\mathcal{A}$ receives $(\mathsf{sk}_0, \mathsf{sk}_1)$ from D.
  It samples $f_{a,b} \leftarrow \mathcal{F}_{key}$ and sends $f(\mathsf{sk}_1)$ to D.
- For polynomially many times,
  - $\mathcal{A}$ receives (c, g, m') from D where $m' \in \mathcal{M}$ and $g \in \mathcal{F}_{msg}$. It computes $m = \mathsf{Dec}(\mathsf{sk}_0, c)$.
  - Parsing $m' = \{m'_i\}_{i\in[w]}$ and $g(m) = \{m_i\}_{i\in[w]}$, for each $i \in [w]$, it sends the pair (m'_i, m_i) to $\mathcal{C}$ and receives a ciphertext c_i.
  - $\mathcal{A}$ sends $(\mathsf{pk}, \{c_i\}_{i\in[w]})$ to D.
- Finally, $\mathcal{A}$ outputs whatever D outputs.

In the above execution, $\mathcal{A}$ has the same advantage as D, which is non-negligible. However, since the El Gamal encryption scheme is CPA secure, no such $\mathcal{A}$ can exist and therefore no such D can exist.

This concludes the proof.

3.2 Instantiating UOT Protocol

We re-visit the definition of updatable oblivious transfer (UOT) from [AHKP24]. This is a functionality between a chooser, a sender and several updaters. The chooser has as input a choice bit b and the sender has as input two strings $a_0, a_1 \in \{0,1\}^\ell$. The UOT functionality accepts these and returns the string $S_0 = a_b$. Then each jth updater has as input a transformation $\sigma : \{0,1\}^\ell \to \{0,1\}^\ell$ which the functionality accepts and returns $S_j = \sigma(S_{j-1})$. We require a protocol that securely realizes this in the presence of a semi-honest adversary corrupting the chooser, and all-but-one of the sender and updaters.

Recall that we use UOT in the SCALES protocol to ensure that an evaluator of the final function garbling receives the correct labels corresponding the the function inputs in order to correctly evaluate. Each client first acts as the chooser in an independent UOT protocol, using their input bit to the SCALES protocol as the choice bit. The garbler acts as the sender for it. For the portion of the projective input that is determined by this input bit, it uses the labels with the 0 and 1 semantic values as the inputs to the UOT senders message. Each rerandomizer acts as an updater that inputs a transformation that would update these labels. Finally, the clients reveal the labels corresponding to the inputs to the final garbling. For this it needs to only decode the UOT message from the last server and so its computation does not grow with the number of computation servers or the function size.

Definition 15. *Let* GS *be a projective garbling scheme (Definition 3) with projective input alphabet* $\mathcal{L}$ *(Definition 3). Then* GS *is* ***compatible*** *with a UOT protocol* UOT *(Definition 12) if* $\mathcal{L} \equiv \mathcal{S}$*, where* $\mathcal{S}$ *is the space of each input of the sender* S *in the UOT scheme* UOT*. Further,* $\mathcal{F}_{\mathcal{L}} \equiv \mathcal{F}_{\Sigma}$*, where* $\mathcal{F}_{\mathcal{L}} = \{\pi_{\mathsf{En}} : \mathcal{L} \to \mathcal{L}\}$ *is the space of all transformations that a rerandomizing algorithm could output, and* $\mathcal{F}_{\Sigma} = \{\sigma : \mathcal{S} \to \mathcal{S}\}$ *is the space of all update functions for* UOT*.*

In [AHKP24], UOT is constructed from a 2-round bit-OT with reusable first message and rerandomizable second message. In this section, we construct a UOT protocol that is compatible with the RGS from Lemma 2, with the following additional property.

Lemma 7. *There exists a UOT protocol (Definition 12) that is compatible (Definition 15) with the RGS in Lemma 2 where the size of each message in the protocol is* $O(\kappa)$ *bits, for computational security parameter* κ*.*

We construct the UOT protocol compatible with the RGS in Section 3.1 from linearly homomorphic encryption (LHE), formalized in Definition 7. The proof for the above lemma then follows directly from Lemma 8 in the ensuing text, when the LHE scheme is instantiated as in [CL15] assuming DDH over class groups of prime order. For completeness, we re-state this as Construction 1 in Section 2. Each ciphertext would consist of two group elements, making the overall complexity $O(\kappa)$-bits.

Updatable OT from LHE. We now present a construction of a semi-honestly secure UOT protocol $\mathsf{UOT} = (\mathsf{OT}_1, \mathsf{OT}_2, \mathsf{OT}_{\mathsf{updt}}, \mathsf{OT}_{\mathsf{fin}}, \mathsf{OT}_{\mathsf{vrfy}})$ (Definition 12) that uses LHE in a black-box way. This protocol closely follows the outline of the construction in [AIR01] and supports update functions in the class of all affine functions over the sender's message space.

Construction 4 *Let* $\mathsf{LHE} = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec}, \mathsf{Eval})$ *defined on a domain of public and private keys, messages and ciphertexts* $\mathcal{PK}, \mathcal{SK}, \mathcal{M}, \mathcal{C}$*, be a linearly homomorphic scheme (Definition 7). Then the algorithms in* $\mathsf{UOT} = (\mathsf{OT}_1, \mathsf{OT}_2, \mathsf{OT}_{\mathsf{updt}}, \mathsf{OT}_{\mathsf{fin}}, \mathsf{OT}_{\mathsf{vrfy}})$ *are initialized as follows:*

- *The sender's message space* $\mathcal{M}$ *is the message space* $\mathcal{M}$ *in* LHE
- *The update function family is* $\Sigma = \{\sigma_{\{a_i\}_{i\in[n]},b}(\{m_i\}_{i\in[n]}) = b + \Sigma_{i\in[n]} a_i \cdot m_i\}_{\{a_i\}_{i\in[n]}\in\mathcal{M}^n, b\in\mathcal{M}}$

These are used to instantiate the following functions:

1. $(\mathsf{aux}, \mathbf{m}_1) \leftarrow \mathsf{OT}_1(b)$

$$(\mathsf{sk}, \mathsf{pk}) \leftarrow \mathsf{LHE.Gen}(1^{\kappa})$$
$$\mathbf{m}_1 \leftarrow \mathsf{LHE.Enc}(\mathsf{pk}, b)$$
$$\mathsf{aux} = \mathsf{sk}$$

2. $\mathbf{m}_2 \leftarrow \mathsf{OT}_2(\mathbf{m}_1, (s_0, s_1))$

$$\textit{parse } s_0, s_1 \in \mathcal{M} \textit{ and } \mathbf{m}_1 = c$$
$$\mathbf{m}_2 \leftarrow \mathsf{LHE.Eval}(c, (s_1 - s_0), s_0)$$

3. $\mathbf{m}_2' \leftarrow \mathsf{OT}_{\mathsf{updt}}(\mathbf{m}_2, \sigma)$

$$\textit{parse } \mathbf{m}_2 = c \textit{ and } \sigma = \sigma_{a,b}(m) = a \cdot m + b$$
$$\mathbf{m}_2' \leftarrow \mathsf{LHE.Eval}(c, a, b)$$

or alternatively, $\mathbf{m}_2' \leftarrow \mathsf{OT}_{\mathsf{updt}}(\{\mathbf{m}_2^i\}_{i\in[n]}, \sigma)$

$$\textit{parse each } \mathbf{m}_2^i = c_i \textit{ and } \sigma = \sigma_{\{a_i\}_{i\in[n]},b}(\{m_i\}_{i\in[n]}) = b + \Sigma_{i\in[n]} a_i \cdot m_i$$
$$\mathbf{m}_2' \leftarrow \mathsf{LHE.Eval}(\{c_i\}_{i\in[n]}, \{a_i\}_{i\in[n]}, b)$$

4. $M \leftarrow \mathsf{OT}_{\mathsf{fin}}(\mathsf{aux}, \mathbf{m}_2)$

$$\begin{aligned} &\textit{parse } \mathsf{aux} = \mathsf{sk} \textit{ and } \mathbf{m}_2 = c \\ &M = \mathsf{LHE.Dec}(\mathsf{sk}, c) \end{aligned}$$

5. $\{s_b, \perp\} \leftarrow \mathsf{OT}_{\mathsf{vrfy}}(M)$

$$\textit{return } M = s_b$$

Lemma 8. *Assume* LHE *is a linearly homomorphic encryption scheme (Definition 7), then Construction 4 is a semi-honestly secure UOT protocol (Definition 12) supporting the same message space $\mathcal{M}$ and the update function family containing all affine functions.*

Proof. We need to prove that the algorithms in UOT as in Construction 4 securely realize $\mathcal{F}_{\mathsf{UOT}}^{\mathcal{M},\Sigma}$ and in order to do so we consider the following PPT simulator $\mathsf{Sim}_{\mathsf{UOT}}$:

- $\mathsf{Sim}_{\mathsf{UOT}}$ receives the corrupt parties' inputs and the functionality output $s_{b,d}$.
- If the receiver is corrupt, $\mathsf{Sim}_{\mathsf{UOT}}$ executes $\mathsf{OT}_1(b)$ exactly as in the real execution and sets $\mathbf{m}_1, \mathsf{aux}$ as part of the view output. Otherwise, it generates a key pair $(\mathsf{sk}, \mathsf{pk}) \leftarrow \mathsf{Gen}(1^\kappa)$, samples a bit $b' \leftarrow \{0,1\}$ and sets only $\mathbf{m}_1 \leftarrow \mathsf{Enc}(\mathsf{pk}, b')$ as part of the view.
- If the sender is corrupt, $\mathsf{Sim}_{\mathsf{UOT}}$ executes $\mathsf{OT}_2(\mathbf{m}_2, (s_0, s_1))$ using the previously generated message and the sender's inputs exactly as in a real execution. Otherwise, if the receiver is corrupted and $\mathsf{Sim}_{\mathsf{UOT}}$ knows $s_{b,0}$, then it sets only $\mathbf{m}_2 \leftarrow \mathsf{Enc}(\mathsf{pk}, s_{b,0})$ as part of the view. If the receiver is honest, then it samples $s_{b,0} \leftarrow \mathcal{M}$ and sets only $\mathbf{m}_2 \leftarrow \mathsf{Enc}(\mathsf{pk}, s_{b,0})$ as part of the view.
- For each updater indexed $i \in [d-1]$, if it is corrupt, $\mathsf{Sim}_{\mathsf{UOT}}$ executes $\mathsf{OT}_{\mathsf{updt}}(\mathbf{m}_2^{i-1}, \sigma_i)$ using the previously generated message and the updater's inputs exactly as in a real execution. Otherwise, if the receiver is corrupted and $\mathsf{Sim}_{\mathsf{UOT}}$ knows $s_{b,i}$, then it sets only $\mathbf{m}_2^i \leftarrow \mathsf{Enc}(\mathsf{pk}, s_{b,i})$ as part of the view. If the receiver is honest, then it samples $s_{b,i} \leftarrow \mathcal{M}$ and sets only $\mathbf{m}_2 \leftarrow \mathsf{Enc}(\mathsf{pk}, s_{b,i})$ as part of the view.
- Corresponding to the last updater's message, $\mathsf{Sim}_{\mathsf{UOT}}$ knows $s_{b,d}$ and sets only $\mathbf{m}_2^d \leftarrow \mathsf{Enc}(\mathsf{pk}, s_{b,d})$ as part of the view.

This simulator produces the a view of the adversary that contains,

$$\begin{aligned} &\mathbf{m}_1, \mathbf{m}_2, \{\mathbf{m}_2^i\}_{i\in[d]}, s_{b,d} \\ &\mathsf{aux}, b, \{s_{b,i-1}\}_{i\in[d]} && \text{if R is corrupt} \\ &s_0, s_1 && \text{if S is corrupt} \\ &\sigma_i && \text{if } U_i \text{ is corrupt for any } i \in [d] \end{aligned}$$

Note that for the case where the receiver R is corrupt, the real and simulated views are identically distributed, owing the the 'privacy' property of LHE which states that a homomorphically transformed ciphertext and a freshly generated one encrypting the same message are identically distributed. Further, by the same argument, the last updater's message in the real and simulated views are also identically distributed. Therefore, going forward, we consider the simulated view only for the honest receiver, sender and updater case. In this case, only the last message $\mathbf{m}_2^d$ encrypts the correct plaintext $s_{b,d}$, while all other OT messages are ciphertexts that encrypt randomly generated messages as in the simulation.

Consider for the sake of contradiction that there exists a PPT distinguisher D that can distinguish the above distribution from that of the real execution with non-negligible probability ϵ. Then for some $i \in [d+1]$, this D can distinguish the hybrids H_{i-1} and H_i with probability at least $\frac{\epsilon}{d+1}$, for the set of hybrids as given below:

- $\mathsf{H}_0 = \{\mathbf{m}_1, \mathbf{m}_2, \{\mathbf{m}_2^i\}_{i\in[d]}, s_{b,d}\} \leftarrow \mathsf{Sim}_{\mathsf{UOT}}$ for the case of honest honest receiver, sender and updaters.
- $\mathsf{H}_1 = \{\mathbf{m}_1^*, \mathbf{m}_2, \{\mathbf{m}_2^i\}_{i\in[d]}, s_{b,d}\}$ where the receiver's message $\mathbf{m}_1^*$ was generated honestly as in the real execution and the rest of the messages are generated exactly as in the previous hybrid distribution.

- $\mathsf{H}_2 = \{\mathbf{m}_1^*, \mathbf{m}_2^*, \{\mathbf{m}_2^i\}_{i\in[d]}, s_{b,d}\}$ where the receiver's message $\mathbf{m}_1^*$ and the sender's message $\mathbf{m}_2^*$ was generated honestly as in the real execution and the rest of the messages are generated exactly as in the previous hybrid distribution.
- $\forall i \in [d-1], \mathsf{H}_{2+i} = \{\mathbf{m}_1^*, \mathbf{m}_2^*, \{\mathbf{m}_2^{i',*}\}_{i'\leq i}, \{\mathbf{m}_2^{i'}\}_{i'>i}, s_{b,d}\}$ where the receiver's, sender's and first i updaters messages are generated honestly as in the real execution and the rest of the messages are generated exactly as in the previous hybrid distribution.

If such a D exists, then it can be used in a black-box way by a PPT adversary $\mathcal{A}_{\mathsf{LHE}}$ to win the CPA security game of the LHE scheme as follows:

- $\mathcal{A}_{\mathsf{LHE}}$ receives from the challenger $\mathcal{C}$ the public key pk.
- $\mathcal{A}_{\mathsf{LHE}}$ has the index i and all the inputs b, (s_0, s_1) and $\{\sigma_j\}_{j\in[d]}$ of the UOT scheme UOT.
- $\mathcal{A}_{\mathsf{LHE}}$ works by creating the view of the first $i-1$ messages in the UOT transcript using the above inputs just as in the real UOT execution.
- For the ith message in the transcript, $\mathcal{A}_{\mathsf{LHE}}$ first sets m_0 as the message (from within $\{b, s_b, \{s_{b,j}\}_{j\in[d]}\}$ as required) whose ciphertext would be created as in the real execution. Next, it sets m_1 as a random message (from $\{0,1\}$ or $\mathcal{M}$ as required) whose ciphertext would be created as in the simulation.
- $\mathcal{A}_{\mathsf{LHE}}$ sends (m_0, m_1) to $\mathcal{C}$ and receives a ciphertext that it sets as the ith OT message $\mathbf{m}$.
- $\mathcal{A}_{\mathsf{LHE}}$ generates the rest of the messages of the UOT transcript as in the simulation and then the complete transcript is given to D.
- Finally, $\mathcal{A}_{\mathsf{LHE}}$ outputs whatever D outputs.

In the above execution, $\mathcal{A}_{\mathsf{LHE}}$ has the same advantage as D which is non-negligible. However, since LHE is a CPA secure LHE scheme, no such $\mathcal{A}_{\mathsf{LHE}}$ can exist and therefore no such D exists. This concludes the proof.

Upgrading to Malicious Security. The updatable OT protocol given above is semi-honestly secure and it remains to discuss how this can be upgraded for use in a maliciously secure SCALES protocol. For this, a similar approach as in [AHKP24] can be used: the sender and updaters' messages can remain unchanged since these will be coupled with black-box zero-knowledge proofs of compatibility with the garbling and rerandomizing. The receivers messages from OT_1 and $\mathsf{OT}_{\mathsf{fin}}$ need to be accompanied by NIZK proofs. Note that since the statements themselves are small, this proof does not need to be succinct in the statement. The algorithm $\mathsf{OT}_{\mathsf{vrfy}}$ (resp. OT_2) needs to produce the final UOT output (resp. sender's message) only if the proof from $\mathsf{OT}_{\mathsf{fin}}$ (resp. OT_1) verifies, and abort otherwise. This approach would additionally need the setup (CRS or RO) that the NIZK would require.

Equivalence of LHE and UOT. Next, we show that a UOT protocol is equivalent to a linearly homomorphic encryption scheme (that is function hiding). We do so by presenting a construction of an LHE scheme (Definition 7) that makes black-box use of a semi-honestly secure UOT protocol supporting affine update functions (Definition 12).

Construction 5 *Let* $\mathsf{UOT} = (\mathsf{OT}_1, \mathsf{OT}_2, \mathsf{OT}_{\mathsf{updt}}, \mathsf{OT}_{\mathsf{fin}}, \mathsf{OT}_{\mathsf{vrfy}})$ *be a semi-honestly secure UOT protocol (Definition 12 – with reusable receiver's message) with a ring* $\mathcal{M}$ *as the message domain and* $\Sigma = \{\sigma_{\{a_i\}_{i\in[n]},b}(\{m_i\}_{i\in[n]}) = b + \Sigma_{i\in[n]} a_i \cdot m_i\}_{\{a_i\}_{i\in[n]}\in\mathcal{M}^n, b\in\mathcal{M}}$ *as the function domain. Then the algorithms in* $\mathsf{LHE} = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec}, \mathsf{Eval})$ *are initialized as follows:*

- *The plaintext space* $\mathcal{M}$ *is the senders input space* $\mathcal{M}$
- *The ciphertext space* $\mathcal{C}$ *is the space of all senders (or updaters) messages* $\mathbf{m}_2$
- *The public key space* $\mathcal{PK}$ *is the space of all receivers messages* $\mathbf{m}_1$
- *The secret key space* $\mathcal{SK}$ *is the space of all receivers private state* aux

These are used to instantiate the following functions:

1. $(\mathsf{sk}, \mathsf{pk}) \leftarrow \mathsf{Gen}(1^\kappa)$

$$(\mathsf{aux}, \mathbf{m}_1) \leftarrow \mathsf{OT}_1(0)$$
$$\mathsf{pk} = \mathbf{m}_1$$
$$\mathsf{sk} = \mathsf{aux}$$

2. $c \leftarrow \mathsf{Enc}(\mathsf{pk}, m)$

$$\textit{parse } m \in \mathcal{M} \textit{ and } \mathsf{pk} = \mathbf{m}_1$$
$$c \leftarrow \mathsf{OT}_2(\mathbf{m}_1, (m, m))$$

3. $m = \mathsf{Dec}(\mathsf{sk}, c)$

$$\textit{parse } \mathsf{sk} = \mathsf{aux} \textit{ and } c = \mathbf{m}_2$$
$$m \leftarrow \mathsf{OT}_{\mathsf{vrfy}}(\mathsf{OT}_{\mathsf{fin}}(\mathsf{aux}, \mathbf{m}_2))$$

4. $c \leftarrow \mathsf{Eval}(\{c_i\}_{i\in[n]}, \{a_i\}_{i\in[n]}, b)$

$$\textit{parse each } c_i = \mathbf{m}_2^i \textit{ and } \sigma = \sigma_{\{a_i\}_{i\in[n]},b}(\{m_i\}_{i\in[n]}) = b + \Sigma_{i\in[n]} a_i \cdot m_i$$
$$c \leftarrow \mathsf{OT}_{\mathsf{updt}}(\{\mathbf{m}_2^i\}_{i\in[n]}, \sigma)$$

Lemma 9. *Assume* UOT *is a semi-honestly secure UOT protocol (Definition 12) where the update function family* Σ *contains all affine functions* $\sigma : \mathcal{M}^n \times \mathcal{M}^n \times \mathcal{M} \to \mathcal{M}$ *and the senders and updaters message spaces are identically distributed. Then Construction 5 is a linearly homomorphic encryption scheme (Definition 7).*

Proof. The 'correctness' of the linear homomorphic encryption scheme in Construction 5 follows from the fact that the protocol UOT securely and correctly realizes $\mathcal{F}_{\mathsf{UOT}}^{\mathcal{M},\Sigma}$. The 'privacy' of LHE (Definition 7) follows from the fact that the space of senders and updaters messages in the UOT protocol are identically distributed. It now remains to show that LHE additionally satisfies 'CPA security'.

Consider for the sake of contradiction that there exists a PPT distinguisher D that can violate the CPA security of the scheme LHE in Construction 5 with non-negligible advantage ϵ. That is, for any number of pairs $(m_i^0, m_i^1) \in \mathcal{M}^2$ it can distinguish between the following distributions:

$$\mathsf{H}_0 = \{\mathbf{m}_1, \{\mathbf{m}_2^i, m_i^0, m_i^1\}_{i\in[n]}\} \quad \substack{(\mathsf{aux},\mathbf{m}_1)\leftarrow\mathsf{OT}_1(0) \\ \forall i\in[n], \mathbf{m}_2^i \leftarrow \mathsf{OT}_2(\mathbf{m}_1,(m_i^0,m_i^0))}$$

$$\mathsf{H}_1 = \{\mathbf{m}_1, \{\mathbf{m}_2^i, m_i^0, m_i^1\}_{i\in[n]}\} \quad \substack{(\mathsf{aux},\mathbf{m}_1)\leftarrow\mathsf{OT}_1(0) \\ \forall i\in[n], \mathbf{m}_2^i \leftarrow \mathsf{OT}_2(\mathbf{m}_1,(m_i^1,m_i^1))}$$

Consider the following hybrid distribution that uses the PPT UOT simulator $\mathsf{Sim}_{\mathsf{UOT}}$ for the case where the sender and receiver are both honest:

$$\mathsf{H}' = \{\mathbf{m}_1, \{\mathbf{m}_2^i, m_i^0, m_i^1\}_{i\in[n]}\} \quad \substack{(\mathsf{aux},\mathbf{m}_1)\leftarrow\mathsf{Sim}_{\mathsf{UOT}}(1^\kappa) \\ \forall i\in[n], \mathbf{m}_2^i \leftarrow \mathsf{Sim}_{\mathsf{UOT}}(\mathbf{m}_1)}$$

The distinguisher should be able to distinguish between either H_0 and H' or H_1 and H' with advantage at least $\frac{\epsilon}{2}$. Then we can show that this distinguisher can be used in a black-box way by a PPT adversary $\mathcal{A}_{\mathsf{UOT}}$ to distinguish between a real and simulated view of the UOT protocol for an honest receiver and honest sender. For any $b \in \{0, 1\}$, $\mathcal{A}_{\mathsf{UOT}}$ would work as follows:

- $\mathcal{A}_{\mathsf{UOT}}$ sends the honest receiver's input bit 0 to the challenger and receives the message $\mathbf{m}_1$.
- $\mathcal{A}_{\mathsf{UOT}}$ sends $\mathsf{pk} = \mathbf{m}_1$ to the distinguisher D.
- For as many times as D wants, it sends $\mathcal{A}_{\mathsf{UOT}}$ two messages $m_i^0, m_i^1 \in \mathcal{M}$, that $\mathcal{A}_{\mathsf{UOT}}$ sends sends to the challenger.
- For any pair of messages the challenger receives, it sends back a message $\mathbf{m}_2^i$ that $\mathcal{A}_{\mathsf{UOT}}$ passes to the distinguisher. Note that if the challenger created all the messages using the simulator, then the resulting distribution that D sees is like H' and otherwise it sees H_b.
- Finally, $\mathcal{A}_{\mathsf{UOT}}$ outputs whatever D outputs.

In the above execution, $\mathcal{A}_{\mathsf{UOT}}$ has the same advantage as D that is non-negligible. However, since the UOT protocol securely realizes $\mathcal{F}_{\mathsf{UOT}}^{\mathcal{M},\Sigma}$, no such $\mathcal{A}_{\mathsf{UOT}}$ can exist and therefore no such D can exist.

4 Extending the Applicability of SCALES

In this section we present extensions that use secure SCALES protocols for bounded function classes given in the previous section. In Section 4.1 we present a general transformation from a semi-honestly (resp. maliciously) secure SCALES protocol for bounded function classes to a protocol for general function families with the complexity of each server growing only sub-linearly in the complete function size. Informally, this is done by re-writing the function as a composition of several functions that lie in the bounded function class and computing each, in sequence, using a separate SCALES protocol as described in Section 3. Although this comes at the cost of the number of client messages growing with the number of function components, the overall communication complexity is greatly reduced as compared to protocols in prior work [AHKP22].

In Section 4.2 we discuss techniques that can lift semi-honestly secure SCALES protocols to achieve guaranteed output delivery (GOD – in the dishonest majority setting) in the presence of malicious servers. This results in protocols secure in the presence of a mixed adversary that can corrupt any number of clients semi-honestly and all-but-one server in each phase maliciously. Note that such protocols, when used in conjunction with the transformation for SCALES with sub-linear server computation, can result in a secure SCALES with GOD in the presence of mixed adversaries that supports long on-going computations where the complete function may not be known in advance.

4.1 SCALES Protocols with Sub-Linear Servers

We present a SCALES protocol that does not require any server to perform computation proportional to the size of the function f being computed by the MPC functionality. This is done by introducing the following design aspects:

- Each function is defined to be 'layered'. That is, a function f can be represented as $f = f_n \circ \cdots \circ f_1$ – a composition of n layers of sub-functions. Each server in our protocols is required to do computation proportional to one such layer.
- As a consequence of the above, the total number of participating ephemeral servers becomes linearly proportional to the number of layers in the function.
- In these protocols, we can no longer let the client computation scale linearly with the number of participating servers. So care needs to be taken that the client computation is independent of the total number of servers.

On a high level, the SCALES protocol works as follows. We use the sub-functions $\{f_i\}_{i\in[n]}$ and a shallow-depth PRF that takes κ-bit keys and inputs in $[n-1]$ (the function index i for f_i), and outputs sufficiently long pseudo-random strings that are used to mask the outputs of these functions. The protocol internally uses SCALES protocols (with linear server computation) as a building-block. Our protocol sets each client's input as the input bit x_i (to be input only to f_1) and a share s_i of the PRF key (to be input to each SCALES sub-protocol). We have all the clients post one message each that acts as the first message for *all* the SCALES sub-protocols. Then our protocol starts by evaluating the first sub-function using a SCALES protocol and publicly posting its output masked by the output of the PRF. Every successive protocol uses the output of the previous protocol as a hard-coded input and computes a masked output of the next sub-function. The last protocol computes the final output and posts it in the clear. If the sub-protocol used is a semi-honestly secure SCALES protocol, then the SCALES protocol resulting from the compilation is semi-honestly secure. Similarly, if the sub-protocol is maliciously secure, then the final protocol is maliciously secure.

We formalize this paradigm below, starting with defining the following functionalities to be computed by the SCALES sub-protocols:

$$F_1: \quad \mathsf{input}: \quad \{x_i \in \{0,1\}, s_i \in \{0,1\}^\kappa\}_{i\in[\mathsf{m}]}$$
$$\mathsf{output}: \quad y_1 = f_1(x) \oplus \mathsf{PRF}(s,1) \quad \text{where} \quad s = \bigoplus_{i\in[\mathsf{m}]} s_i$$

$\forall j > 2 \in [n]$,

$$F_j: \quad \mathsf{input}: \quad \{s_i \in \{0,1\}^\kappa\}_{i\in[\mathsf{m}]}$$
$$\mathsf{output}: \quad y_j = f_j(y_{j-1} \oplus \mathsf{PRF}(s,j-1)) \oplus \mathsf{PRF}(s,j) \quad \text{where} \quad s = \bigoplus_{i\in[\mathsf{m}]} s_i$$

$$F_n: \quad \mathsf{input}: \quad \{s_i \in \{0,1\}^\kappa\}_{i\in[\mathsf{m}]}$$
$$\mathsf{output}: \quad y = f_n(y_{n-1} \oplus \mathsf{PRF}(s,n-1)) \quad \text{where} \quad s = \bigoplus_{i\in[\mathsf{m}]} s_i$$

We denote by $\{\Pi_k = \{\{\mathsf{C}_{\mathsf{msg}_p}^{\ i}\}_{i\in[\mathsf{m}]}, \{\mathsf{S}_{\mathsf{msg}_p}^{\ j}\}_{j\in[d]}\}_{p\in[P]}\}_{k\in\mathbb{N}}$ is a P-phase secure SCALES protocol family in terms of its next-message functions. Within it, for a protocol Π_j realizing a functionality F_j as above, $\mathsf{C}_{\mathsf{msg}_p}^{\ i}$ denotes the next-message function of the ith client for the pth phase. $\mathsf{S}_{\mathsf{msg}_p}^{\ j}$ denotes the computation of the jth server in the pth phase. Given this notation, we formalize a SCALES protocol with sub-linear server computation below.

Protocol SCALES Protocol for f with Sub-linear Servers

Let κ be a computational security parameter. Let $\mathcal{C} = \{C_i\}_{i\in[\mathsf{m}]}$ be the set of clients where each C_i has input bit x_i. Let $\mathcal{B}$ be the bulletin-board. Notation marked in gray is only applicable for F_1. Let $\{\Pi_k = \{\{\mathsf{C}_{\mathsf{msg}_p}^{\ i}\}_{i\in[\mathsf{m}]}, \{\mathsf{S}_{\mathsf{msg}_p}^{\ j}\}_{j\in[d]}\}_{p\in[P]}\}_{k\in\mathbb{N}}$ be a P-phase secure SCALES protocol family. The SCALES protocol for $f = f_n \circ \cdots \circ f_1$ operates as follows:

– $\forall i \in [\mathsf{m}]$, each client C_i with input bit x_i computes the following:

$$(\mathsf{aux}_i, m_i^1) \leftarrow \mathsf{C}_{\mathsf{msg}_1}^{\ i}(x_i)$$
$$\forall k \in [\kappa], \text{ sample } s_{i,k} \leftarrow \{0,1\} \quad (\mathsf{aux}_{k,i}, m_i^{k,1}) \leftarrow \mathsf{C}_{\mathsf{msg}_1}^{\ i}(s_{i,k})$$
$$\mathsf{publish} \qquad m_i^1, \{m_i^{k,1}\}_{k\in[\kappa]}$$

– $\forall \ell \in [n]$, let F_ℓ be the functionality to be computed corresponding to the ℓth layer sub-function f_ℓ.

- $\forall j \in [d]$, each server S_j computes the following:

$$m_j^1 \leftarrow \mathsf{S}_{\mathsf{msg}_1}^{\ j}(\{m_i^1, \{m_i^{k,1}\}_{k\in[\kappa]}\}_{i\in[\mathsf{m}]}, \{m_{j'}^1\}_{j'\in[j-1]})$$
$$\mathsf{publish} \qquad m_j^1$$

- For each phase $p \neq 1 \in [P]$,
  - $\forall i \in [\mathsf{m}]$, each client C_i computes the following:

$$m_i^p \leftarrow \mathsf{C}_{\mathsf{msg}_p}^{\ i}(\mathsf{aux}_i, m_d^{p-1})$$
$$\forall k \in [\kappa], \quad m_i^{k,p} \leftarrow \mathsf{C}_{\mathsf{msg}_p}^{\ i}(\mathsf{aux}_{k,i}, m_d^{p-1})$$
$$\mathsf{publish} \qquad m_i^p, \{m_i^{k,p}\}_{k\in[\kappa]}$$

  - $\forall j \in [d]$, each server S_j computes the following:

$$m_j^p \leftarrow \mathsf{S}_{\mathsf{msg}_p}^{\ j}(\{m_i^{p'}\{m_i^{k,p'}\}_{k\in[\kappa]}\}_{i\in[\mathsf{m}],p'<p}, \{m_j^{p'}\}_{j\in[d],p'<p})$$
$$\mathsf{publish} \qquad m_j^p$$

  - The output y_ℓ can be publicly inferred from $\{m_j^P\}_{j\in[d]}$
- all parties accept $y = y_n$ as the output of the protocol.

Fig. 9 Secure SCALES Protocol with Sub-linear Servers

Theorem 1. *Assuming* PRF *is a pseudo-random function with key space* $\{0,1\}^\kappa$ *and* $\{\Pi_k = \{\{\mathsf{C}_{\mathsf{msg}_p}^i\}_{i\in[\mathsf{m}]}, \{\mathsf{S}_{\mathsf{msg}_p}^j\}_{j\in[d]}\}_{p\in[P]}\}_{k\in\mathbb{N}}$ *is a* P*-phase semi-honestly secure SCALES protocol family (Definition 10) for a function family* $\mathcal{F}$ *where* $\{F_k\}_{k\in[n]} \subseteq \mathcal{F}$*, Figure 9 is a semi-honestly secure SCALES protocol (Definition 10) computing any function* f*.*

Proof. We need to show that the protocol in Figure 9 semi-honestly securely computes the SCALES functionality (Figure 3) for the function f if all underlying SCALES sub-protocols are semi-honestly secure. Letting $f = f_1 \circ \cdots \circ f_n$ be the different layers, we can re-write the protocol in the $\{\mathcal{F}_{F_i}\}_{i\in[n]}$-hybrid where each $\mathcal{F}_{F_i}$ is the SCALES functionality (Figure 3) for computing F_i (the function that internally unmasks the function input of f_i, computes the function, and then outputs a freshly masked output of f_i). The protocol would proceed as follows:

- Each client C_j sends its inputs $(x_j, \{s_{j,k}\}_{k\in[\kappa]})$ to $\mathcal{F}_{F_1}$ (out of which $\{s_{j,k}\}_{k\in[\kappa]}$ is also the input to all other functionalities in $\{\mathcal{F}_{F_i}\}_{i>1\in[n]}$) and receives y_1.
- For each $i > 1 \in [n]$, y_{i-1} determines $\mathcal{F}_{F_i}$ and all the clients C_j receive y_i.
- All clients set $y = y_n$ as their output.

We can now describe a PPT simulator Sim that can simulate the view of a semi-honest adversary (corrupting all-but-one server in each phase and all-but-one client) in the above SCALES protocol as follows:

- Sim has access to the inputs and randomness $\{x_j, \{s_{j,k}\}_{k\in[\kappa]}\}_{j\in\mathcal{A}}$ of all the corrupt parties.
- Sim enters $\{x_j\}_{j\in\mathcal{A}}$ into the functionality and receives $f(x)$.
- For each $i \in [n-1]$, letting ℓ_i be the output length of function f_i, Sim samples a string $y_i \in \{0,1\}^{\ell_i}$ uniformly at random.
- Sim returns to the adversary y_1 as the output of $\mathcal{F}_{F_1}$.
- For each $i > 1 \in [n-1]$, Sim sends y_i as the output of $\mathcal{F}_{F_i}$.
- Finally, Sim sends $y_n = f(x)$ to the adversary.

In order to show that the view produced by the above simulator is computationally indistinguishable from the real view, let us consider the following set of hybrids:

- H_0 : This hybrid experiment produces the view that is identically distributed as the simulated view above.

$$\mathsf{H}_0 = \{\{x_j, \{s_{j,k}\}_{k\in[\kappa]}\}_{j\in\mathcal{A}}, \{y_i\}_{i\in[n]}\} \leftarrow \mathsf{Sim}$$

- H_1 : This hybrid experiment uses the inputs of the honest parties to compute each element y_i that the adversary receives. The difference between this hybrid experiment and the simulation is that for each intermediate layer function f_i, its input is correctly unmasked, and then the output is correctly computed and masked with a different truly random string, instead of simply sampling a random string as the output as in the simulation. Note however, that the result of this masking is identically distributed as the output of the simulator.

$$\begin{aligned}\mathsf{H}_1 = \{&\{x_j, \{s_{j,k}\}_{k\in[\kappa]}\}_{j\in\mathcal{A}}, y_1 = f_1(x) \oplus r_1,\\ &\{y_i = r_i \oplus f_i(y_{i-1} \oplus r_{i-1})\}_{i>1\in[n-1]}, y_n = f(x)\}_{\{r_i \leftarrow \{0,1\}^{|f_i(y_{i-1}\oplus r_{i-1})|}\}_{i\in[n-1]}}\end{aligned}$$

- H_2 : This hybrid experiment is distributed identically as the real execution and differs from the previous hybrid in that the strings used to mask the intermediate outputs are pseudo-random instead of truly random. Further, the key used in the PRF is computed using shares contributed by both honest and corrupt clients alike. When there exists at least one honest client, the resulting key is independently

distributed as compared to the key-shares possessed by the corrupt parties and is therefore unknown to the adversary.

$$\mathsf{H}_2 = \{\{x_j, \{s_{j,k}\}_{k\in[\kappa]}\}_{j\in\mathcal{A}}, y_1 = f_1(x) \oplus r_1, \\ \{y_i = r_i \oplus f_i(y_{i-1} \oplus r_{i-1})\}_{i>1\in[n-1]}, y_n = f(x)\}_{\{r_i=\mathsf{PRF}(s,i);s=\bigoplus_{j\in[\mathsf{m}]} s_j\}_{i\in[n-1]}}$$

Consider for the sake of contradiction that there exists a PPT distinguisher D that can distinguish between the distributions output by H_1 and H_2 with non-negligible advantage ϵ. Then we show that this can be used in a black-box way by a PPT adversary $\mathcal{A}_{\mathsf{PRF}}$ to violate the pseudo-randomness of PRF. $\mathcal{A}_{\mathsf{PRF}}$ behaves as follows:

- $\mathcal{A}_{\mathsf{PRF}}$ has oracle access to either a truly random function $\mathcal{R}$ or a pseudo-random function PRF with a secret key.
- It uses the inputs x to first correctly compute the intermediate outputs z_i of all the sub-functions in $\{f_i\}_{i\in[n]}$.
- Then for each $i \in [n-1]$, $\mathcal{A}_{\mathsf{PRF}}$ queries the oracle with i and receives r_i as response. It sets $y_n = f(x)$ and $y_i = r_i \oplus z_i$ for each $i \in [n-1]$. Note that this is distributed as in H_1 if the oracle was a random function and as in H_2 otherwise.
- $\mathcal{A}_{\mathsf{PRF}}$ samples key shares $\{s_{j,k}\}_{k\in[\kappa],j\in\mathcal{A}}$ for the corrupt parties uniformly at random and sends the distinguisher D the tuple $(\{x_j, \{s_{j,k}\}_{k\in[\kappa]}\}_{j\in\mathcal{A}}, \{y_i\}_{i\in[n]})$.
- Finally, $\mathcal{A}_{\mathsf{PRF}}$ outputs whatever D outputs.

In the above execution, the advantage of $\mathcal{A}_{\mathsf{PRF}}$ is the same as that of D, which is non-negligible. However since PRF is a pseudo-random function, it follows that not such $\mathcal{A}_{\mathsf{PRF}}$ can exist and therefore no such D can exist. Therefore the hybrids H_1 and H_2 are computationally indistinguishable. This concludes the proof.

Theorem 2. *Assuming* PRF *is a pseudo-random function with key space* $\{0,1\}^\kappa$ *and* $\{\Pi_k = \{\{\mathsf{C}_{\mathsf{msg}_p}^{\ i}\}_{i\in[\mathsf{m}]}, \{\mathsf{S}_{\mathsf{msg}_p}^{\ j}\}_{j\in[d]}\}_{p\in[P]}\}_{k\in\mathbb{N}}$ *is a* P*-phase maliciously secure SCALES protocol family (Definition 11) for a function family* $\mathcal{F}$ *where* $\{F_k\}_{k\in[n]} \subseteq \mathcal{F}$*, Figure 9 is a maliciously secure SCALES protocol (Definition 11) computing any function* f*.*

Proof. We need to show that the protocol in Figure 9 maliciously securely computes the SCALES functionality (Figure 4) for the function f if all the underlying SCALES sub-protocols are maliciously secure. Letting $f = f_1 \circ \cdots \circ f_n$ be the different layers, we can re-write the protocol in the $\{\mathcal{F}_{F_i}\}_{i\in[n]}$-hybrid where each $\mathcal{F}_{F_i}$ is the SCALES functionality (Figure 4) for computing F_i (the function that internally unmasks the function input of f_i, computes the function, and then outputs a freshly masked output of f_i). The protocol would proceed as follows:

- Each client C_j sends its inputs $(x_j, \{s_{j,k}\}_{k\in[\kappa]})$ to $\mathcal{F}_{F_1}$ (out of which $\{s_{j,k}\}_{k\in[\kappa]}$ is also the input to all other functionalities in $\{\mathcal{F}_{F_i}\}_{i>1\in[n]}$) and, if the adversary does not cause the functionality to abort, they receive y_1.
- For each $i > 1 \in [n]$, y_{i-1} determines $\mathcal{F}_{F_i}$ and, again if the functionality does not abort, all the clients C_j receive y_i.
- Finally, all clients set $y = y_n$ as their output.

We can now describe a PPT simulator Sim that can simulate the view of a malicious adversary (corrupting all-but-one server in each phase and all-but-one client) in the above SCALES protocol as follows:

- Sim accepts the inputs and PRF key-shares $\{x_j, \{s_{j,k}\}_{k\in[\kappa]}\}_{j\in\mathcal{A}}$ of all the corrupt parties while emulating $\mathcal{F}_{F_1}$.
- Sim inspects if $\{x_j\}_{j\in\mathcal{A}}$ are valid inputs and then enters $\{x_j\}_{j\in\mathcal{A}}$ into the functionality and receives $f(x)$. Otherwise, it asks the functionality to abort.
- For each $i \in [n-1]$, letting ℓ_i be the output length of function f_i, Sim samples a string $y_i \in \{0,1\}^{\ell_i}$ uniformly at random.

– Sim completes the emulation of $\mathcal{F}_{F_1}$ by returning to the adversary y_1 as the output of $\mathcal{F}_{F_1}$ and waiting for OK. If this is not received, the simulation aborts.
– Otherwise, for each $i > 1 \in [n-1]$, Sim sends y_i as the output of $\mathcal{F}_{F_i}$ and similarly waits to receive OK. If this is not received, the simulation aborts.
– Finally, Sim sends $y_n = f(x)$ to the adversary, emulating $\mathcal{F}_{F_n}$ and if the adversary sends OK, then Sim sends OK to the functionality computing f.

Note that the above interaction of the adversary with the simulator in the ideal execution only aborts whenever the real execution would. If the simulation does not abort, in order to show that the view produced by the above simulator is computationally indistinguishable from the real view, let us consider the following set of hybrids:

– H_0 : This hybrid experiment produces the view that is identically distributed as the simulated view above, for when the interaction does not abort. This corresponds to the simulator accepting the adversary's inputs (that are always valid) and then using $f(x)$ received from the ideal functionality to emulate the outputs of all the functionalities $\mathcal{F}_{F_i}$, by outputting truly random strings for each.

$$\mathsf{H}_0 = \{\{x_j, \{s_{j,k}\}_{k\in[\kappa]}\}_{j\in\mathcal{A}}, \{y_i\}_{i\in[n]}\} \leftarrow \langle \mathcal{A}, \mathsf{Sim}\rangle$$

– H_1 : This hybrid experiment uses the inputs of the honest parties to compute each element y_i that the adversary receives. The difference between this hybrid experiment and the simulation above is that for each intermediate layer function f_i, its input is correctly unmasked, and then the output is correctly computed and masked with a different truly random string, instead of simply sampling a random string as the output as in the simulation. Note however, that the result of this masking is identically distributed as the output of the simulator.

$$\begin{aligned}\mathsf{H}_1 = \{\{x_j, \{s_{j,k}\}_{k\in[\kappa]}\}_{j\in\mathcal{A}}, y_1 = f_1(x) \oplus r_1,\\ \{y_i = r_i \oplus f_i(y_{i-1} \oplus r_{i-1})\}_{i>1\in[n-1]}, y_n = f(x)\}_{\{r_i \leftarrow \{0,1\}^{|f_i(y_{i-1}\oplus r_{i-1})|}\}_{i\in[n-1]}}\end{aligned}$$

– H_2 : This hybrid experiment is distributed identically as the real execution and differs from the previous hybrid in that the strings used to mask the intermediate outputs are pseudo-random instead of truly random. Further, the key used in the PRF is computed using shares contributed by both honest and corrupt clients alike. When there exists at least one honest client, the resulting key is independently distributed as compared to the key-shares chosen by the corrupt parties and is therefore unknown to the adversary.

$$\begin{aligned}\mathsf{H}_2 = \{\{x_j, \{s_{j,k}\}_{k\in[\kappa]}\}_{j\in\mathcal{A}}, y_1 = f_1(x) \oplus r_1,\\ \{y_i = r_i \oplus f_i(y_{i-1} \oplus r_{i-1})\}_{i>1\in[n-1]}, y_n = f(x)\}_{\{r_i = \mathsf{PRF}(s,i); s = \bigoplus_{j\in[m]} s_j\}_{i\in[n-1]}}\end{aligned}$$

Consider for the sake of contradiction that there exists a PPT distinguisher D that can distinguish between the distributions output by H_1 and H_2 with non-negligible advantage ϵ. Then we show that this can be used in a black-box way by a PPT adversary $\mathcal{A}_{\mathsf{PRF}}$ to violate the pseudo-randomness of PRF. $\mathcal{A}_{\mathsf{PRF}}$ behaves as follows:

– $\mathcal{A}_{\mathsf{PRF}}$ has oracle access to either a truly random function $\mathcal{R}$ or a pseudo-random function PRF with a secret key.
– $\mathcal{A}_{\mathsf{PRF}}$ first receives from D the inputs $\{x_j\}_{j\in\mathcal{A}}$ and key shares $\{s_{j,k}\}_{k\in[\kappa],j\in\mathcal{A}}$ of the corrupt parties.
– $\mathcal{A}_{\mathsf{PRF}}$ then uses the inputs x (of the honest parties and those chosen by D for the corrupt parties) to first correctly compute the intermediate outputs z_i of all the sub-functions in $\{f_i\}_{i\in[n]}$.
– Then for each $i \in [n-1]$, $\mathcal{A}_{\mathsf{PRF}}$ queries the oracle with i and receives r_i as response. It sets $y_n = f(x)$ and $y_i = r_i \oplus z_i$ for each $i \in [n-1]$. Note that this is distributed as in H_1 if the oracle was a random function and as in H_2 otherwise.
– $\mathcal{A}_{\mathsf{PRF}}$ sends the distinguisher D the tuple $\{y_i\}_{i\in[n]}$.

– Finally, $\mathcal{A}_{\mathsf{PRF}}$ outputs whatever D outputs.

In the above execution, the advantage of $\mathcal{A}_{\mathsf{PRF}}$ is the same as that of D, which is non-negligible. However since PRF is a pseudo-random function, it follows that not such $\mathcal{A}_{\mathsf{PRF}}$ can exist and therefore no such D can exist. Therefore the hybrids H_1 and H_2 are computationally indistinguishable. This concludes the proof.

Efficiently instantiating the SCALES sub-protocols. In the semi-honest setting, the SCALES protocol that we present in Section 3 can be used to instantiate each sub-protocol in the above construction. Note that our protocol works for any function with a PBP representation and, assuming there exist PRFs in $\mathbf{NC}^1$, this is a 2-phase protocol with server computation $O(|F_i|\kappa)$ (where $|F_i| = w\ell$ if F_i can be represented using a length-ℓ width-w PBP), and client computation $O(\kappa)$. In the malicious setting, one can use the protocol of [AHKP24], with the RGS instantiated as in Section 3.1 and UOT protocol as in Section 3.2. This results in a 3-phase protocol in the CRS model with server computation $O(|F_i|\kappa^2)$ and and client computation $O(\kappa^2)$. In summary, instantiating the protocols this way results in the following:

Corollary 1. *Assuming the hardness of the DDH problem, there exists a semi-honestly secure SCALES protocol with clients posting $n+1$ messages and server computation linear in $\frac{1}{n}^{th}$ of the functionality size.*

Corollary 2. *Assuming the hardness of the DDH problem, there exists a maliciously secure SCALES protocol with clients posting $2n+1$ messages and server computation linear in $\frac{1}{n}^{th}$ of the functionality size.*

4.2 Towards Achieving GOD in SCALES

The SCALES model typically features protocols in the dishonest majority setting, and therefore, all known maliciously secure protocols in the SCALES model achieve security with abort. It remains open to design a protocol that achieves guaranteed output delivery (GOD), even at the cost of relaxing the corruption model. In the ensuing text, we informally outline how one can design such a protocol in the presence of a *mixed* adversary that can maliciously corrupt a (dishonest majority) fraction of the server pool and *semi-honestly* corrupt any number of clients.

Note that SCALES protocols in this relaxed model are not subject to the lower bound of [Cle86], which shows that achieving GOD in the dishonest-majority setting is impossible. The reason is twofold. First, [Cle86] applies only in the fully malicious setting and does not rule out cases where some parties are semi-honest. This distinction has been observed before; for example, [HM20] achieves GOD for generic MPC in the dishonest-majority setting under mixed adversaries. Second, in SCALES protocols, the corruption model (semi-honest vs. malicious) depends on the party's role (client vs. server). One might argue that functionalities such as randomness generation, which do not involve inputs, could be carried out solely by the servers, reducing the setting to one of purely malicious corruption and thus falling under Cleve's impossibility. However, SCALES protocols handle only deterministic functionalities, whereas randomized functionalities are implemented by having the clients supply shares of the randomness needed for the functionality. Since the clients can only be semi-honestly corrupted, Cleve's impossibility result does not apply.

We discuss two techniques that can be used to upgrade any semi-honestly secure SCALES protocol to a protocol with guaranteed output delivery in the presence of such a mixed adversary – offering protocols that are secure when all-but-one server in each phase is maliciously corrupted and any number of clients is semi-honestly corrupted. A central observation for both transformations is that in order to lift from semi-honest security to GOD in the presence of malicious servers, it only remains to specify measures to be taken against corrupt servers deviating from the protocol. Not only should honest parties be able to detect malicious behaviour, but we also require that no such behaviour can cause the protocol to abort.

Note that the structure of a SCALES protocol, wherein each server's computation is sequential and in independent rounds, makes it very convenient to rewind the state of the protocol to before a malicious server's message (that could potentially cause it to abort). Taking advantage of this fact, here is the generic approach we apply: we can have each computation server prove in (non-interactive) zero-knowledge that it performed its tasks semi-honestly. If a server is corrupt and its proof does not verify, its participation in the protocol is

simply ignored, and the server elected in the next phase carries on the computation from the previous step. However, for such a protocol to be secure, all the involved parties, including the lightweight clients, must be convinced of the server's computation. There are two approaches that could enable this, and we outline them below.

Using zk-SNARKs. One way to force semi-honest server computation is through the use of generic zk-SNARKs. Although non-black-box, there exist such proofs for any statement capturing the PPT computation of the server. Further, these proofs are non-interactive, fitting well with our stateless servers (provers); and support succinct public verification that can be carried out even by computationally bounded clients.

The protocol satisfying GOD would roughly proceed as follows. In each phase, after the clients have (semi-honestly) posted their message, in the first round, a computation server follows the semi-honest protocol and, along with its message, posts a zk-SNARK proving that it followed the base protocol. Such a proof can be verified by clients and servers alike. For every successive round, the computation server first verifies the computation of all the servers before it. If there exists a proof from a corrupt server that does not verify, this server simply ignores that servers contribution to the computation (and all the protocol messages that build upon it). Instead, it extends the computation of the last server on the semi-honest chain of computation. Once again, all clients and servers can keep track of the overall state of the protocol, effectively re-winding the protocol to ignore corrupt computation. At the end of the phase, all clients can still recognize the last semi-honest server message and post their message (starting the next phase) as a function of it. This ensures that servers deviating from the semi-honest SCALES protocol specification cannot cause the computation to abort.

Using Modular NIZKs. While the above protocol is of theoretical interest, an unfortunate drawback of the outlined protocol is that zk-SNARKs incur very expensive prover computation. This serves as motivation to try and fit cheaper non-interactive zero-knowledge (NIZK) proofs where verification is not-necessarily succinct into our use-case. An observation that makes this possible is that most generic zk-SNARKs and NIZKs are modular: the statement to be proven can be broken down into many related atomic sub-statements, each of which can be potentially proven separately[10]. Therefore, when a verifier finds a proof that is incorrect, it can point to the first such sub-proof that does not verify. Light-weight parties can then be convinced by checking only that sub-proof. A similar approach has been used in [BCR+24] to achieve GOD in the YOSO setting in the presence of dishonest-majority computation committees.

On a high level, a SCALES protocol using such NIZKs would work the same as the previous approach, but with the following exceptions. The use of zk-SNARKs for computation servers can be replaced by NIZKs and each computation server additionally posts, for all proofs before it that did not verify, the corresponding indices so that the clients can keep track of the semi-honest chain. This would however, come at the cost of longer phases as, at any point of time, the clients can only trust a prefix of the semi-honest chain (the suffix would still require an honest verifier to point out incorrect proofs). This concludes the discussion on achieving GOD in SCALES.

Acknowledgments. Anasuya Acharya was supported by the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme under grant agreement No 101124977 (DECRYPSYS). Carmit Hazay was supported by the United States-Israel Binational Science Foundation (BSF) through Grant No.2020277. Vladimir Kolesnikov was supported in part by NSF awards CNS-2246354 and CCF-2217070. Manoj Prabhakaran was supported in part by the IIT Bombay Trust Lab and the Algorand Centres of Excellence programme managed by Algorand Foundation.

[10] the custom zk-proofs in [AHKP24] for correct garbling and rernadomizing are also modular when the underlying RGS involves the gate-by-gate garbling of a circuit: correctness of computation can be proven separately for each gate

References

AHKP22. Anasuya Acharya, Carmit Hazay, Vladimir Kolesnikov, and Manoj Prabhakaran. SCALES - MPC with small clients and larger ephemeral servers. In *TCC 2022*, pages 502–531, 2022.

AHKP24. Anasuya Acharya, Carmit Hazay, Vladimir Kolesnikov, and Manoj Prabhakaran. Malicious security for SCALES - outsourced computation with ephemeral servers. In *CRYPTO 2024*, pages 3–38, 2024.

AIK11. Benny Applebaum, Yuval Ishai, and Eyal Kushilevitz. How to garble arithmetic circuits. In *FOCS*, pages 120–129, 2011.

AIR01. William Aiello, Yuval Ishai, and Omer Reingold. Priced oblivious transfer: How to sell digital goods. In *EUROCRYPT 2001*, pages 119–135, 2001.

Bar86. David A. Mix Barrington. Bounded-width polynomial-size branching programs recognize exactly those languages in nc^1. In *Proceedings of the 18th Annual ACM Symposium on Theory of Computing*, pages 1–5, 1986.

Bar89. David A. Mix Barrington. Bounded-width polynomial-size branching programs recognize exactly those languages in nc^1. *J. Comput. Syst. Sci.*, 38(1):150–164, 1989.

BCR+24. Rishabh Bhadauria, James Hsin-yu Chiang, Divya Ravi, Jure Sternad, and Sophia Yakoubov. Minimizing the use of the honest majority in YOSO MPC with guaranteed output delivery. *IACR Cryptol. ePrint Arch.*, page 2059, 2024.

BDO23. Lennart Braun, Ivan Damgård, and Claudio Orlandi. Secure multiparty computation from threshold encryption based on class groups. In *Advances in Cryptology - CRYPTO 2023 - 43rd Annual International Cryptology Conference, CRYPTO 2023, Santa Barbara, CA, USA, August 20-24, 2023, Proceedings, Part I*, pages 613–645, 2023.

BEP23. Alexander Bienstock, Daniel Escudero, and Antigoni Polychroniadou. On linear communication complexity for (maximally) fluid MPC. In *Advances in Cryptology - CRYPTO 2023 - 43rd Annual International Cryptology Conference, CRYPTO 2023, Santa Barbara, CA, USA, August 20-24, 2023, Proceedings, Part I*, pages 263–294, 2023.

BGG+20. Fabrice Benhamouda, Craig Gentry, Sergey Gorbunov, Shai Halevi, Hugo Krawczyk, Chengyu Lin, Tal Rabin, and Leonid Reyzin. Can a public blockchain keep a secret? In *TCC*, pages 260–290, 2020.

BHHO08. Dan Boneh, Shai Halevi, Michael Hamburg, and Rafail Ostrovsky. Circular-secure encryption from decision diffie-hellman. In *CRYPTO*, pages 108–125, 2008.

Bry86. Randall Bryant. Graph-based algorithms for boolean function manipulation. *IEEE Transactions on Computers*, C-35(8):677–691, 1986.

CGG+21. Arka Rai Choudhuri, Aarushi Goel, Matthew Green, Abhishek Jain, and Gabriel Kaptchuk. Fluid MPC: secure multiparty computation with dynamic participants. In *CRYPTO*, pages 94–123, 2021.

CL15. Guilhem Castagnos and Fabien Laguillaumie. Linearly homomorphic encryption from ddh. In *CT-RSA 2015*, pages 487–505, 2015.

Cle86. Richard Cleve. Limits on the security of coin flips when half the processors are faulty (extended abstract). In *Proceedings of the 18th Annual ACM Symposium on Theory of Computing, May 28-30, 1986, Berkeley, California, USA*, pages 364–369, 1986.

DDG+23. Bernardo David, Giovanni Deligios, Aarushi Goel, Yuval Ishai, Anders Konring, Eyal Kushilevitz, Chen-Da Liu-Zhang, and Varun Narayanan. Perfect MPC over layered graphs. In *Advances in Cryptology - CRYPTO 2023 - 43rd Annual International Cryptology Conference, CRYPTO 2023, Santa Barbara, CA, USA, August 20-24, 2023, Proceedings, Part I*, pages 360–392, 2023.

DKLN24. Giovanni Deligios, Anders Konring, Chen-Da Liu-Zhang, and Varun Narayanan. Statistical layered MPC. In *Theory of Cryptography - 22nd International Conference, TCC 2024, Milan, Italy, December 2-6, 2024, Proceedings, Part IV*, pages 362–394, 2024.

EMP25. Daniel Escudero, Elisaweta Masserova, and Antigoni Polychroniadou. Brief announcement: Towards scalable YOSO MPC via packed secret-sharing. In *Proceedings of the ACM Symposium on Principles of Distributed Computing, PODC 2025, Hotel Las Brisas Huatulco, Huatulco, Mexico, June 16-20, 2025*, pages 162–165, 2025.

GHK+21. Craig Gentry, Shai Halevi, Hugo Krawczyk, Bernardo Magri, Jesper Buus Nielsen, Tal Rabin, and Sophia Yakoubov. YOSO: you only speak once - secure MPC with stateless ephemeral roles. In *CRYPTO*, pages 64–93, 2021.

GHM+21. Craig Gentry, Shai Halevi, Bernardo Magri, Jesper Buus Nielsen, and Sophia Yakoubov. Random-index PIR and applications. In *TCC 2021*, pages 32–61, 2021.

GHV10. Craig Gentry, Shai Halevi, and Vinod Vaikuntanathan. i-hop homomorphic encryption and rerandomizable yao circuits. In *CRYPTO*, pages 155–172, 2010.

GMPS21. Vipul Goyal, Elisaweta Masserova, Bryan Parno, and Yifan Song. Blockchains enable non-interactive MPC. *IACR Cryptol. ePrint Arch.*, page 1233, 2021.

HM20. Martin Hirt and Marta Mularczyk. Efficient MPC with a mixed adversary. In *1st Conference on Information-Theoretic Cryptography, ITC 2020, June 17-19, 2020, Boston, MA, USA*, pages 3:1–3:23, 2020.

HvdHJ25. Raphael Heitjohann, Jonas von der Heyden, and Tibor Jager. Rerandomizable garbling, revisited. *IACR Cryptol. ePrint Arch.*, page 843, 2025.

IP07. Yuval Ishai and Anat Paskin. Evaluating branching programs on encrypted data. In Salil P. Vadhan, editor, *Theory of Cryptography, 4th Theory of Cryptography Conference, TCC 2007, Amsterdam, The Netherlands, February 21-24, 2007, Proceedings*, volume 4392 of *Lecture Notes in Computer Science*, pages 575–594. Springer, 2007.

Kil88. Joe Kilian. Founding crytpography on oblivious transfer. In *STOC*, page 20–31, 1988.

KJGB06. Louis Kruger, Somesh Jha, Eu-Jin Goh, and Dan Boneh. Secure function evaluation with ordered binary decision diagrams. In *Proceedings of the 13th ACM Conference on Computer and Communications Security*, CCS '06, page 410–420, New York, NY, USA, 2006. Association for Computing Machinery.

Kol05. Vladimir Kolesnikov. Gate evaluation secret sharing and secure one-round two-party computation. In *ASIACRYPT 2005*, pages 136–155, 2005.

KRY22. Sebastian Kolby, Divya Ravi, and Sophia Yakoubov. Towards efficient YOSO MPC without setup. *IACR Cryptol. ePrint Arch.*, page 187, 2022.

Lip09. Helger Lipmaa. First CPIR protocol with data-dependent computation. In Dong Hoon Lee and Seokhie Hong, editors, *Information, Security and Cryptology - ICISC 2009, 12th International Conference, Seoul, Korea, December 2-4, 2009, Revised Selected Papers*, volume 5984 of *Lecture Notes in Computer Science*, pages 193–210. Springer, 2009.

LMR+24. Chen-Da Liu-Zhang, Elisaweta Masserova, João Ribeiro, Pratik Soni, and Sri Aravinda Krishnan Thyagarajan. Improved YOSO randomness generation with worst-case corruptions. In *Financial Cryptography and Data Security - 28th International Conference, FC 2024, Willemstad, Curaçao, March 4-8, 2024, Revised Selected Papers, Part II*, pages 73–89, 2024.

LMR+25. Chen-Da Liu-Zhang, Elisaweta Masserova, João Ribeiro, Pratik Soni, and Sri Aravinda Krishnan Thyagarajan. Efficient distributed randomness generation from minimal assumptions where parties speak sequentially once. In *Advances in Cryptology - EUROCRYPT 2025 - 44th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Madrid, Spain, May 4-8, 2025, Proceedings, Part V*, pages 176–206, 2025.

NRO22. Jesper Buus Nielsen, João Ribeiro, and Maciej Obremski. Public randomness extraction with ephemeral roles and worst-case corruptions. In *Advances in Cryptology - CRYPTO 2022 - 42nd Annual International Cryptology Conference, CRYPTO 2022, Santa Barbara, CA, USA, August 15-18, 2022, Proceedings, Part I*, pages 127–147, 2022.

PVW08. Chris Peikert, Vinod Vaikuntanathan, and Brent Waters. A framework for efficient and composable oblivious transfer. In *CRYPTO*, 2008.

RS22. Rahul Rachuri and Peter Scholl. Le mans: Dynamic and fluid MPC for dishonest majority. In *Advances in Cryptology - CRYPTO 2022 - 42nd Annual International Cryptology Conference, CRYPTO 2022, Santa Barbara, CA, USA, August 15-18, 2022, Proceedings, Part I*, pages 719–749, 2022.

Yao86. Andrew Chi-Chih Yao. How to generate and exchange secrets (extended abstract). In *FOCS*, pages 162–167, 1986.

A On the concrete size of PBPs

To implement a function as a PBP, instead of relying on Barrington's theorem [Bar86] to transform a boolean circuit, one can often use a more efficient direct construction. As an illustration, for the case of 32-bit integer addition, one can use modular arithmetic and Chinese Remainder Theorem to design a PBP, such that in our SCALES protocol for sums bounded by 2^{32}, each client posts only 111 group elements (irrespective of how many other clients are present) while each ephemeral server posts under $2200n$ group elements (where n is the number of clients).

The details of the PBP design for this protocol are as follows: firstly, note that addition modulo an integer q can be directly implemented as a PBP of width q and length equal to the number of inputs. But this by itself does not scale well when the sum being computed can be large (e.g., sum $< N = 2^{32}$), since we will need

$q \geq N$ (to avoid overflow). Instead, we express an upper bound $N' \geq N$ as a product of small prime powers (e.g., $2^4 \times 3^2 \times 5 \times 7 \times 11 \times 13 \times 17 \times 19 \times 23 = N' > N = 2^{32}$) such that these prime powers sum up to a relatively small number M (in our example, $2^4 + 3^2 + 5 + 7 + 11 + 13 + 17 + 19 + 23 = 120 = M$). Since every integer in the range $[0, N-1]$ has a unique CRT representation, it is secure to compute the sum modulo each of these prime powers in parallel. The total input size of each client across these nine computations is 37 bits, and the total size (sum over width $\times$ length) of all the PBPs is $527n$, where n is the number of clients.

In contrast, 32-bit integer addition even using the best RGC construction in prior work would require 32 full-adder circuits (with 5 binary gates each). Assuming the security parameter is $\kappa = 128$, the size of each garbled gate would be $4 \times 128 \times 3$ group elements and the RGC size would be $n \times 5 \times 32 \times 4 \times 128 \times 3 = 245,760n$ group elements.