Sublinear-Communication Layered MPC from HSS

Anasuya Acharya[1⋆], Pierre Meyer[2], Divya Ravi[3], and Rahul Satish[4]

[1] Aalborg University, anasuyaa@es.aau.dk
[2] Aarhus University, pierre.meyer@cs.au.dk
[3] University of Amsterdam, d.ravi@uva.nl
[4] IT University of Copenhagen, rahs@itu.dk

Abstract. The server-aided Secure Multi-Party Computation (MPC) model with dynamic participation, where servers may join or leave during the computation is particularly well-suited for practical, long-running applications. Unfortunately, existing protocols supporting dynamic participation either incur communication costs that scale linearly with the circuit size or rely on the heavy hammer of Fully Homomorphic Encryption (FHE). This contrasts with "classic" MPC protocols, where sublinear communication can be achieved without FHE. Focusing on the Layered MPC framework (introduced by David, Ishai, Konring, Kushilevitz, and Narayanan, CRYPTO 2023), we show that this overhead is not inherent. We construct sublinear-communication Layered MPC protocols while maintaining a minimal number of online servers – only two or three per layer. Our constructions rely on a layered-friendly adaptation of Homomorphic Secret Sharing (HSS), which we formalize as *HSS with Layered Resharing*.

Keywords: Layered MPC, Homomorphic Secret Sharing, Communication Complexity

1 Introduction

Secure Multi-Party Computation [GMW87, Yao86, BGW88, CCD88] allows mutually distrusting parties to collaboratively compute a function on their private inputs, ensuring that nothing beyond the intended result is revealed. For large-scale applications, the server-aided model of MPC is often adopted, where clients provide their inputs and delegate the computational workload to a set of dedicated servers. Since large-scale computations can be long-lived, it is desirable to support dynamic server participation rather than requiring the same set of servers to remain online throughout. This has led to renewed interest in the mobile adversary model [OY91], inspiring a line of recent works [GHK+21, CGG+21, AHKP22, DDG+23] that propose ephemeral servers, where the set of computing parties can change over time—typically from one round to the next.

Prominent frameworks in the outsourced setting with dynamic servers, such as YOSO [GHK+21], SCALES [AHKP22], Fluid MPC [BEP24, BEP23], Layered MPC [DDG+23, DKLZN24] considers the notion of a 'committee': This refers to the set of ephemeral servers that are online during a given round to perform a part of the computation, after which they may go offline, passing the torch, in the form of the computation state, to the next committee that becomes online. While these frameworks provide mechanisms for dynamic server participation, their performance is constrained by the size of each committee and the amount of computation state exchanged between consecutive committees.

⋆ This work was done while the author was affiliated with Aarhus University, Denmark.

When considering deployments of such protocols, two key aspects become relevant, namely (a) the size of the committee, as the number of servers that must be simultaneously online directly impacts the protocol's latency and coordination needed; (b) the size of the computation state being transferred between consecutive committees, which determines the communication overhead. The existing protocols supporting dynamic participation, typically instantiated with a general n-party committees either achieve communication complexity that remains linear in the size of the circuit being computed [DDG+23, BEP23, BEP24] or rely on heavy tools such as fully-homomorphic encryption (FHE) [KRY24]. This is in contrast to MPC protocols that do not support dynamic participation, where total communication complexity sublinear in the circuit size is achievable without FHE (in the two-party setting, under DDH [BGI16], quasipolynomial LPN [CM21], DCR [FGJS17, OSY21, RS21], or QR+LPN [BCM22]). This raises the question of whether the additional communication overhead is inherent to supporting dynamic participation, or whether it is possible to match the communication efficiency of protocols without dynamic participation.

In this work, we focus on the Layered MPC framework [DDG+23] (where a committee is only allowed to communicate with its subsequent committee) and ask the following natural question:

> *Are there layered MPC protocols that achieve sublinear communication complexity, without using FHE?*

We answer this question affirmatively using homomorphic secret sharing (HSS) [BGI16], which allows a client to distribute shares of its input so that servers locally compute shares of the output, which the client can later reconstruct. A desirable feature of our protocols is that they only require two (or three depending on the security guarantees) servers to be online in any given round.

1.1 Our Contributions

We first recall the layered MPC model of [DDG+23], which structures parties into layers, with each layer comprising n ephemeral parties that interact only with those in the subsequent layer. The parties within a layer are referred to as forming a 'committee' and the adversary is allowed to corrupt up to a threshold t among n parties within each committee.

Broadly, our contributions consist of designing communication-efficient layered MPC protocols in the computational setting with $n = 2$ or $n = 3$ parties in each computation layer, secure against an adversary that corrupts $t = 1$ among the two or three parties, respectively. The protocols can accommodate an arbitrary number of input and output parties.

In particular, our protocols are the first layered MPC protocols to achieve total communication complexity that grows sublinearly with the circuit size, without using FHE. The key conceptual insight of our work is that HSS is inherently layered-friendly. Specifically, we adapt Distributed Evaluation Homomorphic Encryption (DEHE) [BGI16], an HSS variant that supports inputs from multiple clients, to the layered setting. We show that DEHE intermediate computations can be efficiently chained across consecutive layers, enabling layered MPC protocols with sublinear communication complexity (assuming the circuit has a "layered"[5] structure). To formalize

[5] A circuit is layered [GJ11] if all gates and inputs are arranged into layers, such that any wire only connects one layer to the next, but each input may occur multiple times at different layers. This is a standard restriction in the literature on sublinear-communication MPC from HSS, and we refer to [CCMN24] for an extended discussion.

this approach, we identify and abstract the properties a DEHE scheme must satisfy to be compatible with layered computation, which we believe is of independent interest.

Table 1 summarizes our results, and provides a comparison with the existing state-of-the-art protocols. We provide a detailed description of each contribution below and formally state the corresponding theorems.

Our first result is in the semi-honest setting, where corrupted parties follow the protocol honestly, but the adversary can observe their internal state.

Theorem (Informal) 1 *Assuming Distributed-Evaluation Homomorphic Encryption (which can be instantiated from DCR), there exists a layered MPC protocol with two parties per computation layer, that achieves semi-honest security against adversaries corrupting all but one party in each layer and attains communication complexity* $\mathcal{O}(|\mathbf{C}|/\log|\mathbf{C}|)$*, where* $|\mathbf{C}|$ *denotes the circuit size.*

In the malicious setting, where the corrupt parties can arbitrarily misbehave, we present three protocols with different levels of security: The first protocol achieves *security with abort*, where correctness of output (if obtained) holds, but it is possible for the adversary to obtain the output while depriving honest parties of it. The second protocol achieves *fairness*, where either all or none of the parties obtain the output. Notably, the fair protocol assumes exactly one honest majority committee (specifically, 3 parties with at most 1 corruption). This assumption is both necessary and minimal (as fair MPC is known to be impossible in dishonest majority [Cle86]). The final protocol achieves the strongest guarantee of *guaranteed output delivery (GOD)* where no matter what the adversary does, it cannot prevent honest parties from obtaining the output. Our GOD construction assumes three parties with at most one malicious corruption in each layer.

Theorem (Informal) 2 *Assuming Distributed-Evaluation Homomorphic Encryption, there exists a layered MPC protocol with two parties per computation layer satisfying malicious security with abort in the presence of adversaries corrupting all-but-one parties in each layer and achieving communication complexity* $\mathcal{O}(|\mathbf{C}|/\log|\mathbf{C}|)$*, where* $|\mathbf{C}|$ *denotes the circuit size.*

Theorem (Informal) 3 *Consider a layered setting where each computation layer, except the last, consists of two parties, while the final computation layer consists of three. Using DCR-based Homomorphic Secret Sharing, there exists a layered MPC protocol in this setting that achieves fairness against malicious adversaries corrupting all-but-one party in each layer and attains communication complexity* $\mathcal{O}(|\mathbf{C}|/\log|\mathbf{C}|)$*, where* $|\mathbf{C}|$ *denotes the circuit size.*

Theorem (Informal) 4 *Consider a layered setting where each computation layer consists of three parties. Assuming Distributed-Evaluation Homomorphic Encryption, there exists a layered MPC protocol in this setting that achieves guaranteed output delivery against malicious adversaries corrupting all-but-one party in each layer and attains communication complexity* $\mathcal{O}(|\mathbf{C}|/\log|\mathbf{C}|)$*, where* $|\mathbf{C}|$ *denotes the circuit size.*

[6] Our fair construction requires the last computation committee to satisfy $n = 3, t = 1$

Model	Corruption	Resilience	Security	Assumption	CC
Layered [DDG+23]	$t < n/3$	Perfect	GOD	–	$\mathcal{O}(n^9 \cdot \|C\|)$
Layered [DDG+23]	$t < n/3$	Computational	GOD	OWF	$\kappa \cdot 2^{\mathcal{O}(n)} + \mathcal{O}(n^5 \cdot \|C\|)$
Layered [DGLZ23]	$t < n/2$	Computational	GOD	LWE/DLog, CRS	$\mathcal{O}(\text{poly}(n) \cdot \|C\| \cdot \kappa)$
Layered [DKLZN24]	$t < n/2$	Statistical	GOD	–	$\mathcal{O}(n^{18} \cdot \|C\| \cdot \lambda)$
Layered [BEP23]	$t < n/2$	Statistical	abort	–	$\mathcal{O}(n \cdot \|C\|)$
Fluid (with preprocessing) GOE [BEP23]	$t < n$	Statistical	abort	–	$\mathcal{O}(n \cdot \|C\|)$
Layered [BEP24]	$t < n/3$	Perfect	abort	–	$\mathcal{O}(n \cdot \|C\|)$
Layered **[This work]**	$t = 1, n = 2$	Computational	semi-honest	DCR	$\mathcal{O}(\frac{\|C\|}{\log \|C\|})$
Layered **[This work]**	$t = 1, n = 2$	Computational	abort	DCR	$\mathcal{O}(\frac{\|C\|}{\log \|C\|})$
Layered **[This work]**	$t = 1, n = 2^6$	Computational	fair	DCR	$\mathcal{O}(\frac{\|C\|}{\log \|C\|})$
Layered **[This work]**	$t = 1, n = 3$	Computational	GOD	DCR	$\mathcal{O}(\frac{\|C\|}{\log \|C\|})$

Table 1. Communication Complexity (CC) of MPC with Dynamic Committees. Here $|C|$ denotes the number of gates, κ denotes the security parameter, t denotes the corruption threshold and n denotes the size of each committee.

1.2 Related Work

Sublinear MPC from HSS. HSS has been used to achieve sublinear-communication MPC (but not in the layered setting), in many ways: there are protocols making a direct use of HSS [BGI16], protocols using HSS to generate correlated randomness [CM21, CKY25], protocols using non-black-box properties of HSS constructions [BCM23, ARS24] revolving around Trapdoor Hash-Functions [DGI+19], and sublinear-size Garbled Circuits from HSS [ILL25, MORS25].

Layered MPC. [DDG+23] introduced the layered MPC framework where 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. The first layered MPC protocols appeared in [DDG+23] and [DKLZN24] for the perfect and statistical settings respectively, achieving optimal resilience of $t < n/3$ and $t < n/2$. Recent works further explore multi-layered MPC that reduces communication by allowing parties to speak for more than one layer [DMS25, ZYFZ25], and protocols achieving only security with abort in the layered setting [BEP23, BEP24].

Alternate models with dynamic participation. We now discuss the state-of-the-art in other related models, such as YOSO, PASSO, SCALES and Fluid MPC.

A key distinction between YOSO and layered MPC is that YOSO allows private messages to parties who will speak in the future (while keeping their identities anony-

mous), whereas layered MPC only requires channels between consecutive committees. In the YOSO model, [GHK+21] proposed information-theoretic and computationally secure protocols achieving guaranteed output delivery under an honest-majority assumption per committee. Follow-up works [KRY22, BDO23, EMP25] improve round complexity, scalability, and efficiency.

Inspired by YOSO and [NRO22], [LZMR+24, LZMR+25] present the *PASSO* (PArties Speak Sequentially Once) model. In their constructions, parties speak exactly once in sequence, facilitating the computation of functions for distributed randomness generation.

The *SCALES* model was introduced in [AHKP22], where clients are allowed to send multiple messages. Building on the semi-honest construction in [AHKP22] tolerating a majority of corruptions per committee, the malicious security with abort protocol appeared in [AHKP24]. The SCALES model is orthogonal to both YOSO/PASSO and layered MPC: unlike YOSO and PASSO, it does not support sending private messages to future participants, and unlike layered MPC, it assumes that input holders maintain state across messages.

Finally, the *Fluid MPC* model [CGG+21] allows participants to join and leave the computation dynamically. General fluid protocols permit committees to remain active for multiple rounds, while maximally-fluid protocols are equivalent to layered MPC. [CGG+21] presents maximally-fluid protocols in the honest-majority setting, with subsequent work extending to dishonest-majority [RS22] and communication-efficient constructions [BEP23].

1.3 Technical Overview

Since our goal is to design an MPC protocol in the layered setting whose communication complexity is sublinear in the circuit size, it makes sense to try and determine which of the known two-party sublinear-communication protocols (in the "traditional" setting for MPC) are amenable to the constraints of the layered setting. There are two main categories of protocols:

1. *Constant-round sublinear-communication protocols.* In principle, minimally interactive protocols should be the easiest to use in the layered setting. In fact, if parties do not interact, we need not use more than one layer of server, thereby mostly negating the challenges of the ephemeral setting. Such protocols can either be based on Fully Homomorphic Encryption (or, in principle, on Fully Homomorphic Secret Sharing; however the only known construction is obtained by combining FHE with spooky encryption [DHRW16]), or on sublinear-size Garbled Circuits [MORS25, ILL25]. While the latter may be suited to the semi-honest honest setting, it is unclear how one would achieve malicious security without the heavy hammer of Succinct Non-Interactive Arguments (SNARGs) for P.
2. *Interactive sublinear-communication protocols.* All other known protocols follow a common template of decomposing a circuit into low-depth chunks, and securely computing each of these chunks one-by-one (with a round of "interactive bootstrapping" needed to convert the output of the previous chunk into an appropriately format input for the next chunk). Our first observation is that such protocol design is compatible with layered MPC, provided each committee can securely compute one chunk, and then securely "pass on" its state to the next committee, that they may compute the next chunk.

A bit more concretely, suppose a deep circuit C is viewed as the composition of many shallow circuits $C = C_m \circ \cdots \circ C_1$. If two parties initially hold HSS shares of a PRF key

k (known to neither party) and the one-time-padded ciphertext $\mathsf{ct}_0 = x + \mathsf{PRF}(k, 0)$, they can homomorphically evaluate the function $K \mapsto C_1(\mathsf{ct}_0 - \mathsf{PRF}(K, 0)) + \mathsf{PRF}(K, 1)$ on their HSS shares of k. Note that provided the PRF is not too deep, neither is this function. The parties now hold additive shares of $\mathsf{ct}_1 := C_1(x) + \mathsf{PRF}(K, 1)$, which they can exchange in order to reconstruct ct_1 (this is the "interactive bootstrapping" step). They now have everything they need (ct_1 and HSS shares of k) to repeat the process and compute shares of $\mathsf{ct}_2 = C_2(\mathsf{ct}_1 - \mathsf{PRF}(k, 1)) + \mathsf{PRF}(k, 2)$. Reconstructing the one-time-padded ciphertexts ct_i from their shares is not a challenge in the layered setting, as each committee member can simply broadcast its share towards the next committee. The problem however, is that the HSS shares of k must somehow be passed on to the next committee without revealing k...

HSS with Layered Resharing. The key observation of this work is that many constructions of HSS (a.k.a. "HSS following the BGI template", or "DDLog-based HSS") have input shares of the following form (where x is the input to be shared): both parties' shares contain the *same* ciphertext $\mathsf{ct} = \mathsf{Enc}_{\mathsf{sk}}(x)$, under a key sk known to neither party, as well as an additive share of the secret key sk (this last part is sometimes called an "evaluation key"). This means that each committee can securely "reshare" their input shares to the next committee by (1) broadcasting the ciphertext ct and (2) additively splitting their shares of the secret key, so that the next committee may obtain their own evaluation keys. The formalism of Distributed-Evaluation Homomorphic Encryption from [BGI16] captures the structure we want on the input shares: a ciphertext common to both parties, and an input-independent evaluation-key.

Setup. For ease of exposition, we present our constructions under the assumption of a trusted setup that communicates only with the input layer. This setup can be instantiated using existing layered MPC protocols (Table 1) and requires only a constant number of computation committees, since it is independent of the size of the target functionality.

Semi-Honest Layered MPC. We decompose the layered circuit into a sequence of $\mathbf{NC}^1$ sub-circuits (similar to [Cou19, CCMN24]), each evaluated locally by a layer of two servers using 2-party DEHE, with intermediate outputs feeding into the next layer. This raises two challenges in the layered setting: DEHE outputs additive shares that should not be reconstructed, and servers in the subsequent layers require appropriate evaluation keys. We address the latter using the above new abstraction of DEHE with layered resharing, which avoids any upfront setup tied to the number of layers. To handle the former, each layer computes a masked output using a one-time pad derived from a shared PRF key, allowing safe reconstruction. We defer to Section 3 for a more detailed overview.

Maliciously Secure Layered MPC with Abort. We build on the semi-honest construction outlined above and extend it to achieve malicious security with abort. Instead of relying on computationally heavy tools such as non-interactive zero-knowledge proofs, we show that simple digital signature–based consistency checks suffice to detect misbehavior in intermediate computations by corrupt parties. Additionally, we present an alternative construction that further reduces the computational overhead of achieving malicious security by leveraging DEHE schemes whose reconstruction is based on distributed discrete logarithms. We refer to Section 4 for further details.

Maliciously Secure Layered MPC with Fairness. For our fair construction, we adapt the approach of [IKP+16] to the layered setting. The main idea to securely compute a function f with fairness is to use a layered unanimous abort protocol (where honest parties agree on the output) to instead compute a related functionality f' – where, f'computes two-out-of-three threshold shares of the output of f to the 3-party committee. This enables fairness: if the protocol computing f' aborts, the adversary learns nothing from its single share. If not, the two honest parties suffice to reconstruct the output.

Maliciously Secure Layered MPC with GOD. The core idea of our GOD protocol, which involves three parties with at most one corruption per layer, is to run three parallel 2-party layered computations, one for each pair of parties, with unanimous abort. Since at least one pair is entirely honest, that execution is guaranteed to be robust, ensuring the computation continues. Since branching into three executions at each intermediate step would cause the number of executions per layer to grow prohibitively, the main challenge is enabling honest parties to identify and agree on a single correct instance to continue at each step. We refer to Section 6 for further details.

2 Preliminaries

We refer the reader to Appendix A for additional preliminaries.

Notation. We denote by κ the computational security parameter and by $\mu(\cdot)$ a negligible function. We denote vectors and bit strings as follows, e.g. $\vec{x}$. By $[n]$ we denote the set $\{1, \ldots, n\}$, and $\mathbb{F}$ denotes a finite field. We denote 'accept' and 'reject' (or 'abort') respectively using the symbols $\top$ and $\bot$. The additive sharing of x is denoted by $\langle x \rangle$.

MPC Notation. Let $\mathcal{P} = \{P_i\}_{i \in [n]}$ be a set of n parties involved in MPC. Each party P_i has input x_i. For simplicity and without loss of generality, we work over functionalities for which $x_i \in \{0, 1\}$. For every tuple of inputs $\vec{x} = (x_i, \ldots, x_n)$, the vector output by f is a random variable $(f_1(\vec{x}), \ldots, f_n(\vec{x}))$ ranging over tuples of strings where P_i receives $f_i(\vec{x})$. The following notation describes a functionality: $(x_1, \ldots, x_n) \mapsto (f_1(\vec{x}), \ldots, f_n(\vec{x}))$

2.1 Layered MPC

A layered MPC protocol [DDG+23] can be viewed as a special case of standard MPC (formalized in Appendix A) with a general adversary structure, specialized in the following way: (1) the interaction pattern is defined by a layered graph; (2) the adversary can corrupt at most t parties in each layer.

Definition 1 *A **layered protocol** is a synchronous protocol Π over secure point-to-point channels and a broadcast channel, with the following special features.*

- ***Parties:** Suppose there are $N = n(d+1)$ parties partitioned into $d+1$ layers $\mathcal{L}_i$, $0 \leq i \leq d$, where $|\mathcal{L}_i| = n$. Parties in the first layer $\mathcal{L}_0$ and the last layer $\mathcal{L}_d$ are referred to as input clients and output clients respectively.*

– ***Interaction Pattern:*** *The interaction consists of d rounds, where in round i parties in $\mathcal{L}_{i-1}$ may send messages to parties in $\mathcal{L}_i$ over secure point-to-point channels. By default, we additionally allow each party in $\mathcal{L}_{i-1}$ to send a broadcast message to all parties in $\mathcal{L}_i$.*
– ***Functionalities:*** *We consider functionalities f that take inputs from input clients and deliver outputs to output clients.*
– ***Adversaries:*** *We consider adversaries who may corrupt any number of input and output clients, and additionally corrupt t parties in each intermediate layer $\mathcal{L}_i$, $0 < i < d$. We consider active, rushing, adaptive adversaries.*

We say that a protocol Π is a layered MPC protocol for f if it realizes f in the standard sense of (standalone) secure MPC.

In this paper, we consider a slight variation of the layered MPC framework where the parties are not necessarily distributed evenly across all layers. In particular, the number of input and output clients could differ from the number of parties in each intermediate layer. We denote such protocols as (m, n, z)-**layered protocol** where m and z refer to the number of input and output clients respectively; while n refers to the number of parties in each intermediate layer. We denote parties in $\mathcal{L}_i$ $(1 \leq i < d)$ as $\{S_j^i\}_{j\in[n]}$.

2.2 Distributed-Evaluation Homomorphic Encryption

We adapt the definition of Distributed-Evaluation Homomorphic Encryption[7] from [BGI16], in three ways:

– we weaken it from a public-key to a secret-key flavor (this is because we do not need such public-key properties in our protocols),
– we strengthen the correctness requirement so as to enforce a negligible error (whereas the original definition of [BGI16] was stated in a more general way, allowing for a noticeable correctness error),
– we strengthen the selective security game to require that the same secret key can be used to generate many pairs of evaluation keys, provided the adversary only knows one key from each pair.

We note that this stronger notion of security is naturally satisfied by all the main instantiations (from DDH [BGI16], DCR [FGJS17, OSY21, RS21], Class Groups [ADOS22], and LWE [BKS19]), without alteration[8], and that this stronger notion of correctness is satisfied by some instantiations from DCR [OSY21, RS21] (or LWE [BKS19], provided the noise-to-modulus ratio is superpolynomial).

Definition 1 (Secret-Key Distributed-Evaluation Homomorphic Encryption, adapted from [BGI16]). A (2-party) *Distributed Evaluation Homomorphic Encryption* (DEHE) scheme for a class of programs $\mathcal{P}$ is a tuple of p.p.t. algorithms $\mathsf{DEHE} = (\mathsf{KeyGen}, \mathsf{Gen}, \mathsf{Enc}, \mathsf{Eval})$ with the following syntax and properties:

– $\mathsf{KeyGen}(1^\kappa) \to \mathsf{sk}$: On input the security parameter 1^κ, the *secret-key generation algorithm* KeyGen outputs a secret key sk.

[7] This notion is very closely related to the notion of a "Distributed Discrete Logarithm" [BGI16].
[8] Although this notion is by no means inherent. In fact, it is straightforward to convert *any* DEHE (satisfying the standard notion of security) into one which provably does not satisfy this stronger notion.

- $\mathsf{Gen}(1^\kappa, \mathsf{sk}) \to (\mathsf{ek}_0, \mathsf{ek}_1)$: On input the secret key sk, the *evaluation key generation algorithm* Gen outputs a pair of evaluation keys $(\mathsf{ek}_0, \mathsf{ek}_1)$.
- $\mathsf{Enc}(\mathsf{sk}, x) \to \mathsf{ct}$: On input a secret key sk, and an input $x \in \{0,1\}$, the *encryption algorithm* Enc outputs a ciphertext ct.
- $\mathsf{Eval}(b, \mathsf{ek}_\sigma, (\mathsf{ct}_i)_{i\in[n]}, P, \beta) \to y_b$: On input a party index $b \in \{0,1\}$, an evaluation key ek_b, a vector of n ciphertexts $(\mathsf{ct}_1, \ldots, \mathsf{ct}_n)$, a program P with n input bits and m output bits, and an integer $\beta \geq 2$, the *homomorphic evaluation algorithm* Eval outputs $y_b \in \mathbb{Z}_\beta^m$, constituting party b's share of an output $y \in \{0,1\}^m$. When β is omitted it is understood to be $\beta = 2$.

The algorithms KeyGen, Gen and Enc are PPT algorithms, whereas Eval can run in time polynomial in input length and $1/\delta$. The algorithms $(\mathsf{KeyGen}, \mathsf{Gen}, \mathsf{Enc}, \mathsf{Eval})$ satisfy the following correctness and security requirements.

- ***Correctness.*** For every polynomial p there is a negligible ν such that for every positive integer κ, input $(x_1, \ldots, x_n) \in \{0,1\}^n$, program $P \in \mathcal{P}$ with input length n, and integer $\beta \geq 2$, where $|P| \leq p(\kappa)$, we have

$$\Pr\left[\begin{array}{l} y_0 + y_1 = P(x_1, \ldots, x_n): \\ \end{array} \begin{array}{l} \mathsf{sk} \leftarrow \mathsf{KeyGen}(1^\kappa); \\ (\mathsf{ek}_0, \mathsf{ek}_1) \leftarrow \mathsf{Gen}(\mathsf{sk}); \\ \text{For } i \in [n]: \\ \quad \mathsf{ct}_i \leftarrow \mathsf{Enc}(\mathsf{sk}, x_i); \\ \text{For } b \in \{0,1\}: \\ \quad y_b \leftarrow \mathsf{Eval}(b, \mathsf{ek}_b, \{\mathsf{ct}_i\}_{i\in[n]}, P, \beta) \end{array}\right] \geq 1 - \nu(\kappa) \ .$$

  where addition of y_0 and y_1 is carried out modulo β.

- ***Multi-instance Selective Security.*** We say that a secret-key DEHE scheme satisfies multi-instance selective security[9] if For every $m \in [\mathrm{poly}(\kappa)]$, for every $\sigma_1, \ldots, \sigma_m \in \{0,1\}$ (the sequence of corrupted party's indices), there exists a p.p.t. simulator Sim such that for every sequence of inputs $(x_1, \ldots, x_n) \in \mathcal{R}^n$,

$$\left\{ (\{\mathsf{ek}_{i,\sigma_i}\}_{i\in[m]}, \{\mathsf{ct}_i\}_{i\in[n]}) \;\middle|\; \begin{array}{r} \mathsf{sk} \xleftarrow{\$} \mathsf{KeyGen}(1^\kappa) \\ \text{for } i \in [m],\ (\mathsf{ek}_{i,0}, \mathsf{ek}_{i,1}) \xleftarrow{\$} \mathsf{Gen}(\mathsf{sk}) \\ \text{for } i \in [n],\ \mathsf{ct}_i \xleftarrow{\$} \mathsf{Enc}(\mathsf{sk}, x_i) \end{array} \right\} \overset{c}{\approx} \{\mathsf{Sim}(1^\kappa, n, m)\} \ .$$

In this paper, we focus on the DCR-based instantiation to exemplify the usefulness of HSS in the layered setting. Notably, the DDH-based instantiation [BGI16] suffers from an inverse-polynomial correctness error, which is a non-trivial hurdle in the layered setting with malicious security.

Theorem 2 (DEHE from DCR, adapted[10] from [OSY21, theorem 4.4], [RS21, theorem 20]). *Assuming the Decisional Composite Residuosity assumption (DCR) [Pai99], there exists Secret-Key Distributed-Evaluation Homomorphic*

[9] In contrast, the security notion of [BGI16, Definition 2.2] can be seen as *single-instance* selective security, as it does not (necessarily) allow for sampling multiple evaluation key-pairs with respect to the same public key.

[10] As stated, [OSY21, theorem 4.4] assumes the KDM security (with respect to linear functions of the key) of the Paillier encryption scheme. The Paillier cryptosystem can however be substituted for the Damgård-Jurik-ElGamal cryptosystem [BMO+25, figure 2], for which this notion of KDM security provably follows from DCR [BMO+25, Theorem 2].

Encryption (definition 1) DEHE *such that* $\mathsf{DEHE.Gen}(1^\kappa, \cdot)$ *can be computed by a Restricted-Multiplication Straight-line Program ([BGI16, definition 3.1]) (or equivalently, by an* NC^1 *circuit).*

Theorem 2 follows immediately from the observation that in constructions of homomorphic secret sharing following the template of [BGI16], evaluation keys are simply additive shares of the secret key.

3 Semi-Honestly Secure Layered MPC

In this section we construct a layered MPC protocol satisfying security in the presence of semi-honest adversaries. Our protocol requires two servers per layer of computation and enjoys communication complexity sublinear in the size of the functionality to be computed. The protocol requires future horizon one (the computation of parties in any layer depends only on the messages of parties from the previous layer). Further, the protocol supports long on-going computation where the complete functionality is not known at the beginning and may evolve throughout the protocol progression.

The protocol uses the following building-blocks:

- A two-party Distributed Evaluation Homomorphic Encryption scheme (Section 2.2) $\mathsf{DEHE} = (\mathsf{KeyGen}, \mathsf{Gen}, \mathsf{Enc}, \mathsf{Eval})$ for RMS programs, such that the Gen algorithm is an RMS program. This property is satisfied by virtually all known group-based instantiations of DEHE, in particular the DDLog-based constructions of [OSY21, RS21].
- A PRF (Definition 12)F which, for each hardcoded choice of PRF input, can be computed as an $\mathbf{NC}^1$ circuit of the key. [11]

Let $\mathbf{C} = \mathbf{C}_\ell \circ \cdots \circ \mathbf{C}_1$ be a layered circuit where for each $i \in [\ell]$, $\mathbf{C}_i : \{0,1\}^{\ell_i} \to \{0,1\}^{\ell_{i+1}} \in \mathbf{NC}^1$. At a high-level, the computation at each layer $\mathcal{L}_i$ corresponds to computation of $\mathbf{C}_i$ but in an augmented fashion: Since the outputs at each intermediate layer should remain private, instead of computing the output of the respective circuit directly, a masked version of the output is computed – where the mask is derived from a PRF key α whose encryption is given via the setup to the input clients and subsequently propagated across committees. Concretely, the computation at layer $\mathcal{L}_i$ would involve computing $\mathbf{C}_i(y_{i-1} - r_{i-1}) + r_i$, where the mask of the previous layer r_{i-1} is first removed to obtain the true input to $\mathbf{C}_i$, and the resulting output is subsequently re-masked with a fresh mask r_i. Both r_{i-1} and r_i are derived deterministically from the same PRF key α.

The primary remaining challenge is ensuring that the DEHE evaluation keys are appropriately setup for the committees to carry out the computation. Corresponding to the first layer, the evaluation keys can simply be additively shared across the clients via setup, which suffices to ensure that the intended party in the first committee can subsequently reconstruct them. However for the subsequent committees, these evaluation keys cannot be propagated in a simple one-to-one manner (i.e. Party b ($b \in \{0,1\}$) in a committee simply gives its evaluation key to Party b' ($b' \in \{0,1\}$) in the next committee) – since doing so would allow an adversary corrupting Party $(1-b)$ and b' to learn both evaluation keys. To address this, we leverage a property of the DEHE scheme: the same secret key k can be used to generate many pairs of

[11] Such PRFs are implied by many standard assumptions, with varied algebraic structures: DDH/factoring/k-LIN [NR97, NR04, NRR00, LW09, Vio13], ring-LWE [BPR12], and more recently from variants of LPN [DJK25].

evaluation keys. This fresh generation of evaluation keys is performed by an RMS program evaluated under DEHE, which takes the fixed key k as input. The encryption of this fixed key k is provided to the input client by the setup, and this corresponding ciphertext gets propagated across committees thereafter.

In the following, we first describe a functionality $\mathcal{F}_{\mathsf{Setup}}$ (Algorithm 1) that distributes setup information into the input layer $\mathcal{L}_0$. Then, we describe the computation performed within the DEHE for each layer, followed by a series of DEHE-based protocols Π_i (Algorithm 2 - Algorithm 4) that describe the computation by the parties in $\mathcal{L}_i$. Finally, we present our main result: a protocol (Algorithm 5) that realizes any functionality in the $\mathcal{F}_{\mathsf{Setup}}$-hybrid.

The Setup functionality. The setup functionality $\mathcal{F}_{\mathsf{Setup}}$ (described in Algorithm 1) samples a PRF key α and a DEHE secret key k and distributes the corresponding ciphertexts to the clients in input layer $\mathcal{L}_0$. The first set of DEHE evaluation keys are additively shared among the clients. Further, the setup also sends the set of ciphertexts corresponding to all possible inputs of clients.

Note that due to the use of ciphertext ct_{k}, we require DEHE circular security. However, this can be avoided by having a different secret key for each layer (as elaborated in section 4).

Figure 1 Setup Functionality $\mathcal{F}_{\mathsf{Setup}}$

Computation:

- Sample a PRF key $\alpha \leftarrow \{0,1\}^\kappa$
- Generate keys $\mathsf{k} \leftarrow \mathsf{DEHE.KeyGen}(1^\kappa)$ and $(\mathsf{ek}_0^1, \mathsf{ek}_1^1) \leftarrow \mathsf{DEHE.Gen}(\mathsf{k})$
- Compute ciphertexts $(\mathsf{ct}_\alpha, \mathsf{ct}_{\mathsf{k}}) \leftarrow \mathsf{DEHE.Enc}(\mathsf{k}, (\alpha, \mathsf{k}))$
- Compute additive shares $\langle \mathsf{ek}_b^1 \rangle_j$ such that $\sum_{j \in [\ell_1]} \langle \mathsf{ek}_b^1 \rangle_j = \mathsf{ek}_b^1$ (for $b \in \{0,1\}$).
- $\forall j \in [\ell_1]$ and $b \in \{0,1\}$, compute $\mathsf{ct}_{j,b} \leftarrow \mathsf{Enc}(\mathsf{k}, b)$.

Outputs:

- $\forall j \in [\ell_1]$, C_j receives $(\mathsf{ct}_\alpha, \mathsf{ct}_{\mathsf{k}}, \langle \mathsf{ek}_0^1 \rangle_j, \langle \mathsf{ek}_1^1 \rangle_j, \mathsf{ct}_{j,0}, \mathsf{ct}_{j,1})$.

Computation. We now describe the computation performed using DEHE. Informally, as described above, each layer $\mathcal{L}_i$ computes $\mathbf{C}_i$ in a masked fashion, while also preparing the corresponding pairs of evaluation keys for the subsequent layer (a step that is unnecessary for the final layer). More formally, corresponding to each $i \in [\ell]$, we define a computation circuit P_i as follows:

Parameters: $i, y_{i-1} \in \{0,1\}^{\ell_i}$ if $i > 1$

Input: $\alpha, \mathsf{k} \in \{0,1\}^\kappa$, $y_0 \in \{0,1\}^{\ell_1}$ if $i = 1$

Compute: $(\mathsf{ek}'_0, \mathsf{ek}'_1) \leftarrow \mathsf{DEHE.Gen}(\mathsf{k}; F_\alpha(i, 0))$ if $i < \ell$

$$r_i = F_\alpha(i,1) \qquad r_{i-1} = F_\alpha(i-1,1)$$

$$y_i = \begin{cases} \mathbf{C}_i(y_{i-1}) + r_i & \text{if } i = 1 \\ \mathbf{C}_i(y_{i-1} - r_{i-1}) + r_i & \text{if } i < \ell \\ \mathbf{C}_i(y_{i-1} - r_{i-1}) & \text{otherwise} \end{cases}$$

Output:

$$\begin{cases} \mathsf{ek}'_0, \mathsf{ek}'_1, y_i & \text{if } i < \ell \\ y_i & \text{otherwise} \end{cases}$$

Layer Computation Sub-Protocols Π_i. We now describe a series of layered protocols Π_i. The protocol Π_i is described in Algorithm 2 for $i = 1$, in Algorithm 3 for $1 < i < \ell$, and in Algorithm 4 for $i = \ell$.

Figure 2 Protocol Π_1

Building Blocks: Let $\mathsf{DEHE} = (\mathsf{KeyGen}, \mathsf{Gen}, \mathsf{Enc}, \mathsf{Eval})$ be a two-party DEHE scheme (Definition 1).

Computation Layer $\mathcal{L}_1$:

- S_0^1 has inputs $\mathsf{ek}_0^1, \mathsf{ct}_\alpha, \mathsf{ct}_\mathsf{k}, \{\mathsf{ct}_j\}_{j\in[\ell_1]}$
- S_1^1 has inputs $\mathsf{ek}_1^1, \mathsf{ct}_\alpha, \mathsf{ct}_\mathsf{k}, \{\mathsf{ct}_j\}_{j\in[\ell_1]}$
- $S_\sigma^1(\sigma \in \{0,1\})$ does the following:
  - Compute $([\mathsf{ek}_0^2]_\sigma, [\mathsf{ek}_1^2]_\sigma, [y_1]_\sigma) \leftarrow \mathsf{DEHE.Eval}(\sigma, \mathsf{ek}_\sigma^1, (\mathsf{ct}_\alpha, \mathsf{ct}_\mathsf{k}, \{\mathsf{ct}_j\}_{j\in[\ell_1]}), P_1(1))$.
  - Broadcast $([y_1]_\sigma, \mathsf{ct}_\alpha, \mathsf{ct}_\mathsf{k})$ to parties S_0^2 and S_1^2.
  - Send $[\mathsf{ek}_b^2]_\sigma$ privately to S_b^2 for $b \in \{0,1\}$.

Computation Layer $\mathcal{L}_2$: Each party S_b^2 $(b \in \{0,1\})$ receives $([\mathsf{ek}_b^2]_0, [y_1]_0, \mathsf{ct}_\alpha, \mathsf{ct}_\mathsf{k})$ and $([\mathsf{ek}_b^2]_1, [y_1]_1, \mathsf{ct}_\alpha, \mathsf{ct}_\mathsf{k})$ from S_0^1 and S_1^1 respectively and does the following:

- Compute $\mathsf{ek}_b^2 = [\mathsf{ek}_b^2]_0 + [\mathsf{ek}_b^2]_1$
- Compute $y_1 = [y_1]_0 + [y_1]_1$.
- Output $\mathsf{ek}_b^2, y_1, \mathsf{ct}_\alpha, \mathsf{ct}_\mathsf{k}$.

Figure 3 Protocol Π_i $(1 < i < \ell)$

Building Blocks: Let $\mathsf{DEHE} = (\mathsf{KeyGen}, \mathsf{Gen}, \mathsf{Enc}, \mathsf{Eval})$ be a two-party DEHE scheme (Definition 1).

Computation Layer $\mathcal{L}_i$:

- S_0^i has inputs $\mathsf{ek}_0^i, y_{i-1}, \mathsf{ct}_\alpha, \mathsf{ct}_\mathsf{k}$.
- S_1^i has inputs $\mathsf{ek}_1^i, y_{i-1}, \mathsf{ct}_\alpha, \mathsf{ct}_\mathsf{k}$.
- $S_\sigma^i(\sigma \in \{0,1\})$ does the following:
  - Compute $([\mathsf{ek}_0^{i+1}]_\sigma, [\mathsf{ek}_1^{i+1}]_\sigma, [y_i]_\sigma) \leftarrow \mathsf{Eval}(\sigma, \mathsf{ek}_\sigma^i, (\mathsf{ct}_\alpha, \mathsf{ct}_\mathsf{k}), P_i(i, y_{i-1}))$.
  - Broadcast $([y_i]_\sigma, \mathsf{ct}_\alpha, \mathsf{ct}_\mathsf{k})$ to parties S_0^{i+1} and S_1^{i+1}.
  - Send $[\mathsf{ek}_b^{i+1}]_\sigma$ privately to S_b^{i+1} for $b \in \{0,1\}$.

Computation Layer $\mathcal{L}_{i+1}$: Each party S_b^{i+1} $(b \in \{0,1\})$ receives $([\mathsf{ek}_b^{i+1}]_0, [y_i]_0, \mathsf{ct}_\alpha, \mathsf{ct}_\mathsf{k})$ and $([\mathsf{ek}_b^{i+1}]_1, [y_i]_1, \mathsf{ct}_\alpha, \mathsf{ct}_\mathsf{k})$ from S_0^i and S_1^i respectively and does the following:

- Compute $\mathsf{ek}_b^{i+1} = [\mathsf{ek}_b^{i+1}]_0 + [\mathsf{ek}_b^{i+1}]_1$.
- Compute $y_i = [y_i]_0 + [y_i]_1$.
- Output $\mathsf{ek}_b^{i+1}, y_i, \mathsf{ct}_\alpha, \mathsf{ct}_\mathsf{k}$.

Figure 4 Protocol Π_ℓ

Building Blocks: Let $\mathsf{DEHE} = (\mathsf{KeyGen}, \mathsf{Gen}, \mathsf{Enc}, \mathsf{Eval})$ be a two-party DEHE scheme (Definition 1).

Computation Layer $\mathcal{L}_\ell$:

- S_0^ℓ has inputs $\mathsf{ek}_0^\ell, y_{\ell-1}, \mathsf{ct}_\alpha, \mathsf{ct}_\mathsf{k}$.
- S_1^ℓ has inputs $\mathsf{ek}_1^\ell, y_{\ell-1}, \mathsf{ct}_\alpha, \mathsf{ct}_\mathsf{k}$.
- Each $S_\sigma^\ell(\sigma \in \{0,1\})$ computes $[y_\ell]_\sigma \leftarrow \mathsf{Eval}(\sigma, \mathsf{ek}_\sigma^\ell, (\mathsf{ct}_\alpha, \mathsf{ct}_\mathsf{k}), P_\ell(\ell, y_{\ell-1}))$.
- Broadcast $[y_\ell]_\sigma$ to the output layer.

Output Layer $\mathcal{L}_{\ell+1}$: Each output party $P_i^{\ell+1}$ receives $[y_\ell]_\sigma$ from S_σ^ℓ $(\sigma \in \{0,1\})$ and outputs $y_\ell = [y_\ell]_0 + [y_\ell]_1$.

The final theorem is stated below.

Theorem 3. *Let* $\mathsf{DEHE} = (\mathsf{KeyGen}, \mathsf{Gen}, \mathsf{Enc}, \mathsf{Eval})$ *be a two-party DEHE scheme (Definition 1) and let* F *be a PRF (Definition 12) The layered MPC protocol* Π_{sh} *in Algorithm 5 securely realizes the layered functionality* f *in the* $\mathcal{F}_{\mathsf{Setup}}$*-hybrid (Algorithm 1) satisfying semi-honest security against an adversary that can corrupt all-but-one party in* $\mathcal{L}_0$ *and* $\mathcal{L}_{\ell+1}$*, and one party in each* $\mathcal{L}_i, i \in [\ell]$.

The complete proof for this can be found in Appendix C.

Figure 5 Semi-Honest Secure Protocol Π_{sh} in the $\mathcal{F}_{\mathsf{Setup}}$-hybrid model

Let $\mathcal{F}_{\mathsf{Setup}}$ be the functionality given in Algorithm 1. Let $\Pi_1, \forall i > 1 \in [\ell - 1], \Pi_i$ and Π_ℓ be the functionalities in Algorithms 2 to 4 respectively.

Input Layer $\mathcal{L}_0$: For $j \in [\ell_1]$, each party $P_j^0 \in \mathcal{L}_0$ has an input $x_1[j]$ and does the following:
- Receive $(\mathsf{ct}_\alpha, \mathsf{ct}_\mathsf{k}, \langle \mathsf{ek}_0^1 \rangle_j, \langle \mathsf{ek}_0^1 \rangle_j, \mathsf{ct}_{j,0}, \mathsf{ct}_{j,1})$ from $\mathcal{F}_{\mathsf{Setup}}$.
- For each $\sigma \in \{0,1\}$: Broadcast $(\mathsf{ct}_{j,x_1[j]}, \mathsf{ct}_\alpha, \mathsf{ct}_\mathsf{k})$ and send $\langle \mathsf{ek}_\sigma^1 \rangle_j$ privately to S_σ^1.

Computation Layer $\mathcal{L}_1$: Each server S_σ^1 $(\sigma \in \{0,1\})$ does the following:
- Receive $(\mathsf{ct}_{j,x_1[j]}, \mathsf{ct}_\alpha, \mathsf{ct}_\mathsf{k}, \langle \mathsf{ek}_\sigma^1 \rangle_j)$ from each P_j^0 $(j \in [\ell_1])$.
- Set $\mathsf{ek}_\sigma^1 = \sum_{j \in [\ell_1]} \langle \mathsf{ek}_\sigma^1 \rangle_j$ and invoke protocol Π_1 with input $(\mathsf{ek}_\sigma^1, \mathsf{ct}_\alpha, \mathsf{ct}_\mathsf{k}, \{\mathsf{ct}_j\}_{j \in [\ell_1]})$.

Computation Layers $\mathcal{L}_i (1 < i \leq \ell)$: Each server S_σ^i $(\sigma \in \{0,1\})$ does the following:
- Receive $(\mathsf{ek}_\sigma^i, y_{i-1}, \mathsf{ct}_\alpha, \mathsf{ct}_\mathsf{k})$ as output from Π_{i-1}.
- Invoke Π_i with input $(\mathsf{ek}_\sigma^i, y_{i-1}, \mathsf{ct}_\alpha, \mathsf{ct}_\mathsf{k})$.

Output Layer $\mathcal{L}_{\ell+1}$: Each party $P_i^{\ell+1} \in \mathcal{L}_{\ell+1}$ outputs y_ℓ after completing Π_ℓ.

Communication Complexity. The total communication complexity of protocol Π_{sh} is $O(\kappa \cdot (m + \ell) + \frac{|\mathbf{C}|}{\log |\mathbf{C}|})$, where κ is the security parameter and m is the number of input bits. We note that the total communication by parties in the input layer is $O(m \cdot \kappa)$, constituted by the input ciphertexts. Each server in the intermediate computation layer $\mathcal{L}_i$ broadcasts a message of size $O(|y_i| + \kappa) = O(W + \kappa)$ (where W denotes the width of the circuit $\mathbf{C}$). Thus, the total communication complexity for the protocol is $O(\kappa \cdot (m + \ell) + \ell \cdot W)$, where ℓ denotes the number of layers. It follows from the circuit decomposition that $\ell = D / \log |\mathbf{C}|$[12], where D is the circuit depth of $\mathbf{C}$. Therefore the communication complexity is $O(\kappa \cdot (m + \ell) + \frac{|\mathbf{C}|}{\log |\mathbf{C}|})$.

4 Layered MPC Protocols Satisfying Security with Abort

In this section we formalize a layered MPC protocol satisfying security with unanimous abort in the presence of malicious adversaries. This protocol can be used to compute any functionality wherein all parties in the output layer receive the same output. Our protocol requires two servers per layer of computation and enjoy communication complexity sublinear in the size of the layered functionality to be computed. Further, the protocol supports long on-going computation where the complete functionality is not known at the beginning and may evolve throughout the protocol progression.

The protocol we present works by simply augmenting the passively secure protocol presented in Section 3 with digital signature-based consistency checks that ensure each party participated in the protocol semi-honestly. This construction is simple, has a clean description and has communication complexity that is asymptotically sub-linear in the size of the functionality being computed. In Appendix B, we additionally present another such protocol that is specific to a particular instantiation of the underlying HSS scheme and is concretely efficient. For both protocols, we also discuss how these can be extended to support functionalities with differing outputs to each output party, and functionalities represented using arithmetic circuits, with inputs that are not binary.

[12] The details of how a layered boolean circuit can be decomposed into a layered circuit over $\mathbf{NC}^1$ / branching programs appears in [BGI16, Cou19].

Protocol based on General Primitives. We first present a layered MPC protocol whose description is black-box in the following three building-blocks:

- A two-party Distributed Evaluation Homomorphic Encryption scheme (Definition 1) $\mathsf{DEHE} = (\mathsf{KeyGen}, \mathsf{Gen}, \mathsf{Enc}, \mathsf{Eval})$ for RMS programs, such that the Gen algorithm is an RMS program.
- A PRF (Definition 12)F which, once the input is hard-coded, can be computed as an $\mathbf{NC}^1$ function of the key.[13]
- A Signature Scheme (Definition 14)$\mathcal{S} = (\mathsf{Gen}, \mathsf{Sign}, \mathsf{Verify})$, whose description is within $\mathbf{NC}^1$ and message space $\mathcal{M}$ is the circuit output space.

In the following, we first present a functionality $\mathcal{F}_{\mathsf{Setup}}$ that distributes setup information into the input layer $\mathcal{L}_0$. Then we describe the computation performed within the RMS program for each layer. Using these, we formalize DEHE-based sub-protocols Π_i that captures the computation of a pair of parties in each ith computation layer and the messages sent to the next layer. Finally, we present our main result: a protocol that realizes any functionality in the $\mathcal{F}_{\mathsf{Setup}}$-hybrid.

The Setup Functionality. Our protocol uses a setup functionality that,

- distributes messages among input parties in $\mathcal{L}_0$ that allow it to verifiably encrypt the protocol inputs to the first computation layer $\mathcal{L}_1$
- (verifiably) additively secret-shares the DEHE evaluation keys for the first computation layer among the parties in $\mathcal{L}_0$. Each party in $\mathcal{L}_0$ is given its share signed using a secret signing-key.
- broadcasts the signature verification key vk, and DEHE encryption of the PRF key k and signing key sk.

This functionality $\mathcal{F}_{\mathsf{Setup}}$ is detailed in Algorithm 6 below. It does not take any inputs but uses all the building-blocks stated above to generate the mentioned setup information, distributing it among parties in $\mathcal{L}_0$.

Figure 6 Setup Functionality $\mathcal{F}_{\mathsf{Setup}}$

Let $\mathsf{DEHE} = (\mathsf{KeyGen}, \mathsf{Gen}, \mathsf{Enc}, \mathsf{Eval})$ be a two-party DEHE scheme (Definition 1). Let F be a PRF (Definition 12)and $\mathcal{S} = (\mathsf{Gen}, \mathsf{Sign}, \mathsf{Verify})$ be a signature scheme (Definition 14). $\mathcal{F}_{\mathsf{Setup}}$ works as follows:

Computation:

- generate $\mathsf{k} \leftarrow \mathsf{DEHE.KeyGen}(1^\kappa)$ and $(\mathsf{ek}_0^1, \mathsf{ek}_1^1) \leftarrow \mathsf{DEHE.Gen}(\mathsf{k})$
- generate $(\mathsf{sk}, \mathsf{vk}) \leftarrow \mathcal{S}.\mathsf{Gen}(1^\kappa)$
- sample a PRF key $\alpha \leftarrow \{0,1\}^\kappa$
- compute $(\mathsf{ct}_\alpha, \mathsf{ct}_\mathsf{k}, \mathsf{ct}_\mathsf{sk}) \leftarrow \mathsf{DEHE.Enc}(\mathsf{k}, (\alpha, \mathsf{k}, \mathsf{sk}))$
- $\forall j \in [\ell_1]$, $b \in \{0,1\}$, compute $\mathsf{ct}_{j,b} \leftarrow \mathsf{DEHE.Enc}(\mathsf{k}, b)$ and $\sigma^{\mathsf{ct}}_{b,j} \leftarrow \mathcal{S}.\mathsf{Sign}(\mathsf{sk}, (0, b, j, \mathsf{ct}_{j,b}))$
- for $b \in \{0,1\}$, sample $\{\langle \mathsf{ek}_b^1 \rangle_j\}_{j \in [\ell_1]}$ such that $\mathsf{ek}_b^1 = \sum_{j \in [\ell_1]} \langle \mathsf{ek}_b^1 \rangle_j$
- for $b \in \{0,1\}, j \in [\ell_1]$, compute $\sigma_{b,j} = \mathcal{S}.\mathsf{Sign}(\mathsf{sk}, (0, b, j, \langle \mathsf{ek}_b^1 \rangle_j))$

Outputs:

- $\forall j \in [\ell_1]$, C_j receives

$$\{b : \mathsf{ct}_{j,b}, \sigma^{\mathsf{ct}}_{j,b}\}_{b \in \{0,1\}}, \mathsf{vk}, \mathsf{ct}_\alpha, \mathsf{ct}_\mathsf{k}, \mathsf{ct}_\mathsf{sk}, \{\langle \mathsf{ek}_b^1 \rangle_j, \sigma_{b,j}\}_{b \in \{0,1\}}$$

[13] Such PRFs are known from many standard algebraic assumptions: DDH/factoring [NR97, NR04, NRR00]

Overview of the protocol. The core idea behind this protocol is to verify the correctness of parties' actions without relying on mechanisms like zero-knowledge proofs. The approach for achieving this is described below.

Verifying input sharing by clients: The functionality $\mathcal{F}_{\mathsf{Setup}}$ generates ciphertexts for every possible input of each client C_j and distributes the complete set to all clients. Each client then selects the ciphertext corresponding to its actual input and sends it, together with the signature on it, to the servers in $\mathcal{L}_1$. The servers in $\mathcal{L}_1$ can then verify the signature and abort if the check does not pass.

Verifying common information: The common information given to clients as part of setup (such as the signature verification key, and ciphertexts corresponding to the PRF key, DEHE secret key and signing key) can simply be propagated through the layers. Verification is straightforward: each party checks that it received identical values from all parties in the previous layer. Since at least one honest party is present in every layer, this suffices to ensure the correctness of the common information.

Verifying DEHE evaluation: The DEHE evaluation at layer $\mathcal{L}_i$ computes the intermediate output y_i, following the same procedure as in the semi-honest construction described in Section 3. To guarantee that y_i is correctly transmitted to $\mathcal{L}_{i+1}$, the evaluation at $\mathcal{L}_i$ additionally generates a signature on y_i using a key sk (whose DEHE ciphertext is part of the common information). Consequently, any attempt by a malicious server in $\mathcal{L}_i$ to alter y_i would be detected, as it would fail the signature verification performed at layer $\mathcal{L}_{i+1}$. Lastly, note that the broadcast by parties in $\mathcal{L}_\ell$ ensures that all output parties reach unanimous agreement.

DEHE Computation. We now describe the computation performed using DEHE. Let $\mathbf{C} = \mathbf{C}_\ell \circ \cdots \circ \mathbf{C}_1$ be a layered circuit where for each $i \in [\ell]$, $\mathbf{C}_i : \{0,1\}^{\ell_i} \to \{0,1\}^{\ell_{i+1}} \in \mathbf{NC}^1$. Corresponding to each $i \in [\ell]$, we define a computation circuit P_i as follows:

Parameters: i, $\begin{cases} y_{i-1} \in \{0,1\}^{\ell_i} & \text{for } i > 1 \end{cases}$

Input: $\alpha, \mathsf{k}, \mathsf{sk} \in \{0,1\}^\kappa$, $\begin{cases} y_0 \in \{0,1\}^{\ell_1} & \text{for } i = 1 \end{cases}$

Compute:

$$(\mathsf{ek}'_0, \mathsf{ek}'_1) = \mathsf{DEHE.Gen}(\mathsf{k}; F_\alpha(i,0)) \qquad \text{if } i < \ell$$

$$b \in \{0,1\}, \sigma_b^{\mathsf{ek}} = \mathcal{S}.\mathsf{Sign}(\mathsf{sk}, (i, b, \mathsf{ek}'_b); F_\alpha(3i+b, 1)) \text{ if } i < \ell$$

$$r_i = F_\alpha(i,2) \qquad r_{i-1} = F_\alpha(i-1,2)$$

$$y_i = \begin{cases} \mathbf{C}_i(y_{i-1}) + r_i & \text{if } i = 1 \\ \mathbf{C}_i(y_{i-1} - r_{i-1}) + r_i & \text{if } i < \ell \\ \mathbf{C}_i(y_{i-1} - r_{i-1}) & \text{otherwise} \end{cases}$$

$$\sigma_i = \mathcal{S}.\mathsf{Sign}(\mathsf{sk}, (i, 2, y_i); F_\alpha(3i+2, 1))$$

Output:

$$\begin{cases} \mathsf{ek}'_0, \sigma_0^{\mathsf{ek}}, \mathsf{ek}'_1, \sigma_1^{\mathsf{ek}}, y_i, \sigma_i & \text{if } i < \ell \\ y_i, \sigma_i & \text{otherwise} \end{cases}$$

The layer computation sub-protocols. We describe sub-protocols Π_i for each computation layer $i \in [\ell]$ in the layered MPC protocol in Algorithm 7 for $i = 1$, in Algorithm 8 for $1 < i < \ell$, and in Algorithm 9 for $i = \ell$.

Layered MPC satisfying Security with Unanimous Abort. We now present a layered MPC protocol that can compute any functionality securely with unanimous abort, where the functionality provides the same outputs to each party in the output layer. This complete layered MPC protocol is detailed Algorithm 10 below. We also later discuss how to extend such a protocol to realize functionalities where each output party receives a different output.

Figure 7 Layer protocol Π_1

Let $\mathsf{DEHE} = (\mathsf{KeyGen}, \mathsf{Gen}, \mathsf{Enc}, \mathsf{Eval})$ be a two-party DEHE scheme (Definition 1) and $\mathcal{S} = (\mathsf{Gen}, \mathsf{Sign}, \mathsf{Verify})$ be a signature scheme (Definition 14). The protocol Π_1 works as follows:

Computation Layer $\mathcal{L}_1$: for $b \in \{0,1\}$, each S_b^1 does the following:

- it has inputs $\mathsf{vk}, \mathsf{ek}_b, \mathsf{ct}_\alpha, \mathsf{ct}_\mathsf{k}, \mathsf{ct}_\mathsf{sk}, \mathsf{ct}_{y_0}, \sigma_{y_0}$
- if $\forall i \in [\ell_1]$, it holds that $\top \leftarrow \mathcal{S}.\mathsf{Verify}(\mathsf{vk}, \mathsf{ct}_{y_0}[i], \sigma_{y_0}[i])$,
  - compute $([\mathsf{ek}'_0]_b, [\sigma_0^{\mathsf{ek}}]_b, [\mathsf{ek}'_1]_b, [\sigma_1^{\mathsf{ek}}]_b, [y_1]_b, [\sigma_1]_b) \leftarrow \mathsf{DEHE.Eval}(b, \mathsf{ek}_b, (\mathsf{ct}_\alpha, \mathsf{ct}_\mathsf{k}, \mathsf{ct}_\mathsf{sk}, \mathsf{ct}_{y_0}), P_1(1))$

  else, abort and output $\perp$
- broadcast $\mathsf{vk}, \mathsf{ct}_\alpha, \mathsf{ct}_\mathsf{k}, \mathsf{ct}_\mathsf{sk}, [y_1]_b, [\sigma_1]_b$
- send $[\mathsf{ek}'_0]_b, [\sigma_0^{\mathsf{ek}}]_b$ to S_0^2
- send $[\mathsf{ek}'_1]_b, [\sigma_1^{\mathsf{ek}}]_b$ to S_1^2

Computation Layer $\mathcal{L}_2$: for $b \in \{0,1\}$, each S_b^2 does the following:

- it receives $\{\mathsf{vk}^{b'}, \mathsf{ct}_\alpha^{b'}, \mathsf{ct}_\mathsf{k}^{b'}, \mathsf{ct}_\mathsf{sk}^{b'}, [\mathsf{ek}_b]_{b'}, [\sigma_b^{\mathsf{ek}}]_{b'}, [y_1]_{b'}, [\sigma_1]_{b'}\}_{b' \in \{0,1\}}$ from $\mathcal{L}_1$
- **Consistency Checks:**
  - if $\mathsf{vk}^0 \neq \mathsf{vk}^1 = \mathsf{vk}$, abort and output $\perp$
  - set $\mathsf{ek}_b = [\mathsf{ek}_b]_0 + [\mathsf{ek}_b]_1$ and $\sigma_b^{\mathsf{ek}} = [\sigma_b^{\mathsf{ek}}]_0 + [\sigma_b^{\mathsf{ek}}]_1$
  - if $\top \not\leftarrow \mathcal{S}.\mathsf{Verify}(\mathsf{vk}, (i, b, \mathsf{ek}_b), \sigma_b^{\mathsf{ek}})$, abort and output $\perp$
  - set $y_1 = [y_1]_0 + [y_1]_1$ and $\sigma_1 = [\sigma_1]_0 + [\sigma_1]_1$
  - if $\top \not\leftarrow \mathcal{S}.\mathsf{Verify}(\mathsf{vk}, (i, 2, y_1), \sigma_1)$, abort and output $\perp$
  - if $\mathsf{ct}_\alpha^0 \neq \mathsf{ct}_\alpha^1 = \mathsf{ct}_\alpha$, abort and output $\perp$
  - if $\mathsf{ct}_\mathsf{k}^0 \neq \mathsf{ct}_\mathsf{k}^1 = \mathsf{ct}_\mathsf{k}$, abort and output $\perp$
  - if $\mathsf{ct}_\mathsf{sk}^0 \neq \mathsf{ct}_\mathsf{sk}^1 = \mathsf{ct}_\mathsf{sk}$, abort and output $\perp$
- output $\mathsf{vk}, \mathsf{ct}_\alpha, \mathsf{ct}_\mathsf{k}, \mathsf{ct}_\mathsf{sk}, \mathsf{ek}_b, y_1$

Figure 8 Layer protocol Π_i

Let DEHE be a two-party DEHE scheme (Definition 1) and $\mathcal{S} = (\mathsf{Gen}, \mathsf{Sign}, \mathsf{Verify})$ be a signature scheme (Definition 14). The protocol Π_i works as follows:

Computation Layer $\mathcal{L}_i$: for $b \in \{0,1\}$, each S_b^i does the following:

- it has inputs $\mathsf{vk}, \mathsf{ct}_\alpha, \mathsf{ct}_\mathsf{k}, \mathsf{ct}_\mathsf{sk}, \mathsf{ek}_b, y_{i-1}$
- compute $([\mathsf{ek}'_0]_b, [\sigma_0^{\mathsf{ek}}]_b, [\mathsf{ek}'_1]_b, [\sigma_1^{\mathsf{ek}}]_b, [y_i]_b, [\sigma_i]_b) \leftarrow \mathsf{DEHE.Eval}(b, \mathsf{ek}_b, (\mathsf{ct}_\alpha, \mathsf{ct}_\mathsf{k}, \mathsf{ct}_\mathsf{sk}), P_i(i, y_{i-1}))$
- broadcast $\mathsf{vk}, \mathsf{ct}_\alpha, \mathsf{ct}_\mathsf{k}, \mathsf{ct}_\mathsf{sk}, [y_i]_b, [\sigma_i]_b$
- send $[\mathsf{ek}'_0]_b, [\sigma_0^{\mathsf{ek}}]_b$ to S_0^{i+1}
- send $[\mathsf{ek}'_1]_b, [\sigma_1^{\mathsf{ek}}]_b$ to S_1^{i+1}

Computation Layer $\mathcal{L}_{i+1}$: for $b \in \{0,1\}$, each S_b^{i+1} does the following:

- it receives $\{\mathsf{vk}^{b'}, \mathsf{ct}_\alpha^{b'}, \mathsf{ct}_\mathsf{k}^{b'}, \mathsf{ct}_\mathsf{sk}^{b'}, [\mathsf{ek}_b]_{b'}, [\sigma_b^{\mathsf{ek}}]_{b'}, [y_i]_{b'}, [\sigma_i]_{b'}\}_{b' \in \{0,1\}}$ from $\mathcal{L}_i$
- **Consistency Checks:**
  - if $\mathsf{vk}^0 \neq \mathsf{vk}^1 = \mathsf{vk}$, abort and output $\perp$
  - set $\mathsf{ek}_b = [\mathsf{ek}_b]_0 + [\mathsf{ek}_b]_1$ and $\sigma_b^{\mathsf{ek}} = [\sigma_b^{\mathsf{ek}}]_0 + [\sigma_b^{\mathsf{ek}}]_1$
  - if $\top \not\leftarrow \mathcal{S}.\mathsf{Verify}(\mathsf{vk}, (i, b, \mathsf{ek}_b), \sigma_b^{\mathsf{ek}})$, abort and output $\perp$
  - set $y_i = [y_i]_0 + [y_i]_1$ and $\sigma_i = [\sigma_i]_0 + [\sigma_i]_1$
  - if $\top \not\leftarrow \mathcal{S}.\mathsf{Verify}(\mathsf{vk}, (i, 2, y_i), \sigma_i)$, abort and output $\perp$
  - if $\mathsf{ct}_\alpha^0 \neq \mathsf{ct}_\alpha^1 = \mathsf{ct}_\alpha$, abort and output $\perp$
  - if $\mathsf{ct}_\mathsf{k}^0 \neq \mathsf{ct}_\mathsf{k}^1 = \mathsf{ct}_\mathsf{k}$, abort and output $\perp$
  - if $\mathsf{ct}_\mathsf{sk}^0 \neq \mathsf{ct}_\mathsf{sk}^1 = \mathsf{ct}_\mathsf{sk}$, abort and output $\perp$
- output $\mathsf{vk}, \mathsf{ct}_\alpha, \mathsf{ct}_\mathsf{k}, \mathsf{ct}_\mathsf{sk}, \mathsf{ek}_b, y_i$

Theorem 4. *Let* $\mathsf{DEHE} = (\mathsf{KeyGen}, \mathsf{Gen}, \mathsf{Enc}, \mathsf{Eval})$ *be a two-party circular-secure DEHE scheme (Definition 1). Let* F *be a PRF (Definition 12)and* $\mathcal{S} = (\mathsf{Gen}, \mathsf{Sign}, \mathsf{Verify})$ *be a signature scheme (Definition 14). The layered MPC protocol* $\Pi_\perp$ *in Algorithm 10*

Figure 9 Layer protocol Π_ℓ

Let $\mathsf{DEHE} = (\mathsf{KeyGen}, \mathsf{Gen}, \mathsf{Enc}, \mathsf{Eval})$ be a two-party DEHE scheme (Definition 1) and $\mathcal{S} = (\mathsf{Gen}, \mathsf{Sign}, \mathsf{Verify})$ be a signature scheme (Definition 14). The protocol Π_ℓ works as follows:

Computation Layer $\mathcal{L}_\ell$: for $b \in \{0,1\}$, each S_b^ℓ does the following:

- it has inputs $\mathsf{vk}, \mathsf{ct}_\alpha, \mathsf{ct}_\mathsf{k}, \mathsf{ct}_\mathsf{sk}, \mathsf{ek}_b, y_{\ell-1}$
- compute $([y_\ell]_b, [\sigma_\ell]_b) \leftarrow \mathsf{DEHE.Eval}(b, \mathsf{ek}_b, (\mathsf{ct}_\alpha, \mathsf{ct}_\mathsf{k}, \mathsf{ct}_\mathsf{sk}), P_\ell(\ell, y_{\ell-1}))$
- broadcast $\mathsf{vk}, [y_\ell]_b, [\sigma_\ell]_b$

Output Layer $\mathcal{L}_{\ell+1}$: each output party $P_i^{\ell+1}$ does the following:

- receive $\{\mathsf{vk}_b, [y_\ell]_b, [\sigma_\ell]_b\}_{b\in\{0,1\}}$ from $\mathcal{L}_\ell$
- **Consistency Checks:**
  - if $\mathsf{vk}_0 \neq \mathsf{vk}_1 = \mathsf{vk}$, abort and output $\perp$
  - set $y_\ell = [y_\ell]_0 + [y_\ell]_1$ and $\sigma_\ell = [\sigma_\ell]_0 + [\sigma_\ell]_1$
  - if $\top \not\leftarrow \mathcal{S}.\mathsf{Verify}(\mathsf{vk}, (l, 2, y_\ell), \sigma_\ell)$, abort and output $\perp$
- output y_ℓ

Figure 10 Maliciously Secure Layered MPC satisfying Security with Abort

Let $\mathcal{F}_\mathsf{Setup}$ be the functionality given in Algorithm 6. Let Π_1, $\forall i > 1 \in [l-1], \Pi_i$ and Π_ℓ be the functionalities in Algorithms 7 to 9 respectively. Let $\mathcal{S} = (\mathsf{Gen}, \mathsf{Sign}, \mathsf{Verify})$ be a signature scheme (Definition 14). Then the layered protocol $\Pi_\perp$ in the $\mathcal{F}_\mathsf{Setup}$-hybrid operates as follows:

- **Input Layer $\mathcal{L}_0$:** Each party $P_i^0 \in \mathcal{L}_0$ has as input x_i to the protocol and operates as follows:
  - receive $\{b : \mathsf{ct}_{i,b}, \sigma^\mathsf{ct}_{i,b}\}_{b\in\{0,1\}}, \mathsf{vk}, \mathsf{ct}_\alpha, \mathsf{ct}_\mathsf{k}, \mathsf{ct}_\mathsf{sk}, \{\langle \mathsf{ek}_b^1\rangle_i, \sigma_{b,i}\}_{b\in\{0,1\}}$ from $\mathcal{F}_\mathsf{Setup}$
  - for $b \in \{0,1\}$, send the tuple $\left(\langle \mathsf{ek}_b^1\rangle_i, \sigma_{b,i}\right)$ to S_b^1
  - broadcast $\mathsf{ct}_{i,x_i}, \sigma^\mathsf{ct}_{i,x_i}, \mathsf{vk}, \mathsf{ct}_\alpha, \mathsf{ct}_\mathsf{k}, \mathsf{ct}_\mathsf{sk}$
- **Computation Layer $\mathcal{L}_1$:** Each party S_b^1 for $b \in \{0,1\}$ does the following:
  - receive $\left(\langle \mathsf{ek}_b^1\rangle_i, \sigma_{b,i}, \mathsf{ct}_{i,x_i}, \sigma^\mathsf{ct}_{i,x_i}, \mathsf{vk}^i, \mathsf{ct}^i_\alpha, \mathsf{ct}^i_\mathsf{k}, \mathsf{ct}^i_\mathsf{sk}\right)$ from each $P_i^0 \in \mathcal{L}_0$
  - Consistency Checks:
    - set $\mathsf{vk} = \mathsf{vk}^1$ and if $\exists P_i^0 \in \mathcal{L}_0$ s.t. $\mathsf{vk}^i \neq \mathsf{vk}$, then $\perp$
    - set $\mathsf{ct}_\alpha = \mathsf{ct}^1_\alpha$ and if $\exists P_i^0 \in \mathcal{L}_0$ s.t. $\mathsf{ct}^i_\alpha \neq \mathsf{ct}_\alpha$, then $\perp$
    - set $\mathsf{ct}_\mathsf{k} = \mathsf{ct}^1_\mathsf{k}$ and if $\exists P_i^0 \in \mathcal{L}_0$ s.t. $\mathsf{ct}^i_\mathsf{k} \neq \mathsf{ct}_\mathsf{k}$, then $\perp$
    - set $\mathsf{ct}_\mathsf{sk} = \mathsf{ct}^1_\mathsf{sk}$ and if $\exists P_i^0 \in \mathcal{L}_0$ s.t. $\mathsf{ct}^i_\mathsf{sk} \neq \mathsf{ct}_\mathsf{sk}$, then $\perp$
    - if $\forall i \in [\ell_1]$, it holds that $\top \leftarrow \mathcal{S}.\mathsf{Verify}(\mathsf{vk}, \langle \mathsf{ek}_b^1\rangle_i, \sigma_{b,i})$,
      - compute $\mathsf{ek}_b = \sum_{i\in[\ell_1]} \langle \mathsf{ek}_b^1\rangle_i$

      else, abort and output $\perp$
    - set $\mathsf{ct}_{y_0} = \{\mathsf{ct}_{i,x_i}\}_{i\in[\ell_0]}$ and $\sigma_{y_0} = \{\sigma^\mathsf{ct}_{i,x_i}\}_{i\in[\ell_0]}$
  - invoke protocol Π_1 using $\mathsf{vk}, \mathsf{ek}_b, \mathsf{ct}_\alpha, \mathsf{ct}_\mathsf{k}, \mathsf{ct}_\mathsf{sk}, \mathsf{ct}_{y_0}, \sigma_{y_0}$ as inputs
- $\forall i > 1 \in [\ell]$ **Computation Layer $\mathcal{L}_i$:** Each S_b^i for $b \in \{0,1\}$ does the following:
  - receive $\mathsf{vk}, \mathsf{ct}_\alpha, \mathsf{ct}_\mathsf{k}, \mathsf{ct}_\mathsf{sk}, \mathsf{ek}_b, y_{i-1}$ from the execution of Π_{i-1} and continue by invoking Π_i with these inputs
- **Output Layer $\mathcal{L}_{\ell+1}$:** Each party $P_i^{\ell+1} \in \mathcal{L}_{\ell+1}$ outputs y_ℓ after completing Π_ℓ

securely realizes the layered functionality f in the $\mathcal{F}_\mathsf{Setup}$-hybrid (Algorithm 6) satisfying security with unanimous abort tolerating a malicious adversary that can corrupt all-but-one party in $\mathcal{L}_0$ and $\mathcal{L}_{\ell+1}$, and one party in each $\mathcal{L}_i, i \in [\ell]$.

The complete proof for this can be found in Appendix D.

Communication Complexity. The overall communication complexity of $\Pi_\perp$ (Algorithm 10) is $O(\kappa(m+\ell) + \frac{|C|}{\log|C|})$ where κ is the security parameter, m is the number of input bits, ℓ is the number of $\mathbf{NC}^1$ layers in the circuit and $|C|$ denotes the circuit

size. This follows from the fact that the communication overhead of $\Pi_{\perp}$ as compared to Π_{sh} is derived from:

- For input size m, the setup $\mathcal{F}_{\mathsf{Setup}}$ sends an additional $O(\kappa + m\kappa)$ communication to parties in $\mathcal{L}_0$.
- For each $i \in [\ell]$, each server $S^i \in \mathcal{L}_i$ sends an additional $O(\kappa + |y_i|)$ bits to each server $S^{i+1} \in \mathcal{L}_{i+1}$.

Handling Boolean vs Arithmetic Circuits. In our handling of the protocol $\Pi_{\perp}$ above, note that we write the complete computation performed by HSS as operating on Boolean inputs and outputs. However, this is done only for ease of exposition and the protocol easily generalizes to handle intermediate values, outputs and masks over a larger arithmetic ring.

Handling Functionalities with Differing Outputs. The protocol in Algorithm 10 securely realizes any functionality where each party in the output layer $\mathcal{L}_{\ell+1}$ receives the same functionality output. This is an artifact of the protocol Π_{ℓ} (Algorithm 9) in which consistency is ensured by having the last computation layer $\mathcal{L}_{\ell}$ *broadcast* all its messages into $\mathcal{L}_{\ell+1}$. This can be extended to a protocol wherein each output party can receive a different output by tweaking the setup $\mathcal{F}_{\mathsf{Setup}}$ and the layered functionality f being computed as follows:

- For each $i \in [n]$, let $f_i(x)$ be the output that the output party P_i expects.
- $\mathcal{F}_{\mathsf{Setup}}$ can use the PRF F to generate a unique mask m_i for each P_i using the key k, and hand this output party its mask.
- The functionality f that $\Pi_{\perp}$ now computes is $f(x) = \{f_i(x) \oplus m_i\}_{i \in [n]}$ and this output set is received by each output party, that can now only unmask $f_i(x)$ using m_i.

Removing Circular Security. Note that the setup functionality $\mathcal{F}_{\mathsf{Setup}}$ outputs a DEHE ciphertext ct_{k}, that is propagated through each layer in the protocol in Algorithm 10. Therefore the security of this protocol inherently relies on the circular security of the DEHE scheme. This can be avoided by modifying the protocol in the following way:

- within $\mathcal{F}_{\mathsf{Setup}}$, for each $i \in [\ell]$, sample $\mathsf{k}_i \leftarrow \mathsf{DEHE.KeyGen}(1^{\kappa})$ and compute $(\mathsf{ct}^i_{\alpha}, \mathsf{ct}^i_{\mathsf{k}}, \mathsf{ct}^i_{\mathsf{sk}}) \leftarrow \mathsf{DEHE.Enc}(\mathsf{k}_i, (\alpha, \mathsf{k}_{i+1}, \mathsf{sk}))$ – here α is the PRF key and sk is the signing key
- input ciphertexts are computed under k_1 only, and each client receives the set of all ciphertexts $\{(\mathsf{ct}^i_{\alpha}, \mathsf{ct}^i_{\mathsf{k}}, \mathsf{ct}^i_{\mathsf{sk}})\}_{i \in [\ell]}$
- for each $i \in [\ell]$, the tuple $(\mathsf{ct}^i_{\alpha}, \mathsf{ct}^i_{\mathsf{k}}, \mathsf{ct}^i_{\mathsf{sk}})$ is propagated till the ith computation layer $\mathcal{L}_i$ and the DEHE evaluation within this layer uses evaluation keys generated from k_i – generated within the RMS program in layer $\mathcal{L}_{i-1}$

The above modification would result in the overall communication complexity of the protocol being $O(\kappa(m + \ell^2) + \frac{|C|}{\log |C|})$, incurring a quadratic dependence on the number of layers ℓ, instead of linear.

Weak Identifiable Abort. The protocol in Algorithm 10 can also be modified slightly to ensure that parties in the output layer $\mathcal{L}_{\ell+1}$ learn the identities of at least one computation layer $i \in [\ell]$ where one of the parties is corrupt, if the protocol has aborted. This incriminates both the honest and corrupt party in that computation layer. This can be done as follows. If the consistency checks to be performed by parties

in any layer $\mathcal{L}_j$ cause the protocol to abort, broadcast $(\perp, j-1)$ to the next layer with the identities of both parties in $\mathcal{L}_{j-1}$. In any subsequent layer $\mathcal{L}_k$, if an honest party receives the same such message from both parties in the previous layer, it broadcasts this messages into $\mathcal{L}_{k+1}$. Otherwise, it broadcasts $(\perp, k-1)$ to the next layer with the identities of both parties in $\mathcal{L}_{k-1}$. This way, the parties of the output layer learn the identities of the latest layer that contains a party that behaved dishonestly in the protocol.

5 Layered MPC Protocol Satisfying Fairness

In this section we present a layered protocol that can securely compute any functionality in the presence of malicious adversaries while guaranteeing fairness when one party in each intermediate layer is corrupted and arbitrary corruption in the input and output layers. More precisely, we present a generic transform that takes any secure layered protocol for that functionality with *two* parties in each intermediate computation layer (like that in Section 4) satisfying security with abort, and convert it into a protocol where the last computation layer (before the output layer) requires *three* parties and tolerates one corruption. On a high level, our transform ensures that if the protocol has not aborted until this layer, output is guaranteed to all parties beyond it in the output layer. Notably, our assumption of one honest majority committee (constituted by the aforementioned three-party committee with one corruption) is both necessary and minimal (as fair MPC is known to be impossible in dishonest majority [Cle86]).

Notation. For any $m, z \in \mathbb{N}$ and layered functionality f, let $\Pi_\perp$ be a $(m, 2, z)$-layered protocol realizing f. Let $\Pi_\perp$ have $d+1$ layers and satisfy security with abort in the presence of arbitrary corruption in the input and output layer, $\mathcal{L}_0$ and $\mathcal{L}_d$ respectively, and one corrupt party in all other layers $\mathcal{L}_i$, for all $i \in [d-1]$.

Transformation. We present a generic transform that, can produce a protocol Π_{fair} that can compute any functionality f using $d+2$ layers, given a protocol $\Pi_\perp$ computing a slightly different functionality f' using $d+1$ layers. In more detail, for any function f over input $\vec{x}$, we can define a functionality f' as follows:

$$f'(\vec{x}) = \mathsf{RSS.Share}(f(\vec{x})) \tag{1}$$

where $\mathsf{RSS.Share}$ is a PPT algorithm for an honest secret-sharing scheme with robust reconstruction (Definition 10). The multi-party layered functionality f' distributes one output share from RSS to each party in its output layer $\mathcal{L}_d$.

The high-level idea behind the layered protocol Π_{fair} achieving fairness, is to use a protocol $\Pi_\perp$ realizing f' to distribute output shares into an honest-majority committee $\mathcal{L}_d$ (with three parties and tolerating one corruption). Then each of these parties can send their shares to each party in the output committee $\mathcal{L}_{d+1}$, that can now reconstruct the output. Note that if the computation has progressed without aborting until layer $\mathcal{L}_d$ is invoked, as long as only one party in that layer is corrupted, there is guaranteed to exist enough honest shares with each output party to reconstruct the output. This transformed protocol is detailed in Algorithm 11 below.

Theorem 5. *Let* RSS *be a secret-sharing scheme with robust reconstruction (Definition 10) and* $\Pi_\perp$ *be a* $(m, 2, z)$*-layered protocol realizing the functionality* f' *(Equation (1)) and satisfying security with abort in the presence of malicious adversaries*

Figure 11 Maliciously Secure Layered MPC satisfying Fairness

Let $\mathsf{RSS} = (\mathsf{Share}, \mathsf{Recon})$ be a 2-out-of-3 secret-sharing scheme with robust reconstruction (Definition 10). Let $\Pi_\perp$ be a $(m, 2, z)$-layered protocol over $d+1$ layers, maliciously securely realizing the functionality f' (Equation (1)) with security with abort tolerating all-but-one corrupt input and output parties and one corrupt party in each intermediate layer. The protocol Π_{fair} works as follows:

- **Input Layer $\mathcal{L}_0$:** m input parties $\{P_0^j\}_{j\in[m]}$ comprise this layer. Each party P_0^j enters its input x_j to the functionality f'.
- **Computation Layer $\mathcal{L}_d$:** three parties P_d^1, P_d^2 and P_d^3 comprise this layer. Each party P_d^j for $j \in [3]$,
  - accepts from the layered functionality f' a share y_j (that is the jth share of a 2-out-of-3 robust secret-sharing of $f(\vec{x})$).
  - broadcasts its share y_j to each party in the output layer.
- **Output Layer $\mathcal{L}_{d+1}$:** For $i \in [z]$, each output party P_{d+1}^i,
  - accepts a set of RSS shares $\{y_j\}_{j\in[3]}$ from the previous layer.
  - computes and outputs $f(\vec{x}) = \mathsf{RSS.Recon}(\{y_j\}_{j\in[3]})$

corrupting all-but-one input and output parties and one party in each intermediate computation layer. Then the protocol in Algorithm 11 is a layered MPC protocol (Definition 1) realizing f and satisfying fairness in the presence of malicious adversaries corrupting all-but-one input and output parties and one party in each intermediate computation layer, with exactly one among the computation layers comprising of three parties.

The complete proof can be found in Appendix E.

Communication Complexity. Asymptotically, the protocol given in Algorithm 11 has the same communication complexity as that of the base protocol $\Pi_\perp$. Therefore, if this base protocol $\Pi_\perp$ is instantiated using the protocol we present in Section 4, the transform results in a layered MPC protocol satisfying fairness with communication complexity sub-linear in size of the functionality representation.

6 Layered MPC Protocol Satisfying GOD

We now present a layered MPC protocol that achieves Guaranteed Output Delivery (GOD) against a malicious adversary corrupting at most one party per computation layer. In contrast to the protocol in Section 4, which provides security with unanimous abort using two parties per layer, this construction utilizes *three* parties per computation layer.

Conceptually, the protocol can be viewed as running three non-blackbox executions of the aborting layered MPC protocol from Section 4 in parallel, one for each unordered pair of parties among the three parties S_1^i, S_2^i, S_3^i for computation layer $\mathcal{L}_i$. We index these threads by the unordered pairs $K_i = \{(S_1^i, S_2^i), (S_2^i, S_3^i), (S_3^i, S_1^i)\}$. Each execution thread maintains its own authenticated intermediate state and evaluation keys.

Since the adversary corrupts at most one party per computation layer, at least one of these executions is between two honest parties and therefore proceeds without abort. Executions involving the corrupted party may deviate, but such deviations are detected by honest parties in the immediately succeeding layer via local consistency checks. As a result, corrupted executions are deterministically discarded, while at least one honest execution survives in every layer and propagates a valid authenticated

state to the end, yielding guaranteed output delivery. As a result, although multiple execution threads may exist transiently, at the end of each computation layer all honest parties agree on a unique intermediate value y_i, which is propagated to the next layer. Since at least one honest execution survives every layer, the protocol guarantees progress and ensures that a valid authenticated output is delivered at the end.

To realize this efficiently, we modify the setup functionality to support three parallel executions and the underlying computation the parties do in the DEHE such that the protocol yields guaranteed output delivery in the end.

Modified Setup Functionality. In the GOD protocol, the setup phase must support multiple parallel execution threads in each computation layer. To this end, the setup functionality generates evaluation material independently for each execution thread, while using a single global verification key to authenticate all setup information.

Formally, we define a modified setup functionality $\mathcal{F}_{\mathsf{Setup}}^{\mathsf{GOD}}$ (Algorithm 12) as an extension of $\mathcal{F}_{\mathsf{Setup}}$ (Algorithm 6) that generates independent evaluation keys and corresponding signatures per thread, but signs all material under a single verification key. All public verification information is shared across threads, while evaluation keys are generated independently to ensure that executions remain unlinkable.

Moreover, additive secret sharing used in the aborting setup is replaced by a robust secret sharing scheme with threshold $t < n/2$. This ensures that setup material can be reconstructed correctly despite malicious behavior, which is necessary to guarantee output delivery.

Figure 12 Setup Functionality $\mathcal{F}_{\mathsf{Setup}}^{\mathsf{GOD}}$

Let $K_1 = \{k_{12}, k_{23}, k_{31}\}$ be the set of threads assigned to pairs $\{S_1^1, S_2^1\}$, $\{S_2^1, S_3^1\}$, and $\{S_3^1, S_1^1\}$ respectively.

Computation:

- generate $\mathsf{k} \leftarrow \mathsf{DEHE.KeyGen}(1^\kappa)$
- for each thread $k \in K_1$, compute $(\mathsf{ek}_{0,k}^1, \mathsf{ek}_{1,k}^1) \leftarrow \mathsf{DEHE.Gen}(\mathsf{k})$.
- generate $(\mathsf{sk}, \mathsf{vk}) \leftarrow \mathcal{S}.\mathsf{Gen}(1^\kappa)$ and sample PRF key $\alpha \leftarrow \{0,1\}^\kappa$
- compute $(\mathsf{ct}_\alpha, \mathsf{ct}_\mathsf{k}, \mathsf{ct}_\mathsf{sk}) \leftarrow \mathsf{DEHE.Enc}(\mathsf{k}, (\alpha, \mathsf{k}, \mathsf{sk}))$
- $\forall j \in [\ell_1]$, $b \in \{0,1\}$, compute $\mathsf{ct}_{j,b} \leftarrow \mathsf{DEHE.Enc}(\mathsf{k}, b)$ and $\sigma_{j,b}^{\mathsf{ct}} \leftarrow \mathcal{S}.\mathsf{Sign}(\mathsf{sk}, (0, j, b, \mathsf{ct}_{j,b}))$
- for each thread $k \in K_1$, $b \in \{0,1\}$, compute shares $\{\langle \mathsf{ek}_{b,k}^1 \rangle_j \leftarrow \mathsf{Share}(\mathsf{ek}_{b,k}^1)\}_{j \in [\ell_1]}$
- For each $k \in K_1$, $b \in \{0,1\}$, $j \in [\ell_1]$, sign shares $\sigma_{b,k,j} = \mathcal{S}.\mathsf{Sign}(\mathsf{sk}, (0, b, k, j, \langle \mathsf{ek}_{b,k}^1 \rangle_j))$

Outputs:

- **Private Output to C_j in $\mathcal{L}_0$:** send the key shares $\{\langle \mathsf{ek}_{b,k}^1 \rangle_j, \sigma_{b,k,j}\}_{k \in K_1, b \in \{0,1\}}$ and input ciphertexts $\{\mathsf{ct}_{j,b}, \sigma_{j,b}^{\mathsf{ct}}\}_{b \in \{0,1\}}$
- **Public Output:** broadcast the verification key vk and ciphertexts $(\mathsf{ct}_\alpha, \mathsf{ct}_\mathsf{k}, \mathsf{ct}_\mathsf{sk})$

Modified DEHE Computation. Let $\mathbf{C} = \mathbf{C}_\ell \circ \cdots \circ \mathbf{C}_1$ be a layered circuit where for each $i \in [\ell]$, $\mathbf{C}_i : \{0,1\}^{\ell_i} \to \{0,1\}^{\ell_{i+1}} \in \mathbf{NC}^1$. As in the case with $\Pi_\perp$, we define a computation circuit P_i corresponding to each $i \in [\ell]$. However, we modify the computation so that each execution thread in layer i generates evaluation key material for all execution threads in the subsequent layer.

Since honest parties do not know which party (if any) is corrupted, they cannot determine in advance which execution will survive. Consequently, for every $k \in K_i$

and every $k' \in K_{i+1}$, the execution prepares a distinct evaluation key pair intended for use in the execution $P_{i+1,k'}$, resulting in a total of $|K_i| \cdot |K_{i+1}| = 9$ evaluation key pairs per layer.

Each evaluation key pair is derived using a PRF with explicit domain separation on the index of the target execution in the next layer, ensuring that all evaluation keys are fresh and independent across both execution threads and layers. Evaluation keys associated with corrupted executions are identified and discarded in later layers; we defer the details of this pruning procedure to the description of the modified sub-protocols. Corresponding to each layer i and thread $k \in K_i$, we define a computation circuit $P_{i,k}$ as follows:

Parameters: $i, k, \begin{cases} y_{i-1} \in \{0,1\}^{\ell_i} & \text{for } i > 1 \\ y_0^* \in \{0,1\}^{\ell_1^*} & \text{for } i = 1 \end{cases}$

Input: $\alpha, \mathsf{k}, \mathsf{sk} \in \{0,1\}^\kappa, \begin{cases} y_0' \in \{0,1\}^{\ell_1 - \ell_1^*} & \text{for } i = 1 \end{cases}$

Compute: if $i = 1$, $y_0 = y_0' \cup y_0^*$

$$\text{if } i < \ell, \forall k' \in K_{i+1}, \begin{cases} (\mathsf{ek}'_{0,k,k'}, \mathsf{ek}'_{1,k,k'}) = \mathsf{DEHE.Gen}(\mathsf{k}; F_\alpha(i,k,k',0)) \\ \sigma^{\mathsf{ek}}_{0,k,k'} = \mathcal{S}.\mathsf{Sign}(\mathsf{sk}, (i,0,k,k',\mathsf{ek}'_{0,k,k'}); F_\alpha(3i,k,k',1)) \\ \sigma^{\mathsf{ek}}_{1,k,k'} = \mathcal{S}.\mathsf{Sign}(\mathsf{sk}, (i,1,k,k',\mathsf{ek}'_{1,k,k'}); F_\alpha(3i+1,k,k',1)) \end{cases}$$

$$r_i = F_\alpha(i,0,0,2) \qquad r_{i-1} = F_\alpha(i-1,0,0,2)$$

$$y_i = \begin{cases} \mathbf{C}_i(y_{i-1}) + r_i & \text{if } i = 1 \\ \mathbf{C}_i(y_{i-1} - r_{i-1}) + r_i & \text{if } i < \ell \\ \mathbf{C}_i(y_{i-1} - r_{i-1}) & \text{otherwise} \end{cases}$$

$$\sigma_i = \mathcal{S}.\mathsf{Sign}(\mathsf{sk}, (i,2,0,0,y_i); F_\alpha(3i+2,0,0,1))$$

Output: $y_i, \sigma_i, \begin{cases} \{\mathsf{ek}'_{0,k,k'}, \sigma^{\mathsf{ek}}_{0,k,k'}, \mathsf{ek}'_{1,k,k'}, \sigma^{\mathsf{ek}}_{1,k,k'}\}_{k' \in K_{i+1}} & \text{if } i < \ell \end{cases}$

The modified layer computation protocols. We now give a brief overview of the modified sub-protocols used to evaluate the layered circuit, before formally specifying them.

The Layer Protocol Π_1. The GOD variant of the first computation layer protocol Π_1 differs from its abort counterpart by executing multiple parallel threads and replacing aborts with thread elimination; as specified in the modified setup functionality and DEHE computation, each thread $k \in K_1$ generates evaluation key pairs for all successor threads in K_2.

Unlike the abort variant, invalid signed inputs do not cause an immediate abort. Instead, inputs that fail signature verification are locally replaced with agreed-upon default values, ensuring that every thread produces a syntactically valid output. At the end of Π_1, parties in the next layer explicitly perform consistency checks across all threads, discard failing threads, and deterministically select a surviving thread lexicographically, as shown in Algorithm 13 below.

Layer Protocol Π_i for $i \in \{2, \ldots, \ell - 1\}$. For intermediate layers, the GOD variant of Π_i differs from the abort protocol only in how inconsistencies are handled. As in Π_1, each thread $k \in K_i$ produces evaluation key pairs for all successor threads in K_{i+1}, and parties in the next layer perform explicit consistency checks across all incoming threads.

In contrast to the abort variant, failures in signature verification or mismatched common information do not cause an immediate abort. Instead, threads that fail

Figure 13 Layer protocol Π_1

Let $\mathsf{DEHE} = (\mathsf{KeyGen}, \mathsf{Gen}, \mathsf{Enc}, \mathsf{Eval})$ be a two-party DEHE scheme (Definition 1) and $\mathcal{S} = (\mathsf{Gen}, \mathsf{Sign}, \mathsf{Verify})$ be a signature scheme (Definition 14). The protocol Π_1 works as follows:

Computation Layer $\mathcal{L}_1$: for each $k \in K_1$, $b \in \{0,1\}$, $S^1_{b,k}$ does the following:

- it has inputs $\mathsf{vk}, \mathsf{ek}_{b,k}, \mathsf{ct}_\alpha, \mathsf{ct}_\mathsf{k}, \mathsf{ct}_\mathsf{sk}, \mathsf{ct}_{y_0}, \sigma_{y_0}$
- set $\mathsf{ct}_{y_0'} = \{i : \mathsf{ct}_{y_0}[i]\}$ for every $i \in [\ell_1]$, where $\top \leftarrow \mathcal{S}.\mathsf{Verify}(\mathsf{vk}, (0,0,i,\mathsf{ct}_{y_0}[i]), \sigma_{y_0}[i])$
- set $y_0^* = \{i : \mathsf{Default}[i]\}$ for every $i \in [\ell_1]$, where $\bot \leftarrow \mathcal{S}.\mathsf{Verify}(\mathsf{vk}, (0,0,i,\mathsf{ct}_{y_0}[i]), \sigma_{y_0}[i])$ for some agreed upon set of default values $\mathsf{Default}$
- compute $(\{[\mathsf{ek}'_{0,k,k'}]_b, [\sigma^{\mathsf{ek}}_{0,k,k'}]_b, [\mathsf{ek}'_{1,k,k'}]_b, [\sigma^{\mathsf{ek}}_{1,k,k'}]_b\}_{k' \in K_{i+1}}, [y_1]_{b,k}, [\sigma_1]_{b,k}) \leftarrow \mathsf{DEHE.Eval}(b, \mathsf{ek}_{b,k}, (\mathsf{ct}_\alpha, \mathsf{ct}_\mathsf{k}, \mathsf{ct}_\mathsf{sk}, \mathsf{ct}_{y_0'}), P_{1,k}(1,k,y_0^*))$
- broadcast $\mathsf{vk}_k, \mathsf{ct}_{\alpha,k}, \mathsf{ct}_{\mathsf{k},k}, \mathsf{ct}_{\mathsf{sk},k}, [y_1]_{b,k}, [\sigma_1]_{b,k}$
- for each thread $k' \in K_2$, send $[\mathsf{ek}'_{0,k,k'}]_b, [\sigma^{\mathsf{ek}}_{0,k,k'}]_b$ to $S^2_{0,k'}$
- for each thread $k' \in K_2$, send $[\mathsf{ek}'_{1,k,k'}]_b, [\sigma^{\mathsf{ek}}_{1,k,k'}]_b$ to $S^2_{1,k'}$

Computation Layer $\mathcal{L}_2$: For each $k' \in K_2$, $b \in \{0,1\}$, $S^2_{b,k'}$ does the following:

- Receives $\{\mathsf{vk}_k^{b'}, \mathsf{ct}_{\alpha,k}^{b'}, \mathsf{ct}_{\mathsf{k},k}^{b'}, \mathsf{ct}_{\mathsf{sk},k}^{b'}, [\mathsf{ek}_{b,k,k'}]_{b'}, [\sigma^{\mathsf{ek}}_{b,k,k'}]_{b'}, [y_1]_{b',k}, [\sigma_1]_{b',k}\}_{b' \in \{0,1\}, k \in K_1}$ from $\mathcal{L}_1$ and defines $\mathsf{FailThread} = \emptyset$
- **Consistency Checks:** For each $k \in K_1$, do the following:
  - define $\mathsf{FailFlag} = \mathsf{false}$.
  - set $\mathsf{ek}_{b,k,k'} = [\mathsf{ek}_{b,k,k'}]_0 + [\mathsf{ek}_{b,k,k'}]_1$ and $\sigma^{\mathsf{ek}}_{b,k,k'} = [\sigma^{\mathsf{ek}}_{b,k,k'}]_0 + [\sigma^{\mathsf{ek}}_{b,k,k'}]_1$
  - if $\top \not\leftarrow \mathcal{S}.\mathsf{Verify}(\mathsf{vk}, (i,b,k,k',\mathsf{ek}_{b,k,k'}), \sigma^{\mathsf{ek}}_{b,k,k'})$, set $\mathsf{FailFlag} = \mathsf{true}$
  - if $\mathsf{vk}_k^0 \neq \mathsf{vk}_k^1 = \mathsf{vk}_k$, set $\mathsf{FailFlag} = \mathsf{true}$
  - set $y_{1,k} = [y_1]_{0,k} + [y_1]_{1,k}$ and $\sigma_{1,k} = [\sigma_1]_{0,k} + [\sigma_1]_{1,k}$
  - if $\top \not\leftarrow \mathcal{S}.\mathsf{Verify}(\mathsf{vk}, (i,2,0,0,y_{1,k}), \sigma_{1,k})$, set $\mathsf{FailFlag} = \mathsf{true}$
  - if $\mathsf{ct}^0_{\alpha,k} \neq \mathsf{ct}^1_{\alpha,k} = \mathsf{ct}_{\alpha,k}$, set $\mathsf{FailFlag} = \mathsf{true}$
  - if $\mathsf{ct}^0_{\mathsf{k},k} \neq \mathsf{ct}^1_{\mathsf{k},k} = \mathsf{ct}_{\mathsf{k},k}$, set $\mathsf{FailFlag} = \mathsf{true}$
  - if $\mathsf{ct}^0_{\mathsf{sk},k} \neq \mathsf{ct}^1_{\mathsf{sk},k} = \mathsf{ct}_{\mathsf{sk},k}$, set $\mathsf{FailFlag} = \mathsf{true}$
  - if $\mathsf{FailFlag} == \mathsf{true}$, set $\mathsf{FailThread} = \mathsf{FailThread} \cup \{k\}$
- select k^* as the lexicographically smallest element in $K_1 \setminus \mathsf{FailThread}$
- output $(\mathsf{vk}, \mathsf{ct}_\alpha, \mathsf{ct}_\mathsf{k}, \mathsf{ct}_\mathsf{sk}, \mathsf{ek}_{b,k'}, y_1) = (\mathsf{vk}_{k^*}, \mathsf{ct}_{\alpha,k^*}, \mathsf{ct}_{\mathsf{k},k^*}, \mathsf{ct}_{\mathsf{sk},k^*}, \mathsf{ek}_{b,k^*,k'}, y_{1,k^*})$

the consistency checks are discarded, and parties deterministically select a surviving thread lexicographically, as specified in Algorithm 14. This ensures that at least one honest thread continues to propagate a valid authenticated state to the next layer.

Layer Protocol Π_ℓ. The GOD variant of Π_ℓ differs from the abort protocol only in its handling of inconsistencies. Instead of aborting, output parties discard threads that fail verification and deterministically select the lexicographically smallest surviving thread, as specified in Algorithm 15. The output associated with this thread is used.

Layered MPC with Guaranteed Output Delivery. The GOD protocol Π_{GOD} is obtained by replacing the setup functionality and sub-protocols of the abort variant with their GOD counterparts. We also modify the input layer additionally to ensure that we obtain GOD. In contrast to the abort variant, where evaluation keys are additively shared and correctness requires all but one input party to be honest, the GOD protocol relies on majority reconstruction of setup material and evaluation keys. Since the setup functionality distributes secret shares of evaluation keys across the input layer, guaranteed reconstruction requires the input layer itself to contain an honest majority. To relax this requirement, the protocol may introduce an auxiliary dummy committee that participates alongside the input committee and guarantees an honest majority, thereby allowing the original input layer to tolerate all but one corruption as in the abort setting. Moreover, additive sharing in the setup phase is replaced by a robust

Figure 14 Layer protocol Π_i

Let DEHE be a two-party DEHE scheme (Definition 1) and $\mathcal{S} = (\mathsf{Gen}, \mathsf{Sign}, \mathsf{Verify})$ be a signature scheme (Definition 14). The protocol Π_i works as follows:

Computation Layer $\mathcal{L}_i$: for each $k \in K_i$, $b \in \{0,1\}$, $S^i_{b,k}$ does the following:

- it has inputs $\mathsf{vk}, \mathsf{ct}_\alpha, \mathsf{ct}_\mathsf{k}, \mathsf{ct}_\mathsf{sk}, \mathsf{ek}_{b,k}, y_{i-1}$
- compute $(\{[\mathsf{ek}'_{0,k,k'}]_b, [\sigma^{\mathsf{ek}}_{0,k,k'}]_b, [\mathsf{ek}'_{1,k,k'}]_b, [\sigma^{\mathsf{ek}}_{1,k,k'}]_b\}_{k' \in K_{i+1}}, [y_i]_{b,k}, [\sigma_i]_{b,k}) \leftarrow \mathsf{DEHE.Eval}(b, \mathsf{ek}_{b,k}, (\mathsf{ct}_\alpha, \mathsf{ct}_\mathsf{k}, \mathsf{ct}_\mathsf{sk}), P_{i,k}(i, k, y_{i-1}))$
- broadcast $\mathsf{vk}, \mathsf{ct}_\alpha, \mathsf{ct}_\mathsf{k}, \mathsf{ct}_\mathsf{sk}, [y_i]_{b,k}, [\sigma_i]_{b,k}$
- for each thread $k' \in K_{i+1}$, send $[\mathsf{ek}'_{0,k,k'}]_b, [\sigma^{\mathsf{ek}}_{0,k,k'}]_b$ to $S^{i+1}_{0,k'}$
- for each thread $k' \in K_{i+1}$, send $[\mathsf{ek}'_{1,k,k'}]_b, [\sigma^{\mathsf{ek}}_{1,k,k'}]_b$ to $S^{i+1}_{1,k'}$

Computation Layer $\mathcal{L}_{i+1}$: For each $k' \in K_{i+1}$, $b \in \{0,1\}$, $S^{i+1}_{b,k'}$ does the following:

- receives $\{\mathsf{vk}^{b'}_k, \mathsf{ct}^{b'}_{\alpha,k}, \mathsf{ct}^{b'}_{\mathsf{k},k}, \mathsf{ct}^{b'}_{\mathsf{sk},k}, [\mathsf{ek}_{b,k,k'}]_{b'}, [\sigma^{\mathsf{ek}}_{b,k,k'}]_{b'}, [y_i]_{b',k}, [\sigma_i]_{b',k}\}_{b' \in \{0,1\}, k \in K_i}$ from $\mathcal{L}_i$ and defines $\mathsf{FailThread} = \emptyset$
- **Consistency Checks:** for each $k \in K_i$, do the following:
  - define $\mathsf{FailFlag} = \mathsf{false}$
  - set $\mathsf{ek}_{b,k,k'} = [\mathsf{ek}_{b,k,k'}]_0 + [\mathsf{ek}_{b,k,k'}]_1$ and $\sigma^{\mathsf{ek}}_{b,k,k'} = [\sigma^{\mathsf{ek}}_{b,k,k'}]_0 + [\sigma^{\mathsf{ek}}_{b,k,k'}]_1$
  - if $\top \not\leftarrow \mathcal{S}.\mathsf{Verify}(\mathsf{vk}, (i, b, k, k', \mathsf{ek}_{b,k,k'}), \sigma^{\mathsf{ek}}_{b,k,k'})$, set $\mathsf{FailFlag} = \mathsf{true}$
  - if $\mathsf{vk}^0_k \neq \mathsf{vk}^1_k = \mathsf{vk}_k$, set $\mathsf{FailFlag} = \mathsf{true}$
  - set $y_{i,k} = [y_i]_{0,k} + [y_i]_{1,k}$ and $\sigma_{i,k} = [\sigma_i]_{0,k} + [\sigma_i]_{1,k}$
  - if $\top \not\leftarrow \mathcal{S}.\mathsf{Verify}(\mathsf{vk}, (i, 2, 0, 0, y_{i,k}), \sigma_{i,k})$, set $\mathsf{FailFlag} = \mathsf{true}$
  - if $\mathsf{ct}^0_{\alpha,k} \neq \mathsf{ct}^1_{\alpha,k} = \mathsf{ct}_{\alpha,k}$, set $\mathsf{FailFlag} = \mathsf{true}$
  - if $\mathsf{ct}^0_{\mathsf{k},k} \neq \mathsf{ct}^1_{\mathsf{k},k} = \mathsf{ct}_{\mathsf{k},k}$, set $\mathsf{FailFlag} = \mathsf{true}$
  - if $\mathsf{ct}^0_{\mathsf{sk},k} \neq \mathsf{ct}^1_{\mathsf{sk},k} = \mathsf{ct}_{\mathsf{sk},k}$, set $\mathsf{FailFlag} = \mathsf{true}$
  - if $\mathsf{FailFlag} == \mathsf{true}$, set $\mathsf{FailThread} = \mathsf{FailThread} \cup \{k\}$
- select k^* as the lexicographically smallest element in $K_i \setminus \mathsf{FailThread}$
- output $(\mathsf{vk}, \mathsf{ct}_\alpha, \mathsf{ct}_\mathsf{k}, \mathsf{ct}_\mathsf{sk}, \mathsf{ek}_{b,k'}, y_i) = (\mathsf{vk}_{k^*}, \mathsf{ct}_{\alpha,k^*}, \mathsf{ct}_{\mathsf{k},k^*}, \mathsf{ct}_{\mathsf{sk},k^*}, \mathsf{ek}_{b,k^*,k'}, y_{i,k^*})$

Figure 15 Layer protocol Π_ℓ

Let DEHE be a two-party DEHE scheme (Definition 1) and $\mathcal{S} = (\mathsf{Gen}, \mathsf{Sign}, \mathsf{Verify})$ be a signature scheme (Definition 14). The protocol Π_ℓ works as follows:

Computation Layer $\mathcal{L}_\ell$: for each $k \in K_l$, $b \in \{0,1\}$, each $S^\ell_{b,k}$ does the following:

- it has inputs $\mathsf{vk}, \mathsf{ct}_\alpha, \mathsf{ct}_\mathsf{k}, \mathsf{ct}_\mathsf{sk}, \mathsf{ek}_{b,k}, y_{\ell-1}$
- compute $([y_\ell]_{b,k}, [\sigma_\ell]_{b,k}) \leftarrow \mathsf{DEHE.Eval}(b, \mathsf{ek}_b, (\mathsf{ct}_\alpha, \mathsf{ct}_\mathsf{k}, \mathsf{ct}_\mathsf{sk}), P_{\ell,k}(\ell, k, y_{\ell-1}))$
- broadcast $\mathsf{vk}, [y_\ell]_{b,k}, [\sigma_\ell]_{b,k}$

Output Layer $\mathcal{L}_{\ell+1}$: each output party $P^{\ell+1}_i$ does the following:

- receive $\{\mathsf{vk}^b_k, [y_\ell]_{b,k}, [\sigma_\ell]_{b,k}\}_{b \in \{0,1\}}$ from $\mathcal{L}_\ell$.
- define $\mathsf{FailThread} = \emptyset$
- **Consistency Checks:** for each $k \in K_i$, do the following:
  - define $\mathsf{FailFlag} = \mathsf{false}$
  - if $\mathsf{vk}^0_k \neq \mathsf{vk}^1_k = \mathsf{vk}$, set $\mathsf{FailFlag} = \mathsf{true}$
  - set $y_{\ell,k} = [y_\ell]_{0,k} + [y_\ell]_{1,k}$ and $\sigma_{\ell,k} = [\sigma_\ell]_{0,k} + [\sigma_\ell]_{1,k}$
  - if $\top \not\leftarrow \mathcal{S}.\mathsf{Verify}(\mathsf{vk}, (l, 2, 0, 0, y_\ell), \sigma_\ell)$, set $\mathsf{FailFlag} = \mathsf{true}$
  - if $\mathsf{FailFlag} == \mathsf{true}$, set $\mathsf{FailThread} = \mathsf{FailThread} \cup \{k\}$
- for the lexicographically smallest element $k^* \in K_l \setminus \mathsf{FailThread}$ and output y_{ℓ,k^*}

secret sharing scheme with threshold $t < n/2$, ensuring that reconstruction succeeds whenever a strict majority of shares is honest.

Together, these changes ensure that the protocol never aborts due to malformed inputs or inconsistent setup information. Instead, invalid contributions are filtered out by majority checks, and at least one valid execution thread per layer always survives,

guaranteeing correct evaluation and output delivery for any layered circuit in the $\mathcal{F}_{\mathsf{Setup}}^{\mathsf{GOD}}$-hybrid model.

Figure 16 Maliciously Secure Layered MPC satisfying GOD

Let $\mathcal{F}_{\mathsf{Setup}}^{\mathsf{GOD}}$ be the functionality given in Algorithm 12. Let $\Pi_1, \forall i > 1 \in [l-1], \Pi_i$ and Π_ℓ be the functionalities in Algorithms 13 to 15 respectively. Let $\mathcal{S} = (\mathsf{Gen}, \mathsf{Sign}, \mathsf{Verify})$ be a signature scheme (Definition 14). Then the layered protocol Π_{GOD} in the $\mathcal{F}_{\mathsf{Setup}}^{\mathsf{GOD}}$-hybrid operates as follows:

- **Input Layer $\mathcal{L}_0$:** Each party $P_i^0 \in \mathcal{L}_0$ has as input x_i and operates as follows:
  - receive $\{b, \mathsf{ct}_{i,b}, \sigma_{i,b}^{\mathsf{ct}}\}_{b\in\{0,1\}}, \mathsf{vk}, \mathsf{ct}_\alpha, \mathsf{ct}_\mathsf{k}, \mathsf{ct}_\mathsf{sk}, \{\langle \mathsf{ek}_{b,k}^1\rangle_i, \sigma_{b,k,i}^{\mathsf{ek}}\}_{b\in\{0,1\},k\in K_1}$ from $\mathcal{F}_{\mathsf{Setup}}$ and, for $b \in \{0,1\}$, $k \in K_1$ send the tuple $\left(\langle \mathsf{ek}_{b,k}^1\rangle_i, \sigma_{b,k,i}\right)$ to $S_{b,k}^1$
  - broadcast $\mathsf{ct}_{i,x_i}, \sigma_{i,x_i}^{\mathsf{ct}}, \mathsf{vk}, \mathsf{ct}_\alpha, \mathsf{ct}_\mathsf{k}, \mathsf{ct}_\mathsf{sk}$
- **Computation Layer $\mathcal{L}_1$:** For $k \in K_1, b \in \{0,1\}$, $S_{b,k}^1$ does the following:
  - receive $\left(\langle \mathsf{ek}_{b,k}^1\rangle_i, \sigma_{b,k,i}, \mathsf{ct}_{i,x_i}, \sigma_{i,x_i}^{\mathsf{ct}}, \mathsf{vk}^i, \mathsf{ct}_\alpha^i, \mathsf{ct}_\mathsf{k}^i, \mathsf{ct}_\mathsf{sk}^i\right)$ from each $P_i^0 \in \mathcal{L}_0$
  - set $\mathsf{vk} = \mathsf{vk}^*$ such that $|\{ P_i \in \mathcal{L}_0 : \mathsf{vk}^i = \mathsf{vk}^* \}| > \frac{|\mathcal{L}_0|}{2}$.
  - set $\mathsf{ct}_\alpha = \mathsf{ct}_\alpha^*$ such that $|\{ P_i \in \mathcal{L}_0 : \mathsf{ct}_\alpha^i = \mathsf{ct}_\alpha^* \}| > \frac{|\mathcal{L}_0|}{2}$.
  - set $\mathsf{ct}_\mathsf{k} = \mathsf{ct}_\mathsf{k}^*$ such that $|\{ P_i \in \mathcal{L}_0 : \mathsf{ct}_\mathsf{k}^i = \mathsf{ct}_\mathsf{k}^* \}| > \frac{|\mathcal{L}_0|}{2}$.
  - set $\mathsf{ct}_\mathsf{sk} = \mathsf{ct}_\mathsf{sk}^*$ such that $|\{ P_i \in \mathcal{L}_0 : \mathsf{ct}_\mathsf{sk}^i = \mathsf{ct}_\mathsf{sk}^* \}| > \frac{|\mathcal{L}_0|}{2}$.
  - Let $I_{\mathsf{valid}} := \{ i \in [\ell_1] : \top \leftarrow \mathcal{S}.\mathsf{Verify}(\mathsf{vk}, \langle \mathsf{ek}_{b,k}^1\rangle_i, \sigma_{b,k,i}) \}$. Compute $\mathsf{ek}_{b,k} \leftarrow \mathsf{Reconstruct}\left(\{\langle \mathsf{ek}_{b,k}^1\rangle_i\}_{i\in I_{\mathsf{valid}}}\right)$.
  - set $\mathsf{ct}_{y_0} = \{\mathsf{ct}_{i,x_i}\}_{i\in[\ell_1]}$ and $\sigma_{y_0} = \{\sigma_{i,x_i}^{\mathsf{ct}}\}_{i\in[\ell_1]}$
  - invoke protocol Π_1 using $\mathsf{vk}, \{\mathsf{ek}_{b,k}\}_{k\in K_1}, \mathsf{ct}_\alpha, \mathsf{ct}_\mathsf{k}, \mathsf{ct}_\mathsf{sk}, \mathsf{ct}_{y_0}, \sigma_{y_0}$ as inputs.
- **Computation Layer $\mathcal{L}_i$ $(1 \le i \le l)$:** Each S_j^i for $j \in [3]$ does the following:
  - receive $\mathsf{vk}, \mathsf{ct}_\alpha, \mathsf{ct}_\mathsf{k}, \mathsf{ct}_\mathsf{sk}, \{\mathsf{ek}_{b,k}\}_{b\in\{0,1\},k\in K_i}, y_{i-1}$ from the execution of Π_{i-1} and continue by invoking Π_i with these inputs
- **Output Layer $\mathcal{L}_{\ell+1}$:** Each party $P_i^{\ell+1} \in \mathcal{L}_{\ell+1}$ outputs y_ℓ after completing $\Pi_{\ell,k}$

Theorem 6. *Let* $\mathsf{DEHE} = (\mathsf{KeyGen}, \mathsf{Gen}, \mathsf{Enc}, \mathsf{Eval})$ *be a two-party DEHE scheme (Definition 1). Let* F *be a PRF (Definition 12) and* $\mathcal{S} = (\mathsf{Gen}, \mathsf{Sign}, \mathsf{Verify})$ *be a signature scheme (Definition 14). The layered MPC protocol* Π_{GOD} *in Algorithm 16 securely realizes the layered functionality* f *in the* $\mathcal{F}_{\mathsf{Setup}}^{\mathsf{GOD}}$*-hybrid satisfying security with guaranteed output delivery tolerating a malicious adversary that can corrupt less than half the parties in* $\mathcal{L}_0$*, all-but-one parties in* $\mathcal{L}_{\ell+1}$*, and one party in each* $\mathcal{L}_i, i \in [\ell]$*.*

The complete proof for this can be found in Appendix F.

Acknowledgments. Anasuya Acharya and Pierre Meyer were supported by the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme under grant agreement No 101124977 (DECRYPSYS). Divya Ravi was supported by NWO (Dutch Research Council) grant with number VI.Veni.242.295 and by NWO Gravitation Project CiCS (Challenges in Cyber Security). Rahul Satish was supported by the Independent Research Fund Denmark (IRFD) grant number 0165-00079B.

References

ADOS22. D. Abram, I. Damgård, C. Orlandi, and P. Scholl. An algebraic framework for silent preprocessing with trustless setup and active security. In *CRYPTO 2022, Part IV*, *LNCS* 13510, pages 421–452. Springer, Cham, August 2022.

AHKP22. A. Acharya, C. Hazay, V. Kolesnikov, and M. Prabhakaran. SCALES - MPC with small clients and larger ephemeral servers. In *TCC 2022, Part II*, *LNCS* 13748, pages 502–531. Springer, Cham, November 2022.

AHKP24. A. Acharya, C. Hazay, V. Kolesnikov, and M. Prabhakaran. Malicious security for SCALES - outsourced computation with ephemeral servers. In *CRYPTO 2024, Part IX*, *LNCS* 14928, pages 3–38. Springer, Cham, August 2024.

ARS24. D. Abram, L. Roy, and P. Scholl. Succinct homomorphic secret sharing. In *EUROCRYPT 2024, Part VI*, *LNCS* 14656, pages 301–330. Springer, Cham, May 2024.

BCM22. E. Boyle, G. Couteau, and P. Meyer. Sublinear secure computation from new assumptions. In *TCC 2022, Part II*, *LNCS* 13748, pages 121–150. Springer, Cham, November 2022.

BCM23. E. Boyle, G. Couteau, and P. Meyer. Sublinear-communication secure multiparty computation does not require FHE. In *EUROCRYPT 2023, Part II*, *LNCS* 14005, pages 159–189. Springer, Cham, April 2023.

BDO23. L. Braun, I. Damgård, and C. Orlandi. Secure multiparty computation from threshold encryption based on class groups. In *CRYPTO 2023, Part I*, *LNCS* 14081, pages 613–645. Springer, Cham, August 2023.

BEP23. A. Bienstock, D. Escudero, and A. Polychroniadou. On linear communication complexity for (maximally) fluid MPC. In *CRYPTO 2023, Part I*, *LNCS* 14081, pages 263–294. Springer, Cham, August 2023.

BEP24. A. Bienstock, D. Escudero, and A. Polychroniadou. Perfectly secure fluid MPC with abort and linear communication complexity. *CiC*, 1(4):28, 2024.

BGI16. E. Boyle, N. Gilboa, and Y. Ishai. Breaking the circuit size barrier for secure computation under DDH. In *CRYPTO 2016, Part I*, *LNCS* 9814, pages 509–539. Springer, Berlin, Heidelberg, August 2016.

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

BKS19. E. Boyle, L. Kohl, and P. Scholl. Homomorphic secret sharing from lattices without FHE. In *EUROCRYPT 2019, Part II*, *LNCS* 11477, pages 3–33. Springer, Cham, May 2019.

BMO+25. A. R. Behera, P. Meyer, C. Orlandi, L. Roy, and P. Scholl. Privately constrained PRFs from DCR: Puncturing and bounded waring rank. In *TCC 2025, Part I*, *LNCS* 16268, pages 301–332. Springer, Cham, December 2025.

BPR12. A. Banerjee, C. Peikert, and A. Rosen. Pseudorandom functions and lattices. In *EUROCRYPT 2012*, *LNCS* 7237, pages 719–737. Springer, Berlin, Heidelberg, April 2012.

CCD88. D. Chaum, C. Crépeau, and I. Damgård. Multiparty unconditionally secure protocols (extended abstract). In *20th ACM STOC*, pages 11–19. ACM Press, May 1988.

CCMN24. P. Charbit, G. Couteau, P. Meyer, and R. Naserasr. A note on low-communication secure multiparty computation via circuit depth-reduction. In *TCC 2024, Part IV*, *LNCS* 15367, pages 167–199. Springer, Cham, December 2024.

CGG+21. A. R. Choudhuri, A. Goel, M. Green, A. Jain, and G. Kaptchuk. Fluid MPC: Secure multiparty computation with dynamic participants. In *CRYPTO 2021, Part II*, *LNCS* 12826, pages 94–123, Virtual Event, August 2021. Springer, Cham.

CKY25. G. Couteau, N. Kumar, and X. Ye. Multiparty homomorphic secret sharing and more from LPN and MQ. In *TCC 2025, Part II*, *LNCS* 16269, pages 498–530. Springer, Cham, December 2025.

Cle86. R. Cleve. Limits on the security of coin flips when half the processors are faulty (extended abstract). In *18th ACM STOC*, pages 364–369. ACM Press, May 1986.

CM21. G. Couteau and P. Meyer. Breaking the circuit size barrier for secure computation under quasi-polynomial LPN. In *EUROCRYPT 2021, Part II*, *LNCS* 12697, pages 842–870. Springer, Cham, October 2021.

Cou19. G. Couteau. A note on the communication complexity of multiparty computation in the correlated randomness model. In *EUROCRYPT 2019, Part II*, *LNCS* 11477, pages 473–503. Springer, Cham, May 2019.

DDG+23. B. David, G. Deligios, A. Goel, Y. Ishai, A. Konring, E. Kushilevitz, C.-D. Liu-Zhang, and V. Narayanan. Perfect MPC over layered graphs. In *CRYPTO 2023, Part I*, *LNCS* 14081, pages 360–392. Springer, Cham, August 2023.

DGI+19. N. Döttling, S. Garg, Y. Ishai, G. Malavolta, T. Mour, and R. Ostrovsky. Trapdoor hash functions and their applications. In *CRYPTO 2019, Part III*, *LNCS* 11694, pages 3–32. Springer, Cham, August 2019.

DGLZ23. G. Deligios, A. Goel, and C.-D. Liu-Zhang. Maximally-fluid MPC with guaranteed output delivery. Cryptology ePrint Archive, Report 2023/415, 2023.

DHRW16. Y. Dodis, S. Halevi, R. D. Rothblum, and D. Wichs. Spooky encryption and its applications. In *CRYPTO 2016, Part III*, *LNCS* 9816, pages 93–122. Springer, Berlin, Heidelberg, August 2016.

DJK25. Y. Ding, A. Jain, and I. Komargodski. A new approach for LPN-based pseudorandom functions: Low-depth and key-homomorphic. In *57th ACM STOC*, pages 1898–1909. ACM Press, June 2025.

DKLZN24. G. Deligios, A. Konring, C.-D. Liu-Zhang, and V. Narayanan. Statistical layered MPC. In *TCC 2024, Part IV*, *LNCS* 15367, pages 362–394. Springer, Cham, December 2024.

DMS25. B. David, A. Mondal, and R. Satish. Rumors MPC: GOD for dynamic committees, low communication via constant-round chat, 2025.

EMP25. D. Escudero, E. Masserova, and A. Polychroniadou. Towards scalable YOSO MPC via packed secret-sharing. In *ASIACRYPT 2025, Part V*, *LNCS* 16249, pages 68–101. Springer, Singapore, December 2025.

FGJS17. N. Fazio, R. Gennaro, T. Jafarikhah, and W. E. Skeith, III. Homomorphic secret sharing from paillier encryption. In *ProvSec 2017*, *LNCS* 10592, pages 381–399. Springer, Cham, October 2017.

GGM84. O. Goldreich, S. Goldwasser, and S. Micali. On the cryptographic applications of random functions. In *CRYPTO'84*, *LNCS* 196, pages 276–288. Springer, Berlin, Heidelberg, August 1984.

GHK+21. C. Gentry, S. Halevi, H. Krawczyk, B. Magri, J. B. Nielsen, T. Rabin, and S. Yakoubov. YOSO: You only speak once - secure MPC with stateless ephemeral roles. In *CRYPTO 2021, Part II*, *LNCS* 12826, pages 64–93, Virtual Event, August 2021. Springer, Cham.

GJ11. A. Gál and J.-T. Jang. The size and depth of layered boolean circuits. *Information Processing Letters*, 111(5):213–217, 2011.

GMW87. O. Goldreich, S. Micali, and A. Wigderson. How to play any mental game or A completeness theorem for protocols with honest majority. In *19th ACM STOC*, pages 218–229. ACM Press, May 1987.

Gol06. O. Goldreich. *Foundations of Cryptography: Volume 1*. Cambridge University Press, USA, 2006.

IKP+16. Y. Ishai, E. Kushilevitz, M. Prabhakaran, A. Sahai, and C.-H. Yu. Secure protocol transformations. In *CRYPTO 2016, Part II*, *LNCS* 9815, pages 430–458. Springer, Berlin, Heidelberg, August 2016.

ILL25. Y. Ishai, H. Li, and H. Lin. A unified framework for succinct garbling from homomorphic secret sharing. In *CRYPTO 2025, Part IV*, *LNCS* 16003, pages 390–425. Springer, Cham, August 2025.

KL14. J. Katz and Y. Lindell. *Introduction to Modern Cryptography, Second Edition*. Chapman & Hall/CRC, 2nd edition, 2014.

KRY22. S. Kolby, D. Ravi, and S. Yakoubov. Towards efficient YOSO MPC without setup. Cryptology ePrint Archive, Report 2022/187, 2022.

KRY24. S. Kolby, D. Ravi, and S. Yakoubov. Constant-round YOSO MPC without setup. *IACR Commun. Cryptol.*, 1(3):30, 2024.

LW09. A. B. Lewko and B. Waters. Efficient pseudorandom functions from the decisional linear assumption and weaker variants. In *ACM CCS 2009*, pages 112–120. ACM Press, November 2009.

LZMR+24. C.-D. Liu-Zhang, E. Masserova, J. Ribeiro, P. Soni, and S. A. K. Thyagarajan. Improved YOSO randomness generation with worst-case corruptions. In *FC 2024, Part II*, *LNCS* 14745, pages 73–89. Springer, Cham, March 2024.

LZMR+25. C.-D. Liu-Zhang, E. Masserova, J. Ribeiro, P. Soni, and S. A. K. Thyagarajan. Efficient distributed randomness generation from minimal assumptions where PArties speak sequentially once. In *EUROCRYPT 2025, Part V*, *LNCS* 15605, pages 176–206. Springer, Cham, May 2025.

MORS25. P. Meyer, C. Orlandi, L. Roy, and P. Scholl. Silent circuit relinearisation: Sublinear-size (boolean and arithmetic) garbled circuits from DCR. In *CRYPTO 2025, Part IV*, *LNCS* 16003, pages 426–458. Springer, Cham, August 2025.

NR97. M. Naor and O. Reingold. Number-theoretic constructions of efficient pseudo-random functions. In *38th FOCS*, pages 458–467. IEEE Computer Society Press, October 1997.

NR04. M. Naor and O. Reingold. Number-theoretic constructions of efficient pseudo-random functions. In *Proceedings 38th Annual Symposium on Foundations of Computer Science*, pages 458–467, 2004.

NRO22. J. B. Nielsen, J. L. Ribeiro, and M. Obremski. Public randomness extraction with ephemeral roles and worst-case corruptions. In *CRYPTO 2022, Part I*, *LNCS* 13507, pages 127–147. Springer, Cham, August 2022.

NRR00. M. Naor, O. Reingold, and A. Rosen. Pseudo-random functions and factoring (extended abstract). In *32nd ACM STOC*, pages 11–20. ACM Press, May 2000.

OSY21. C. Orlandi, P. Scholl, and S. Yakoubov. The rise of paillier: Homomorphic secret sharing and public-key silent OT. In *EUROCRYPT 2021, Part I*, *LNCS* 12696, pages 678–708. Springer, Cham, October 2021.

OY91. R. Ostrovsky and M. Yung. How to withstand mobile virus attacks (extended abstract). In *10th ACM PODC*, pages 51–59. ACM, August 1991.

Pai99. P. Paillier. Public-key cryptosystems based on composite degree residuosity classes. In *EUROCRYPT'99*, *LNCS* 1592, pages 223–238. Springer, Berlin, Heidelberg, May 1999.

RS21. L. Roy and J. Singh. Large message homomorphic secret sharing from DCR and applications. In *CRYPTO 2021, Part III*, *LNCS* 12827, pages 687–717, Virtual Event, August 2021. Springer, Cham.

RS22. R. Rachuri and P. Scholl. Le mans: Dynamic and fluid MPC for dishonest majority. In *CRYPTO 2022, Part I*, *LNCS* 13507, pages 719–749. Springer, Cham, August 2022.

Sha79. A. Shamir. How to share a secret. *Commun. ACM*, 22(11):612–613, November 1979.

Vio13. E. Viola. The communication complexity of addition. In *24th SODA*, pages 632–651. ACM-SIAM, January 2013.

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

ZYFZ25. Y. Zeng, K. Yang, D. Feng, and M. Zhang. Ion: Concretely efficient submaximal-fluid MPC with linear communication. Cryptology ePrint Archive, Paper 2025/1508, 2025.

A Additional Preliminaries

Definition 7. *Two ensembles of probability distributions* $X = \{X_\kappa\}_{\kappa\in\mathbb{N}}$ *and* $Y = \{Y_\kappa\}_{\kappa\in\mathbb{N}}$ *are* computationally indistinguishable*, denoted* $X \overset{\text{c}}{\approx} Y$*, if for every PPT distinguisher* D*, there exists a negligible function* $\epsilon(\cdot)$ *such that for every* $\kappa \in \mathbb{N}$*,*

$$\left| \Pr_{x\leftarrow X_\kappa}[D(1^\kappa, x) = 1] - \Pr_{y\leftarrow Y_\kappa}[D(1^\kappa, y) = 1] \right| < \epsilon(\kappa).$$

A.1 Secure Multiparty Computation

Let $\mathcal{P} = \{P_i\}_{i\in[n]}$ be a set of n parties. MPC involves the computation of a random process that maps a tuple of inputs to a tuple of outputs (one for each party). This random process is referred to as a functionality $f : \{0,1\}^* \times \cdots \times \{0,1\}^* \to \{0,1\}^* \times \cdots \times \{0,1\}^*$ where $f = (f_1, \ldots, f_n)$. Each party P_i has input x_i. For simplicity and without loss of generality, we work over functionalities for which $x_i \in \{0,1\}$. For every tuple of inputs $\vec{x} = (x_i, \ldots, x_n)$, the vector output by f is a random variable $(f_1(\vec{x}), \ldots, f_n(\vec{x}))$ ranging over tuples of strings where P_i receives $f_i(\vec{x})$. The following notation describes a functionality:

$$(x_1, \ldots, x_n) \mapsto (f_1(\vec{x}), \ldots, f_n(\vec{x}))$$

We prove the security of our MPC protocols in the presence of malicious adversaries in the dishonest majority setting.

In the semi-honest setting, an adversary can control a subset of the participating parties and always follows the protocol specification but may try to learn additional information from the transcript of messages received and its internal state. Let $f = (f_1, \ldots, f_n)$ be a multiparty functionality and Π be an n-party protocol computing f in t rounds. The view of party P_i in an execution of Π on inputs $\vec{x} = (x_1, \ldots, x_n)$ is,

$$\mathbf{View}_{\Pi,i}(\kappa, \vec{x}) = (\kappa, x_i, r_i, \{m^j_{k\to i}\}_{j\in[t],k\neq i\in[n]})$$

where κ is the security parameter, r_i is the content of P_i's internal random tape, and each $m^j_{k\to i}$ represents the message received from party P_k in round j. The output of P_i in an execution of Π on $\vec{x}$ is denoted $\mathsf{output}_{\Pi,i}(\vec{x})$ and can be computed from $\mathbf{View}_{\Pi,i}(\kappa, \vec{x})$. The set of corrupted parties is denoted by $Z \subset \mathcal{P}$ and the set of honest parties by $\overline{Z}$. We extend the view notation to capture any subset of parties, denoting by $\mathbf{View}_{\Pi,T}(\kappa, \vec{x})$ the joint view of all parties in $T \subseteq \mathcal{P}$ on $(\kappa, \vec{x})$.

Definition 8. *Let* f *and* Π *be as above, and let* $\mathcal{Z} \subseteq 2^\mathcal{P}$ *be an adversary structure. The protocol* Π *is said to securely compute* f *in the presence of a* ***PPT semi-honest adversary*** *tolerating adversary structure* $\mathcal{Z}$ *if there exists a PPT simulator* Sim *such that for every* $Z \in \mathcal{Z}$*,*

$$\left\{ \mathsf{Sim}(1^\kappa, Z, \{x_i, f_i(\vec{x})\}_{i\in Z}), \{f_i(\vec{x})\}_{i\notin Z} \right\}_{\kappa\in\mathbb{N},\vec{x}\in\{0,1\}^*}$$
$$\overset{\text{c}}{\approx} \left\{ \mathbf{View}_{\Pi,Z}(\kappa, \vec{x}), \mathsf{output}_{\Pi,\overline{Z}}(\kappa, \vec{x}) \right\}_{\kappa\in\mathbb{N},\vec{x}\in\{0,1\}^*}$$

where κ *is the computational security parameter. The distribution is considered over the randomness of the simulator and random tapes of parties in the protocol.*

In the malicious setting, an adversary controlling a subset of the participating parties can have them deviate arbitrarily from the protocol specification. Security in this setting is defined by comparing an execution of the functionality in an ideal model and an execution of the protocol in the real model. Depending on the security guarantee required, the ideal model execution is defined differently for the case of security with abort and fairness, and guaranteed output delivery (GOD).

Execution in the ideal model (Security with Abort). An ideal execution is an interaction between all the parties in $\mathcal{P}$ and a trusted party $\mathcal{F}$, wherein the parties submit their input to the trusted party that computes the output and returns it. An honest party receives input for the computation and forwards it to the trusted party. In contrast, a corrupt party can replace its input with an arbitrary value of the same length. On receiving these, $\mathcal{F}$ computes the output and first sends the outputs of the corrupt parties to the adversary. The adversary then decides whether the honest parties will receive their outputs from $\mathcal{F}$ or an abort symbol $\perp$. This notion is called *unanimous abort*, since all honest parties agree on whether an abort occurred. A weaker abort notion, namely *selective abort* that does not have agreement, allows the adversary to additionally specify to $\mathcal{F}$ which honest parties are selected to receive the output. For a multiparty functionality $f = (f_1, \ldots, f_n)$, let $\mathcal{A}$ be a non-uniform probabilistic machine and $Z \subset \mathcal{P}$ be the set of corrupted parties. Then the ideal execution of f on inputs $(\kappa, \vec{x})$, auxiliary input z to $\mathcal{A}$ and security parameter κ is defined as the output tuple of the honest parties and the adversary from this execution and is denoted as $\mathbf{IDEAL}^{\perp}_{f,\mathcal{A}(z),Z}(\kappa, \vec{x})$.

Execution in the ideal model (Fairness). This is a similar ideal execution as described above. Here again, honest parties receive inputs for the computation and forward it to the trusted party. A corrupt party can replace its input with an arbitrary value of the same length. If these inputs contain an abort symbol $\perp$, $\mathcal{F}$ does not compute the output, and none of the parties receive it. Instead, $\mathcal{F}$ sends $\perp$ to the honest parties. Otherwise, $\mathcal{F}$ computes the output and simultaneously sends it to *both the adversary and the honest parties*. For a multiparty functionality $f = (f_1, \ldots, f_n)$, let $\mathcal{A}$ be a non-uniform probabilistic machine and $Z \subset \mathcal{P}$ be the set of corrupted parties. Then the ideal execution of f on inputs $(\kappa, \vec{x})$, auxiliary input z to $\mathcal{A}$ and security parameter κ is defined as the output tuple of the honest parties and the adversary from this execution and is denoted as $\mathbf{IDEAL}^{\mathsf{fair}}_{f,\mathcal{A}(z),Z}(\kappa, \vec{x})$.

Execution in the ideal model (GOD). In this ideal execution, honest parties receive inputs for the computation and forward it to the trusted party. A corrupt party can replace its input with an arbitrary value of the same length. On receiving all inputs, $\mathcal{F}$ *always* computes the output and simultaneously sends it to both the adversary and the honest parties. For a multiparty functionality $f = (f_1, \ldots, f_n)$, let $\mathcal{A}$ be a non-uniform probabilistic machine and $Z \subset \mathcal{P}$ be the set of corrupted parties. Then the ideal execution of f on inputs $(\kappa, \vec{x})$, auxiliary input z to $\mathcal{A}$ and security parameter κ is defined as the output tuple of the honest parties and the adversary from this execution and is denoted as $\mathbf{IDEAL}^{\mathsf{GOD}}_{f,\mathcal{A}(z),Z}(\kappa, \vec{x})$.

Execution in the real model. This is an interaction directly among the parties in $\mathcal{P}$ wherein the honest parties follow the instructions in the protocol Π. The adversary $\mathcal{A}$ sends all messages in place of the corrupted parties and may follow any arbitrary strategy. Letting Π be a protocol that computes f as above, $\mathcal{A}$ be a non-uniform probabilistic machine, and let $Z \subset \mathcal{P}$ be the set of corrupted parties, the real execution

of Π on inputs $(\kappa, \vec{x})$, auxiliary input z to $\mathcal{A}$ and security parameter κ is defined as the output tuple of the honest parties and the adversary from this execution and is denoted as $\mathbf{REAL}_{\Pi,\mathcal{A}(z),Z}(\kappa, \vec{x})$.

Definition 9. *Let f and Π be as above, and let $\mathcal{Z} \subseteq 2^{\mathcal{P}}$ be an adversary structure. The protocol Π is said to securely compute f with unanimous abort (resp. fairness and GOD) in the presence of **PPT malicious adversaries** tolerating adversary structure $\mathcal{Z}$ if there exists a PPT adversary* Sim *in the ideal model such that for every PPT $\mathcal{A}$ in the real model and $Z \in \mathcal{Z}$,*

$$\left\{\mathbf{IDEAL}^{\perp}_{f,\mathsf{Sim}(z),Z}(\kappa, \vec{x})\right\}_{\kappa\in\mathbb{N},\vec{x},z\in\{0,1\}^*} \overset{c}{\approx} \left\{\mathbf{REAL}_{\Pi,\mathcal{A}(z),Z}(\kappa, \vec{x})\right\}_{\kappa\in\mathbb{N},\vec{x},z\in\{0,1\}^*}$$

resp.,

$$\left\{\mathbf{IDEAL}^{\mathsf{fair}}_{f,\mathsf{Sim}(z),Z}(s, \vec{x})\right\}_{s\in\mathbb{N},\vec{x},z\in\{0,1\}^*} \overset{c}{\approx} \left\{\mathbf{REAL}_{\Pi,\mathcal{A}(z),Z}(s, \vec{x})\right\}_{s\in\mathbb{N},\vec{x},z\in\{0,1\}^*}$$

$$\left\{\mathbf{IDEAL}^{\mathsf{GOD}}_{f,\mathsf{Sim}(z),Z}(s, \vec{x})\right\}_{s\in\mathbb{N},\vec{x},z\in\{0,1\}^*} \overset{c}{\approx} \left\{\mathbf{REAL}_{\Pi,\mathcal{A}(z),Z}(s, \vec{x})\right\}_{s\in\mathbb{N},\vec{x},z\in\{0,1\}^*}$$

where κ is the computational security parameter. The distribution is considered over the adversaries' randomness and all the parties' random tapes.

A.2 Robust Secret-Sharing

We re-state the definition of a threshold secret-sharing scheme with robust reconstruction (RSS) below.

Definition 10 (Robust Secret-Sharing). *For parameters $n, t \in \mathbb{N}$, where $t \leq n$, and message space $\mathcal{M}$, a tuple of PPT algorithms* $\mathsf{RSS} = (\mathsf{Share}, \mathsf{Recon})$ *is a **robust secret-sharing scheme** if the following holds:*

- *for any $m \in \mathcal{M}$, $[m] = \{[m]_i\}_{i\in[n]} \leftarrow \mathsf{Share}(m)$ denotes a t-out-of-n secret-sharing of some message m and for any $\mathcal{H} \subseteq [n]$ we use $[m]_{\mathcal{H}} = \{[m]_i\}_{i\in\mathcal{H}}$.*
- *t-**reconstruction:** for all $m \in \mathcal{M}$ and all $R \subseteq [n]$ with $|R| \geq t$ it always holds that,*

$$\mathsf{Recon}(\mathsf{Share}(m)_R) = m$$

- *t-**robustness:** for any $m \in \mathcal{M}$, $[m] \leftarrow \mathsf{Share}(m)$, $\mathcal{H} \subseteq [n]$ s.t. $|\mathcal{H}| \geq t$, for any (possibly maliciously generated) $[m]' \supset [m]_{\mathcal{H}}$ it should hold that,*

$$\mathsf{Recon}([m]') = m$$

- *t-**privacy:** for all $m^0, m^1 \in \mathcal{M}$ and all $\mathcal{C} \subseteq [n]$ with $|\mathcal{C}| < t$ it holds that,*

$$\mathsf{Share}(m^0)_{\mathcal{C}} \equiv \mathsf{Share}(m^1)_{\mathcal{C}}$$

- *t-**patching:** let S be the domain of the shares $[m]_i$. Let $T \subseteq [n]$ with $|T| < t$. Let $s_T = \{[s]_i\}_{i\in T}$ be s.t. $[s]_i \in S$. Let $\{[m]_i\}_{i\in[n]} \leftarrow \mathsf{Share}(m)_{|s_T}$ denote the distribution given by a random sample $\{[m]_i\}_{i\in[n]} \leftarrow \mathsf{Share}(m)$ under the restriction that $[m]_i = [s]_i$ for all $i \in T$. Let $\mathsf{Share}(m)_{|s_T} = \perp$ if no secret-sharing is consistent with s_T.*
  We say that RSS *has t-patching if for all messages $m \in \mathcal{M}$ and all s_T it holds that $\mathsf{Share}(m)_{|s_T} \neq \perp$ and there exists a PPT algorithm* Patch *such that,*

$$\mathsf{Patch}(m, s_T) \equiv \mathsf{Share}(m)_{|s_T}$$

Looking ahead, our construction of a maliciously secure layered protocol satisfying fairness uses a 2-out-of-3 RSS scheme as a building-block. This can be instantiated using the Error-Correcting-Secret Sharing (ECSS) scheme of [IKP+16] – this scheme augments Shamir's threshold secret sharing [Sha79] with message authentication codes (which enables t-robustness as it allows detection of malicious shares). This supports the t-patching property because, in Shamir secret sharing, the underlying $(t-1)$ degree polynomial allows fewer than t shares to be fixed, and the remaining shares can be efficiently filled in to form a valid sharing consistent with a given secret.

A.3 Commitment Schemes

A commitment scheme Com is a 2-party protocol between a committer C with an input x and a receiver R in which C samples randomness r and creates a 'commitment' to the value x. This commitment should be hidden: no information about x should be revealed. At a later point in time, C can send a 'decommit' message to R showing that the value committed to was indeed x. This property by which C cannot deceive R into believing that the value committed to was something different $x' \neq x$ is called the binding property of the commitment. A *deterministic* commitment scheme is one where the algorithm Com does not accept randomness as input. It is computationally hiding, statistically binding, and is additively homomorphic.

Definition 11 (Deterministic Commitment). *A PPT algorithm* Com *is a* ***deterministic commitment scheme*** *for* $m \in \mathcal{M}$ *if it satisfies:*

- ***Statistical Binding:*** $\forall$ *adversary* $\mathcal{A}$ *there exists a negligible function* μ *s.t.,*

$$\Pr[(c, m_0, m_1) \leftarrow \mathcal{A}(1^s) | c = \mathsf{Com}(m_0);\ c = \mathsf{Com}(m_1);\ m_0 \neq m_1 \in \mathcal{M}] < \mu(s)$$

- ***Computationally Hiding:*** *For random choices of messages it holds that,*

$$\{\mathsf{Com}(m_0)\}_{m_0 \leftarrow \mathcal{M}} \overset{c}{\approx} \{\mathsf{Com}(m_1)\}_{m_1 \leftarrow \mathcal{M}}$$

- ***Decommitment*** *to a commitment* τ *is* m *such that* $\tau = \mathsf{Com}(m)$
- *The commitment is* ***additively homomorphic*** *if there exists operators* $\circ$ *and* $*$ *such that,* $\forall m_0, m_1, m_2 \in \mathcal{M}$*,*

$$\mathsf{Com}(m_0) * \mathsf{Com}(m_1) \circ m_2 = \mathsf{Com}(m_0 + m_1 + m_2)$$

A.4 Pseudorandom Functions (PRF)

We restate the definition of PRFs from [Gol06, GGM84].

Definition 12 (Pseudorandom Functions). *An* l*-bit function ensemble* $\mathcal{F} = \{F_\kappa : \{0,1\}^\kappa \times \{0,1\}^{n(\kappa)} \to \{0,1\}^{m(\kappa)}\}_{\kappa \in \mathbb{N}}$ *is called* ***pseudorandom*** *if for all probabilistic poly-time adversary* $\mathcal{A}$*, and all sufficiently large* κ*, we have:*

$$\left| \Pr[\mathcal{A}^{F_\kappa(k;\cdot)}(1^\kappa) = 1] - \Pr[\mathcal{A}^{R_\kappa(\cdot)}(1^\kappa) = 1] \right| \leq \mu(\kappa)$$

where $k \leftarrow \{0,1\}^\kappa$ *and* $R = \{R_\kappa\}_{\kappa \in \mathbb{N}}$ *is the uniform* l*-bit function ensemble.*

In our protocols, we use PRFs in $\mathbf{NC}^1$ (with respect to the key, on hard-coded inputs) [NR97, BPR12] to mask intermediate values in the circuit that our protocol needs to compute.

A.5 Message Authentication Codes (MACs)

We restate the definition of message authentication codes (MACs) from [Gol06] below. Looking ahead, in our work we use MACs (where the MAC tag size is linear in the size of the message being authenticated) in $\mathbf{NC}^1$ to prove that the computation was done correctly. Prior work in [KL14] present a construction of such a MAC from PRFs.

Definition 13 (Message Authentication Codes). *A tuple of PPT algorithms* $\Sigma = (\mathsf{Gen}, \mathsf{MAC}, \mathsf{Verify})$ *is a MAC scheme for message space* $\mathcal{M}$ *if,*

- ***Correctness:*** *for all* $m \in \mathcal{M}$,

$$\Pr[\top = \mathsf{Verify}(\mathsf{sk}, \sigma, m) \mid \mathsf{sk} \leftarrow \mathsf{Gen}(1^\kappa) \; ; \; \sigma \leftarrow \mathsf{MAC}(\mathsf{sk}, m)] = 1$$

- ***Succinctness:*** $\forall m \in \mathcal{M}, \mathsf{sk} \leftarrow \mathsf{Gen}(1^\kappa)$, *it holds that,*

$$|\sigma \leftarrow \mathsf{MAC}(\mathsf{sk}, m)| = O(|m|)$$

- ***Unforgeability:*** $\forall$ *PPT adversaries* $\mathcal{A}$, $\exists$ *a negligible function* μ *s.t.* $\mathcal{A}$ *has advantage* $< \mu(\kappa)$ *in the following security game,*
  - *challenger* $\mathcal{C}_\Sigma$ *samples* $\mathsf{sk} \leftarrow \mathsf{Gen}(1^\kappa)$
  - *for as many times as* $\mathcal{A}$ *wants, it can query* $\mathsf{MAC}(\mathsf{sk}, \cdot)$ *with messages* m_i *and receive* σ_i. *Let* $\mathcal{Q}$ *be the set of all queries* $\mathcal{A}$ *makes*
  - *finally* $\mathcal{A}$ *needs to output* (m', σ')
  - *the output of this game is* $\top$ *if* $m' \notin \mathcal{Q}$ *and* $\top \leftarrow \mathsf{Verify}(\mathsf{sk}, \sigma', m')$

Signature Schemes. We restate the definition of signature schemes from [Gol06] below. Looking ahead, in our work we use signatures in $\mathbf{NC}^1$ to prove that the parties of the last computation layer performed their tasks correctly. Note that most prior signatures scheme based on the DDH assumption have this property.

Definition 14 (Signature Schemes). *A tuple of PPT algorithms* $\mathcal{S} = (\mathsf{Gen}, \mathsf{Sign}, \mathsf{Verify})$ *is a signature scheme for message space* $\mathcal{M}$ *if,*

- ***Correctness:*** *for all* $m \in \mathcal{M}$,

$$\Pr[\top = \mathsf{Verify}(\mathsf{vk}, \sigma, m) \mid (\mathsf{sk}, \mathsf{vk}) \leftarrow \mathsf{Gen}(1^\kappa) \; ; \; \sigma \leftarrow \mathsf{Sign}(\mathsf{sk}, m)] = 1$$

- ***Unforgeability:*** $\forall$ *PPT adversaries* $\mathcal{A}$, $\exists$ *a negligible function* μ *s.t.* $\mathcal{A}$ *has advantage* $< \mu(\kappa)$ *in the following security game,*
  - *challenger* $\mathcal{C}_\mathcal{S}$ *samples* $(\mathsf{sk}, \mathsf{vk}) \leftarrow \mathsf{Gen}(1^\kappa)$ *and gives* vk *to* $\mathcal{A}$
  - *for as many times as* $\mathcal{A}$ *wants, it can query* $\mathsf{Sign}(\mathsf{sk}, \cdot)$ *with messages* m_i *and receive* σ_i. *Let* $\mathcal{Q}$ *be the set of all queries* $\mathcal{A}$ *makes*
  - *finally* $\mathcal{A}$ *needs to output* (m', σ')
  - *the output of this game is* $\top$ *if* $m' \notin \mathcal{Q}$ *and* $\top \leftarrow \mathsf{Verify}(\mathsf{vk}, \sigma', m')$

B Layered MPC satisfying Security with Abort

In this section we present a layered MPC protocol that is a modification of the protocol in Section 4. This protocol is specific to 2-party DEHE schemes (Definition 1) and uses PRF-based message authentication codes (MACs – Definition 13) to perform consistency checks between consecutive layers of computation. As a result, within the RMS program of each computation layer $\mathcal{L}_i$ (except that of the last layer $\mathcal{L}_\ell$), all

the computation beyond $\mathbf{C}_i$ is composed of PRF calls only. This makes the resulting protocol more concretely efficient, but at the cost that any malicious behaviour of parities in level $\mathcal{L}_i$ will cause the protocol to abort only at level $\mathcal{L}_{i+2}$ (In contrast, in Section 4, malicious behaviour in $\mathcal{L}_i$ causes the protocol to abort already at level $\mathcal{L}_{i+1}$). In more detail, this protocol makes use of the following building-blocks:

- A PRF (Definition 12) F, whose description is within $\mathbf{NC}^1$ (with respect to the key, on hard-coded inputs).
- A two-party Distributed Evaluation Homomorphic Encryption scheme (Definition 1) $\mathsf{DEHE} = (\mathsf{KeyGen}, \mathsf{Gen}, \mathsf{Enc}, \mathsf{Eval})$ for RMS programs. We observe that in such schemes, the algorithm $(\mathsf{ek}_0, \mathsf{ek}_1) \leftarrow \mathsf{Gen}(\mathsf{k})$ works by simply additively sharing the secret key k. Therefore, it can be re-written in a non-black-box way as:

$$\mathsf{ek}_0 = F_\alpha(i, 0) \qquad \mathsf{ek}_1 = \mathsf{k} - \mathsf{ek}_0$$

  where $F_\alpha(i, 0)$ is the PRF-generated randomness for this algorithm for the ith computation layer, using secret PRF key α.
- A MAC scheme $\Sigma = (\mathsf{Gen}, \mathsf{MAC}, \mathsf{Verify})$ (Definition 13) where the algorithms MAC and Verify are based only on PRF calls.
- A deterministic commitment scheme Com (Definition 11) that is additively homomorphic.
- A Signature Scheme (Definition 14) $\mathcal{S} = (\mathsf{Gen}, \mathsf{Sign}, \mathsf{Verify})$.

The Setup Functionality. The setup functionality for this protocol is similar to Algorithm 6, with the difference that it additionally generates and broadcasts a ciphertext encrypting the MAC key. Details can be found in Algorithm 17.

Figure 17 Setup Functionality $\mathcal{F}_{\mathsf{Setup}}$

Let $\mathsf{DEHE} = (\mathsf{KeyGen}, \mathsf{Gen}, \mathsf{Enc}, \mathsf{Eval})$ be a two-party DEHE scheme (Definition 1). Let F be a PRF (Definition 12), $\Sigma = (\mathsf{Gen}, \mathsf{MAC}, \mathsf{Verify})$ be a MAC scheme (Definition 13), $\mathcal{S} = (\mathsf{Gen}, \mathsf{Sign}, \mathsf{Verify})$ be a signature scheme (Definition 14) and Com be an additively homomorphic deterministic commitment scheme (Definition 11). $\mathcal{F}_{\mathsf{Setup}}$ works as follows:

Computation:

- generate $\mathsf{k} \leftarrow \mathsf{DEHE.KeyGen}(1^\kappa)$ and $(\mathsf{ek}_0^1, \mathsf{ek}_1^1) \leftarrow \mathsf{DEHE.Gen}(\mathsf{k})$
- generate $(\mathsf{sk}, \mathsf{vk}) \leftarrow \mathcal{S}.\mathsf{Gen}(1^\kappa)$ and $\mathsf{sk}_\Sigma \leftarrow \Sigma.\mathsf{Gen}(1^\kappa)$
- compute $\mathsf{pk} = \mathsf{Com}(\mathsf{k})$
- sample a PRF key $\alpha \leftarrow \{0,1\}^\kappa$
- compute $(\mathsf{ct}_\alpha, \mathsf{ct}_\mathsf{k}, \mathsf{ct}_\mathsf{sk}, \mathsf{ct}_\Sigma) \leftarrow \mathsf{DEHE.Enc}(\mathsf{k}, (\alpha, \mathsf{k}, \mathsf{sk}, \mathsf{sk}_\Sigma))$
- $\forall j \in [\ell_1]$, $b \in \{0,1\}$, compute $\mathsf{ct}_{j,b} \leftarrow \mathsf{DEHE.Enc}(\mathsf{k}, b)$ and $\sigma^{\mathsf{ct}}_{b,j} \leftarrow \mathcal{S}.\mathsf{Sign}(\mathsf{sk}, (o, b, j, \mathsf{ct}_{j,b}))$
- for $b \in \{0,1\}$, sample $\{\langle \mathsf{ek}_b^1 \rangle_j\}_{j \in [\ell_1]}$ such that $\mathsf{ek}_b^1 = \sum_{j \in [\ell_1]} \langle \mathsf{ek}_b^1 \rangle_j$
- for $b \in \{0,1\}, j \in [\ell_1]$, compute $\sigma_{b,j} = \mathcal{S}.\mathsf{Sign}(\mathsf{sk}, (0, b, j, \langle \mathsf{ek}_b^1 \rangle_j))$

Outputs:

- $\forall j \in [\ell_1]$, C_j receives

$$\{b : \mathsf{ct}_{j,b}, \sigma^{\mathsf{ct}}_{j,b}\}_{b \in \{0,1\}}, \mathsf{vk}, \mathsf{pk}, \mathsf{ct}_\alpha, \mathsf{ct}_\mathsf{k}, \mathsf{ct}_\mathsf{sk}, \mathsf{ct}_\Sigma, \{\langle \mathsf{ek}_b^1 \rangle_j, \sigma_{b,j}\}_{b \in \{0,1\}}$$

DEHE Computation. We now describe the computation performed using DEHE. Let $\mathbf{C} = \mathbf{C}_\ell \circ \cdots \circ \mathbf{C}_1$ be a layered circuit where for each $i \in [\ell]$, $\mathbf{C}_i : \{0,1\}^{\ell_i} \to \{0,1\}^{\ell_{i+1}} \in \mathbf{NC}^1$. We define one RMS program with respect to each such $i \in [\ell]$. We define the computation circuit P_1 as follows:

$$
\begin{aligned}
&\text{Parameters:} && 1 \\
&\text{Input:} && \alpha, \mathsf{k}, \mathsf{sk}_\Sigma \in \{0,1\}^\kappa, y_0 \in \{0,1\}^{\ell_1} \\
&\text{Compute:} && \mathsf{ek}'_0 = F_\alpha(i,0) \qquad \mathsf{ek}'_1 = \mathsf{k} - \mathsf{ek}'_0 \\
& && y_1 = \mathbf{C}_1(y_0) + F_\alpha(1,2) \qquad \sigma_1 = \Sigma.\mathsf{MAC}(\mathsf{sk}_\Sigma, (1, y_1)) \\
&\text{Output:} && \mathsf{ek}'_0, \mathsf{ek}'_1, y_1, \sigma_1
\end{aligned}
$$

Corresponding to each $i > 1 \in [\ell - 1]$, we define a computation circuit P_i as follows:

$$
\begin{aligned}
&\text{Parameters:} && i, y_{i-1}, \sigma_{i-1} \in \{0,1\}^{\ell_i} \\
&\text{Input:} && \alpha, \mathsf{k}, \mathsf{sk}_\Sigma \in \{0,1\}^\kappa \\
&\text{Compute:} && \mathsf{ek}'_0 = F_\alpha(i,0) \qquad \mathsf{ek}'_1 = \mathsf{k} - \mathsf{ek}'_0 \\
& && r_i = F_\alpha(i,2) \qquad r_{i-1} = F_\alpha(i-1,2) \\
& && y_i = \mathbf{C}_i(y_{i-1} - r_{i-1}) + r_i \qquad \sigma_i = \Sigma.\mathsf{MAC}(\mathsf{sk}_\Sigma, (i, y_i)) \\
&\text{Output:} && \begin{cases} \mathsf{ek}'_0, \mathsf{ek}'_1, y_i, \sigma_i & \text{if } \top = \Sigma.\mathsf{Verify}(\mathsf{sk}_\Sigma, y_{i-1}, \sigma_{i-1}) \\ \bot & \text{otherwise} \end{cases}
\end{aligned}
$$

Similarly, we define P_ℓ as follows:

$$
\begin{aligned}
&\text{Parameters:} && \ell, y_{\ell-1}, \sigma_{\ell-1} \in \{0,1\}^{\ell_\ell} \\
&\text{Input:} && \alpha, \mathsf{sk}, \mathsf{sk}_\Sigma \in \{0,1\}^\kappa \\
&\text{Compute:} && y_\ell = \mathbf{C}_\ell(y_{\ell-1} - F_\alpha(\ell-1,2)) \\
& && \sigma_\ell = \mathcal{S}.\mathsf{Sign}(\mathsf{sk}, (\ell, y_\ell); F_\alpha(\ell,2)) \\
&\text{Output:} && \begin{cases} y_\ell, \sigma_\ell & \text{if } \top = \Sigma.\mathsf{Verify}(\mathsf{sk}_\Sigma, (\ell-1, y_{\ell-1}), \sigma_{\ell-1}) \\ \bot & \text{otherwise} \end{cases}
\end{aligned}
$$

The layer computation sub-protocols. We describe sub-protocols Π_i for each computation layer $i \in [\ell]$ in the layered MPC protocol in Algorithm 18 for $i = 1$, in Algorithm 19 for $1 < i < \ell$, and in Algorithm 20 for $i = \ell$.

Layered MPC satisfying Security with Unanimous Abort. We now present a layered MPC protocol that can compute any functionality securely with unanimous abort, where the functionality provides the same outputs to each party in the output layer. This complete layered MPC protocol is detailed Algorithm 21 below.

Theorem 15. *Let* $\mathsf{DEHE} = (\mathsf{KeyGen}, \mathsf{Gen}, \mathsf{Enc}, \mathsf{Eval})$ *be a two-party DEHE scheme (Definition 1). Let* F *be a PRF (Definition 12),* $\Sigma = (\mathsf{Gen}, \mathsf{MAC}, \mathsf{Verify})$ *be a MAC scheme (Definition 13),* $\mathcal{S} = (\mathsf{Gen}, \mathsf{Sign}, \mathsf{Verify})$ *be a signature scheme (Definition 14) and* Com *be a deterministic commitment scheme (Definition 11). The layered MPC protocol* $\Pi_\perp$ *in Algorithm 21 securely realizes the layered functionality* f *in the* $\mathcal{F}_{\mathsf{Setup}}$*-hybrid (Algorithm 17) satisfying security with unanimous abort tolerating a malicious adversary that can corrupt all-but-one party in* $\mathcal{L}_0$ *and* $\mathcal{L}_{\ell+1}$*, and one party in each* $\mathcal{L}_i, i \in [\ell]$.

Figure 18 Layer protocol Π_1

Let $\mathsf{DEHE} = (\mathsf{KeyGen}, \mathsf{Gen}, \mathsf{Enc}, \mathsf{Eval})$ be a two-party DEHE scheme (Definition 1), $\mathcal{S} = (\mathsf{Gen}, \mathsf{Sign}, \mathsf{Verify})$ be a signature scheme (Definition 14) and Com be an additively homomorphic deterministic commitment scheme (Definition 11). The protocol Π_1 works as follows:

Computation Layer $\mathcal{L}_1$: for $b \in \{0,1\}$, each S_b^1 does the following:

- it has inputs $\mathsf{vk}, \mathsf{pk}, \mathsf{ek}_b, \mathsf{ct}_\alpha, \mathsf{ct}_\mathsf{k}, \mathsf{ct}_\mathsf{sk}, \mathsf{ct}_\Sigma, \mathsf{ct}_{y_0}, \sigma_{y_0}$
- if $\forall i \in [\ell_1]$, it holds that $\top \leftarrow \mathcal{S}.\mathsf{Verify}(\mathsf{vk}, (0, i, \mathsf{ct}_{y_0}[i]), \sigma_{y_0}[i])$,
  - $([\mathsf{ek}'_0]_b, [\mathsf{ek}'_1]_b, [y_1]_b, [\sigma_1]_b) \leftarrow \mathsf{DEHE.Eval}(b, \mathsf{ek}_b, (\mathsf{ct}_\alpha, \mathsf{ct}_\mathsf{k}, \mathsf{ct}_\Sigma, \mathsf{ct}_{y_0}), P_1(1))$

  else, abort and output $\perp$
- compute $\mathsf{pk}_{b,0} = \mathsf{Com}([\mathsf{ek}'_0]_b)$ and $\mathsf{pk}_{b,1} = \mathsf{Com}([\mathsf{ek}'_1]_b)$
- broadcast $\mathsf{vk}, \mathsf{pk}, \mathsf{ct}_\alpha, \mathsf{ct}_\mathsf{k}, \mathsf{ct}_\mathsf{sk}, \mathsf{ct}_\Sigma, [y_1]_b, [\sigma_1]_b, \mathsf{pk}_{b,0}, \mathsf{pk}_{b,1}$
- send $[\mathsf{ek}'_0]_b$ to S_0^2
- send $[\mathsf{ek}'_1]_b$ to S_1^2

Computation Layer $\mathcal{L}_2$: for $b \in \{0,1\}$, each S_b^2 does the following:

- receive $\{\mathsf{vk}^{b'}, \mathsf{pk}^{b'}, \mathsf{ct}_\alpha^{b'}, \mathsf{ct}_\mathsf{k}^{b'}, \mathsf{ct}_\mathsf{sk}^{b'}, \mathsf{ct}_\Sigma^{b'}, [\mathsf{ek}_b]_{b'}, \mathsf{pk}_{b',0}, \mathsf{pk}_{b',1}, [y_1]_{b'}, [\sigma_1]_{b'}\}_{b' \in \{0,1\}}$ from $\mathcal{L}_1$
- **Consistency Checks:**
  - if $\mathsf{vk}^0 \neq \mathsf{vk}^1 = \mathsf{vk}$, abort and output $\perp$
  - if $\mathsf{pk}^0 \neq \mathsf{pk}^1 = \mathsf{pk}$, abort and output $\perp$
  - if $\mathsf{pk}_{0,b} \neq \mathsf{Com}([\mathsf{ek}_b]_0)$ and $\mathsf{pk}_{1,b} \neq \mathsf{Com}([\mathsf{ek}_b]_1)$, abort and output $\perp$
  - set $\mathsf{ek}_b = [\mathsf{ek}_b]_0 + [\mathsf{ek}_b]_1$ and $\mathsf{pk}_0 = \mathsf{pk}_{0,0} + \mathsf{pk}_{0,1}$ and $\mathsf{pk}_1 = \mathsf{pk}_{1,0} + \mathsf{pk}_{1,1}$
  - if $\mathsf{pk} \neq \mathsf{pk}_0 + \mathsf{pk}_1$, abort and output $\perp$
  - if $\mathsf{ct}_\alpha^0 \neq \mathsf{ct}_\alpha^1 = \mathsf{ct}_\alpha$, abort and output $\perp$
  - if $\mathsf{ct}_\mathsf{k}^0 \neq \mathsf{ct}_\mathsf{k}^1 = \mathsf{ct}_\mathsf{k}$, abort and output $\perp$
  - if $\mathsf{ct}_\mathsf{sk}^0 \neq \mathsf{ct}_\mathsf{sk}^1 = \mathsf{ct}_\mathsf{sk}$, abort and output $\perp$
  - if $\mathsf{ct}_\Sigma^0 \neq \mathsf{ct}_\Sigma^1 = \mathsf{ct}_\Sigma$, abort and output $\perp$
- set $y_1 = [y_1]_0 + [y_1]_1$ and $\sigma_1 = [\sigma_1]_0 + [\sigma_1]_1$
- output $\mathsf{vk}, \mathsf{pk}, \mathsf{ct}_\alpha, \mathsf{ct}_\mathsf{k}, \mathsf{ct}_\mathsf{sk}, \mathsf{ct}_\Sigma, \mathsf{ek}_b, y_1, \sigma_1$

Proof. We need to show that the protocol presented in Algorithm 21 realizes the required layered functionality f in the presence of a malicious, adaptive, rushing adversary $\mathcal{A}$ that can corrupt all-but-one party in $\mathcal{L}_0$ and $\mathcal{L}_{\ell+1}$, and one party in each intermediate layer $\mathcal{L}_i$, for all $i \in [\ell]$.

Adaptive Security through Secure Erasures. In the layered setting where each party only sends one message, our protocol supports adaptive security using secure erasures: each party participating in the protocol computes the messages it needs to send, then erases its state of the protocol before sending the messages to the next layer as specified. This makes it such that when an adaptive adversary corrupts a party that has already sent its message, it does not gain access to that party's state in the protocol. Therefore, it suffices to design protocols that are only selectively secure and these can be lifted to adaptive security in the layered setting using secure erasures.

Going forward, we present a proof that the protocol $\Pi_\perp$ in Algorithm 21 is selectively secure in the presence of a malicious rushing adversary $\mathcal{A}$ that can corrupt all-but-one party in $\mathcal{L}_0$ and $\mathcal{L}_{\ell+1}$, and one party in each intermediate layer $\mathcal{L}_i$, for all $i \in [\ell]$. The proof for this follows from a PPT simulation algorithm $\mathsf{Sim}_\perp$ that we define for the protocol $\Pi_\perp$. Before detailing the simulator itself, let us first formalize essential notation below.

Notation. Let $\mathcal{F}_\mathsf{Setup}$ be the functionality given in Algorithm 17. Since we show that $\Pi_\perp$ realizes f in the $\mathcal{F}_\mathsf{Setup}$-hybrid, our simulator $\mathsf{Sim}_\perp$ needs to also internally emulate $\mathcal{F}_\mathsf{Setup}$ and the messages that the adversary needs to receive from it. Let

Figure 19 Layer protocol Π_i

Let $\mathsf{DEHE} = (\mathsf{KeyGen}, \mathsf{Gen}, \mathsf{Enc}, \mathsf{Eval})$ be a two-party DEHE scheme (Definition 1), $\mathcal{S} = (\mathsf{Gen}, \mathsf{Sign}, \mathsf{Verify})$ be a signature scheme (Definition 14) and Com be an additively homomorphic deterministic commitment scheme (Definition 11). The protocol Π_i works as follows:

Computation Layer $\mathcal{L}_i$: for $b \in \{0,1\}$, each S_σ^b does the following:

- it has inputs $\mathsf{vk}, \mathsf{pk}, \mathsf{ct}_\alpha, \mathsf{ct}_\mathsf{k}, \mathsf{ct}_\mathsf{sk}, \mathsf{ct}_\Sigma, \mathsf{ek}_b, y_{i-1}, \sigma_{i-1}$
- $([\mathsf{ek}'_0]_b, [\mathsf{ek}'_1]_b, [y_i]_b, [\sigma_i]_b) \leftarrow \mathsf{DEHE.Eval}(b, \mathsf{ek}_b, (\mathsf{ct}_\alpha, \mathsf{ct}_\mathsf{k}, \mathsf{ct}_\Sigma), P_i(i, y_{i-1}, \sigma_{i-1}))$
- compute $\mathsf{pk}_{b,0} = \mathsf{Com}([\mathsf{ek}'_0]_b)$ and $\mathsf{pk}_{b,1} = \mathsf{Com}([\mathsf{ek}'_1]_b)$
- broadcast $\mathsf{vk}, \mathsf{pk}, \mathsf{ct}_\alpha, \mathsf{ct}_\mathsf{k}, \mathsf{ct}_\mathsf{sk}, \mathsf{ct}_\Sigma, [y_i]_b, [\sigma_i]_b, \mathsf{pk}_{b,0}, \mathsf{pk}_{b,1}$
- send $[\mathsf{ek}'_0]_b$ to S_0^2
- send $[\mathsf{ek}'_1]_b$ to S_1^2

Computation Layer $\mathcal{L}_{i+1}$: for $b \in \{0,1\}$, each S_b^{i+1} does the following:

- receive $\{\mathsf{vk}^{b'}, \mathsf{pk}^{b'}, \mathsf{ct}_\alpha^{b'}, \mathsf{ct}_\mathsf{k}^{b'}, \mathsf{ct}_\mathsf{sk}^{b'}, \mathsf{ct}_\Sigma^{b'}, [\mathsf{ek}_b]_{b'}, \mathsf{pk}_{b',0}, \mathsf{pk}_{b',1}, [y_i]_{b'}, [\sigma_i]_{b'}\}_{b' \in \{0,1\}}$ from $\mathcal{L}_1$
- **Consistency Checks:**
  - if $\mathsf{vk}^0 \neq \mathsf{vk}^1 = \mathsf{vk}$, abort and output $\perp$
  - if $\mathsf{pk}^0 \neq \mathsf{pk}^1 = \mathsf{pk}$, abort and output $\perp$
  - if $\mathsf{pk}_{0,b} \neq \mathsf{Com}([\mathsf{ek}_b]_0)$ and $\mathsf{pk}_{1,b} \neq \mathsf{Com}([\mathsf{ek}_b]_1)$, abort and output $\perp$
  - set $\mathsf{ek}_b = [\mathsf{ek}_b]_0 + [\mathsf{ek}_b]_1$ and $\mathsf{pk}_0 = \mathsf{pk}_{0,0} + \mathsf{pk}_{0,1}$ and $\mathsf{pk}_1 = \mathsf{pk}_{1,0} + \mathsf{pk}_{1,1}$
  - if $\mathsf{pk} \neq \mathsf{pk}_0 + \mathsf{pk}_1$, abort and output $\perp$
  - if $\mathsf{ct}_\alpha^0 \neq \mathsf{ct}_\alpha^1 = \mathsf{ct}_\alpha$, abort and output $\perp$
  - if $\mathsf{ct}_\mathsf{k}^0 \neq \mathsf{ct}_\mathsf{k}^1 = \mathsf{ct}_\mathsf{k}$, abort and output $\perp$
  - if $\mathsf{ct}_\mathsf{sk}^0 \neq \mathsf{ct}_\mathsf{sk}^1 = \mathsf{ct}_\mathsf{sk}$, abort and output $\perp$
  - if $\mathsf{ct}_\Sigma^0 \neq \mathsf{ct}_\Sigma^1 = \mathsf{ct}_\Sigma$, abort and output $\perp$
- set $y_i = [y_i]_0 + [y_i]_1$ and $\sigma_i = [\sigma_i]_0 + [\sigma_i]_1$
- output $\mathsf{vk}, \mathsf{pk}, \mathsf{ct}_\alpha, \mathsf{ct}_\mathsf{k}, \mathsf{ct}_\mathsf{sk}, \mathsf{ct}_\Sigma, \mathsf{ek}_b, y_i, \sigma_i$

Figure 20 Layer protocol Π_ℓ

Let $\mathsf{DEHE} = (\mathsf{KeyGen}, \mathsf{Gen}, \mathsf{Enc}, \mathsf{Eval})$ be a two-party DEHE scheme (Definition 1), $\mathcal{S} = (\mathsf{Gen}, \mathsf{Sign}, \mathsf{Verify})$ be a signature scheme (Definition 14) and Com be an additively homomorphic deterministic commitment scheme (Definition 11). The protocol Π_ℓ works as follows:

Computation Layer $\mathcal{L}_\ell$: for $b \in \{0,1\}$, each S_b^ℓ does the following:

- it has inputs $\mathsf{vk}, \mathsf{pk}, \mathsf{ct}_\alpha, \mathsf{ct}_\mathsf{k}, \mathsf{ct}_\mathsf{sk}, \mathsf{ct}_\Sigma, \mathsf{ek}_b, y_{\ell-1}, \sigma_{\ell-1}$
- compute $([y_\ell]_b, [\sigma_\ell]_b) \leftarrow \mathsf{DEHE.Eval}(b, \mathsf{ek}_b, (\mathsf{ct}_\alpha, \mathsf{ct}_\mathsf{k}, \mathsf{ct}_\mathsf{sk}), P_\ell(\ell, y_{\ell-1}, \sigma_{\ell-1}))$
- broadcast $\mathsf{vk}, [y_\ell]_b, [\sigma_\ell]_b$

Output Layer $\mathcal{L}_{\ell+1}$: each output party $P_i^{\ell+1}$ does the following:

- receive $\{\mathsf{vk}_b, [y_\ell]_b, [\sigma_\ell]_b\}_{b \in \{0,1\}}$ from $\mathcal{L}_\ell$
- **Consistency Checks:**
  - if $\mathsf{vk}_0 \neq \mathsf{vk}_1 = \mathsf{vk}$, abort and output $\perp$
  - set $y_\ell = [y_\ell]_0 + [y_\ell]_1$ and $\sigma_\ell = [\sigma_\ell]_0 + [\sigma_\ell]_1$
  - if $\top \not\leftarrow \mathcal{S}.\mathsf{Verify}(\mathsf{vk}, (\ell, y_\ell), \sigma_\ell)$, abort and output $\perp$
- output y_ℓ

$\mathsf{DEHE} = (\mathsf{KeyGen}, \mathsf{Gen}, \mathsf{Enc}, \mathsf{Eval})$ be a two-party DEHE scheme (Definition 1). With respect to this, let $\mathsf{Sim}_\mathsf{DEHE}(\cdot)$ be the PPT simulator for DEHE that, for $n, \ell \in \mathbb{N}$, outputs

$$(\{\mathsf{ek}_i\}_{i \in [\ell]}, \{\mathsf{ct}_i\}_{i \in [n]}) \leftarrow \mathsf{Sim}_\mathsf{DEHE}(1^\kappa, n, \ell)$$

where n is the DEHE input size, and ℓ is the number of layers in the computation. Let $\mathcal{S} = (\mathsf{Gen}, \mathsf{Sign}, \mathsf{Verify})$ be a signature scheme (Definition 14). Let $\Sigma = (\mathsf{Gen}, \mathsf{MAC}, \mathsf{Verify})$ be a MAC scheme (Definition 13), and Com be a deterministic commitment scheme (Definition 11). For the protocol $\Pi_\perp$ in Algorithm 21, let $\mathsf{Sim}_\perp$

Figure 21 Maliciously Secure Layered MPC satisfying Security with Abort

Let $\mathcal{F}_{\mathsf{Setup}}$ be the functionality given in Algorithm 6. Let Π_1, $\forall i > 1 \in [l-1], \Pi_i$ and Π_ℓ be the functionalities in Algorithms 7 to 9 respectively. Let $\mathcal{S} = (\mathsf{Gen}, \mathsf{Sign}, \mathsf{Verify})$ be a signature scheme (Definition 14) and Com be a deterministic commitment scheme (Definition 11). Then the layered protocol $\Pi_\perp$ in the $\mathcal{F}_{\mathsf{Setup}}$-hybrid operates as follows:

- **Input Layer $\mathcal{L}_0$:** Each party $P_i^0 \in \mathcal{L}_0$ has as input x_i to the protocol and operates as follows:
  - receive $\{b : \mathsf{ct}_{i,b}, \sigma^{\mathsf{ct}}_{i,b}\}_{b\in\{0,1\}}, \mathsf{vk}, \mathsf{pk}, \mathsf{ct}_\alpha, \mathsf{ct}_\mathsf{k}, \mathsf{ct}_\mathsf{sk}, \mathsf{ct}_\Sigma, \{\langle \mathsf{ek}_b^1\rangle_i, \sigma_{b,i}\}_{b\in\{0,1\}}$ from $\mathcal{F}_{\mathsf{Setup}}$
  - for $b \in \{0,1\}$, send the tuple $\left(\langle \mathsf{ek}_b^1\rangle_i, \sigma_{b,i}\right)$ to S_b^1
  - broadcast $\mathsf{ct}_{i,x_i}, \sigma^{\mathsf{ct}}_{i,x_i}, \mathsf{vk}, \mathsf{pk}, \mathsf{ct}_\alpha, \mathsf{ct}_\mathsf{k}, \mathsf{ct}_\mathsf{sk}, \mathsf{ct}_\Sigma$
- **Computation Layer $\mathcal{L}_1$:** Each party S_b^1 for $b \in \{0,1\}$ does the following:
  - receive $\left(\langle \mathsf{ek}_b^1\rangle_i, \sigma_{b,i}, \mathsf{ct}_{i,x_i}, \sigma^{\mathsf{ct}}_{i,x_i}, \mathsf{vk}^i, \mathsf{pk}^i, \mathsf{ct}^i_\alpha, \mathsf{ct}^i_\mathsf{k}, \mathsf{ct}^i_\mathsf{sk}, \mathsf{ct}^i_\Sigma\right)$ from each $P_i^0 \in \mathcal{L}_0$
  - Consistency Checks:
    - set $\mathsf{vk} = \mathsf{vk}^1$ and if $\exists P_i^0 \in \mathcal{L}_0$ s.t. $\mathsf{vk}^i \neq \mathsf{vk}$, then $\perp$
    - set $\mathsf{pk} = \mathsf{pk}^1$ and if $\exists P_i^0 \in \mathcal{L}_0$ s.t. $\mathsf{pk}^i \neq \mathsf{pk}$, then $\perp$
    - set $\mathsf{ct}_\alpha = \mathsf{ct}^1_\alpha$ and if $\exists P_i^0 \in \mathcal{L}_0$ s.t. $\mathsf{ct}^i_\alpha \neq \mathsf{ct}_\alpha$, then $\perp$
    - set $\mathsf{ct}_\mathsf{k} = \mathsf{ct}^1_\mathsf{k}$ and if $\exists P_i^0 \in \mathcal{L}_0$ s.t. $\mathsf{ct}^i_\mathsf{k} \neq \mathsf{ct}_\mathsf{k}$, then $\perp$
    - set $\mathsf{ct}_\mathsf{sk} = \mathsf{ct}^1_\mathsf{sk}$ and if $\exists P_i^0 \in \mathcal{L}_0$ s.t. $\mathsf{ct}^i_\mathsf{sk} \neq \mathsf{ct}_\mathsf{sk}$, then $\perp$
    - set $\mathsf{ct}_\Sigma = \mathsf{ct}^1_\Sigma$ and if $\exists P_i^0 \in \mathcal{L}_0$ s.t. $\mathsf{ct}^i_\Sigma \neq \mathsf{ct}_\Sigma$, then $\perp$
    - if $\forall i \in [\ell_0]$, it holds that $\top \leftarrow \mathcal{S}.\mathsf{Verify}(\mathsf{vk}, (i, b, \langle \mathsf{ek}_b^1\rangle_i), \sigma_{b,i})$,
      - compute $\mathsf{ek}_b = \sum_{i\in[\ell_1]} \langle \mathsf{ek}_b^1\rangle_i$

      else, abort and output $\perp$
    - set $\mathsf{ct}_{y_0} = \{\mathsf{ct}_{i,x_i}\}_{i\in[\ell_0]}$ and $\sigma_{y_0} = \{\sigma^{\mathsf{ct}}_{i,x_i}\}_{i\in[\ell_0]}$
  - invoke protocol Π_1 using $\mathsf{vk}, \mathsf{pk}, \mathsf{ek}_b, \mathsf{ct}_\alpha, \mathsf{ct}_\mathsf{k}, \mathsf{ct}_\mathsf{sk}, \mathsf{ct}_\Sigma, \mathsf{ct}_{y_0}, \sigma_{y_0}$ as inputs
- $\forall i > 1 \in [\ell]$ **Computation Layer $\mathcal{L}_i$:** Each S_b^i for $b \in \{0,1\}$ does the following:
  - receive $\mathsf{vk}, \mathsf{pk}, \mathsf{ct}_\alpha, \mathsf{ct}_\mathsf{k}, \mathsf{ct}_\mathsf{sk}, \mathsf{ct}_\Sigma, \mathsf{ek}_b, y_{i-1}, \sigma_{i-1}$ from the execution of Π_{i-1} and continue by invoking Π_i with these inputs
- **Output Layer $\mathcal{L}_{\ell+1}$:** Each party $P_i^{\ell+1} \in \mathcal{L}_{\ell+1}$ outputs y_ℓ after completing Π_ℓ

be the PPT algorithm that simulates the view produced by an interaction with a PPT malicious adversary $\mathcal{A}$ corrupting all-but-one parties in the input and output layer and one party in each intermediate layer. For each $i \in [\ell+1] \cup \{0\}$ let $\mathcal{H}_i \subseteq \mathcal{L}_i$ be the set of honest parties in layer $\mathcal{L}_i$. Let $\mathcal{C}_i = \mathcal{L}_i - \mathcal{H}_i$ be the set of corrupt parties in this layer.

Simulation. The simulator $\mathsf{Sim}_\perp$ is detailed in Algorithms 22 and 23. It uses the DEHE simulator $\mathsf{Sim}_{\mathsf{DEHE}}$ given above in a black-box way.

Hybrids. It remains to prove that the joint view produced by the simulator $\mathsf{Sim}_\perp$ is computationally indistinguishable from that of the real execution of the protocol. We do so by considering the following intermediate hybrid experiments:

- H_0 : This hybrid experiment is defined to be the same as the interaction of the adversary $\mathcal{A}$ in the simulation with $\mathsf{Sim}_\perp$.
- H_1 : This is an intermediate hybrid experiment that differs from the simulation $\mathsf{Sim}_\perp$ in the following way: the execution only aborts if the **consistency checks** given in Algorithms 18 to 21 (performed on the values $\mathsf{Sim}_\perp$ internally computed for the honest party and those received from $\mathcal{A}$) would cause one. In all other aspects, H_1 and the previous hybrid are the same. This experiment is detailed in Algorithm 24. Looking ahead, we can show that the view produced by this hybrid experiment H_1 is computationally indistinguishable from that of H_0 via a reduction to the unforgeability property of the digital signature scheme $\mathcal{S}$.

Figure 22 Simulator $\mathsf{Sim}_\perp$ for Layer protocol $\Pi_\perp$ in the $\mathcal{F}_{\mathsf{Setup}}$-hybrid

Let $\mathsf{DEHE} = (\mathsf{KeyGen}, \mathsf{Gen}, \mathsf{Enc}, \mathsf{Eval})$ be a two-party DEHE scheme (Definition 1). Let $\mathcal{S} = (\mathsf{Gen}, \mathsf{Sign}, \mathsf{Verify})$ be a signature scheme (Definition 14), F be a PRF (Definition 12), $\Sigma = (\mathsf{Gen}, \mathsf{MAC}, \mathsf{Verify})$ be a MAC scheme (Definition 13), and Com be a deterministic commitment scheme (Definition 11). The simulator $\mathsf{Sim}_\perp$ works as follows:

$\mathsf{Sim}_\perp$ first emulates $\mathcal{F}_{\mathsf{Setup}}$ and does the following:
- generate $(\mathsf{sk}, \mathsf{vk}) \leftarrow \mathcal{S}.\mathsf{Gen}(1^\kappa)$ and $\mathsf{sk}_\Sigma \leftarrow \Sigma.\mathsf{Gen}(1^\kappa)$
- sample $\mathsf{k}' \leftarrow \{0,1\}^{|\text{DEHE key}|}$ and compute $\mathsf{pk} = \mathsf{Com}(\mathsf{k}')$
- let $n = |\text{PRF key}| + |\text{DEHE key}| + |\mathsf{sk}| + |\mathsf{sk}_\Sigma|$
- compute $(\{\mathsf{ek}_i^*\}_{i\in[\ell]}, \mathsf{ct}_\alpha^*, \mathsf{ct}_\mathsf{k}^*, \mathsf{ct}_\mathsf{sk}^*, \mathsf{ct}_\Sigma^*, \{\mathsf{ct}_{j,0}^*, \mathsf{ct}_{j,1}^*\}_{j\in[\ell_1]}) \leftarrow \mathsf{Sim}_\mathsf{DEHE}(1^\kappa, n + 2\ell_1, \ell)$
- $\forall j \in [\ell_1], b \in \{0,1\}$, compute $\sigma_{b,j}^{\mathsf{ct}*} \leftarrow \mathcal{S}.\mathsf{Sign}(\mathsf{sk}, (0, b, j, \mathsf{ct}_{j,b}^*))$
- for $S_b^1 \in \mathcal{C}_1$, sample $\{\langle \mathsf{ek}_b^1 \rangle_j\}_{j\in[\ell_1]}$ such that $\mathsf{ek}_1^* = \sum_{j\in[\ell_1]} \langle \mathsf{ek}_b^1 \rangle_j$
- sample $\{\langle \mathsf{ek}_{\neg b}^1 \rangle_j\}_{j\in[\ell_1]}$ uniformly at random
- for $b' \in \{0,1\}, j \in [\ell_1]$, compute $\sigma_{b',j} = \mathcal{S}.\mathsf{Sign}(\mathsf{sk}, (0, b', j, \langle \mathsf{ek}_{b'}^1 \rangle_j))$
- $\mathcal{A}$ receives, for each corrupt $C_j \in \mathcal{C}_0$, the following tuple from $\mathsf{Sim}_\perp$:

$$\{b' : \mathsf{ct}_{j,b'}^*, \sigma_{j,b'}^{\mathsf{ct}*}\}_{b'\in\{0,1\}}, \mathsf{vk}, \mathsf{pk}, \mathsf{ct}_\alpha^*, \mathsf{ct}_\mathsf{k}^*, \mathsf{ct}_\mathsf{sk}^*, \mathsf{ct}_\Sigma^*, \{\langle \mathsf{ek}_{b'}^1 \rangle_j, \sigma_{b',j}\}_{b'\in\{0,1\}}$$

$\mathsf{Sim}_\perp$, emulating the first round of $\Pi_\perp$, does the following:
- For each $C_j \in \mathcal{H}_0$, sample $x_j \leftarrow \{0,1\}$ and send to $\mathcal{A}$ the tuple: $\langle \mathsf{ek}_b^1 \rangle_j, \sigma_{b,j}, \mathsf{ct}_{j,x_j}^*, \sigma_{j,x_j}^{\mathsf{ct}*}, \mathsf{vk}, \mathsf{pk}, \mathsf{ct}_\alpha^*, \mathsf{ct}_\mathsf{k}^*, \mathsf{ct}_\mathsf{sk}^*, \mathsf{ct}_\Sigma^*$
- On behalf of $S_{\neg b}^1 \in \mathcal{H}_1$, $\mathsf{Sim}_\perp$ accepts from $\mathcal{A}$, for each $C_j \in \mathcal{C}_0$ the tuple, $\langle \mathsf{ek}_{\neg b}^1 \rangle_j, \sigma_{\neg b,j}, \mathsf{ct}_{j,x_j}, \sigma_{j,x_j}^{\mathsf{ct}}, \mathsf{vk}, \mathsf{pk}, \mathsf{ct}_\alpha^*, \mathsf{ct}_\mathsf{k}^*, \mathsf{ct}_\mathsf{sk}^*, \mathsf{ct}_\Sigma^*$
  - send $\perp$ to $\mathcal{F}_\perp$ if this does not match the internal computation
  - **Input Extraction:** For each $C_j \in \mathcal{C}_0$,
    - if $\mathsf{ct}_{j,x_j} \notin \{\mathsf{ct}_{j,0}^*, \mathsf{ct}_{j,1}^*\}$, then abort and send $\perp$ to $\mathcal{F}_\perp$
    - else, set $x_j = 0$ if $\mathsf{ct}_{j,x_j} == \mathsf{ct}_{j,0}^*$ and $x_j = 1$ otherwise

    Finally, send $\{x_j\}_{C_j\in\mathcal{C}_0}$ as the adversary's input to $\mathcal{F}_\perp$
- $\mathsf{Sim}_\perp$ (emulating protocol Π_1) internally computes, $([\mathsf{ek}_0']_b, [\mathsf{ek}_1']_b, [y_1]_b, [\sigma_1]_b) \leftarrow \mathsf{DEHE.Eval}(b, \mathsf{ek}_1^*, (\mathsf{ct}_\alpha^*, \mathsf{ct}_\mathsf{k}^*, \mathsf{ct}_\Sigma^*, \mathsf{ct}_{y_0}^*), P_1(1))$ as the corrupt S_b^1 would, by setting $\mathsf{ct}_{y_0} = \{\mathsf{ct}_{i,x_i}^*\}_{C_i\in\mathcal{L}_0}$

$\mathsf{Sim}_\perp$ samples $\{y_i\}_{i\in[\ell-1]}$ uniformly at random and does the following:
- for $i \in [\ell - 1]$, compute $\sigma_i = \Sigma.\mathsf{MAC}(\mathsf{sk}_\Sigma, y_i)$

- H_2 : This is an intermediate hybrid experiment that differs from the previous hybrid in the following way: the honest parties' inputs $\{x_j\}_{C_j\in\mathcal{H}_0}$ are used, along with the corrupt parties' inputs $\{x_j^*\}_{C_j\in\mathcal{C}_0}$ that were extracted, to compute the intermediate computation values $\{y_i\}_{i\in[\ell]}$ using the masks $\{r_i\}_{i\in[\ell-1]}$ – like in the real execution. The masks $\{r_i\}_{i\in[\ell-1]}$ are sampled uniformly at random and these honest parties' inputs are also used when choosing input ciphertexts in y_0. In all other aspects, H_2 and the previous hybrid are the same. This experiment is detailed in Algorithms 25 and 26. The view produced by this hybrid experiment H_2 is identically distributed to that of H_1 since the masks act as one-time-pads making the intermediate values in $\{y_i\}_{i\in[\ell]}$ be uniform random strings. The choice of initial simulated ciphertexts using $\{x_j\}_{C_j\in\mathcal{H}_0}$ (as opposed to uniformly at random) also is identically distributed as the ciphertexts themselves are uncorrelated from the choice.
- H_3 : This is an intermediate hybrid experiment that differs from the previous hybrid in the following way: instead of using $\mathsf{Sim}_\mathsf{DEHE}$ to simulate the corrupt evaluation keys and all ciphertexts, sample the PRF key α, a MAC key sk_Σ and compute the real DEHE key k, evaluation keys $\{\mathsf{ek}_i^*\}_{i\in[\ell]}$ (using fresh randomness),

Figure 23 Simulator $\mathsf{Sim}_\perp$ for Layer protocol $\Pi_\perp$ in the $\mathcal{F}_{\mathsf{Setup}}$-hybrid

Let $\mathsf{DEHE} = (\mathsf{KeyGen}, \mathsf{Gen}, \mathsf{Enc}, \mathsf{Eval})$ be a two-party DEHE scheme (Definition 1). Let $\mathcal{S} = (\mathsf{Gen}, \mathsf{Sign}, \mathsf{Verify})$ be a signature scheme (Definition 14), F be a PRF (Definition 12), $\Sigma = (\mathsf{Gen}, \mathsf{MAC}, \mathsf{Verify})$ be a MAC scheme (Definition 13), and Com be a deterministic commitment scheme (Definition 11).

$\forall i \in [\ell - 1]$, $\mathsf{Sim}_\perp$, emulating the ith computation round of $\Pi_\perp$, does the following:

- On behalf of $S^i_{\neg b} \in \mathcal{H}_i$, do the following:
  - compute $[y_i]_{\neg b} = y_i - [y_i]_b$ and $[\sigma_i]_{\neg b} = \sigma_i - [\sigma_i]_b$
  - for $S^{i+1}_{b'} \in \mathcal{C}_{i+1}$, set $[\mathsf{ek}_{b'}]_{\neg b} = \mathsf{ek}^*_i - [\mathsf{ek}'_{b'}]_b$ and $[\mathsf{ek}_{\neg b'}]_{\neg b} = \mathsf{k}' - \mathsf{ek}^*_i - [\mathsf{ek}'_{\neg b'}]_b$
  - compute $\mathsf{pk}_{b',\neg b} = \mathsf{Com}([\mathsf{ek}_{b'}]_{\neg b})$ and $\mathsf{pk}_{\neg b',\neg b} = \mathsf{Com}([\mathsf{ek}_{\neg b'}]_{\neg b})$

  send $([y_i]_{\neg b}, [\sigma_i]_{\neg b}, [\mathsf{ek}_{b'}]_{\neg b}, \mathsf{pk}_{b',\neg b}, \mathsf{pk}_{\neg b',\neg b}, \mathsf{vk}, \mathsf{pk}, \mathsf{ct}^*_\alpha, \mathsf{ct}^*_\mathsf{k}, \mathsf{ct}^*_\mathsf{sk}, \mathsf{ct}^*_\Sigma)$ to $\mathcal{A}$
- $\mathsf{Sim}_\perp$ accepts $([y_i]_b, [\sigma_i]_b, [\mathsf{ek}'_{\neg b'}]_b, \mathsf{pk}_{b',b}, \mathsf{pk}_{\neg b',b}, \mathsf{vk}, \mathsf{pk}, \mathsf{ct}^*_\alpha, \mathsf{ct}^*_\mathsf{k}, \mathsf{ct}^*_\mathsf{sk}, \mathsf{ct}^*_\Sigma)$ from $\mathcal{A}$ and if it does not match internal computation, send $\perp$ to $\mathcal{F}_\perp$
- $\mathsf{Sim}_\perp$ (emulating protocol Π_{i+1}) computes, $([\mathsf{ek}'_0]_{b'}, [\mathsf{ek}'_1]_{b'}, [y_{i+1}]_{b'}, [\sigma_{i+1}]_{b'}) \leftarrow \mathsf{DEHE.Eval}(b', \mathsf{ek}^*_{i+1}, (\mathsf{ct}^*_\alpha, \mathsf{ct}^*_\mathsf{k}, \mathsf{ct}^*_\Sigma), P_{i+1}(i+1, y_i, \sigma_i))$ as $S^{i+1}_{b'}$ would

Before the final round, $\mathsf{Sim}_\perp$ receives from $\mathcal{F}_\perp$ the functionality output $f(x)$.

- set $y_\ell = f(x)$ and $\sigma_\ell = \mathcal{S}.\mathsf{Sign}(\mathsf{sk}, (\ell, y_\ell))$ and on behalf of $S^\ell_{\neg b} \in \mathcal{H}_\ell$, do the following:
  - compute $([y_\ell]_{b'}, [\sigma_\ell]_{b'}) \leftarrow \mathsf{DEHE.Eval}(b', \mathsf{ek}^*_\ell, (\mathsf{ct}^*_\alpha, \mathsf{ct}^*_\mathsf{k}, \mathsf{ct}^*_\mathsf{sk}), P_\ell(\ell, y_{\ell-1}, \sigma_{\ell-1}))$ as $S^\ell_{b'}$ would
  - compute $[y_\ell]_{\neg b} = y_\ell - [y_\ell]_b$ and $[\sigma_\ell]_{\neg b} = \sigma_\ell - [\sigma_\ell]_b$

  send $([y_i]_{\neg b}, [\sigma_i]_{\neg b}, \mathsf{vk})$ to $\mathcal{A}$
- $\forall P^{\ell+1}_j \in \mathcal{H}_{i+1}$, $\mathsf{Sim}_\perp$ accepts $([y_i]_b, [\sigma_i]_b, \mathsf{vk})$ from $\mathcal{A}$ and, if this is not the same as that internally computed, send $\perp$ to $\mathcal{F}_\perp$. Otherwise send $\top$ to $\mathcal{F}_\perp$.

and ciphertexts – like in the real execution. In all other aspects, H_3 and the previous hybrid are the same. This experiment is detailed in Algorithms 27 and 28. We can show that the view produced by this hybrid experiment H_3 is computationally indistinguishable from that of H_2 via a reduction to the multi-instance selective security of DEHE.

- H_4 : This is an intermediate hybrid experiment that differs from the previous hybrid in the following way: all the MACs $\{\sigma_i\}_{i\in[\ell]}$ over intermediate outputs of the DEHE evaluation are verified honestly as in the real execution. This experiment is detailed in Algorithms 29 and 30. We can show that the view produced by this hybrid experiment H_4 is computationally indistinguishable from that of H_3 via a reduction to the unforgeability property of the MAC scheme Σ.
- H_5 : This is an intermediate hybrid experiment that differs from the previous hybrid in the following way: all the values committed using the deterministic commitment scheme Com are derived from the DEHE key generation algorithm, instead of being sampled independently. This experiment is detailed in Algorithms 31 and 32. Both distributions of inputs to Com are uniform random distributions and we can show that the view produced by this hybrid experiment H_5 is computationally indistinguishable from that of H_4 via a reduction to the computational hiding property of Com.
- H_6 : This experiment is the same as the real execution. Note that this differs from the previous experiment only in that all the masks $\{r_i\}_{i\in[\ell-1]}$, the randomness used to re-generate the DEHE evaluation keys and the signing and MAC randomness within the RMS programs is pseudorandomly generated from the PRF F keyed with α. We can show that the view produced by this hybrid experiment H_6 is computationally indistinguishable from that of H_5 via a reduction to the pseudorandomness of F.

Figure 24 Hybrid Experiment H_1

Let $\mathsf{DEHE} = (\mathsf{KeyGen}, \mathsf{Gen}, \mathsf{Enc}, \mathsf{Eval})$ be a two-party DEHE scheme (Definition 1). Let $\mathcal{S} = (\mathsf{Gen}, \mathsf{Sign}, \mathsf{Verify})$ be a signature scheme (Definition 14), F be a PRF (Definition 12), $\Sigma = (\mathsf{Gen}, \mathsf{MAC}, \mathsf{Verify})$ be a MAC scheme (Definition 13), and Com be a deterministic commitment scheme (Definition 11).

H_1 first emulates $\mathcal{F}_{\mathsf{Setup}}$ and does the following:
- generate $(\mathsf{sk}, \mathsf{vk}) \leftarrow \mathcal{S}.\mathsf{Gen}(1^\kappa)$ and $\mathsf{sk}_\Sigma \leftarrow \Sigma.\mathsf{Gen}(1^\kappa)$
- sample $\mathsf{k}' \leftarrow \{0,1\}^{|\text{DEHE key}|}$ and compute $\mathsf{pk} = \mathsf{Com}(\mathsf{k}')$
- let $n = |\text{PRF key}| + |\text{DEHE key}| + |\mathsf{sk}| + |\mathsf{sk}_\Sigma|$
- compute $(\{\mathsf{ek}_i^*\}_{i\in[\ell]}, \mathsf{ct}_\alpha^*, \mathsf{ct}_\mathsf{k}^*, \mathsf{ct}_\mathsf{sk}^*, \mathsf{ct}_\Sigma^*, \{\mathsf{ct}_{j,0}^*, \mathsf{ct}_{j,1}^*\}_{j\in[\ell_1]}) \leftarrow \mathsf{Sim}_{\mathsf{DEHE}}(1^\kappa, n + 2\ell_1, \ell)$
- $\forall j \in [\ell_1], b \in \{0,1\}$, compute $\sigma_{b,j}^{\mathsf{ct}*} \leftarrow \mathcal{S}.\mathsf{Sign}(\mathsf{sk}, (1, b, j, \mathsf{ct}_{j,b}^*))$
- for $S_b^1 \in \mathcal{C}_1$, sample $\{\langle \mathsf{ek}_b^1 \rangle_j\}_{j\in[\ell_1]}$ such that $\mathsf{ek}_1^* = \sum_{j\in[\ell_1]} \langle \mathsf{ek}_b^1 \rangle_j$
- sample $\{\langle \mathsf{ek}_{\neg b}^1 \rangle_j\}_{j\in[\ell_1]}$ uniformly at random
- for $b' \in \{0,1\}, j \in [\ell_1]$, compute $\sigma_{b',j} = \mathcal{S}.\mathsf{Sign}(\mathsf{sk}, (1, b', j, \langle \mathsf{ek}_{b'}^1 \rangle_j))$
- $\mathcal{A}$ receives, for each corrupt $C_j \in \mathcal{C}_0$, the following tuple:

$$\{b' : \mathsf{ct}_{j,b'}^*, \sigma_{j,b'}^{\mathsf{ct}*}\}_{b'\in\{0,1\}}, \mathsf{vk}, \mathsf{pk}, \mathsf{ct}_\alpha^*, \mathsf{ct}_\mathsf{k}^*, \mathsf{ct}_\mathsf{sk}^*, \mathsf{ct}_\Sigma^*, \{\langle \mathsf{ek}_{b'}^1 \rangle_j, \sigma_{b',j}\}_{b'\in\{0,1\}}$$

H_1, emulating the first round of $\Pi_\perp$, does the following:
- For each $C_j \in \mathcal{H}_0$, sample $x_j \leftarrow \{0,1\}$ and send to $\mathcal{A}$ the tuple: $\langle \mathsf{ek}_b^1 \rangle_j, \sigma_{b,j}, \mathsf{ct}_{j,x_j}^*, \sigma_{j,x_j}^{\mathsf{ct}*}, \mathsf{vk}, \mathsf{pk}, \mathsf{ct}_\alpha^*, \mathsf{ct}_\mathsf{k}^*, \mathsf{ct}_\mathsf{sk}^*, \mathsf{ct}_\Sigma^*$
- On behalf of $S_{\neg b}^1 \in \mathcal{H}_1$, accept from $\mathcal{A}$, for each $C_j \in \mathcal{C}_0$ the tuple, $\langle \mathsf{ek}_{\neg b}^1 \rangle_j, \sigma_{\neg b,j}, \mathsf{ct}_{j,x_j}, \sigma_{j,x_j}^{\mathsf{ct}}, \mathsf{vk}, \mathsf{pk}, \mathsf{ct}_\alpha^*, \mathsf{ct}_\mathsf{k}^*, \mathsf{ct}_\mathsf{sk}^*, \mathsf{ct}_\Sigma^*$ abort if any consistency checks given in $\Pi_\perp$ fail
- H_1, (emulating protocol Π_1) internally computes, $([\mathsf{ek}_0']_b, [\mathsf{ek}_1']_b, [y_1]_b, [\sigma_1]_b) \leftarrow \mathsf{DEHE.Eval}(b, \mathsf{ek}_1^*, (\mathsf{ct}_\alpha^*, \mathsf{ct}_\mathsf{k}^*, \mathsf{ct}_\Sigma^*, \mathsf{ct}_{y_0}^*), P_1(1))$ as the corrupt S_b^1 would, by setting $\mathsf{ct}_{y_0} = \{\mathsf{ct}_{i,x_i}^*\}_{C_i\in\mathcal{L}_0}$

H_1 samples $\{y_i\}_{i\in[\ell-1]}$ uniformly at random and does the following:
- for $i \in [\ell-1]$, compute $\sigma_i = \Sigma.\mathsf{MAC}(\mathsf{sk}_\Sigma, (i, y_i))$

$\forall i \in [\ell-1]$, H_1, emulating the ith computation round of $\Pi_\perp$, does the following:
- On behalf of $S_{\neg b}^i \in \mathcal{H}_i$, do the following:
  - compute $[y_i]_{\neg b} = y_i - [y_i]_b$ and $[\sigma_i]_{\neg b} = \sigma_i - [\sigma_i]_b$
  - for $S_{b'}^{i+1} \in \mathcal{C}_{i+1}$, set $[\mathsf{ek}_{b'}]_{\neg b} = \mathsf{ek}_i^* - [\mathsf{ek}_{b'}']_b$ and $[\mathsf{ek}_{\neg b'}]_{\neg b} = \mathsf{k}' - \mathsf{ek}_i^* - [\mathsf{ek}_{\neg b'}']_b$
  - compute $\mathsf{pk}_{b',\neg b} = \mathsf{Com}([\mathsf{ek}_{b'}]_{\neg b})$ and $\mathsf{pk}_{\neg b',\neg b} = \mathsf{Com}([\mathsf{ek}_{\neg b'}]_{\neg b})$
  - abort if any **consistency checks** given in Π_i fail

  send $([y_i]_{\neg b}, [\sigma_i]_{\neg b}, [\mathsf{ek}_{b'}]_{\neg b}, \mathsf{pk}_{b',\neg b}, \mathsf{pk}_{\neg b',\neg b}, \mathsf{vk}, \mathsf{pk}, \mathsf{ct}_\alpha^*, \mathsf{ct}_\mathsf{k}^*, \mathsf{ct}_\mathsf{sk}^*, \mathsf{ct}_\Sigma^*)$ to $\mathcal{A}$
- H_1 accepts $([y_i]_b, [\sigma_i]_b, [\mathsf{ek}_{\neg b'}']_b, \mathsf{pk}_{b',b}, \mathsf{pk}_{\neg b',b}, \mathsf{vk}, \mathsf{pk}, \mathsf{ct}_\alpha^*, \mathsf{ct}_\mathsf{k}^*, \mathsf{ct}_\mathsf{sk}^*, \mathsf{ct}_\Sigma^*)$ from $\mathcal{A}$
- H_1, emulating protocol Π_{i+1}, computes, $([\mathsf{ek}_0']_{b'}, [\mathsf{ek}_1']_{b'}, [y_{i+1}]_{b'}, [\sigma_{i+1}]_{b'}) \leftarrow \mathsf{DEHE.Eval}(b', \mathsf{ek}_{i+1}^*, (\mathsf{ct}_\alpha^*, \mathsf{ct}_\mathsf{k}^*, \mathsf{ct}_\Sigma^*), P_{i+1}(i+1, y_i, \sigma_i))$ as $S_{b'}^{i+1}$ would

Before the final round, H_1 sets $y_\ell = f(x)$ and $\sigma_\ell = \mathcal{S}.\mathsf{Sign}(\mathsf{sk}, (\ell, y_\ell))$.
- on behalf of $S_{\neg b}^\ell \in \mathcal{H}_\ell$, do the following:
  - compute $([y_\ell]_{b'}, [\sigma_\ell]_{b'}) \leftarrow \mathsf{DEHE.Eval}(b', \mathsf{ek}_\ell^*, (\mathsf{ct}_\alpha^*, \mathsf{ct}_\mathsf{k}^*, \mathsf{ct}_\mathsf{sk}^*), P_\ell(\ell, y_{\ell-1}, \sigma_{\ell-1}))$
  - compute $[y_\ell]_{\neg b} = y_\ell - [y_\ell]_b$ and $[\sigma_\ell]_{\neg b} = \sigma_\ell - [\sigma_\ell]_b$

  send $([y_i]_{\neg b}, [\sigma_i]_{\neg b}, \mathsf{vk})$ to $\mathcal{A}$
- $\forall P_j^{\ell+1} \in \mathcal{H}_{i+1}$, H_1 accepts $([y_i]_b, [\sigma_i]_b, \mathsf{vk})$ from $\mathcal{A}$ and, if **consistency checks** given in Π_ℓ fail, abort the protocol.

Claim. Assuming $\mathcal{S}$ is a digital signature scheme (Definition 14) satisfying unforgeability, $\mathsf{H}_0 \overset{c}{\approx} \mathsf{H}_1$.

Proof. Recall that the hybrid experiment H_0 behaves the same way as $\mathsf{Sim}_\perp$ (Algorithms 22 and 23) and H_1 (Algorithm 24) differs only in that the consistency checks that H_1 performs on behalf of the honest parties involve verifying signatures on

messages, even if they do not match the internally computed messages by the hybrid experiment. In contrast, in H_0, the protocol aborts when the adversary's messages are not the same as that the simulation computes. The only cases where H_0 aborts but H_1 does not would correspond to an instance where the adversary has forged a digital signature on a message of its choice, and has provided correct additive shares of the same, causing it to verify correctly on the honest party's end.

We can show that both experiments will produce exactly the same view, except with negligible probability by reduction to the unforgeability property of the underlying signature scheme $\mathcal{S}$ (Definition 14). Consider for the sake of contradiction that there exists a PPT adversary $\mathsf{D}_\perp$ that can, with non-negligible probability ϵ, produce a view distributed as in H_1 but not in H_0. We show that if this were possible, then $\mathsf{D}_\perp$ can be used in a black-box way by a PPT adversary $\mathsf{D}_\mathcal{S}$ to violate the unforgeability property of the signature scheme $\mathcal{S}$. This distinguisher $\mathsf{D}_\mathcal{S}$ would work as follows:

- $\mathsf{D}_\mathcal{S}$ has a verification key vk and oracle access to a signing algorithm $\mathcal{O}_\mathcal{S}$ parameterized with the corresponding secret signing key sk. It uses this to sign messages and starts interacting with $\mathsf{D}_\perp$ exactly as given in the simulation H_0 in Algorithms 22 and 23.
- If for any ($\mathcal{F}_{\mathsf{Setup}}$ or DEHE output) message that $\mathsf{D}_\perp$ sends, it is not the case that it equals the message that $\mathsf{D}_\mathcal{S}$ has internally computed, reconstruct the value m and corresponding signature σ using both the honest and corrupt party's additive shares. If $\top \leftarrow \mathcal{S}.\mathsf{Verify}(\mathsf{vk}, m, \sigma)$, output (m, σ) and continue the protocol as in H_1 (Algorithm 24). Otherwise, abort the protocol.

Note that in the above, if the protocol has aborted without $\mathsf{D}_\mathcal{S}$ outputting a forgery (m, σ), the view of $\mathsf{D}_\perp$ is the same as that of H_0. Otherwise, it is the same as that in the experiment H_1. The advantage of $\mathsf{D}_\mathcal{S}$ is the same as that of $\mathsf{D}_\perp$ not inducing the view of H_0, which is non-negligible by assumption. However, since $\mathcal{S}$ is an unforgeable digital signature scheme (Definition 14), no such $\mathsf{D}_\mathcal{S}$ can exist, yielding a contradiction.

Claim. Assuming DEHE is a secure two-server DEHE scheme (Definition 1) satisfying multi-instance selective security, $\mathsf{H}_2 \overset{c}{\approx} \mathsf{H}_3$.

Proof. Consider for the sake of contradiction that there exists a PPT distinguisher $\mathsf{D}_\perp$ that can differentiate with non-negligible probability ϵ between the view produced by the hybrid experiments H_2 and H_3. Then this can be used in a black-box way by a PPT distinguisher $\mathsf{D}_{\mathsf{DEHE}}$ to violate the multi-instance selective security of DEHE for $m = \ell$, with the same advantage. This distinguisher $\mathsf{D}_{\mathsf{DEHE}}$ would work as follows:

- $\mathsf{D}_{\mathsf{DEHE}}$ gives a challenger ℓ and the size of the input $n + 2\ell_1$ (as defined in Algorithm 22) and receives the tuple $(\{\mathsf{ek}_i^*\}_{i\in[\ell]}, \mathsf{ct}_\alpha^*, \mathsf{ct}_\mathsf{k}^*, \mathsf{ct}_\mathsf{sk}^*, \mathsf{ct}_\Sigma^*, \{\mathsf{ct}_{j,0}^*, \mathsf{ct}_{j,1}^*\}_{j\in[\ell_1]})$.
- $\mathsf{D}_{\mathsf{DEHE}}$ then uses this tuple to complete the interaction in the hybrid experiment exactly as in H_2.
- Note that if $(\{\mathsf{ek}_i^*\}_{i\in[\ell]}, \mathsf{ct}_\alpha^*, \mathsf{ct}_\mathsf{k}^*, \mathsf{ct}_\mathsf{sk}^*, \mathsf{ct}_\Sigma^*, \{\mathsf{ct}_{j,0}^*, \mathsf{ct}_{j,1}^*\}_{j\in[\ell_1]}) \leftarrow \mathsf{Sim}_{\mathsf{DEHE}}$, the view produced would be identically distributed as H_2. Otherwise, this would be the same as that of H_3.
- Finally, $\mathsf{D}_{\mathsf{DEHE}}$ sends this view to $\mathsf{D}_\perp$ and outputs whatever it would output.

Note that in the above, the advantage of $\mathsf{D}_{\mathsf{DEHE}}$ is the same as that of $\mathsf{D}_\perp$, which is non-negligible. However, since DEHE is a secure DEHE scheme (Definition 1), no such $\mathsf{D}_{\mathsf{DEHE}}$ can exist, yielding a contradiction.

Figure 25 Hybrid Experiment H_2

Let $\mathsf{DEHE} = (\mathsf{KeyGen}, \mathsf{Gen}, \mathsf{Enc}, \mathsf{Eval})$ be a two-party DEHE scheme (Definition 1). Let $\mathcal{S} = (\mathsf{Gen}, \mathsf{Sign}, \mathsf{Verify})$ be a signature scheme (Definition 14), F be a PRF (Definition 12), $\Sigma = (\mathsf{Gen}, \mathsf{MAC}, \mathsf{Verify})$ be a MAC scheme (Definition 13), and Com be a deterministic commitment scheme (Definition 11).

H_2 first emulates $\mathcal{F}_{\mathsf{Setup}}$ and does the following:
- generate $(\mathsf{sk}, \mathsf{vk}) \leftarrow \mathcal{S}.\mathsf{Gen}(1^\kappa)$ and $\mathsf{sk}_\Sigma \leftarrow \Sigma.\mathsf{Gen}(1^\kappa)$
- sample $\mathsf{k}' \leftarrow \{0,1\}^{|\text{DEHE key}|}$ and compute $\mathsf{pk} = \mathsf{Com}(\mathsf{k}')$
- let $n = |\text{PRF key}| + |\text{DEHE key}| + |\mathsf{sk}| + |\mathsf{sk}_\Sigma|$
- compute $(\{\mathsf{ek}_i^*\}_{i\in[\ell]}, \mathsf{ct}_\alpha^*, \mathsf{ct}_\mathsf{k}^*, \mathsf{ct}_\mathsf{sk}^*, \mathsf{ct}_\Sigma^*, \{\mathsf{ct}_{j,0}^*, \mathsf{ct}_{j,1}^*\}_{j\in[\ell_1]}) \leftarrow \mathsf{Sim}_{\mathsf{DEHE}}(1^\kappa, n + 2\ell_1, \ell)$
- $\forall j \in [\ell_1], b \in \{0,1\}$, compute $\sigma_{b,j}^{\mathsf{ct}*} \leftarrow \mathcal{S}.\mathsf{Sign}(\mathsf{sk}, (j, b, \mathsf{ct}_{j,b}^*))$
- for $S_b^1 \in \mathcal{C}_1$, sample $\{\langle \mathsf{ek}_b^1 \rangle_j\}_{j\in[\ell_1]}$ such that $\mathsf{ek}_1^* = \sum_{j\in[\ell_1]} \langle \mathsf{ek}_b^1 \rangle_j$
- sample $\{\langle \mathsf{ek}_{\neg b}^1 \rangle_j\}_{j\in[\ell_1]}$ uniformly at random
- for $b' \in \{0,1\}, j \in [\ell_1]$, compute $\sigma_{b',j} = \mathcal{S}.\mathsf{Sign}(\mathsf{sk}, (j, b', \langle \mathsf{ek}_{b'}^1 \rangle_j))$
- $\mathcal{A}$ receives, for each corrupt $C_j \in \mathcal{C}_0$, the following tuple:

$$\{b' : \mathsf{ct}_{j,b'}^*, \sigma_{j,b'}^{\mathsf{ct}*}\}_{b'\in\{0,1\}}, \mathsf{vk}, \mathsf{pk}, \mathsf{ct}_\alpha^*, \mathsf{ct}_\mathsf{k}^*, \mathsf{ct}_\mathsf{sk}^*, \mathsf{ct}_\Sigma^*, \{\langle \mathsf{ek}_{b'}^1 \rangle_j, \sigma_{b',j}\}_{b'\in\{0,1\}}$$

H_2, emulating the first round of $\Pi_\perp$, does the following:
- For each $C_j \in \mathcal{H}_0$, using honest party's input $x_j \in \{0,1\}$, send to $\mathcal{A}$ the tuple: $\langle \mathsf{ek}_b^1 \rangle_j, \sigma_{b,j}, \mathsf{ct}_{j,x_j}^*, \sigma_{j,x_j}^{\mathsf{ct}*}, \mathsf{vk}, \mathsf{pk}, \mathsf{ct}_\alpha^*, \mathsf{ct}_\mathsf{k}^*, \mathsf{ct}_\mathsf{sk}^*, \mathsf{ct}_\Sigma^*$
- On behalf of $S_{\neg b}^1 \in \mathcal{H}_1$, accept from $\mathcal{A}$, for each $C_j \in \mathcal{C}_0$ the tuple, $\langle \mathsf{ek}_{\neg b}^1 \rangle_j, \sigma_{\neg b,j}, \mathsf{ct}_{j,x_j}, \sigma_{j,x_j}^{\mathsf{ct}}, \mathsf{vk}, \mathsf{pk}, \mathsf{ct}_\alpha^*, \mathsf{ct}_\mathsf{k}^*, \mathsf{ct}_\mathsf{sk}^*, \mathsf{ct}_\Sigma^*$ abort if any consistency checks given in $\Pi_\perp$ fail
- H_2, (emulating protocol Π_1) internally computes, $([\mathsf{ek}_0']_b, [\mathsf{ek}_1']_b, [y_1]_b, [\sigma_1]_b) \leftarrow \mathsf{DEHE.Eval}(b, \mathsf{ek}_1^*, (\mathsf{ct}_\alpha^*, \mathsf{ct}_\mathsf{k}^*, \mathsf{ct}_\Sigma^*, \mathsf{ct}_{y_0}^*), P_1(1))$ as the corrupt S_b^1 would, by setting $\mathsf{ct}_{y_0} = \{\mathsf{ct}_{i,x_i}^*\}_{C_i\in\mathcal{L}_0}$

H_2 samples $\{r_i\}_{i\in[\ell-1]}$ uniformly at random and does the following:
- for $i \in [\ell-1]$, compute $y_i = \begin{cases} \mathbf{C}_i(y_{i-1}) + r_i & \text{if } i = 1 \\ \mathbf{C}_i(y_{i-1} - r_{i-1}) + r_i & \text{if } i < \ell \\ \mathbf{C}_i(y_{i-1} - r_{i-1}) & \text{otherwise} \end{cases}$
- for $i \in [\ell-1]$, compute $\sigma_i = \Sigma.\mathsf{MAC}(\mathsf{sk}_\Sigma, (i, y_i))$

Claim. Assuming Σ is a MAC scheme (Definition 14) satisfying unforgeability, $\mathsf{H}_3 \overset{\text{c}}{\approx} \mathsf{H}_4$.

Proof. Recall that the hybrid experiment H_4 (Algorithms 29 and 30) differs from H_3 (Algorithms 27 and 28) only in that the MAC consistency checks within the RMS programs are performed honestly within H_4, similar to the real execution, where the protocol does not abort even if the MACs do not match the internally computed messages by the hybrid experiment. In H_3 no such checks are performed and the protocol only proceeds without aborting if the messages y_i and corresponding MACs σ_i are the same as that computed by the hybrid experiment. The only cases where H_3 aborts but H_4 does not would correspond to an instance where the adversary has forged a MAC tag on a message of its choice for the secret key sk_Σ, and has provided correct additive shares of the same, causing it to verify correctly on the honest party's end.

We can show that both experiments will produce exactly the same view, except with negligible probability by reduction to the unforgeability property of the underlying MAC scheme Σ (Definition 13). Consider for the sake of contradiction that there exists a PPT adversary $\mathsf{D}_\perp$ that can, with non-negligible probability ϵ, produce a

Figure 26 Hybrid Experiment H_2 – continued

Let $\mathsf{DEHE} = (\mathsf{KeyGen}, \mathsf{Gen}, \mathsf{Enc}, \mathsf{Eval})$ be a two-party DEHE scheme (Definition 1). Let $\mathcal{S} = (\mathsf{Gen}, \mathsf{Sign}, \mathsf{Verify})$ be a signature scheme (Definition 14), F be a PRF (Definition 12), $\Sigma = (\mathsf{Gen}, \mathsf{MAC}, \mathsf{Verify})$ be a MAC scheme (Definition 13), and Com be a deterministic commitment scheme (Definition 11).

$\forall i \in [\ell - 1]$, H_2, emulating the ith computation round of $\Pi_\perp$, does the following:

- On behalf of $S^i_{\neg b} \in \mathcal{H}_i$, do the following:
  - compute $[y_i]_{\neg b} = y_i - [y_i]_b$ and $[\sigma_i]_{\neg b} = \sigma_i - [\sigma_i]_b$
  - for $S^{i+1}_{b'} \in \mathcal{C}_{i+1}$, set $[\mathsf{ek}_{b'}]_{\neg b} = \mathsf{ek}^*_i - [\mathsf{ek}'_{b'}]_b$ and $[\mathsf{ek}_{\neg b'}]_{\neg b} = \mathsf{k}' - \mathsf{ek}^*_i - [\mathsf{ek}'_{\neg b'}]_b$
  - compute $\mathsf{pk}_{b',\neg b} = \mathsf{Com}([\mathsf{ek}_{b'}]_{\neg b})$ and $\mathsf{pk}_{\neg b',\neg b} = \mathsf{Com}([\mathsf{ek}_{\neg b'}]_{\neg b})$
  - abort if any **consistency checks** given in Π_i fail

  send $([y_i]_{\neg b}, [\sigma_i]_{\neg b}, [\mathsf{ek}_{b'}]_{\neg b}, \mathsf{pk}_{b',\neg b}, \mathsf{pk}_{\neg b',\neg b}, \mathsf{vk}, \mathsf{pk}, \mathsf{ct}^*_\alpha, \mathsf{ct}^*_\mathsf{k}, \mathsf{ct}^*_\mathsf{sk}, \mathsf{ct}^*_\Sigma)$ to $\mathcal{A}$
- H_2 accepts $([y_i]_b, [\sigma_i]_b, [\mathsf{ek}'_{\neg b'}]_b, \mathsf{pk}_{b',b}, \mathsf{pk}_{\neg b',b}, \mathsf{vk}, \mathsf{pk}, \mathsf{ct}^*_\alpha, \mathsf{ct}^*_\mathsf{k}, \mathsf{ct}^*_\mathsf{sk}, \mathsf{ct}^*_\Sigma)$ from $\mathcal{A}$
- H_2, emulating protocol Π_{i+1}, computes, $([\mathsf{ek}'_0]_{b'}, [\mathsf{ek}'_1]_{b'}, [y_{i+1}]_{b'}, [\sigma_{i+1}]_{b'}) \leftarrow \mathsf{DEHE.Eval}(b', \mathsf{ek}^*_{i+1}, (\mathsf{ct}^*_\alpha, \mathsf{ct}^*_\mathsf{k}, \mathsf{ct}^*_\Sigma), P_{i+1}(i+1, y_i, \sigma_i))$ as $S^{i+1}_{b'}$ would

Before the final round, H_2 sets $y_\ell = f(x)$ and $\sigma_\ell = \mathcal{S}.\mathsf{Sign}(\mathsf{sk}, (\ell, y_\ell))$.

- on behalf of $S^\ell_{\neg b} \in \mathcal{H}_\ell$, do the following:
  - compute $([y_\ell]_{b'}, [\sigma_\ell]_{b'}) \leftarrow \mathsf{DEHE.Eval}(b', \mathsf{ek}^*_\ell, (\mathsf{ct}^*_\alpha, \mathsf{ct}^*_\mathsf{k}, \mathsf{ct}^*_\mathsf{sk}), P_\ell(\ell, y_{\ell-1}, \sigma_{\ell-1}))$
  - compute $[y_\ell]_{\neg b} = y_\ell - [y_\ell]_b$ and $[\sigma_\ell]_{\neg b} = \sigma_\ell - [\sigma_\ell]_b$

  send $([y_i]_{\neg b}, [\sigma_i]_{\neg b}, \mathsf{vk})$ to $\mathcal{A}$
- $\forall P^{\ell+1}_j \in \mathcal{H}_{i+1}$, H_2 accepts $([y_i]_b, [\sigma_i]_b, \mathsf{vk})$ from $\mathcal{A}$ and, if **consistency checks** given in Π_ℓ fail, abort the protocol.

view distributed as in H_4 but not in H_3. We show that if this were possible, then $\mathsf{D}_\perp$ can be used in a black-box way by a PPT adversary D_Σ to violate the unforgeability property of the MAC scheme Σ. This distinguisher D_Σ would work as follows:

- D_Σ has oracle access to a MAC algorithm $\mathcal{O}_\Sigma$ parameterized with the corresponding secret MAC key sk_Σ. It uses this to create MAC tags for messages and starts interacting with $\mathsf{D}_\perp$ exactly as given in the simulation H_3 in Algorithms 27 and 28.
- If for any message that $\mathsf{D}_\perp$ sends, it is not the case that it equals the message that D_Σ has internally computed, reconstruct the value m and corresponding MAC tag σ using both the honest and corrupt party's additive shares. Output (m, σ) and continue the protocol as in H_3 (Algorithms 27 and 28). Otherwise, abort the protocol.

Note that in the above, if the protocol has aborted without D_Σ outputting a forgery (m, σ), the view of $\mathsf{D}_\perp$ is the same as that of H_3. Otherwise, it is the same as that in the experiment H_4. The advantage of D_Σ is the same as that of $\mathsf{D}_\perp$ not inducing the view of H_3, which is non-negligible by assumption. However, since Σ is an unforgeable MAC scheme (Definition 13), no such D_Σ can exist, yielding a contradiction.

Claim. Assuming Com is a deterministic commitment scheme (Definition 11), $\mathsf{H}_4 \overset{c}{\approx} \mathsf{H}_5$.

Proof. Consider for the sake of contradiction that there exists a PPT distinguisher $\mathsf{D}_\perp$ that can differentiate with non-negligible probability ϵ between the view produced by the hybrid experiments H_4 and H_5. Then this can be used in a black-box way by a PPT distinguisher D_Com to distinguish between having received commitments $\mathsf{pk}, \{\mathsf{pk}^i_{a,b}\}_{i \in [\ell], a, b \in \{0,1\}}$ of values computed as in H_4 and H_5 with the same advantage.

Figure 27 Hybrid Experiment H_3

Let $\mathsf{DEHE} = (\mathsf{KeyGen}, \mathsf{Gen}, \mathsf{Enc}, \mathsf{Eval})$ be a two-party DEHE scheme (Definition 1). Let $\mathcal{S} = (\mathsf{Gen}, \mathsf{Sign}, \mathsf{Verify})$ be a signature scheme (Definition 14), F be a PRF (Definition 12), $\Sigma = (\mathsf{Gen}, \mathsf{MAC}, \mathsf{Verify})$ be a MAC scheme (Definition 13), and Com be a deterministic commitment scheme (Definition 11).

H_3 first emulates $\mathcal{F}_{\mathsf{Setup}}$ and does the following:

- generate $(\mathsf{sk}, \mathsf{vk}) \leftarrow \mathcal{S}.\mathsf{Gen}(1^\kappa)$, $\mathsf{sk}_\Sigma \leftarrow \Sigma.\mathsf{Gen}(1^\kappa)$ and sample $\alpha \leftarrow \{0,1\}^\kappa$
- generate $\mathsf{k} \leftarrow \mathsf{DEHE.KeyGen}(1^\kappa)$ and, for $i \in [\ell]$, $(\mathsf{ek}_0^i, \mathsf{ek}_1^i) \leftarrow \mathsf{DEHE.Gen}(\mathsf{k})$
- compute $(\mathsf{ct}_\alpha^*, \mathsf{ct}_\mathsf{k}^*, \mathsf{ct}_\mathsf{sk}^*, \mathsf{ct}_\Sigma^*) \leftarrow \mathsf{DEHE.Enc}(\mathsf{k}, (\alpha, \mathsf{k}, \mathsf{sk}, \mathsf{sk}_\Sigma))$
- sample $\mathsf{k}' \leftarrow \{0,1\}^{|\text{DEHE key}|}$ and compute $\mathsf{pk} = \mathsf{Com}(\mathsf{k}')$
- $\forall j \in [\ell_1]$, $b \in \{0,1\}$, compute $\mathsf{ct}_{j,b}^* \leftarrow \mathsf{DEHE.Enc}(\mathsf{k}, b)$ and $\sigma_{b,j}^{\mathsf{ct}*} \leftarrow \mathcal{S}.\mathsf{Sign}(\mathsf{sk}, (j, b, \mathsf{ct}_{j,b}^*))$
- for $b \in \{0,1\}$, sample $\{\langle \mathsf{ek}_b^1 \rangle_j\}_{j \in [\ell_1]}$ such that $\mathsf{ek}_b^1 = \sum_{j \in [\ell_1]} \langle \mathsf{ek}_b^1 \rangle_j$
- for $b \in \{0,1\}, j \in [\ell_1]$, compute $\sigma_{b,j} = \mathcal{S}.\mathsf{Sign}(\mathsf{sk}, (j, b, \langle \mathsf{ek}_b^1 \rangle_j))$
- $\forall i \in [\ell], S_b^i \in \mathcal{C}_i$, set $\mathsf{ek}_i^* = \mathsf{ek}_b^i$
- $\mathcal{A}$ receives, for each corrupt $C_j \in \mathcal{C}_0$, the following tuple:

$$\{b' : \mathsf{ct}_{j,b'}^*, \sigma_{j,b'}^{\mathsf{ct}*}\}_{b' \in \{0,1\}}, \mathsf{vk}, \mathsf{pk}, \mathsf{ct}_\alpha^*, \mathsf{ct}_\mathsf{k}^*, \mathsf{ct}_\mathsf{sk}^*, \mathsf{ct}_\Sigma^*, \{\langle \mathsf{ek}_{b'}^1 \rangle_j, \sigma_{b',j}\}_{b' \in \{0,1\}}$$

H_3, emulating the first round of $\Pi_\perp$, does the following:

- For each $C_j \in \mathcal{H}_0$, using honest party's input $x_j \in \{0,1\}$, send to $\mathcal{A}$ the tuple: $\langle \mathsf{ek}_b^1 \rangle_j, \sigma_{b,j}, \mathsf{ct}_{j,x_j}^*, \sigma_{j,x_j}^{\mathsf{ct}*}, \mathsf{vk}, \mathsf{pk}, \mathsf{ct}_\alpha^*, \mathsf{ct}_\mathsf{k}^*, \mathsf{ct}_\mathsf{sk}^*, \mathsf{ct}_\Sigma^*$
- On behalf of $S_{\neg b}^1 \in \mathcal{H}_1$, accept from $\mathcal{A}$, for each $C_j \in \mathcal{C}_0$ the tuple, $\langle \mathsf{ek}_{\neg b}^1 \rangle_j, \sigma_{\neg b,j}, \mathsf{ct}_{j,x_j}, \sigma_{j,x_j}^{\mathsf{ct}}, \mathsf{vk}, \mathsf{pk}, \mathsf{ct}_\alpha^*, \mathsf{ct}_\mathsf{k}^*, \mathsf{ct}_\mathsf{sk}^*, \mathsf{ct}_\Sigma^*$ abort if any consistency checks given in $\Pi_\perp$ fail
- H_3, (emulating protocol Π_1) internally computes, $([\mathsf{ek}_0']_b, [\mathsf{ek}_1']_b, [y_1]_b, [\sigma_1]_b) \leftarrow \mathsf{DEHE.Eval}(b, \mathsf{ek}_1^*, (\mathsf{ct}_\alpha^*, \mathsf{ct}_\mathsf{k}^*, \mathsf{ct}_\Sigma^*, \mathsf{ct}_{y_0}^*), P_1(1))$ as the corrupt S_b^1 would, by setting $\mathsf{ct}_{y_0} = \{\mathsf{ct}_{i,x_i}^*\}_{C_i \in \mathcal{L}_0}$

H_3 samples $\{r_i\}_{i \in [\ell-1]}$ uniformly at random and does the following:

- for $i \in [\ell - 1]$, compute $y_i = \begin{cases} \mathbf{C}_i(y_{i-1}) + r_i & \text{if } i = 1 \\ \mathbf{C}_i(y_{i-1} - r_{i-1}) + r_i & \text{if } i < \ell \\ \mathbf{C}_i(y_{i-1} - r_{i-1}) & \text{otherwise} \end{cases}$
- for $i \in [\ell - 1]$, compute $\sigma_i = \Sigma.\mathsf{MAC}(\mathsf{sk}_\Sigma, (i, y_i))$

Note that in both experiments the distributions induced for all of the messages committed is the uniform distribution. This distinguisher $\mathsf{D}_{\mathsf{Com}}$ would work as follows:

- Let $\mathcal{C}$ be a challenger that has internally sampled a bit $b \leftarrow \{0,1\}$.
- $\mathsf{D}_{\mathsf{Com}}$ starts generating the view as described in H_4 and has sampled both k and k'.
- For each time that the view needs to include a deterministic commitment, $\mathsf{D}_{\mathsf{Com}}$ sends messages m_0 and m_1 (generated as a function of k and k' respectively as in H_4 and H_5) to $\mathcal{C}$ and receives a $\mathsf{pk} = \mathsf{Com}(m_b)$. This is included as part of the view and $\mathsf{D}_{\mathsf{Com}}$ continues generating the view.
- Finally, $\mathsf{D}_{\mathsf{Com}}$ sends this view to $\mathsf{D}_\perp$ and outputs whatever it would output.

Note that in the above, the advantage of $\mathsf{D}_{\mathsf{Com}}$ is the same as that of $\mathsf{D}_\perp$, which is non-negligible. However, since Com is a deterministic commitment scheme satisfying computational hiding for randomly chosen messages (Definition 11), no such $\mathsf{D}_{\mathsf{Com}}$ can exist, yielding a contradiction.

Claim. Assuming F is a PRF (Definition 12), $\mathsf{H}_5 \overset{\text{c}}{\approx} \mathsf{H}_6$.

Figure 28 Hybrid Experiment H_3 – continued

Let $\mathsf{DEHE} = (\mathsf{KeyGen}, \mathsf{Gen}, \mathsf{Enc}, \mathsf{Eval})$ be a two-party DEHE scheme (Definition 1). Let $\mathcal{S} = (\mathsf{Gen}, \mathsf{Sign}, \mathsf{Verify})$ be a signature scheme (Definition 14), F be a PRF (Definition 12), $\Sigma = (\mathsf{Gen}, \mathsf{MAC}, \mathsf{Verify})$ be a MAC scheme (Definition 13), and Com be a deterministic commitment scheme (Definition 11).

$\forall i \in [\ell - 1]$, H_3, emulating the ith computation round of $\Pi_\perp$, does the following:

- On behalf of $S^i_{\neg b} \in \mathcal{H}_i$, do the following:
  - compute $[y_i]_{\neg b} = y_i - [y_i]_b$ and $[\sigma_i]_{\neg b} = \sigma_i - [\sigma_i]_b$
  - for $S^{i+1}_{b'} \in \mathcal{C}_{i+1}$, set $[\mathsf{ek}_{b'}]_{\neg b} = \mathsf{ek}^*_i - [\mathsf{ek}'_{b'}]_b$ and $[\mathsf{ek}_{\neg b'}]_{\neg b} = \mathsf{k}' - \mathsf{ek}^*_i - [\mathsf{ek}'_{\neg b'}]_b$
  - compute $\mathsf{pk}_{b',\neg b} = \mathsf{Com}([\mathsf{ek}_{b'}]_{\neg b})$ and $\mathsf{pk}_{\neg b',\neg b} = \mathsf{Com}([\mathsf{ek}_{\neg b'}]_{\neg b})$
  - abort if any **consistency checks** given in Π_i fail

  send $([y_i]_{\neg b}, [\sigma_i]_{\neg b}, [\mathsf{ek}_{b'}]_{\neg b}, \mathsf{pk}_{b',\neg b}, \mathsf{pk}_{\neg b',\neg b}, \mathsf{vk}, \mathsf{pk}, \mathsf{ct}^*_\alpha, \mathsf{ct}^*_\mathsf{k}, \mathsf{ct}^*_\mathsf{sk}, \mathsf{ct}^*_\Sigma)$ to $\mathcal{A}$
- H_3 accepts $([y_i]_b, [\sigma_i]_b, [\mathsf{ek}'_{\neg b'}]_b, \mathsf{pk}_{b',b}, \mathsf{pk}_{\neg b',b}, \mathsf{vk}, \mathsf{pk}, \mathsf{ct}^*_\alpha, \mathsf{ct}^*_\mathsf{k}, \mathsf{ct}^*_\mathsf{sk}, \mathsf{ct}^*_\Sigma)$ from $\mathcal{A}$
- H_3, emulating protocol Π_{i+1}, computes, $([\mathsf{ek}'_0]_{b'}, [\mathsf{ek}'_1]_{b'}, [y_{i+1}]_{b'}, [\sigma_{i+1}]_{b'}) \leftarrow \mathsf{DEHE.Eval}(b', \mathsf{ek}^*_{i+1}, (\mathsf{ct}^*_\alpha, \mathsf{ct}^*_\mathsf{k}, \mathsf{ct}^*_\Sigma), P_{i+1}(i+1, y_i, \sigma_i))$ as $S^{i+1}_{b'}$ would

Before the final round, H_3 sets $y_\ell = f(x)$ and $\sigma_\ell = \mathcal{S}.\mathsf{Sign}(\mathsf{sk}, (\ell, y_\ell))$.

- on behalf of $S^\ell_{\neg b} \in \mathcal{H}_\ell$, do the following:
  - compute $([y_\ell]_{b'}, [\sigma_\ell]_{b'}) \leftarrow \mathsf{DEHE.Eval}(b', \mathsf{ek}^*_\ell, (\mathsf{ct}^*_\alpha, \mathsf{ct}^*_\mathsf{k}, \mathsf{ct}^*_\mathsf{sk}), P_\ell(\ell, y_{\ell-1}, \sigma_{\ell-1}))$
  - compute $[y_\ell]_{\neg b} = y_\ell - [y_\ell]_b$ and $[\sigma_\ell]_{\neg b} = \sigma_\ell - [\sigma_\ell]_b$

  send $([y_i]_{\neg b}, [\sigma_i]_{\neg b}, \mathsf{vk})$ to $\mathcal{A}$
- $\forall P^{\ell+1}_j \in \mathcal{H}_{i+1}$, H_3 accepts $([y_i]_b, [\sigma_i]_b, \mathsf{vk})$ from $\mathcal{A}$ and, if **consistency checks** given in Π_ℓ fail, abort the protocol.

Proof. Consider for the sake of contradiction that there exists a PPT distinguisher $\mathsf{D}_\perp$ that can differentiate with non-negligible probability ϵ between the view produced by the hybrid experiments H_5 and H_6. Then this can be used in a black-box way by a PPT distinguisher $\mathsf{D}_{\mathsf{PRF}}$ to distinguish between having received oracle access to a PRF F keyed with a random key α, and a random function R with the same advantage. This distinguisher $\mathsf{D}_{\mathsf{PRF}}$ would work as follows:

- Let $\mathcal{O}$ be the oracle that $\mathsf{D}_{\mathsf{PRF}}$ has access to.
- For each $i \in [4\ell]$, $\mathsf{D}_{\mathsf{PRF}}$ queries $\mathcal{O}(i, 1)$ and $\mathcal{O}(i, 2)$, and uses its output to generate the rest of the view exactly as in H_5.
- Note that if $\mathcal{O} = R$, the view produced would be identically distributed as H_5. Otherwise, when $\mathcal{O} = F_\alpha$, this would be the same as that of H_6.
- Finally, $\mathsf{D}_{\mathsf{PRF}}$ sends this view to $\mathsf{D}_\perp$ and outputs whatever it would output.

Note that in the above, the advantage of $\mathsf{D}_{\mathsf{PRF}}$ is the same as that of $\mathsf{D}_\perp$, which is non-negligible. However, since F is a secure pseudo-random function (Definition 12), no such $\mathsf{D}_{\mathsf{PRF}}$ can exist, yielding a contradiction.

C Proof of Theorem 3

Proof. We need to show that the protocol presented in Algorithm 5 realizes the required layered functionality f in the presence of a semi-honest $\mathcal{A}$ that can corrupt all-but-one party in $\mathcal{L}_0$ and $\mathcal{L}_{\ell+1}$, and one party in each intermediate layer $\mathcal{L}_i$, for all $i \in [\ell]$. To prove security, we design a simulator Sim_{sh} that simulates the view produced by an interaction with a PPT semi-honest adversary $\mathcal{A}$ corrupting all-but-one parties in the input and output layer and one party in each intermediate layer.

Figure 29 Hybrid Experiment H_4

Let $\mathsf{DEHE} = (\mathsf{KeyGen}, \mathsf{Gen}, \mathsf{Enc}, \mathsf{Eval})$ be a two-party DEHE scheme (Definition 1). Let $\mathcal{S} = (\mathsf{Gen}, \mathsf{Sign}, \mathsf{Verify})$ be a signature scheme (Definition 14), F be a PRF (Definition 12), $\Sigma = (\mathsf{Gen}, \mathsf{MAC}, \mathsf{Verify})$ be a MAC scheme (Definition 13), and Com be a deterministic commitment scheme (Definition 11).

H_4 first emulates $\mathcal{F}_{\mathsf{Setup}}$ and does the following:

- generate $(\mathsf{sk}, \mathsf{vk}) \leftarrow \mathcal{S}.\mathsf{Gen}(1^\kappa)$, $\mathsf{sk}_\Sigma \leftarrow \Sigma.\mathsf{Gen}(1^\kappa)$ and sample $\alpha \leftarrow \{0,1\}^\kappa$
- generate $\mathsf{k} \leftarrow \mathsf{DEHE.KeyGen}(1^\kappa)$ and, for $i \in [\ell]$, $(\mathsf{ek}_0^i, \mathsf{ek}_1^i) \leftarrow \mathsf{DEHE.Gen}(\mathsf{k})$
- compute $(\mathsf{ct}_\alpha^*, \mathsf{ct}_\mathsf{k}^*, \mathsf{ct}_\mathsf{sk}^*, \mathsf{ct}_\Sigma^*) \leftarrow \mathsf{DEHE.Enc}(\mathsf{k}, (\alpha, \mathsf{k}, \mathsf{sk}, \mathsf{sk}_\Sigma))$
- sample $\mathsf{k}' \leftarrow \{0,1\}^{|\text{DEHE key}|}$ and compute $\mathsf{pk} = \mathsf{Com}(\mathsf{k}')$
- $\forall j \in [\ell_1]$, $b \in \{0,1\}$, compute $\mathsf{ct}_{j,b}^* \leftarrow \mathsf{DEHE.Enc}(\mathsf{k}, b)$ and $\sigma_{b,j}^{\mathsf{ct}*} \leftarrow \mathcal{S}.\mathsf{Sign}(\mathsf{sk}, (j, b, \mathsf{ct}_{j,b}^*))$
- for $b \in \{0,1\}$, sample $\{\langle \mathsf{ek}_b^1 \rangle_j\}_{j \in [\ell_1]}$ such that $\mathsf{ek}_b^1 = \sum_{j \in [\ell_1]} \langle \mathsf{ek}_b^1 \rangle_j$
- for $b \in \{0,1\}, j \in [\ell_1]$, compute $\sigma_{b,j} = \mathcal{S}.\mathsf{Sign}(\mathsf{sk}, (j, b, \langle \mathsf{ek}_b^1 \rangle_j))$
- $\forall i \in [\ell], S_b^i \in \mathcal{C}_i$, set $\mathsf{ek}_i^* = \mathsf{ek}_b^i$
- $\mathcal{A}$ receives, for each corrupt $C_j \in \mathcal{C}_0$, the following tuple:

$$\{b' : \mathsf{ct}_{j,b'}^*, \sigma_{j,b'}^{\mathsf{ct}*}\}_{b' \in \{0,1\}}, \mathsf{vk}, \mathsf{pk}, \mathsf{ct}_\alpha^*, \mathsf{ct}_\mathsf{k}^*, \mathsf{ct}_\mathsf{sk}^*, \mathsf{ct}_\Sigma^*, \{\langle \mathsf{ek}_{b'}^1 \rangle_j, \sigma_{b',j}\}_{b' \in \{0,1\}}$$

H_4, emulating the first round of $\Pi_\perp$, does the following:

- For each $C_j \in \mathcal{H}_0$, using honest party's input $x_j \in \{0,1\}$, send to $\mathcal{A}$ the tuple: $\langle \mathsf{ek}_b^1 \rangle_j, \sigma_{b,j}, \mathsf{ct}_{j,x_j}^*, \sigma_{j,x_j}^{\mathsf{ct}*}, \mathsf{vk}, \mathsf{pk}, \mathsf{ct}_\alpha^*, \mathsf{ct}_\mathsf{k}^*, \mathsf{ct}_\mathsf{sk}^*, \mathsf{ct}_\Sigma^*$
- On behalf of $S_{\neg b}^1 \in \mathcal{H}_1$, accept from $\mathcal{A}$, for each $C_j \in \mathcal{C}_0$ the tuple, $\langle \mathsf{ek}_{\neg b}^1 \rangle_j, \sigma_{\neg b,j}, \mathsf{ct}_{j,x_j}, \sigma_{j,x_j}^{\mathsf{ct}}, \mathsf{vk}, \mathsf{pk}, \mathsf{ct}_\alpha^*, \mathsf{ct}_\mathsf{k}^*, \mathsf{ct}_\mathsf{sk}^*, \mathsf{ct}_\Sigma^*$ abort if any consistency checks given in $\Pi_\perp$ fail
- H_4, (emulating protocol Π_1) internally computes, $([\mathsf{ek}_0']_b, [\mathsf{ek}_1']_b, [y_1]_b, [\sigma_1]_b) \leftarrow \mathsf{DEHE.Eval}(b, \mathsf{ek}_1^*, (\mathsf{ct}_\alpha^*, \mathsf{ct}_\mathsf{k}^*, \mathsf{ct}_\Sigma^*, \mathsf{ct}_{y_0}^*), P_1(1))$ as the corrupt S_b^1 would, by setting $\mathsf{ct}_{y_0} = \{\mathsf{ct}_{i,x_i}^*\}_{C_i \in \mathcal{L}_0}$

H_4 samples $\{r_i\}_{i \in [\ell-1]}$ uniformly at random and does the following:

- for $i \in [\ell-1]$, compute $y_i = \begin{cases} \mathbf{C}_i(y_{i-1}) + r_i & \text{if } i = 1 \\ \mathbf{C}_i(y_{i-1} - r_{i-1}) + r_i & \text{if } i < \ell \\ \mathbf{C}_i(y_{i-1} - r_{i-1}) & \text{otherwise} \end{cases}$
- for $i \in [\ell-1]$, compute $\sigma_i = \Sigma.\mathsf{MAC}(\mathsf{sk}_\Sigma, (i, y_i))$

For each $i \in [\ell+1] \cup \{0\}$ let $\mathcal{H}_i \subseteq \mathcal{L}_i$ be the set of honest parties in layer $\mathcal{L}_i$. Let $\mathcal{C}_i = \mathcal{L}_i - \mathcal{H}_i$ be the set of corrupt parties in this layer. Note that the simulator Sim_{sh} receives as input, the input of the semi-honest corrupt parties in input layer $\mathcal{L}_0$, namely $\{x_1[j]\}_{j \in \mathcal{C}_0}$, as well as the output $y_\ell = f(\{x_1[j]\}_{j \in \ell_1})$.

Adaptive Security through Secure Erasures. In the layered setting where each party only sends one message, our protocol supports adaptive security using secure erasures: each party participating in the protocol computes the messages it needs to send, then erases its state of the protocol before sending the messages to the next layer as specified. This makes it such that when an adaptive adversary corrupts a party that has already sent its message, it does not gain access to that party's state in the protocol. Therefore, it suffices to design protocols that are only selectively secure and these can be lifted to adaptive security in the layered setting using secure erasures.

Going forward, we present a proof that the protocol Π_{sh} in Algorithm 5 is selectively secure in the presence of a semi-honest rushing adversary $\mathcal{A}$ that can corrupt all-but-one party in $\mathcal{L}_0$ and $\mathcal{L}_{\ell+1}$, and one party in each intermediate layer $\mathcal{L}_i$, for all $i \in [\ell]$. The proof for this follows from a PPT simulation algorithm Sim_{sh}

Figure 30 Hybrid Experiment H_4 – continued

Let $\mathsf{DEHE} = (\mathsf{KeyGen}, \mathsf{Gen}, \mathsf{Enc}, \mathsf{Eval})$ be a two-party DEHE scheme (Definition 1). Let $\mathcal{S} = (\mathsf{Gen}, \mathsf{Sign}, \mathsf{Verify})$ be a signature scheme (Definition 14), F be a PRF (Definition 12), $\Sigma = (\mathsf{Gen}, \mathsf{MAC}, \mathsf{Verify})$ be a MAC scheme (Definition 13), and Com be a deterministic commitment scheme (Definition 11).

$\forall i \in [\ell - 1]$, H_4, emulating the ith computation round of $\Pi_\perp$, does the following:

- On behalf of $S^i_{\neg b} \in \mathcal{H}_i$, do the following:
  - compute $[y_i]_{\neg b} = y_i - [y_i]_b$ and $[\sigma_i]_{\neg b} = \sigma_i - [\sigma_i]_b$
  - for $S^{i+1}_{b'} \in \mathcal{C}_{i+1}$, set $[\mathsf{ek}_{b'}]_{\neg b} = \mathsf{ek}^*_i - [\mathsf{ek}'_{b'}]_b$ and $[\mathsf{ek}_{\neg b'}]_{\neg b} = \mathsf{k} - \mathsf{ek}^*_i - [\mathsf{ek}'_{\neg b'}]_b$
  - compute $\mathsf{pk}_{b',\neg b} = \mathsf{Com}([\mathsf{ek}_{b'}]_{\neg b})$ and $\mathsf{pk}_{\neg b',\neg b} = \mathsf{Com}([\mathsf{ek}_{\neg b'}]_{\neg b})$
  - abort if any **consistency checks** given in Π_i fail

  send $([y_i]_{\neg b}, [\sigma_i]_{\neg b}, [\mathsf{ek}_{b'}]_{\neg b}, \mathsf{pk}_{b',\neg b}, \mathsf{pk}_{\neg b',\neg b}, \mathsf{vk}, \mathsf{pk}, \mathsf{ct}^*_\alpha, \mathsf{ct}^*_\mathsf{k}, \mathsf{ct}^*_\mathsf{sk}, \mathsf{ct}^*_\Sigma)$ to $\mathcal{A}$
- H_4 accepts $([y_i]^*_b, [\sigma_i]^*_b, [\mathsf{ek}'_{\neg b'}]_b, \mathsf{pk}_{b',b}, \mathsf{pk}_{\neg b',b}, \mathsf{vk}, \mathsf{pk}, \mathsf{ct}^*_\alpha, \mathsf{ct}^*_\mathsf{k}, \mathsf{ct}^*_\mathsf{sk}, \mathsf{ct}^*_\Sigma)$ from $\mathcal{A}$
- H_4 computes $y^*_i = [y_i]^*_b + [y_i]_{\neg b}$ and $\sigma^*_i = [\sigma_i]^*_b + [\sigma_i]_{\neg b}$. If $\top \neq \Sigma.\mathsf{Verify}(\mathsf{sk}_\Sigma, \sigma^*_i, y^*_i)$, then abort the protocol
- H_4, emulating protocol Π_{i+1}, computes, $([\mathsf{ek}'_0]_{b'}, [\mathsf{ek}'_1]_{b'}, [y_{i+1}]_{b'}, [\sigma_{i+1}]_{b'}) \leftarrow \mathsf{DEHE.Eval}(b', \mathsf{ek}^*_{i+1}, (\mathsf{ct}^*_\alpha, \mathsf{ct}^*_\mathsf{k}, \mathsf{ct}^*_\Sigma), P_{i+1}(i+1, y_i, \sigma_i))$ as $S^{i+1}_{b'}$ would

Before the final round, H_4 sets $y_\ell = f(x)$ and $\sigma_\ell = \mathcal{S}.\mathsf{Sign}(\mathsf{sk}, (\ell, y_\ell))$.

- on behalf of $S^\ell_{\neg b} \in \mathcal{H}_\ell$, do the following:
  - compute $([y_\ell]_{b'}, [\sigma_\ell]_{b'}) \leftarrow \mathsf{DEHE.Eval}(b', \mathsf{ek}^*_\ell, (\mathsf{ct}^*_\alpha, \mathsf{ct}^*_\mathsf{k}, \mathsf{ct}^*_\mathsf{sk}), P_\ell(\ell, y_{\ell-1}, \sigma_{\ell-1}))$
  - compute $[y_\ell]_{\neg b} = y_\ell - [y_\ell]_b$ and $[\sigma_\ell]_{\neg b} = \sigma_\ell - [\sigma_\ell]_b$

  send $([y_i]_{\neg b}, [\sigma_i]_{\neg b}, \mathsf{vk})$ to $\mathcal{A}$
- $\forall P^{\ell+1}_j \in \mathcal{H}_{i+1}$, H_4 accepts $([y_i]_b, [\sigma_i]_b, \mathsf{vk})$ from $\mathcal{A}$ and, if **consistency checks** given in Π_ℓ fail, abort the protocol.

that we define for the protocol Π_{sh}. Before detailing the simulator itself, let us first formalize essential notation below.

Notation. Let $\mathcal{F}_\mathsf{Setup}$ be the functionality given in Algorithm 1. Since we show that Π_{sh} realizes f in the $\mathcal{F}_\mathsf{Setup}$-hybrid, our simulator Sim_{sh} needs to also internally emulate $\mathcal{F}_\mathsf{Setup}$ and the messages that the adversary needs to receive from it. Let $\mathsf{DEHE} = (\mathsf{KeyGen}, \mathsf{Gen}, \mathsf{Enc}, \mathsf{Eval})$ be a two-party DEHE scheme (Definition 1). With respect to this, let $\mathsf{Sim}_\mathsf{DEHE}(\cdot)$ be the PPT simulator for DEHE that, for $n, \ell \in \mathbb{N}$, outputs

$$(\{\mathsf{ek}_i\}_{i\in[\ell]}, \{\mathsf{ct}_i\}_{i\in[n]}) \leftarrow \mathsf{Sim}_\mathsf{DEHE}(1^\kappa, n, \ell)$$

where n is the DEHE input size, and ℓ is the number of layers in the computation.

Simulation. The formal description of the simulator Sim_{sh} is described in Algorithm 33. It uses the DEHE simulator $\mathsf{Sim}_\mathsf{DEHE}$ mentioned above in a black-box way.

Hybrids. It remains to prove that the joint view produced by the simulator Sim_{sh} is computationally indistinguishable from that of the real execution of the protocol. We do so by considering the following intermediate hybrid experiments:

- H_0 : This hybrid experiment is defined to be the same as the interaction of the adversary $\mathcal{A}$ in the simulation with Sim_{sh}.
- H_1 : This is an intermediate hybrid experiment that differs from the previous hybrid in the following way: the honest parties' inputs $\{x_j\}_{C_j \in \mathcal{H}_0}$ are used, along with the corrupt parties' inputs $\{x^*_j\}_{C_j \in \mathcal{C}_0}$, to compute the intermediate computation values $\{y_i\}_{i\in[\ell]}$ using the masks $\{r_i\}_{i\in[\ell-1]}$ – like in the real execution. The masks $\{r_i\}_{i\in[\ell-1]}$ are sampled uniformly at random and these honest parties'

Figure 31 Hybrid Experiment H_5

Let $\mathsf{DEHE} = (\mathsf{KeyGen}, \mathsf{Gen}, \mathsf{Enc}, \mathsf{Eval})$ be a two-party DEHE scheme (Definition 1). Let $\mathcal{S} = (\mathsf{Gen}, \mathsf{Sign}, \mathsf{Verify})$ be a signature scheme (Definition 14), F be a PRF (Definition 12), $\Sigma = (\mathsf{Gen}, \mathsf{MAC}, \mathsf{Verify})$ be a MAC scheme (Definition 13), and Com be a deterministic commitment scheme (Definition 11).

H_5 first emulates $\mathcal{F}_{\mathsf{Setup}}$ and does the following:
- generate $(\mathsf{sk}, \mathsf{vk}) \leftarrow \mathcal{S}.\mathsf{Gen}(1^\kappa)$, $\mathsf{sk}_\Sigma \leftarrow \Sigma.\mathsf{Gen}(1^\kappa)$ and sample $\alpha \leftarrow \{0,1\}^\kappa$
- generate $\mathsf{k} \leftarrow \mathsf{DEHE.KeyGen}(1^\kappa)$ and, for $i \in [\ell]$, $(\mathsf{ek}_0^i, \mathsf{ek}_1^i) \leftarrow \mathsf{DEHE.Gen}(\mathsf{k})$
- compute $(\mathsf{ct}^*_\alpha, \mathsf{ct}^*_\mathsf{k}, \mathsf{ct}^*_\mathsf{sk}, \mathsf{ct}^*_\Sigma) \leftarrow \mathsf{DEHE.Enc}(\mathsf{k}, (\alpha, \mathsf{k}, \mathsf{sk}, \mathsf{sk}_\Sigma))$ and $\mathsf{pk} = \mathsf{Com}(\mathsf{k})$
- $\forall j \in [\ell_1]$, $b \in \{0,1\}$, compute $\mathsf{ct}^*_{j,b} \leftarrow \mathsf{DEHE.Enc}(\mathsf{k}, b)$ and $\sigma^{\mathsf{ct}*}_{b,j} \leftarrow \mathcal{S}.\mathsf{Sign}(\mathsf{sk}, (j, b, \mathsf{ct}^*_{j,b}))$
- for $b \in \{0,1\}$, sample $\{\langle \mathsf{ek}_b^1 \rangle_j\}_{j \in [\ell_1]}$ such that $\mathsf{ek}_b^1 = \sum_{j \in [\ell_1]} \langle \mathsf{ek}_b^1 \rangle_j$
- for $b \in \{0,1\}, j \in [\ell_1]$, compute $\sigma_{b,j} = \mathcal{S}.\mathsf{Sign}(\mathsf{sk}, (j, b, \langle \mathsf{ek}_b^1 \rangle_j))$
- $\forall i \in [\ell], S_b^i \in \mathcal{C}_i$, set $\mathsf{ek}_i^* = \mathsf{ek}_b^i$
- $\mathcal{A}$ receives, for each corrupt $C_j \in \mathcal{C}_0$, the following tuple:

$$\{b' : \mathsf{ct}^*_{j,b'}, \sigma^{\mathsf{ct}*}_{j,b'}\}_{b' \in \{0,1\}}, \mathsf{vk}, \mathsf{pk}, \mathsf{ct}^*_\alpha, \mathsf{ct}^*_\mathsf{k}, \mathsf{ct}^*_\mathsf{sk}, \mathsf{ct}^*_\Sigma, \{\langle \mathsf{ek}_{b'}^1 \rangle_j, \sigma_{b',j}\}_{b' \in \{0,1\}}$$

H_5, emulating the first round of $\Pi_\perp$, does the following:
- For each $C_j \in \mathcal{H}_0$, using honest party's input $x_j \in \{0,1\}$, send to $\mathcal{A}$ the tuple: $\langle \mathsf{ek}_b^1 \rangle_j, \sigma_{b,j}, \mathsf{ct}^*_{j,x_j}, \sigma^{\mathsf{ct}*}_{j,x_j}, \mathsf{vk}, \mathsf{pk}, \mathsf{ct}^*_\alpha, \mathsf{ct}^*_\mathsf{k}, \mathsf{ct}^*_\mathsf{sk}, \mathsf{ct}^*_\Sigma$
- On behalf of $S^1_{\neg b} \in \mathcal{H}_1$, accept from $\mathcal{A}$, for each $C_j \in \mathcal{C}_0$ the tuple, $\langle \mathsf{ek}_{\neg b}^1 \rangle_j, \sigma_{\neg b,j}, \mathsf{ct}_{j,x_j}, \sigma^{\mathsf{ct}}_{j,x_j}, \mathsf{vk}, \mathsf{pk}, \mathsf{ct}^*_\alpha, \mathsf{ct}^*_\mathsf{k}, \mathsf{ct}^*_\mathsf{sk}, \mathsf{ct}^*_\Sigma$ abort if any consistency checks given in $\Pi_\perp$ fail
- H_5, (emulating protocol Π_1) internally computes, $([\mathsf{ek}'_0]_b, [\mathsf{ek}'_1]_b, [y_1]_b, [\sigma_1]_b) \leftarrow \mathsf{DEHE.Eval}(b, \mathsf{ek}_1^*, (\mathsf{ct}^*_\alpha, \mathsf{ct}^*_\mathsf{k}, \mathsf{ct}^*_\Sigma, \mathsf{ct}^*_{y_0}), P_1(1))$ as the corrupt S_b^1 would, by setting $\mathsf{ct}_{y_0} = \{\mathsf{ct}^*_{i,x_i}\}_{C_i \in \mathcal{L}_0}$

H_5 samples $\{r_i\}_{i \in [\ell-1]}$ uniformly at random and does the following:
- for $i \in [\ell-1]$, compute $y_i = \begin{cases} \mathbf{C}_i(y_{i-1}) + r_i & \text{if } i = 1 \\ \mathbf{C}_i(y_{i-1} - r_{i-1}) + r_i & \text{if } i < \ell \\ \mathbf{C}_i(y_{i-1} - r_{i-1}) & \text{otherwise} \end{cases}$
- for $i \in [\ell-1]$, compute $\sigma_i = \Sigma.\mathsf{MAC}(\mathsf{sk}_\Sigma, (i, y_i))$

inputs are also used when choosing input ciphertexts in y_0. In all other aspects, H_1 and the previous hybrid are the same. The view produced by this hybrid experiment H_1 is identically distributed to that of H_0 since the masks act as one-time-pads making the intermediate values in $\{y_i\}_{i \in [\ell]}$ be uniform random strings. The choice of initial simulated ciphertexts using $\{x_j\}_{C_j \in \mathcal{H}_0}$ (as opposed to uniformly at random) also is identically distributed as the ciphertexts themselves are uncorrelated from the choice.

– H_2 : This is an intermediate hybrid experiment that differs from the previous hybrid in the following way: instead of using $\mathsf{Sim}_{\mathsf{DEHE}}$ to simulate the corrupt evaluation keys and all ciphertexts, sample the PRF key α and compute the real DEHE key k, evaluation keys $\{\mathsf{ek}_i^*\}_{i \in [\ell]}$ (using fresh randomness), and ciphertexts – like in the real execution. In all other aspects, H_2 and the previous hybrid are the same. We can show that the view produced by this hybrid experiment H_2 is computationally indistinguishable from that of H_1 via a reduction to the multi-instance selective security of DEHE.

– H_3 : This experiment is the same as the real execution. Note that this differs from the previous experiment only in that all the masks $\{r_i\}_{i \in [\ell-1]}$, the randomness used to re-generate the DEHE evaluation keys and the signing randomness within the

Figure 32 Hybrid Experiment H_5 – continued

Let $\mathsf{DEHE} = (\mathsf{KeyGen}, \mathsf{Gen}, \mathsf{Enc}, \mathsf{Eval})$ be a two-party DEHE scheme (Definition 1). Let $\mathcal{S} = (\mathsf{Gen}, \mathsf{Sign}, \mathsf{Verify})$ be a signature scheme (Definition 14), F be a PRF (Definition 12), $\Sigma = (\mathsf{Gen}, \mathsf{MAC}, \mathsf{Verify})$ be a MAC scheme (Definition 13), and Com be a deterministic commitment scheme (Definition 11).

$\forall i \in [\ell - 1]$, H_5, emulating the ith computation round of $\Pi_\perp$, does the following:

- On behalf of $S^i_{\neg b} \in \mathcal{H}_i$, do the following:
  - compute $[y_i]_{\neg b} = y_i - [y_i]_b$ and $[\sigma_i]_{\neg b} = \sigma_i - [\sigma_i]_b$
  - for $S^{i+1}_{b'} \in \mathcal{C}_{i+1}$, set $[\mathsf{ek}_{b'}]_{\neg b} = \mathsf{ek}^*_i - [\mathsf{ek}'_{b'}]_b$ and $[\mathsf{ek}_{\neg b'}]_{\neg b} = \mathsf{k} - \mathsf{ek}^*_i - [\mathsf{ek}'_{\neg b'}]_b$
  - compute $\mathsf{pk}_{b',\neg b} = \mathsf{Com}([\mathsf{ek}_{b'}]_{\neg b})$ and $\mathsf{pk}_{\neg b',\neg b} = \mathsf{Com}([\mathsf{ek}_{\neg b'}]_{\neg b})$
  - abort if any **consistency checks** given in Π_i fail

  send $([y_i]_{\neg b}, [\sigma_i]_{\neg b}, [\mathsf{ek}_{b'}]_{\neg b}, \mathsf{pk}_{b',\neg b}, \mathsf{pk}_{\neg b',\neg b}, \mathsf{vk}, \mathsf{pk}, \mathsf{ct}^*_\alpha, \mathsf{ct}^*_\mathsf{k}, \mathsf{ct}^*_\mathsf{sk}, \mathsf{ct}^*_\Sigma)$ to $\mathcal{A}$
- H_5 accepts $([y_i]^*_b, [\sigma_i]^*_b, [\mathsf{ek}'_{\neg b'}]_b, \mathsf{pk}_{b',b}, \mathsf{pk}_{\neg b',b}, \mathsf{vk}, \mathsf{pk}, \mathsf{ct}^*_\alpha, \mathsf{ct}^*_\mathsf{k}, \mathsf{ct}^*_\mathsf{sk}, \mathsf{ct}^*_\Sigma)$ from $\mathcal{A}$
- H_5 computes $y^*_i = [y_i]^*_b + [y_i]_{\neg b}$ and $\sigma^*_i = [\sigma_i]^*_b + [\sigma_i]_{\neg b}$. If $\top \neq \Sigma.\mathsf{Verify}(\mathsf{sk}_\Sigma, \sigma^*_i, y^*_i)$, then abort the protocol
- H_5, emulating protocol Π_{i+1}, computes, $([\mathsf{ek}'_0]_{b'}, [\mathsf{ek}'_1]_{b'}, [y_{i+1}]_{b'}, [\sigma_{i+1}]_{b'}) \leftarrow \mathsf{DEHE.Eval}(b', \mathsf{ek}^*_{i+1}, (\mathsf{ct}^*_\alpha, \mathsf{ct}^*_\mathsf{k}, \mathsf{ct}^*_\Sigma), P_{i+1}(i+1, y_i, \sigma_i))$ as $S^{i+1}_{b'}$ would

Before the final round, H_5 sets $y_\ell = f(x)$ and $\sigma_\ell = \mathcal{S}.\mathsf{Sign}(\mathsf{sk}, (\ell, y_\ell))$.

- on behalf of $S^\ell_{\neg b} \in \mathcal{H}_\ell$, do the following:
  - compute $([y_\ell]_{b'}, [\sigma_\ell]_{b'}) \leftarrow \mathsf{DEHE.Eval}(b', \mathsf{ek}^*_\ell, (\mathsf{ct}^*_\alpha, \mathsf{ct}^*_\mathsf{k}, \mathsf{ct}^*_\mathsf{sk}), P_\ell(\ell, y_{\ell-1}, \sigma_{\ell-1}))$
  - compute $[y_\ell]_{\neg b} = y_\ell - [y_\ell]_b$ and $[\sigma_\ell]_{\neg b} = \sigma_\ell - [\sigma_\ell]_b$

  send $([y_i]_{\neg b}, [\sigma_i]_{\neg b}, \mathsf{vk})$ to $\mathcal{A}$
- $\forall P^{\ell+1}_j \in \mathcal{H}_{i+1}$, H_5 accepts $([y_i]_b, [\sigma_i]_b, \mathsf{vk})$ from $\mathcal{A}$ and, if **consistency checks** given in Π_ℓ fail, abort the protocol.

RMS programs is pseudorandomly generated from the PRF F keyed with α. We can show that the view produced by this hybrid experiment H_3 is computationally indistinguishable from that of H_2 via a reduction to the pseudorandomness of F.

The formal reductions proving hybrid indistinguishability follow similarly to the corresponding claims in Appendix D.

D Proof of Theorem 4

Proof. We need to show that the protocol presented in Algorithm 10 realizes the required layered functionality f in the presence of a malicious, adaptive, rushing adversary $\mathcal{A}$ that can corrupt all-but-one party in $\mathcal{L}_0$ and $\mathcal{L}_{\ell+1}$, and one party in each intermediate layer $\mathcal{L}_i$, for all $i \in [\ell]$.

Adaptive Security through Secure Erasures. In the layered setting where each party only sends one message, our protocol supports adaptive security using secure erasures: each party participating in the protocol computes the messages it needs to send, then erases its state of the protocol before sending the messages to the next layer as specified. This makes it such that when an adaptive adversary corrupts a party that has already sent its message, it does not gain access to that party's state in the protocol. Therefore, it suffices to design protocols that are only selectively secure and these can be lifted to adaptive security in the layered setting using secure erasures.

Going forward, we present a proof that the protocol $\Pi_\perp$ in Algorithm 10 is selectively secure in the presence of a malicious rushing adversary $\mathcal{A}$ that can corrupt all-but-one party in $\mathcal{L}_0$ and $\mathcal{L}_{\ell+1}$, and one party in each intermediate layer $\mathcal{L}_i$, for all $i \in [\ell]$. The proof for this follows from a PPT simulation algorithm $\mathsf{Sim}_\perp$ that we

Figure 33 Simulator Sim_{sh} for Layer protocol Π_{sh} in the $\mathcal{F}_{\mathsf{Setup}}$-hybrid

Let $\mathsf{DEHE} = (\mathsf{KeyGen}, \mathsf{Gen}, \mathsf{Enc}, \mathsf{Eval})$ be a two-party DEHE scheme (Definition 1) and F be a PRF (Definition 12).

Sim_{sh} first emulates $\mathcal{F}_{\mathsf{Setup}}$ and does the following:
- Let $n = |\text{PRF key}| + |\text{DEHE key}|$. Compute $(\{\mathsf{ek}_i^*\}_{i\in[\ell]}, \mathsf{ct}_\alpha^*, \mathsf{ct}_\mathsf{k}^*, \{\mathsf{ct}_{j,0}^*, \mathsf{ct}_{j,1}^*\}_{j\in[\ell_1]}) \leftarrow \mathsf{Sim}_{\mathsf{DEHE}}(1^\kappa, n + 2\ell_1, \ell)$
- For $C_j \in \mathcal{C}_0$, sample $\{\langle \mathsf{ek}_b^1\rangle_j\}_{b\in\{0,1\}}$ uniformly at random.
- $\mathcal{A}$ receives, for each corrupt $C_j \in \mathcal{C}_0$, the following tuple from Sim_{sh}:

$$\{b' : \mathsf{ct}_{j,b'}^*\}_{b'\in\{0,1\}}, \mathsf{ct}_\alpha^*, \mathsf{ct}_\mathsf{k}^*, \{\langle \mathsf{ek}_b^1\rangle_j\}_{b\in\{0,1\}}$$

Sim_{sh}, emulating the first round of Π_{sh}, does the following:
- For $S_b^1 \in \mathcal{C}_1$, sample $\{\langle \mathsf{ek}_b^1\rangle_j\}_{j\in\mathcal{H}_0}$ such that $\mathsf{ek}_1^* = \sum_{j\in[\ell_1]}\langle \mathsf{ek}_b^1\rangle_j$.
- For each $C_j \in \mathcal{H}_0$, sample $x_j \leftarrow \{0,1\}$ and send to $\mathcal{A}$ the tuple: $\langle \mathsf{ek}_b^1\rangle_j, \mathsf{ct}_{j,x_j}^*, \mathsf{ct}_\alpha^*, \mathsf{ct}_\mathsf{k}^*$
- On behalf of $S_{\neg b}^1 \in \mathcal{H}_1$, Sim_{sh} accepts from $\mathcal{A}$, for each $C_j \in \mathcal{C}_0$ the tuple, $\langle \mathsf{ek}_{\neg b}^1\rangle_j, \mathsf{ct}_{j,x_j}^*, \mathsf{ct}_\alpha^*, \mathsf{ct}_\mathsf{k}^*$.

Sim_{sh} (emulating protocol Π_1) internally computes, $([\mathsf{ek}_0^2]_b, [\mathsf{ek}_1^2]_b, [y_1]_b) \leftarrow \mathsf{DEHE.Eval}(b, \mathsf{ek}_1^*, (\mathsf{ct}_\alpha^*, \mathsf{ct}_\mathsf{k}^*, \mathsf{ct}_{y_0}), P_1(1))$ as the corrupt S_b^1 would, by setting $\mathsf{ct}_{y_0} = \{\mathsf{ct}_{i,x_i}^*\}_{C_i\in\mathcal{L}_0}$.

$\forall i \in [\ell - 1]$, Sim_{sh}, emulating the ith computation round of Π_{sh}, does the following:
- Sample y_i uniformly at random.
- On behalf of $S_{\neg b}^i \in \mathcal{H}_i$, do the following:
  - compute $[y_i]_{\neg b} = y_i - [y_i]_b$.
  - for $S_{b'}^{i+1} \in \mathcal{C}_{i+1}$, compute $[\mathsf{ek}_{b'}^{i+1}]_{\neg b} = \mathsf{ek}_i^* - [\mathsf{ek}_{b'}^{i+1}]_b$.
  - Send $([y_i]_{\neg b}, [\mathsf{ek}_{b'}^{i+1}]_{\neg b}, \mathsf{ct}_\alpha^*, \mathsf{ct}_\mathsf{k}^*)$ to $\mathcal{A}$
  - Sim_{sh} accepts $([y_i]_b, [\mathsf{ek}_{\neg b'}^{i+1}]_b, \mathsf{ct}_\alpha^*, \mathsf{ct}_\mathsf{k}^*)$ from $\mathcal{A}$.
  - Sim_{sh} (emulating protocol Π_{i+1}) computes, $([\mathsf{ek}_0^{i+2}]_{b'}, [\mathsf{ek}_1^{i+2}]_{b'}, [y_{i+1}]_{b'}) \leftarrow \mathsf{DEHE.Eval}(b', \mathsf{ek}_{i+1}^*, (\mathsf{ct}_\alpha^*, \mathsf{ct}_\mathsf{k}^*), P_{i+1}(i+1, y_i))$ as corrupt $S_{b'}^{i+1}$ would.

For the final round, Sim_{sh} does the following.
- Let y_ℓ denote the function output (which is received as input by Sim_{sh}). Compute $[y_\ell]_{\neg b} = y_\ell - [y_\ell]_b$, where $S_b^\ell \in \mathcal{C}_\ell$.
- Send $[y_\ell]_{\neg b}$ to $\mathcal{A}$.

define for the protocol $\Pi_\perp$. Before detailing the simulator itself, let us first formalize essential notation below.

Notation. Let $\mathcal{F}_{\mathsf{Setup}}$ be the functionality given in Algorithm 6. Since we show that $\Pi_\perp$ realizes f in the $\mathcal{F}_{\mathsf{Setup}}$-hybrid, our simulator $\mathsf{Sim}_\perp$ needs to also internally emulate $\mathcal{F}_{\mathsf{Setup}}$ and the messages that the adversary needs to receive from it. Let $\mathsf{DEHE} = (\mathsf{KeyGen}, \mathsf{Gen}, \mathsf{Enc}, \mathsf{Eval})$ be a two-party DEHE scheme (Definition 1). With respect to this, let $\mathsf{Sim}_{\mathsf{DEHE}}(\cdot)$ be the PPT simulator for DEHE that, for $n, \ell \in \mathbb{N}$, outputs

$$(\{\mathsf{ek}_i\}_{i\in[\ell]}, \{\mathsf{ct}_i\}_{i\in[n]}) \leftarrow \mathsf{Sim}_{\mathsf{DEHE}}(1^\kappa, n, \ell)$$

where n is the DEHE input size, and ℓ is the number of layers in the computation. Let $\mathcal{S} = (\mathsf{Gen}, \mathsf{Sign}, \mathsf{Verify})$ be a signature scheme (Definition 14). For the protocol $\Pi_\perp$ in Algorithm 10, let $\mathsf{Sim}_\perp$ be the PPT algorithm that simulates the view produced by an interaction with a PPT malicious adversary $\mathcal{A}$ corrupting all-but-one parties in the input and output layer and one party in each intermediate layer. For each $i \in [\ell + 1] \cup \{0\}$ let $\mathcal{H}_i \subseteq \mathcal{L}_i$ be the set of honest parties in layer $\mathcal{L}_i$. Let $\mathcal{C}_i = \mathcal{L}_i - \mathcal{H}_i$ be the set of corrupt parties in this layer.

Simulation. The simulator $\mathsf{Sim}_\perp$ is detailed in Algorithm 34. It uses the DEHE simulator $\mathsf{Sim}_{\mathsf{DEHE}}$ given above in a black-box way.

Figure 34 Simulator $\mathsf{Sim}_\perp$ for Layer protocol $\Pi_\perp$ in the $\mathcal{F}_{\mathsf{Setup}}$-hybrid

Let $\mathsf{DEHE} = (\mathsf{KeyGen}, \mathsf{Gen}, \mathsf{Enc}, \mathsf{Eval})$ be a two-party DEHE scheme (Definition 1). Let $\mathcal{S}$ be a signature scheme (Definition 14) and F be a PRF (Definition 12).

$\mathsf{Sim}_\perp$ first emulates $\mathcal{F}_{\mathsf{Setup}}$ and does the following:
- generate $(\mathsf{sk}, \mathsf{vk}) \leftarrow \mathcal{S}.\mathsf{Gen}(1^\kappa)$ and let $n = |\text{PRF key}| + |\text{DEHE key}| + |\mathsf{sk}|$
- compute $(\{\mathsf{ek}_i^*\}_{i\in[\ell]}, \mathsf{ct}_\alpha^*, \mathsf{ct}_\mathsf{k}^*, \mathsf{ct}_\mathsf{sk}^*, \{\mathsf{ct}_{j,0}^*, \mathsf{ct}_{j,1}^*\}_{j\in[\ell_1]}) \leftarrow \mathsf{Sim}_{\mathsf{DEHE}}(1^\kappa, n + 2\ell_1, \ell)$
- $\forall j \in [\ell_1], b \in \{0,1\}$, compute $\sigma_{b,j}^{\mathsf{ct}*} \leftarrow \mathcal{S}.\mathsf{Sign}(\mathsf{sk}, \mathsf{ct}_{j,b}^*)$
- for $S_b^1 \in \mathcal{C}_1$, sample $\{\langle \mathsf{ek}_b^1 \rangle_j\}_{j\in[\ell_1]}$ such that $\mathsf{ek}_1^* = \sum_{j\in[\ell_1]} \langle \mathsf{ek}_b^1 \rangle_j$
- sample $\{\langle \mathsf{ek}_{\neg b}^1 \rangle_j\}_{j\in[\ell_1]}$ uniformly at random
- for $b' \in \{0,1\}, j \in [\ell_1]$, compute $\sigma_{b',j} = \mathcal{S}.\mathsf{Sign}(\mathsf{sk}, \langle \mathsf{ek}_{b'}^1 \rangle_j)$
- $\mathcal{A}$ receives, for each corrupt $C_j \in \mathcal{C}_0$, the following tuple from $\mathsf{Sim}_\perp$:

$$\{b' : \mathsf{ct}_{j,b'}^*, \sigma_{j,b'}^{\mathsf{ct}*}\}_{b'\in\{0,1\}}, \mathsf{vk}, \mathsf{ct}_\alpha^*, \mathsf{ct}_\mathsf{k}^*, \mathsf{ct}_\mathsf{sk}^*, \{\langle \mathsf{ek}_{b'}^1 \rangle_j, \sigma_{b',j}\}_{b'\in\{0,1\}}$$

$\mathsf{Sim}_\perp$, emulating the first round of $\Pi_\perp$, does the following:
- For each $C_j \in \mathcal{H}_0$, sample $x_j \leftarrow \{0,1\}$ and send to $\mathcal{A}$ the tuple: $\langle \mathsf{ek}_b^1 \rangle_j, \sigma_{b,j}, \mathsf{ct}_{j,x_j}^*, \sigma_{j,x_j}^{\mathsf{ct}*}, \mathsf{vk}, \mathsf{ct}_\alpha^*, \mathsf{ct}_\mathsf{k}^*, \mathsf{ct}_\mathsf{sk}^*$
- On behalf of $S_{\neg b}^1 \in \mathcal{H}_1$, $\mathsf{Sim}_\perp$ accepts from $\mathcal{A}$, for each $C_j \in \mathcal{C}_0$ the tuple, $\langle \mathsf{ek}_{\neg b}^1 \rangle_j, \sigma_{\neg b,j}, \mathsf{ct}_{j,x_j}, \sigma_{j,x_j}^{\mathsf{ct}}, \mathsf{vk}, \mathsf{ct}_\alpha^*, \mathsf{ct}_\mathsf{k}^*, \mathsf{ct}_\mathsf{sk}^*$
  - send $\perp$ to $\mathcal{F}_\perp$ if this does not match the internal computation
  - **Input Extraction:** For each $C_j \in \mathcal{C}_0$,
    - if $\mathsf{ct}_{j,x_j} \notin \{\mathsf{ct}_{j,0}^*, \mathsf{ct}_{j,1}^*\}$, then abort and send $\perp$ to $\mathcal{F}_\perp$
    - else, set $x_j = 0$ if $\mathsf{ct}_{j,x_j} == \mathsf{ct}_{j,0}^*$ and $x_j = 1$ otherwise

    Finally, send $\{x_j\}_{C_j\in\mathcal{C}_0}$ as the adversary's input to $\mathcal{F}_\perp$
- $\mathsf{Sim}_\perp$ (emulating protocol Π_1) internally computes, $([\mathsf{ek}_0']_b, [\sigma_0^{\mathsf{ek}}]_b, [\mathsf{ek}_1']_b, [\sigma_1^{\mathsf{ek}}]_b, [y_1]_b, [\sigma_1]_b) \leftarrow \mathsf{DEHE.Eval}(b, \mathsf{ek}_1^*, (\mathsf{ct}_\alpha^*, \mathsf{ct}_\mathsf{k}^*, \mathsf{ct}_\mathsf{sk}^*, \mathsf{ct}_{y_0}^*), P_1(1))$ as the corrupt S_b^1 would, by setting $\mathsf{ct}_{y_0} = \{\mathsf{ct}_{i,x_i}^*\}_{C_i\in\mathcal{L}_0}$

$\mathsf{Sim}_\perp$ samples $\{y_i\}_{i\in[\ell-1]}$ uniformly at random and does the following:
- for $i \in [\ell-1]$, compute $\sigma_i = \mathcal{S}.\mathsf{Sign}(\mathsf{sk}, y_i)$ and $\sigma_{i+1}^{\mathsf{ek}} = \mathcal{S}.\mathsf{Sign}(\mathsf{sk}, \mathsf{ek}_{i+1}^*)$

$\forall i \in [\ell-1]$, $\mathsf{Sim}_\perp$, emulating the ith computation round of $\Pi_\perp$, does the following:
- On behalf of $S_{\neg b}^i \in \mathcal{H}_i$, do the following:
  - compute $[y_i]_{\neg b} = y_i - [y_i]_b$ and $[\sigma_i]_{\neg b} = \sigma_i - [\sigma_i]_b$
  - for $S_{b'}^{i+1} \in \mathcal{C}_{i+1}$, compute $[\mathsf{ek}_{b'}]_{\neg b} = \mathsf{ek}_i^* - [\mathsf{ek}_{b'}']_b$ and $[\sigma_{b'}^{\mathsf{ek}}]_{\neg b} = \sigma_i - [\sigma_{b'}^{\mathsf{ek}}]_b$

  send $([y_i]_{\neg b}, [\sigma_i]_{\neg b}, [\mathsf{ek}_{b'}]_{\neg b}, [\sigma_{b'}^{\mathsf{ek}}]_{\neg b}, \mathsf{vk}, \mathsf{ct}_\alpha^*, \mathsf{ct}_\mathsf{k}^*, \mathsf{ct}_\mathsf{sk}^*)$ to $\mathcal{A}$
- $\mathsf{Sim}_\perp$ accepts $([y_i]_b, [\sigma_i]_b, [\mathsf{ek}_{\neg b'}']_b, [\sigma_{\neg b'}^{\mathsf{ek}}]_b, \mathsf{vk}, \mathsf{ct}_\alpha^*, \mathsf{ct}_\mathsf{k}^*, \mathsf{ct}_\mathsf{sk}^*)$ from $\mathcal{A}$ and if it does not match internal computation, send $\perp$ to $\mathcal{F}_\perp$
- $\mathsf{Sim}_\perp$ (emulating protocol Π_{i+1}) computes, $([\mathsf{ek}_0']_{b'}, [\sigma_0^{\mathsf{ek}}]_{b'}, [\mathsf{ek}_1']_{b'}, [\sigma_1^{\mathsf{ek}}]_{b'}, [y_{i+1}]_{b'}, [\sigma_i]_{b'}) \leftarrow \mathsf{DEHE.Eval}(b', \mathsf{ek}_{i+1}^*, (\mathsf{ct}_\alpha^*, \mathsf{ct}_\mathsf{k}^*, \mathsf{ct}_\mathsf{sk}^*), P_{i+1}(i+1, y_{i+1}))$ as $S_{b'}^{i+1}$ would

Before the final round, $\mathsf{Sim}_\perp$ receives from $\mathcal{F}_\perp$ the functionality output $f(x)$.
- set $y_\ell = f(x)$ and $\sigma_\ell = \mathcal{S}.\mathsf{Sign}(\mathsf{sk}, y_\ell)$ and on behalf of $S_{\neg b}^\ell \in \mathcal{H}_\ell$, do the following:
  - compute $[y_\ell]_{\neg b} = y_\ell - [y_\ell]_b$ and $[\sigma_\ell]_{\neg b} = \sigma_\ell - [\sigma_\ell]_b$

  send $([y_i]_{\neg b}, [\sigma_i]_{\neg b}, \mathsf{vk})$ to $\mathcal{A}$
- $\forall P_j^{\ell+1} \in \mathcal{H}_{i+1}$, $\mathsf{Sim}_\perp$ accepts $([y_i]_b, [\sigma_i]_b, \mathsf{vk})$ from $\mathcal{A}$ and, if this is not the same as that internally computed, send $\perp$ to $\mathcal{F}_\perp$. Otherwise send $\top$ to $\mathcal{F}_\perp$.

Hybrids. It remains to prove that the joint view produced by the simulator $\mathsf{Sim}_\perp$ is computationally indistinguishable from that of the real execution of the protocol. We do so by considering the following intermediate hybrid experiments:

- H_0 : This hybrid experiment is defined to be the same as the interaction of the adversary $\mathcal{A}$ in the simulation with $\mathsf{Sim}_\perp$.
- H_1 : This is an intermediate hybrid experiment that differs from the simulation $\mathsf{Sim}_\perp$ in the following way: the execution only aborts if the **consistency checks** given in Algorithms 7 to 10 (performed on the values $\mathsf{Sim}_\perp$ internally computed for the honest party and those received from $\mathcal{A}$) would cause one. In all other aspects, H_1 and the previous hybrid are the same. This experiment is detailed in Algorithm 35. Looking ahead, we can show that the view produced by this hybrid experiment H_1 is computationally indistinguishable from that of H_0 via a reduction to the unforgeability property of the digital signature scheme $\mathcal{S}$.
- H_2 : This is an intermediate hybrid experiment that differs from the previous hybrid in the following way: the honest parties' inputs $\{x_j\}_{C_j \in \mathcal{H}_0}$ are used, along with the corrupt parties' inputs $\{x_j^*\}_{C_j \in \mathcal{C}_0}$ that were extracted, to compute the intermediate computation values $\{y_i\}_{i \in [\ell]}$ using the masks $\{r_i\}_{i \in [\ell-1]}$ – like in the real execution. The masks $\{r_i\}_{i \in [\ell-1]}$ are sampled uniformly at random and these honest parties' inputs are also used when choosing input ciphertexts in y_0. In all other aspects, H_2 and the previous hybrid are the same. This experiment is detailed in Algorithm 36. The view produced by this hybrid experiment H_2 is identically distributed to that of H_1 since the masks act as one-time-pads making the intermediate values in $\{y_i\}_{i \in [\ell]}$ be uniform random strings. The choice of initial simulated ciphertexts using $\{x_j\}_{C_j \in \mathcal{H}_0}$ (as opposed to uniformly at random) also is identically distributed as the ciphertexts themselves are uncorrelated from the choice.
- H_3 : This is an intermediate hybrid experiment that differs from the previous hybrid in the following way: instead of using $\mathsf{Sim}_{\mathsf{DEHE}}$ to simulate the corrupt evaluation keys and all ciphertexts, sample the PRF key α and compute the real DEHE key k, evaluation keys $\{\mathsf{ek}_i^*\}_{i \in [\ell]}$ (using fresh randomness), and ciphertexts – like in the real execution. In all other aspects, H_3 and the previous hybrid are the same. This experiment is detailed in Algorithm 37. We can show that the view produced by this hybrid experiment H_3 is computationally indistinguishable from that of H_2 via a reduction to the multi-instance selective security of DEHE.
- H_4 : This experiment is the same as the real execution. Note that this differs from the previous experiment only in that all the masks $\{r_i\}_{i \in [\ell-1]}$, the randomness used to re-generate the DEHE evaluation keys and the signing randomness within the RMS programs is pseudorandomly generated from the PRF F keyed with α. We can show that the view produced by this hybrid experiment H_4 is computationally indistinguishable from that of H_3 via a reduction to the pseudorandomness of F.

Claim. Assuming $\mathcal{S}$ is a digital signature scheme (Definition 14) satisfying unforgeability, $\mathsf{H}_0 \overset{c}{\approx} \mathsf{H}_1$.

Proof. Recall that the hybrid experiment H_0 behaves the same way as $\mathsf{Sim}_\perp$ (Algorithm 34) and H_1 (Algorithm 35) differs only in that the consistency checks that H_1 performs on behalf of the honest parties involve verifying signatures on messages, even if they do not match the internally computed messages by the hybrid experiment. In contrast, in H_0, the protocol aborts when the adversary's messages are not the same as that the simulation computes. The only cases where H_0 aborts but H_1 does not would correspond to an instance where the adversary has forged a digital signature on a message of its choice, and has provided correct additive shares of the same, causing it to verify correctly on the honest party's end.

We can show that both experiments will produce exactly the same view, except with negligible probability by reduction to the unforgeability property of the underlying

Figure 35 Hybrid Experiment H_1

Let $\mathsf{DEHE} = (\mathsf{KeyGen}, \mathsf{Gen}, \mathsf{Enc}, \mathsf{Eval})$ be a two-party DEHE scheme (Definition 1). Let $\mathcal{S}$ be a signature scheme (Definition 14) and F be a PRF (Definition 12).

H_1 first emulates $\mathcal{F}_{\mathsf{Setup}}$ and does the following:
- generate $(\mathsf{sk}, \mathsf{vk}) \leftarrow \mathcal{S}.\mathsf{Gen}(1^\kappa)$ and let $n = |\text{PRF key}| + |\text{DEHE key}| + |\mathsf{sk}|$
- compute $(\{\mathsf{ek}_i^*\}_{i\in[\ell]}, \mathsf{ct}_\alpha^*, \mathsf{ct}_\mathsf{k}^*, \mathsf{ct}_\mathsf{sk}^*, \{\mathsf{ct}_{j,0}^*, \mathsf{ct}_{j,1}^*\}_{j\in[\ell_1]}) \leftarrow \mathsf{Sim}_{\mathsf{DEHE}}(1^\kappa, n + 2\ell_1, \ell)$
- $\forall j \in [\ell_1], b \in \{0,1\}$, compute $\sigma_{b,j}^{\mathsf{ct}*} \leftarrow \mathcal{S}.\mathsf{Sign}(\mathsf{sk}, \mathsf{ct}_{j,b}^*)$
- for $S_b^1 \in \mathcal{C}_1$, sample $\{\langle \mathsf{ek}_b^1 \rangle_j\}_{j\in[\ell_1]}$ such that $\mathsf{ek}_1^* = \sum_{j\in[\ell_1]} \langle \mathsf{ek}_b^1 \rangle_j$
- sample $\{\langle \mathsf{ek}_{\neg b}^1 \rangle_j\}_{j\in[\ell_1]}$ uniformly at random
- for $b' \in \{0,1\}, j \in [\ell_1]$, compute $\sigma_{b',j} = \mathcal{S}.\mathsf{Sign}(\mathsf{sk}, \langle \mathsf{ek}_{b'}^1 \rangle_j)$
- $\mathcal{A}$ receives, for each corrupt $C_j \in \mathcal{C}_0$, the following tuple:

$$\{b' : \mathsf{ct}_{j,b'}^*, \sigma_{j,b'}^{\mathsf{ct}*}\}_{b'\in\{0,1\}}, \mathsf{vk}, \mathsf{ct}_\alpha^*, \mathsf{ct}_\mathsf{k}^*, \mathsf{ct}_\mathsf{sk}^*, \{\langle \mathsf{ek}_{b'}^1 \rangle_j, \sigma_{b',j}\}_{b'\in\{0,1\}}$$

H_1, emulating the first round of $\Pi_\perp$, does the following:
- For each $C_j \in \mathcal{H}_0$, sample $x_j \leftarrow \{0,1\}$ and send to $\mathcal{A}$ the tuple: $\langle \mathsf{ek}_b^1 \rangle_j, \sigma_{b,j}, \mathsf{ct}_{j,x_j}^*, \sigma_{j,x_j}^{\mathsf{ct}*}, \mathsf{vk}, \mathsf{ct}_\alpha^*, \mathsf{ct}_\mathsf{k}^*, \mathsf{ct}_\mathsf{sk}^*$
- On behalf of $S_{\neg b}^1 \in \mathcal{H}_1$, accept from $\mathcal{A}$, for each $C_j \in \mathcal{C}_0$ the tuple, $\langle \mathsf{ek}_{\neg b}^1 \rangle_j, \sigma_{\neg b,j}, \mathsf{ct}_{j,x_j}, \sigma_{j,x_j}^{\mathsf{ct}}, \mathsf{vk}, \mathsf{ct}_\alpha^*, \mathsf{ct}_\mathsf{k}^*, \mathsf{ct}_\mathsf{sk}^*$ abort if any consistency checks given in $\Pi_\perp$ fail
- H_1, emulating protocol Π_1, internally computes, $([\mathsf{ek}_0']_b, [\sigma_0^{\mathsf{ek}}]_b, [\mathsf{ek}_1']_b, [\sigma_1^{\mathsf{ek}}]_b, [y_1]_b, [\sigma_1]_b) \leftarrow \mathsf{DEHE.Eval}(b, \mathsf{ek}_1^*, (\mathsf{ct}_\alpha^*, \mathsf{ct}_\mathsf{k}^*, \mathsf{ct}_\mathsf{sk}^*, \mathsf{ct}_{y_0}^*), P_1(1))$ as the corrupt S_b^1 would, by setting $\mathsf{ct}_{y_0} = \{\mathsf{ct}_{i,x_i}^*\}_{C_i\in\mathcal{L}_0}$

H_1 samples $\{y_i\}_{i\in[\ell-1]}$ uniformly at random and does the following:
- for $i \in [\ell-1]$, compute $\sigma_i = \mathcal{S}.\mathsf{Sign}(\mathsf{sk}, y_i)$ and $\sigma_{i+1}^{\mathsf{ek}} = \mathcal{S}.\mathsf{Sign}(\mathsf{sk}, \mathsf{ek}_{i+1}^*)$

$\forall i \in [\ell-1]$, H_1, emulating the ith computation round of $\Pi_\perp$, does the following:
- On behalf of $S_{\neg b}^i \in \mathcal{H}_i$, do the following:
  - compute $[y_i]_{\neg b} = y_i - [y_i]_b$ and $[\sigma_i]_{\neg b} = \sigma_i - [\sigma_i]_b$
  - for $S_{b'}^{i+1} \in \mathcal{C}_{i+1}$, compute $[\mathsf{ek}_{b'}]_{\neg b} = \mathsf{ek}_i^* - [\mathsf{ek}_{b'}']_b$ and $[\sigma_{b'}^{\mathsf{ek}}]_{\neg b} = \sigma_i - [\sigma_{b'}^{\mathsf{ek}}]_b$
  - abort if any **consistency checks** given in Π_i fail

  send $([y_i]_{\neg b}, [\sigma_i]_{\neg b}, [\mathsf{ek}_{b'}]_{\neg b}, [\sigma_{b'}^{\mathsf{ek}}]_{\neg b}, \mathsf{vk}, \mathsf{ct}_\alpha^*, \mathsf{ct}_\mathsf{k}^*, \mathsf{ct}_\mathsf{sk}^*)$ to $\mathcal{A}$
- H_1 accepts $([y_i]_b, [\sigma_i]_b, [\mathsf{ek}_{\neg b'}']_b, [\sigma_{\neg b'}^{\mathsf{ek}}]_b, \mathsf{vk}, \mathsf{ct}_\alpha^*, \mathsf{ct}_\mathsf{k}^*, \mathsf{ct}_\mathsf{sk}^*)$ from $\mathcal{A}$
- H_1, emulating protocol Π_{i+1}, computes, $([\mathsf{ek}_0']_{b'}, [\sigma_0^{\mathsf{ek}}]_{b'}, [\mathsf{ek}_1']_{b'}, [\sigma_1^{\mathsf{ek}}]_{b'}, [y_{i+1}]_{b'}, [\sigma_{i+1}]_{b'}) \leftarrow \mathsf{DEHE.Eval}(b', \mathsf{ek}_{i+1}^*, (\mathsf{ct}_\alpha^*, \mathsf{ct}_\mathsf{k}^*, \mathsf{ct}_\mathsf{sk}^*), P_{i+1}(i+1, y_{i+1}))$ as $S_{b'}^{i+1}$ would

Before the final round, H_1 sets $y_\ell = f(x)$ and $\sigma_\ell = \mathcal{S}.\mathsf{Sign}(\mathsf{sk}, y_\ell)$.
- on behalf of $S_{\neg b}^\ell \in \mathcal{H}_\ell$, do the following:
  - compute $[y_\ell]_{\neg b} = y_\ell - [y_\ell]_b$ and $[\sigma_\ell]_{\neg b} = \sigma_\ell - [\sigma_\ell]_b$

  send $([y_i]_{\neg b}, [\sigma_i]_{\neg b}, \mathsf{vk})$ to $\mathcal{A}$
- $\forall P_j^{\ell+1} \in \mathcal{H}_{i+1}$, H_1 accepts $([y_i]_b, [\sigma_i]_b, \mathsf{vk})$ from $\mathcal{A}$ and, if **consistency checks** given in Π_ℓ fail, abort the protocol.

signature scheme $\mathcal{S}$ (Definition 14). Consider for the sake of contradiction that there exists a PPT adversary $\mathsf{D}_\perp$ that can, with non-negligible probability ϵ, produce a view distributed as in H_1 but not in H_0. We show that if this were possible, then $\mathsf{D}_\perp$ can be used in a black-box way by a PPT adversary $\mathsf{D}_\mathcal{S}$ to violate the unforgeability property of the signature scheme $\mathcal{S}$. This distinguisher $\mathsf{D}_\mathcal{S}$ would work as follows:

- $\mathsf{D}_\mathcal{S}$ has a verification key vk and oracle access to a signing algorithm $\mathcal{O}_\mathcal{S}$ parameterized with the corresponding secret signing key sk. It uses this to sign messages and starts interacting with $\mathsf{D}_\perp$ exactly as given in the simulation H_0 in Algorithm 34.
- If for any ($\mathcal{F}_{\mathsf{Setup}}$ or DEHE output) message that $\mathsf{D}_\perp$ sends, it is not the case that it equals the message that $\mathsf{D}_\mathcal{S}$ has internally computed, reconstruct the value m

and corresponding signature σ using both the honest and corrupt party's additive shares. If $\top \leftarrow \mathcal{S}.\mathsf{Verify}(\mathsf{vk}, m, \sigma)$, output (m, σ) and continue the protocol as in H_1 (Algorithm 35). Otherwise, abort the protocol.

Note that in the above, if the protocol has aborted without $\mathsf{D}_{\mathcal{S}}$ outputting a forgery (m, σ), the view of $\mathsf{D}_{\perp}$ is the same as that of H_0. Otherwise, it is the same as that in the experiment H_1. The advantage of $\mathsf{D}_{\mathcal{S}}$ is the same as that of $\mathsf{D}_{\perp}$ not inducing the view of H_0, which is non-negligible by assumption. However, since $\mathcal{S}$ is an unforgeable digital signature scheme (Definition 14), no such $\mathsf{D}_{\mathcal{S}}$ can exist, yielding a contradiction.

Claim. Assuming DEHE is a secure two-server DEHE scheme (Definition 1) satisfying multi-instance selective security, $\mathsf{H}_2 \overset{c}{\approx} \mathsf{H}_3$.

Proof. Consider for the sake of contradiction that there exists a PPT distinguisher $\mathsf{D}_{\perp}$ that can differentiate with non-negligible probability ϵ between the view produced by the hybrid experiments H_2 and H_3. Then this can be used in a black-box way by a PPT distinguisher $\mathsf{D}_{\mathsf{DEHE}}$ to violate the multi-instance selective security of DEHE for $m = \ell$, with the same advantage. This distinguisher $\mathsf{D}_{\mathsf{DEHE}}$ would work as follows:

- $\mathsf{D}_{\mathsf{DEHE}}$ gives a challenger ℓ and the size of the input $n + 2\ell_1$ (as defined in Algorithm 34) and receives the tuple $(\{\mathsf{ek}_i^*\}_{i\in[\ell]}, \mathsf{ct}_{\alpha}^*, \mathsf{ct}_{\mathsf{k}}^*, \mathsf{ct}_{\mathsf{sk}}^*, \{\mathsf{ct}_{j,0}^*, \mathsf{ct}_{j,1}^*\}_{j\in[\ell_1]})$.
- $\mathsf{D}_{\mathsf{DEHE}}$ then uses this tuple to complete the interaction in the hybrid experiment exactly as in H_2.
- Note that if $(\{\mathsf{ek}_i^*\}_{i\in[\ell]}, \mathsf{ct}_{\alpha}^*, \mathsf{ct}_{\mathsf{k}}^*, \mathsf{ct}_{\mathsf{sk}}^*, \{\mathsf{ct}_{j,0}^*, \mathsf{ct}_{j,1}^*\}_{j\in[\ell_1]}) \leftarrow \mathsf{Sim}_{\mathsf{DEHE}}$, the view produced would be identically distributed as H_2. Otherwise, this would be the same as that of H_3.
- Finally, $\mathsf{D}_{\mathsf{DEHE}}$ sends this view to $\mathsf{D}_{\perp}$ and outputs whatever it would output.

Note that in the above, the advantage of $\mathsf{D}_{\mathsf{DEHE}}$ is the same as that of $\mathsf{D}_{\perp}$, which is non-negligible. However, since DEHE is a secure DEHE scheme (Definition 1), no such $\mathsf{D}_{\mathsf{DEHE}}$ can exist, yielding a contradiction.

Claim. Assuming F is a PRF (Definition 12), $\mathsf{H}_3 \overset{c}{\approx} \mathsf{H}_4$.

Proof. Consider for the sake of contradiction that there exists a PPT distinguisher $\mathsf{D}_{\perp}$ that can differentiate with non-negligible probability ϵ between the view produced by the hybrid experiments H_3 and H_4. Then this can be used in a black-box way by a PPT distinguisher $\mathsf{D}_{\mathsf{PRF}}$ to distinguish between having received oracle access to a PRF F keyed with a random key α, and a random function R with the same advantage. This distinguisher $\mathsf{D}_{\mathsf{PRF}}$ would work as follows:

- Let $\mathcal{O}$ be the oracle that $\mathsf{D}_{\mathsf{PRF}}$ has access to.
- For each $i \in [4\ell]$, $\mathsf{D}_{\mathsf{PRF}}$ queries $\mathcal{O}(i, 1)$ and $\mathcal{O}(i, 2)$, and uses its output to generate the rest of the view exactly as in H_3.
- Note that if $\mathcal{O} = R$, the view produced would be identically distributed as H_3. Otherwise, when $\mathcal{O} = F_{\alpha}$, this would be the same as that of H_4.
- Finally, $\mathsf{D}_{\mathsf{PRF}}$ sends this view to $\mathsf{D}_{\perp}$ and outputs whatever it would output.

Note that in the above, the advantage of $\mathsf{D}_{\mathsf{PRF}}$ is the same as that of $\mathsf{D}_{\perp}$, which is non-negligible. However, since F is a secure pseudo-random function (Definition 12), no such $\mathsf{D}_{\mathsf{PRF}}$ can exist, yielding a contradiction.

Figure 36 Hybrid Experiment H_2

Let $\mathsf{DEHE} = (\mathsf{KeyGen}, \mathsf{Gen}, \mathsf{Enc}, \mathsf{Eval})$ be a two-party DEHE scheme (Definition 1). Let $\mathcal{S}$ be a signature scheme (Definition 14) and F be a PRF (Definition 12).

H_2 first emulates $\mathcal{F}_{\mathsf{Setup}}$ and does the following:
- generate $(\mathsf{sk}, \mathsf{vk}) \leftarrow \mathcal{S}.\mathsf{Gen}(1^\kappa)$ and let $n = |\text{PRF key}| + |\text{DEHE key}| + |\mathsf{sk}|$
- compute $(\{\mathsf{ek}_i^*\}_{i\in[\ell]}, \mathsf{ct}_\alpha^*, \mathsf{ct}_\mathsf{k}^*, \mathsf{ct}_\mathsf{sk}^*, \{\mathsf{ct}_{j,0}^*, \mathsf{ct}_{j,1}^*\}_{j\in[\ell_1]}) \leftarrow \mathsf{Sim}_{\mathsf{DEHE}}(1^\kappa, n + 2\ell_1, \ell)$
- $\forall j \in [\ell_1], b \in \{0,1\}$, compute $\sigma_{b,j}^{\mathsf{ct}*} \leftarrow \mathcal{S}.\mathsf{Sign}(\mathsf{sk}, \mathsf{ct}_{j,b}^*)$
- for $S_b^1 \in \mathcal{C}_1$, sample $\{\langle \mathsf{ek}_b^1 \rangle_j\}_{j\in[\ell_1]}$ such that $\mathsf{ek}_1^* = \sum_{j\in[\ell_1]} \langle \mathsf{ek}_b^1 \rangle_j$
- sample $\{\langle \mathsf{ek}_{\neg b}^1 \rangle_j\}_{j\in[\ell_1]}$ uniformly at random
- for $b' \in \{0,1\}, j \in [\ell_1]$, compute $\sigma_{b',j} = \mathcal{S}.\mathsf{Sign}(\mathsf{sk}, \langle \mathsf{ek}_{b'}^1 \rangle_j)$
- $\mathcal{A}$ receives, for each corrupt $C_j \in \mathcal{C}_0$, the following tuple:

$$\{b' : \mathsf{ct}_{j,b'}^*, \sigma_{j,b'}^{\mathsf{ct}*}\}_{b'\in\{0,1\}}, \mathsf{vk}, \mathsf{ct}_\alpha^*, \mathsf{ct}_\mathsf{k}^*, \mathsf{ct}_\mathsf{sk}^*, \{\langle \mathsf{ek}_{b'}^1 \rangle_j, \sigma_{b',j}\}_{b'\in\{0,1\}}$$

H_2, emulating the first round of $\Pi_\perp$, does the following:
- For each $C_j \in \mathcal{H}_0$, using honest party's input $x_j \in \{0,1\}$, send to $\mathcal{A}$ the tuple: $\langle \mathsf{ek}_b^1 \rangle_j, \sigma_{b,j}, \mathsf{ct}_{j,x_j}^*, \sigma_{j,x_j}^{\mathsf{ct}*}, \mathsf{vk}, \mathsf{ct}_\alpha^*, \mathsf{ct}_\mathsf{k}^*, \mathsf{ct}_\mathsf{sk}^*$
- On behalf of $S_{\neg b}^1 \in \mathcal{H}_1$, accept from $\mathcal{A}$, for each $C_j \in \mathcal{C}_0$ the tuple, $\langle \mathsf{ek}_{\neg b}^1 \rangle_j, \sigma_{\neg b,j}, \mathsf{ct}_{j,x_j}, \sigma_{j,x_j}^{\mathsf{ct}}, \mathsf{vk}, \mathsf{ct}_\alpha^*, \mathsf{ct}_\mathsf{k}^*, \mathsf{ct}_\mathsf{sk}^*$ abort if any consistency checks given in $\Pi_\perp$ fail
- H_2, emulating protocol Π_1, internally computes, $([\mathsf{ek}_0']_b, [\sigma_0^{\mathsf{ek}}]_b, [\mathsf{ek}_1']_b, [\sigma_1^{\mathsf{ek}}]_b, [y_1]_b, [\sigma_1]_b) \leftarrow \mathsf{DEHE.Eval}(b, \mathsf{ek}_1^*, (\mathsf{ct}_\alpha^*, \mathsf{ct}_\mathsf{k}^*, \mathsf{ct}_\mathsf{sk}^*, \mathsf{ct}_{y_0}^*), P_1(1))$ as the corrupt S_b^1 would, by setting $\mathsf{ct}_{y_0} = \{\mathsf{ct}_{i,x_i}^*\}_{C_i\in\mathcal{L}_0}$

H_2 samples $\{r_i\}_{i\in[\ell-1]}$ uniformly at random and does the following:
- for $i \in [\ell-1]$, compute $y_i = \begin{cases} \mathbf{C}_i(y_{i-1}) + r_i & \text{if } i = 1 \\ \mathbf{C}_i(y_{i-1} - r_{i-1}) + r_i & \text{if } i < \ell \\ \mathbf{C}_i(y_{i-1} - r_{i-1}) & \text{otherwise} \end{cases}$
- for $i \in [\ell-1]$, compute $\sigma_i = \mathcal{S}.\mathsf{Sign}(\mathsf{sk}, y_i)$ and $\sigma_{i+1}^{\mathsf{ek}} = \mathcal{S}.\mathsf{Sign}(\mathsf{sk}, \mathsf{ek}_{i+1}^*)$

$\forall i \in [\ell-1]$, H_2, emulating the ith computation round of $\Pi_\perp$, does the following:
- On behalf of $S_{\neg b}^i \in \mathcal{H}_i$, do the following:
  - compute $[y_i]_{\neg b} = y_i - [y_i]_b$ and $[\sigma_i]_{\neg b} = \sigma_i - [\sigma_i]_b$
  - for $S_{b'}^{i+1} \in \mathcal{C}_{i+1}$, compute $[\mathsf{ek}_{b'}]_{\neg b} = \mathsf{ek}_i^* - [\mathsf{ek}_{b'}']_b$ and $[\sigma_{b'}^{\mathsf{ek}}]_{\neg b} = \sigma_i - [\sigma_{b'}^{\mathsf{ek}}]_b$
  - abort if any **consistency checks** given in Π_i fail

  send $([y_i]_{\neg b}, [\sigma_i]_{\neg b}, [\mathsf{ek}_{b'}]_{\neg b}, [\sigma_{b'}^{\mathsf{ek}}]_{\neg b}, \mathsf{vk}, \mathsf{ct}_\alpha^*, \mathsf{ct}_\mathsf{k}^*, \mathsf{ct}_\mathsf{sk}^*)$ to $\mathcal{A}$
- H_2 accepts $([y_i]_b, [\sigma_i]_b, [\mathsf{ek}_{\neg b'}']_b, [\sigma_{\neg b'}^{\mathsf{ek}}]_b, \mathsf{vk}, \mathsf{ct}_\alpha^*, \mathsf{ct}_\mathsf{k}^*, \mathsf{ct}_\mathsf{sk}^*)$ from $\mathcal{A}$
- H_2, emulating protocol Π_{i+1}, computes, $([\mathsf{ek}_0']_{b'}, [\sigma_0^{\mathsf{ek}}]_{b'}, [\mathsf{ek}_1']_{b'}, [\sigma_1^{\mathsf{ek}}]_{b'}, [y_{i+1}]_{b'}, [\sigma_{i+1}]_{b'}) \leftarrow \mathsf{DEHE.Eval}(b', \mathsf{ek}_{i+1}^*, (\mathsf{ct}_\alpha^*, \mathsf{ct}_\mathsf{k}^*, \mathsf{ct}_\mathsf{sk}^*), P_{i+1}(i+1, y_i))$ as $S_{b'}^{i+1}$ would

Before the final round, H_2 sets $y_\ell = f(x)$ and $\sigma_\ell = \mathcal{S}.\mathsf{Sign}(\mathsf{sk}, y_\ell)$.
- on behalf of $S_{\neg b}^\ell \in \mathcal{H}_\ell$, do the following:
  - compute $[y_\ell]_{\neg b} = y_\ell - [y_\ell]_b$ and $[\sigma_\ell]_{\neg b} = \sigma_\ell - [\sigma_\ell]_b$

  send $([y_i]_{\neg b}, [\sigma_i]_{\neg b}, \mathsf{vk})$ to $\mathcal{A}$
- $\forall P_j^{\ell+1} \in \mathcal{H}_{i+1}$, H_2 accepts $([y_i]_b, [\sigma_i]_b, \mathsf{vk})$ from $\mathcal{A}$ and, if **consistency checks** given in Π_ℓ fail, abort the protocol.

E Proof of Theorem 5

Proof. The proof for the theorem follows from a PPT simulator $\mathsf{Sim}_{\mathsf{fair}}$ that we define for the protocol Π_{fair} in Algorithm 11. Before detailing the simulator itself, let us first formalize essential notation below.

Notation. For the protocol $\Pi_\perp$ used in Algorithm 11 that satisfies security with abort, let $\mathsf{Sim}_\perp$ be the PPT algorithm that simulates the view produced by an interaction

Figure 37 Hybrid Experiment H_3

Let $\mathsf{DEHE} = (\mathsf{KeyGen}, \mathsf{Gen}, \mathsf{Enc}, \mathsf{Eval})$ be a two-party DEHE scheme (Definition 1). Let $\mathcal{S}$ be a signature scheme (Definition 14) and F be a PRF (Definition 12).

H_3 first emulates $\mathcal{F}_{\mathsf{Setup}}$ and does the following:
- generate $(\mathsf{sk}, \mathsf{vk}) \leftarrow \mathcal{S}.\mathsf{Gen}(1^\kappa)$ and sample a PRF key $\alpha \leftarrow \{0,1\}^\kappa$
- generate $\mathsf{k} \leftarrow \mathsf{DEHE.KeyGen}(1^\kappa)$ and, for $i \in [\ell]$, $(\mathsf{ek}_0^i, \mathsf{ek}_1^i) \leftarrow \mathsf{DEHE.Gen}(\mathsf{k})$
- compute $(\mathsf{ct}_\alpha^*, \mathsf{ct}_\mathsf{k}^*, \mathsf{ct}_\mathsf{sk}^*) \leftarrow \mathsf{DEHE.Enc}(\mathsf{k}, (\alpha, \mathsf{k}, \mathsf{sk}))$
- $\forall j \in [\ell_1]$, $b \in \{0,1\}$, compute $\mathsf{ct}_{j,b}^* \leftarrow \mathsf{DEHE.Enc}(\mathsf{k}, b)$ and $\sigma_{b,j}^{\mathsf{ct}*} \leftarrow \mathcal{S}.\mathsf{Sign}(\mathsf{sk}, \mathsf{ct}_{j,b}^*)$
- for $b \in \{0,1\}$, sample $\{\langle \mathsf{ek}_b^1 \rangle_j\}_{j\in[\ell_1]}$ such that $\mathsf{ek}_b^1 = \sum_{j\in[\ell_1]} \langle \mathsf{ek}_b^1 \rangle_j$
- for $b \in \{0,1\}, j \in [\ell_1]$, compute $\sigma_{b,j} = \mathcal{S}.\mathsf{Sign}(\mathsf{sk}, \langle \mathsf{ek}_b^1 \rangle_j)$
- $\forall i \in [\ell]$, $S_b^i \in \mathcal{C}_i$, set $\mathsf{ek}_i^* = \mathsf{ek}_b^i$
- $\mathcal{A}$ receives, for each corrupt $C_j \in \mathcal{C}_0$, the following tuple:

$$\{b' : \mathsf{ct}_{j,b'}^*, \sigma_{j,b'}^{\mathsf{ct}*}\}_{b'\in\{0,1\}}, \mathsf{vk}, \mathsf{ct}_\alpha^*, \mathsf{ct}_\mathsf{k}^*, \mathsf{ct}_\mathsf{sk}^*, \{\langle \mathsf{ek}_{b'}^1 \rangle_j, \sigma_{b',j}\}_{b'\in\{0,1\}}$$

H_3, emulating the first round of $\Pi_\perp$, does the following:
- For each $C_j \in \mathcal{H}_0$, using honest party's input $x_j \in \{0,1\}$, send to $\mathcal{A}$ the tuple: $\langle \mathsf{ek}_b^1 \rangle_j, \sigma_{b,j}, \mathsf{ct}_{j,x_j}^*, \sigma_{j,x_j}^{\mathsf{ct}*}, \mathsf{vk}, \mathsf{ct}_\alpha^*, \mathsf{ct}_\mathsf{k}^*, \mathsf{ct}_\mathsf{sk}^*$
- On behalf of $S_{\neg b}^1 \in \mathcal{H}_1$, accept from $\mathcal{A}$, for each $C_j \in \mathcal{C}_0$ the tuple, $\langle \mathsf{ek}_{\neg b}^1 \rangle_j, \sigma_{\neg b,j}, \mathsf{ct}_{j,x_j}, \sigma_{j,x_j}^{\mathsf{ct}}, \mathsf{vk}, \mathsf{ct}_\alpha^*, \mathsf{ct}_\mathsf{k}^*, \mathsf{ct}_\mathsf{sk}^*$ abort if any consistency checks given in $\Pi_\perp$ fail
- H_3, emulating protocol Π_1, internally computes, $([\mathsf{ek}_0']_b, [\sigma_0^{\mathsf{ek}}]_b, [\mathsf{ek}_1']_b, [\sigma_1^{\mathsf{ek}}]_b, [y_1]_b, [\sigma_1]_b) \leftarrow \mathsf{DEHE.Eval}(b, \mathsf{ek}_1^*, (\mathsf{ct}_\alpha^*, \mathsf{ct}_\mathsf{k}^*, \mathsf{ct}_\mathsf{sk}^*, \mathsf{ct}_{y_0}^*), P_1(1))$ as the corrupt S_b^1 would, by setting $\mathsf{ct}_{y_0} = \{\mathsf{ct}_{i,x_i}^*\}_{C_i \in \mathcal{L}_0}$

H_3 samples $\{r_i\}_{i\in[\ell-1]}$ uniformly at random and does the following:
- for $i \in [\ell-1]$, compute $y_i = \begin{cases} \mathbf{C}_i(y_{i-1}) + r_i & \text{if } i = 1 \\ \mathbf{C}_i(y_{i-1} - r_{i-1}) + r_i & \text{if } i < \ell \\ \mathbf{C}_i(y_{i-1} - r_{i-1}) & \text{otherwise} \end{cases}$
- for $i \in [\ell-1]$, compute $\sigma_i = \mathcal{S}.\mathsf{Sign}(\mathsf{sk}, y_i)$ and $\sigma_{i+1}^{\mathsf{ek}} = \mathcal{S}.\mathsf{Sign}(\mathsf{sk}, \mathsf{ek}_{i+1}^*)$

$\forall i \in [\ell-1]$, H_3, emulating the ith computation round of $\Pi_\perp$, does the following:
- On behalf of $S_{\neg b}^i \in \mathcal{H}_i$, do the following:
  - compute $[y_i]_{\neg b} = y_i - [y_i]_b$ and $[\sigma_i]_{\neg b} = \sigma_i - [\sigma_i]_b$
  - for $S_{b'}^{i+1} \in \mathcal{C}_{i+1}$, compute $[\mathsf{ek}_{b'}]_{\neg b} = \mathsf{ek}_i^* - [\mathsf{ek}_{b'}']_b$ and $[\sigma_{b'}^{\mathsf{ek}}]_{\neg b} = \sigma_i - [\sigma_{b'}^{\mathsf{ek}}]_b$
  - abort if any **consistency checks** given in Π_i fail

  send $([y_i]_{\neg b}, [\sigma_i]_{\neg b}, [\mathsf{ek}_{b'}]_{\neg b}, [\sigma_{b'}^{\mathsf{ek}}]_{\neg b}, \mathsf{vk}, \mathsf{ct}_\alpha^*, \mathsf{ct}_\mathsf{k}^*, \mathsf{ct}_\mathsf{sk}^*)$ to $\mathcal{A}$
- H_3 accepts $([y_i]_b, [\sigma_i]_b, [\mathsf{ek}_{\neg b'}']_b, [\sigma_{\neg b'}^{\mathsf{ek}}]_b, \mathsf{vk}, \mathsf{ct}_\alpha^*, \mathsf{ct}_\mathsf{k}^*, \mathsf{ct}_\mathsf{sk}^*)$ from $\mathcal{A}$
- H_3, emulating protocol Π_{i+1}, computes, $([\mathsf{ek}_0']_{b'}, [\sigma_0^{\mathsf{ek}}]_{b'}, [\mathsf{ek}_1']_{b'}, [\sigma_1^{\mathsf{ek}}]_{b'}, [y_{i+1}]_{b'}, [\sigma_{i+1}]_{b'}) \leftarrow \mathsf{DEHE.Eval}(b', \mathsf{ek}_{i+1}^*, (\mathsf{ct}_\alpha^*, \mathsf{ct}_\mathsf{k}^*, \mathsf{ct}_\mathsf{sk}^*), P_{i+1}(i+1, y_i))$ as $S_{b'}^{i+1}$ would

Before the final round, H_3 sets $y_\ell = f(x)$ and $\sigma_\ell = \mathcal{S}.\mathsf{Sign}(\mathsf{sk}, y_\ell)$.
- on behalf of $S_{\neg b}^\ell \in \mathcal{H}_\ell$, do the following:
  - compute $[y_\ell]_{\neg b} = y_\ell - [y_\ell]_b$ and $[\sigma_\ell]_{\neg b} = \sigma_\ell - [\sigma_\ell]_b$

  send $([y_i]_{\neg b}, [\sigma_i]_{\neg b}, \mathsf{vk})$ to $\mathcal{A}$
- $\forall P_j^{\ell+1} \in \mathcal{H}_{i+1}$, H_3 accepts $([y_i]_b, [\sigma_i]_b, \mathsf{vk})$ from $\mathcal{A}$ and, if **consistency checks** given in Π_ℓ fail, abort the protocol.

with a PPT malicious adversary $\mathcal{A}$ corrupting all-but-one parties in the input and output layer and one party in each intermediate layer. For each $i \in [d+1] \cup \{0\}$ let $\mathcal{H}_i \subseteq \mathcal{L}_i$ be the set of honest parties in layer $\mathcal{L}_i$. Let $\mathcal{C}_i = \mathcal{L}_i - \mathcal{H}_i$ be the set of corrupt parties in this layer.

Simulation. Given the above notation, the interactive algorithm $\mathsf{Sim}_{\mathsf{fair}}$ simply works by invoking $\mathsf{Sim}_\perp$ to interact with the adversary for the first $d+1$ layers (up to $\mathcal{L}_d$).

The adversary's inputs are extracted just as it would have been in $\mathsf{Sim}_\perp$ and given to the ideal layered functionality f. $\mathsf{Sim}_{\mathsf{fair}}$ additionally samples a random RSS share (independent of the function output) and uses it to complete the simulation $\mathsf{Sim}_\perp$.

If the adversary $\mathcal{A}$ causes the protocol to abort on or before layer $\mathcal{L}_d$ of computation, the simulation $\mathsf{Sim}_{\mathsf{fair}}$ would send $\perp$ to the functionality f and behaved in the same way to the adversary as the ideal world for security with abort, possibly revealing shares of the function output to the adversary. However, since $\mathcal{L}_d$ has an honest majority, the adversary will never have enough shares to reconstruct the output $f(\vec{x})$.

If the protocol has not aborted and the computation proceeds to the output layer, $\mathsf{Sim}_{\mathsf{fair}}$ receives $f(\vec{x})$ from the functionality and uses share-patching (Definition 10) to generate other RSS shares such that these, and the share that the adversary already knows, reconstruct to $f(\vec{x})$. These shares are given to the adversary. This simulation strategy $\mathsf{Sim}_{\mathsf{fair}}$ is formalized in Algorithm 38 below.

Figure 38 Simulator for the Layered MPC satisfying Fairness

Let $\mathcal{A}$ be a malicious adversary expecting to participate in $\Pi_\perp$ to compute f. For each $i \in [d+1] \cup \{0\}$, $\mathcal{C}_i = \mathcal{L}_i - \mathcal{H}_i$ is the set of corrupt parties in this layer. The simulation $\mathsf{Sim}_{\mathsf{fair}}$ works as follows:

- within layers $\mathcal{L}_0$ to $\mathcal{L}_d$, invoke $\mathsf{Sim}_\perp$ to interact with the adversary $\mathcal{A}$:
  - **Input Extraction:** $\mathsf{Sim}_\perp$ extracts the adversary's inputs $\{x'_j\}_{P_0^j \in \mathcal{C}_0}$ after receiving its messages from $\mathcal{L}_0$ and gives these to the functionality f.
  - **Abort Handling:** sample an RSS share y' uniformly at random and complete the interaction with $\mathsf{Sim}_\perp$. If the interaction aborts, send $\perp$ to f.
- If the above interaction has not aborted, $\mathsf{Sim}_{\mathsf{fair}}$ sends $\top$ to the functionality f and receives $f(\vec{x})$.
- $\mathsf{Sim}_{\mathsf{fair}}$ uses share-patching to generate $\{y', y_1, y_2\} \leftarrow \mathsf{RSS.Patch}(f(\vec{x}), y')$, and sends (y_1, y_2) to $\mathcal{A}$.

Reduction. The indistinguishability of the joint view of the adversary in the real and ideal execution can be proven via a reduction to the security of $\Pi_\perp$. That is, consider for the sake of contradiction that there exists a PPT distinguisher $\mathsf{D}_{\mathsf{fair}}$ that can differentiate with non-negligible probability ϵ between the view of $\mathcal{A}$ produced by $\mathsf{Sim}_{\mathsf{fair}}$ (Algorithm 38) and that in Π_{fair} (Algorithm 11). Then this can be used in a black-box way by a PPT distinguisher $\mathsf{D}_\perp$ to distinguish between the view produced by $\mathsf{Sim}_\perp$ and that in $\Pi_\perp$ (realizing functionality f' in Equation (1)) with the same advantage. The distinguisher $\mathsf{D}_\perp$ would work as follows:

- $\mathsf{D}_\perp$ receives $\mathbf{View}_\perp^{\mathcal{A}}$ from the challenger and needs to output a bit indicating whether this view is $\mathbf{View}_{\Pi_\perp}^{\mathcal{A}}$ or $\mathbf{View}_{\mathsf{Sim}_\perp}^{\mathcal{A}}$, generated in the real or ideal interaction respectively. Note that $\mathbf{View}_\perp^{\mathcal{A}}$ contains both the view of the adversary and the output that the honest parties receive.
- $\mathsf{D}_\perp$ uses all the functionality outputs $[f(\vec{x})]$ of f' and reconstructs $f(\vec{x}) \leftarrow \mathsf{RSS.Recon}([f(\vec{x})])$. It sets $\mathbf{View}_{\mathsf{fair}}^{\mathcal{A}}$ by augmenting the adversary's view with the honest parties' shares in $[f(\vec{x})]$, and setting $f(\vec{x})$ as the output.
  Note that if $\mathbf{View}_\perp^{\mathcal{A}} = \mathbf{View}_{\Pi_\perp}^{\mathcal{A}}$, then $\mathbf{View}_{\mathsf{fair}}^{\mathcal{A}} = \mathbf{View}_{\Pi_{\mathsf{fair}}}^{\mathcal{A}}$. Otherwise, $\mathbf{View}_{\mathsf{fair}}^{\mathcal{A}} = \mathbf{View}_{\mathsf{Sim}_{\mathsf{fair}}}^{\mathcal{A}}$.
- $\mathsf{D}_\perp$ then sends $\mathbf{View}_{\mathsf{fair}}^{\mathcal{A}}$ to $\mathsf{D}_{\mathsf{fair}}$ and outputs whatever it outputs.

Note that in the above, the advantage of $\mathsf{D}_\perp$ is the same as that of $\mathsf{D}_{\mathsf{fair}}$, which is non-negligible. However, since $\Pi_\perp$ is a secure protocol realizing f', no such $\mathsf{D}_\perp$ can exist, yielding a contradiction.

F Proof of Theorem 6

Proof. We need to show that the protocol presented in Algorithm 16 realizes the required layered functionality f in the presence of a malicious, adaptive, rushing adversary $\mathcal{A}$ that can corrupt a minority of parties in $\mathcal{L}_0$, all but one parties in $\mathcal{L}_{\ell+1}$, and one party in each intermediate layer $\mathcal{L}_i$, for all $i \in [\ell]$ with guaranteed output delivery. As discussed in Appendix D, it suffices to design protocols that are only selectively secure and these can be lifted to adaptive security in the layered setting using secure erasures. Going forward, we present a proof that the protocol Π_{GOD} in Algorithm 16 is selectively secure in the presence of a malicious rushing adversary $\mathcal{A}$ that can corrupt a minority of parties in $\mathcal{L}_0$, all but one parties in $\mathcal{L}_{\ell+1}$, and one party in each intermediate layer $\mathcal{L}_i$, for all $i \in [\ell]$.

The proof for this follows from a PPT simulation algorithm $\mathsf{Sim}_{\mathsf{GOD}}$ that we define for the protocol Π_{GOD}. Before detailing the simulator itself, let us first formalize essential notation below.

Notation. Let $\mathcal{F}_{\mathsf{Setup}}^{\mathsf{GOD}}$ be the functionality given in Algorithm 12. Since we show that Π_{GOD} realizes f in the $\mathcal{F}_{\mathsf{Setup}}^{\mathsf{GOD}}$-hybrid, our simulator $\mathsf{Sim}_{\mathsf{GOD}}$ needs to also internally emulate $\mathcal{F}_{\mathsf{Setup}}^{\mathsf{GOD}}$ and the messages that the adversary needs to receive from it. Let $\mathsf{DEHE} = (\mathsf{KeyGen}, \mathsf{Gen}, \mathsf{Enc}, \mathsf{Eval})$ be a two-party DEHE scheme (Definition 1). With respect to this, let $\mathsf{Sim}_{\mathsf{DEHE}}(\cdot)$ be the PPT simulator for DEHE that, for $n, \ell \in \mathbb{N}$, outputs

$$(\{\mathsf{ek}_{i,k}\}_{k \in K_i, i \in [\ell]}, \{\mathsf{ct}_i\}_{i \in [n]}) \leftarrow \mathsf{Sim}_{\mathsf{DEHE}}(1^\kappa, n, 3\ell)$$

where n is the DEHE input size, K_i are the threads running in layer i and ℓ is the number of layers in the computation.

Let $\mathcal{S} = (\mathsf{Gen}, \mathsf{Sign}, \mathsf{Verify})$ be a signature scheme (Definition 14). for the protocol Π_{GOD} in Algorithm 16, let $\mathsf{Sim}_{\mathsf{GOD}}$ be the PPT algorithm that simulates the view produced by an interaction with a PPT malicious adversary $\mathcal{A}$ corrupting a minority of parties in $\mathcal{L}_0$, all but one parties in $\mathcal{L}_{\ell+1}$, and one party in each intermediate layer $\mathcal{L}_i$, for all $i \in [\ell]$. for each $i \in [\ell+1] \cup \{0\}$ let $\mathcal{H}_i \subseteq \mathcal{L}_i$ be the set of honest parties in layer $\mathcal{L}_i$. Let $\mathcal{C}_i = \mathcal{L}_i - \mathcal{H}_i$ be the set of corrupt parties in this layer. We additionally say that $k \in \mathsf{Corrupt}[i]$ if thread $k \in K_i$ contains a corrupt party, and $k \in \mathsf{Honest}[i]$ otherwise.

Simulation. The simulator $\mathsf{Sim}_{\mathsf{GOD}}$ is detailed in Algorithms 39 and 40. It uses the DEHE simulator $\mathsf{Sim}_{\mathsf{DEHE}}$ given above in a black-box way.

Hybrids. It remains to prove that the joint view produced by the simulator $\mathsf{Sim}_{\mathsf{GOD}}$ is computationally indistinguishable from that of the real execution of the protocol. We do so by considering the following intermediate hybrid experiments:

- H_0 : This hybrid experiment is defined to be the same as the interaction of the adversary $\mathcal{A}$ in the simulation with $\mathsf{Sim}_{\mathsf{GOD}}$.
- H_1 : This is an intermediate hybrid experiment that differs from the simulation $\mathsf{Sim}_{\mathsf{GOD}}$ in the following way: any of the executions only abort if the **consistency checks** given in Algorithms 13 to 16 (performed on the values $\mathsf{Sim}_{\mathsf{GOD}}$ internally computed for the honest party and those received from $\mathcal{A}$) would cause one. In all other aspects, H_1 and the previous hybrid are the same. This experiment is detailed in Algorithms 41 and 42. Looking ahead, we can show that the view produced by this hybrid experiment H_1 is computationally indistinguishable from that of H_0 via a reduction to the unforgeability property of the digital signature scheme $\mathcal{S}$.

Figure 39 Simulator $\mathsf{Sim}_{\mathsf{GOD}}$ for Layer protocol Π_{GOD} in the $\mathcal{F}_{\mathsf{Setup}}^{\mathsf{GOD}}$-hybrid

Let $\mathsf{DEHE} = (\mathsf{KeyGen}, \mathsf{Gen}, \mathsf{Enc}, \mathsf{Eval})$ be a two-party DEHE scheme (Definition 1). Let $\mathcal{S}$ be a signature scheme (Definition 14) and F be a PRF (Definition 12).

1. $\mathsf{Sim}_{\mathsf{GOD}}$ first emulates $\mathcal{F}_{\mathsf{Setup}}$ and does the following:
- generate $(\mathsf{sk}, \mathsf{vk}) \leftarrow \mathcal{S}.\mathsf{Gen}(1^\kappa)$ and let $n = |\text{PRF key}| + |\text{DEHE key}| + |\mathsf{sk}|$
- compute $(\{\mathsf{ek}^*_{i+1,b,k,k'}\}_{S^{i+1}_{b,k'} \in \mathcal{C}_{i+1}, k \in K_i, k' \in K_{i+1}, i \in [\ell-1]}, \{\mathsf{ek}^*_{1,b,k}\}_{S^1_{b,k} \in \mathcal{C}_1, k \in K_1}, \mathsf{ct}^*_\alpha, \mathsf{ct}^*_{\mathsf{k}}, \mathsf{ct}^*_{\mathsf{sk}}, \{\mathsf{ct}^*_{j,0}, \mathsf{ct}^*_{j,1}\}_{j \in [\ell_1]}) \leftarrow \mathsf{Sim}_{\mathsf{DEHE}}(1^\kappa, n + 2\ell_1, 6\ell)$
- $\forall j \in [\ell_1], b \in \{0,1\}$, compute $\sigma^{\mathsf{ct}*}_{b,j} \leftarrow \mathcal{S}.\mathsf{Sign}(\mathsf{sk}, \mathsf{ct}^*_{j,b})$
- $\forall S^1_{b,k} \in \mathcal{C}_1$, sample $\{\langle \mathsf{ek}^1_{b,k} \rangle_j\}_{j \in [\ell_1]}$ s.t. $\mathsf{ek}^*_{1,b,k} = \mathsf{Reconstruct}(\{\langle \mathsf{ek}^1_{b,k} \rangle_j\}_{j \in [\ell_1]})$, sample shares $\{\langle \mathsf{ek}^1_{b',k} \rangle_j\}_{j \in [\ell_1], k \in K_1}$ uniformly at random for every $S^1_{b',k} \in \mathcal{H}_1$
- for $b' \in \{0,1\}, k \in K_1, j \in [\ell_1]$, compute $\sigma_{b',k,j} = \mathcal{S}.\mathsf{Sign}(\mathsf{sk}, \langle \mathsf{ek}^1_{b',k} \rangle_j)$
- $\mathcal{A}$ receives, for each corrupt $C_j \in \mathcal{C}_0$, the following tuple from $\mathsf{Sim}_{\mathsf{GOD}}$:

$$\{b' : \mathsf{ct}^*_{j,b'}, \sigma^{\mathsf{ct}*}_{j,b'}\}_{b' \in \{0,1\}}, \mathsf{vk}, \mathsf{ct}^*_\alpha, \mathsf{ct}^*_{\mathsf{k}}, \mathsf{ct}^*_{\mathsf{sk}}, \{\langle \mathsf{ek}^1_{b',k} \rangle_j, \sigma_{b',k,j}\}_{k \in K_1, b' \in \{0,1\}}$$

2. $\mathsf{Sim}_{\mathsf{GOD}}$, emulating the first round of π_{GOD}, does the following:
- for each $C_j \in \mathcal{H}_0$, sample $x_j \leftarrow \{0,1\}$ and send to $\mathcal{A}$ the tuple: $\{\langle \mathsf{ek}^1_{b,k} \rangle_j, \sigma_{b,k,j}\}_{b \in \{0,1\}, k \in K_1}, \mathsf{ct}^*_{j,x_j}, \sigma^{\mathsf{ct}*}_{j,x_j}, \mathsf{vk}, \mathsf{ct}^*_\alpha, \mathsf{ct}^*_{\mathsf{k}}, \mathsf{ct}^*_{\mathsf{sk}}$
- **Input Extraction:** On behalf of every $S^1_{b'} \in \mathcal{H}_1$, $\mathsf{Sim}_{\mathsf{GOD}}$ accepts from $\mathcal{A}$, for each $C_j \in \mathcal{C}_0$ the tuple, $(\mathsf{ct}_{j,x_j}, \sigma^{\mathsf{ct}}_{j,x_j})$. Defines $y^*_0, \mathsf{ct}^*_{y'_0} = \emptyset$. for each $C_j \in \mathcal{C}_0$:
  - if $\mathsf{ct}_{j,x_j} \notin \{\mathsf{ct}^*_{j,0}, \mathsf{ct}^*_{j,1}\}$, then set $x_j = \mathsf{default}[j]$ and add $\{j : x_j\}$ to y^*_0.
  - else, set $x_j = b$ if it received $\mathsf{ct}^*_{j,b}$. Add $\{j : \mathsf{ct}^*_{j,b}\}$ to $\mathsf{ct}^*_{y'_0}$.

  Finally, send $\{x_j\}_{C_j \in \mathcal{C}_0}$ as the adversary's input to $\mathcal{F}_{\mathsf{GOD}}$
- $\mathsf{Sim}_{\mathsf{GOD}}$ (emulating protocol Π_1) internally computes, $(\{[\mathsf{ek}'_{0,k,k'}]_b, [\sigma^{\mathsf{ek}}_{0,k,k'}]_b, [\mathsf{ek}'_{1,k,k'}]_b, [\sigma^{\mathsf{ek}}_{1,k,k'}]_b\}_{k' \in K_{i+1}}, [y_1]_{b,k}, [\sigma_1]_{b,k}) \leftarrow \mathsf{DEHE.Eval}(b, \mathsf{ek}^*_{1,b,k}, (\mathsf{ct}^*_\alpha, \mathsf{ct}^*_{\mathsf{k}}, \mathsf{ct}^*_{\mathsf{sk}}, \mathsf{ct}^*_{y'_0}), P_{1,k}(1, k, y^*_0))$ as the corrupt $S^1_{b,k}$ would.

3. $\mathsf{Sim}_{\mathsf{GOD}}$ samples $\{y_i\}_{i \in [\ell-1]}$ uniformly at random and does the following:
- for $i \in [\ell-1]$, compute $\sigma_i = \mathcal{S}.\mathsf{Sign}(\mathsf{sk}, y_i)$ and for every $S^{i+1}_{b,k'} \in \mathcal{C}_{i+1}, k \in K_i, k' \in K_{i+1}$, compute $\sigma^{\mathsf{ek}}_{i+1,b,k,k'} = \mathcal{S}.\mathsf{Sign}(\mathsf{sk}, \mathsf{ek}^*_{i+1,b,k,k'})$.

- H_2 : This is an intermediate hybrid experiment that differs from the previous hybrid in the following way: the honest parties' inputs $\{x_j\}_{C_j \in \mathcal{H}_0}$ are used, along with the corrupt parties' inputs $\{x^*_j\}_{C_j \in \mathcal{C}_0}$ that were extracted, to compute the intermediate computation values $\{y_i\}_{i \in [\ell]}$ using the masks $\{r_i\}_{i \in [\ell-1]}$ – like in the real execution. The masks $\{r_i\}_{i \in [\ell-1]}$ are sampled uniformly at random and these honest parties' inputs are also used when choosing input ciphertexts in y_0. In all other aspects, H_2 and the previous hybrid are the same. This experiment is detailed in Algorithms 43 and 44. The view produced by this hybrid experiment H_2 is identically distributed to that of H_1 since the masks act as one-time-pads making the intermediate values in $\{y_i\}_{i \in [\ell]}$ be uniform random strings. The choice of initial simulated ciphertexts using $\{x_j\}_{C_j \in \mathcal{H}_0}$ (as opposed to uniformly at random) also is identically distributed as the ciphertexts themselves are uncorrelated from the choice.
- H_3 : This is an intermediate hybrid experiment that differs from the previous hybrid in the following way: instead of using $\mathsf{Sim}_{\mathsf{DEHE}}$ to simulate the corrupt evaluation keys and all ciphertexts, sample the PRF key α and compute the real DEHE key k, evaluation keys $\{\mathsf{ek}^*_i\}_{i \in [\ell]}$ (using fresh randomness), and ciphertexts – like in the real execution. In all other aspects, H_3 and the previous hybrid are the same. This experiment is detailed in Algorithms 45 and 46. We can show that the

Figure 40 Simulator $\mathsf{Sim}_{\mathsf{GOD}}$ – continued

Let $\mathsf{DEHE} = (\mathsf{KeyGen}, \mathsf{Gen}, \mathsf{Enc}, \mathsf{Eval})$ be a two-party DEHE scheme (Definition 1). Let $\mathcal{S}$ be a signature scheme (Definition 14) and F be a PRF (Definition 12).

4. $\mathsf{Sim}_{\mathsf{GOD}}$, $\forall i \in [\ell - 1]$, emulating π_{GOD}'s i^{th} computation round does the following:

- for each thread $k \in \mathsf{Corrupt}[i]$, having a corrupt party $S^i_{b,k}$, do the following:
  - compute $[y_i]_{\neg b,k} = y_i - [y_i]_{b,k}$ and $[\sigma_i]_{\neg b,k} = \sigma_i - [\sigma_i]_{b,k}$
  - for every $k' \in \mathsf{Corrupt}[i+1]$, with corrupt party $S^{i+1}_{b',k'}$, compute $[\mathsf{ek}_{b',k,k'}]_{\neg b} = \mathsf{ek}^*_{i+1,b',k,k'} - [\mathsf{ek}'_{b',k,k'}]_b$ and $[\sigma^{\mathsf{ek}}_{b',k,k'}]_{\neg b} = \sigma_{i,b',k,k'} - [\sigma^{\mathsf{ek}}_{b',k,k'}]_b$
- for each thread $k \in \mathsf{Honest}[i]$, do the following:
  - sample $([y_i]_{0,k}, [y_i]_{1,k})$ such that $[y_i]_{0,k} + [y_i]_{1,k} = y_i$.
  - for $b \in \{0,1\}$, compute signatures $[\sigma_i]_{b,k} = \mathcal{S}.\mathsf{Sign}(\mathsf{sk}, [y_i]_{b,k})$
  - for every $k' \in \mathsf{Corrupt}[i+1]$, with corrupt party $S^{i+1}_{b',k'}$, sample $([\mathsf{ek}_{b',k,k'}]_0, [\mathsf{ek}'_{b',k,k'}]_1)$ such that $[\mathsf{ek}'_{b',k,k'}]_0 + [\mathsf{ek}'_{b',k,k'}]_1 = \mathsf{ek}^*_{i+1,b',k,k'}$, and for $b \in \{0,1\}$, compute signatures $[\sigma^{\mathsf{ek}}_{b',k,k'}]_b = \mathcal{S}.\mathsf{Sign}(\mathsf{sk}, [\mathsf{ek}'_{b',k,k'}]_b)$
- send the following to $\mathcal{A}$:
  - Corresponding to every honest thread $k \in \mathsf{Honest}[i]$, send $\{[y_i]_{b,k}, [\sigma_i]_{b,k}, [\mathsf{ek}'_{b',k,k'}]_b, [\sigma^{\mathsf{ek}}_{b',k,k'}]_b\}_{b\in\{0,1\}}$.
  - Corresponding to every corrupt thread $k \in \mathsf{Corrupt}[i]$, with corrupt party $S^i_{b,k}$, send $([y_i]_{\neg b,k}, [\sigma_i]_{\neg b,k}, \{[\mathsf{ek}_{b',k'}]_{\neg b}, [\sigma^{\mathsf{ek}}_{b',k'}]_{\neg b}\}_{S^{i+1}_{b',k'} \in \mathcal{C}_{i+1}})$.
  - Additionally send $(\mathsf{vk}, \mathsf{ct}^*_\alpha, \mathsf{ct}^*_{\mathsf{k}}, \mathsf{ct}^*_{\mathsf{sk}})$.
- $\mathsf{Sim}_{\mathsf{GOD}}$ accepts $([y_i]_{b,k}, [\sigma_i]_{b,k}, [\mathsf{ek}'_{\neg b',k'}]_b, [\sigma^{\mathsf{ek}}_{\neg b',k'}]_b, \mathsf{vk}, \mathsf{ct}^*_\alpha, \mathsf{ct}^*_{\mathsf{k}}, \mathsf{ct}^*_{\mathsf{sk}})$ from $\mathcal{A}$ for each corrupt $S^i_{b,k}, k \in \mathsf{Corrupt}[i]$ and if it does not match internal computation, abort this thread.
- Choose $k \in K_i$ to be lexicographically the smallest thread that has not been aborted and use values corresponding to this thread for layer $i+1$.
- $\mathsf{Sim}_{\mathsf{GOD}}$ (emulating protocol Π_{i+1}) computes, $(\{[\mathsf{ek}'_{0,k,k'}]_b, [\sigma^{\mathsf{ek}}_{0,k,k'}]_b, [\mathsf{ek}'_{1,k,k'}]_b, [\sigma^{\mathsf{ek}}_{1,k,k'}]_b\}_{k'\in K_{i+2}}, [y_{i+1}]_{b,k}, [\sigma_{i+1}]_{b,k}) \leftarrow \mathsf{DEHE.Eval}(b, \mathsf{ek}_{b,k}, (\mathsf{ct}_\alpha, \mathsf{ct}_{\mathsf{k}}, \mathsf{ct}_{\mathsf{sk}}), P_{i+1,k}(i+1, k, y_i))$ as the corrupt party $S^{i+1}_{b,k}$ would.

5. Before the final round, $\mathsf{Sim}_{\mathsf{GOD}}$ receives from $\mathcal{F}_{\mathsf{GOD}}$ the functionality output $f(x)$.

- set $y_\ell = f(x)$ and $\sigma_\ell = \mathcal{S}.\mathsf{Sign}(\mathsf{sk}, y_\ell)$
- for each thread $k \in \mathsf{Corrupt}[\ell]$, having a corrupt party $S^\ell_{b,k}$, do the following:
  - compute $[y_\ell]_{\neg b,k} = y_\ell - [y_\ell]_{b,k}$ and $[\sigma_\ell]_{\neg b,k} = \sigma_\ell - [\sigma_\ell]_{b,k}$
- for each thread $k \in \mathsf{Honest}[\ell]$, do the following:
  - sample $[y_\ell]_{0,k}, [y_\ell]_{1,k}$ such that $[y_\ell]_{0,k} + [y_\ell]_{1,k} = y_\ell$.
  - for $b \in \{0,1\}$, compute signatures $[\sigma_\ell]_{b,k} = \mathcal{S}.\mathsf{Sign}(\mathsf{sk}, [y_\ell]_{b,k})$.
- send the following to $\mathcal{A}$:
  - Corresponding to every honest thread $k \in \mathsf{Honest}[\ell]$, send $\{[y_\ell]_{b,k}, [\sigma_\ell]_{b,k}\}_{b\in\{0,1\}}$. Additionally send vk.
  - Corresponding to every corrupt thread $k \in \mathsf{Corrupt}[\ell]$, with corrupt party $S^i_{b,k}$, send $([y_\ell]_{\neg b,k}, [\sigma_\ell]_{\neg b,k})$.
- $\forall P^{\ell+1}_j \in \mathcal{H}_{\ell+1}$, $\mathsf{Sim}_{\mathsf{GOD}}$ accepts $(\{[y_\ell]_{b,k}, [\sigma_\ell]_{b,k}\}_{S^\ell_{b,k}\in K_\ell, k\in\mathsf{Corrupt}[\ell]})$ from $\mathcal{A}$ and, if this is not the same as that internally computed, it aborts this thread.

view produced by this hybrid experiment H_3 is computationally indistinguishable from that of H_2 via a reduction to the multi-instance selective security of DEHE.

– H_4 : This experiment is the same as the real execution. Note that this differs from the previous experiment only in that all the masks $\{r_i\}_{i\in[\ell-1]}$, the randomness used to re-generate the DEHE evaluation keys and the signing randomness within the RMS programs is pseudorandomly generated from the PRF F keyed with α. We can show that the view produced by this hybrid experiment H_4 is computationally indistinguishable from that of H_3 via a reduction to the pseudorandomness of F.

Figure 41 Hybrid Experiment H_1

Let $\mathsf{DEHE} = (\mathsf{KeyGen}, \mathsf{Gen}, \mathsf{Enc}, \mathsf{Eval})$ be a two-party DEHE scheme (Definition 1). Let $\mathcal{S}$ be a signature scheme (Definition 14) and F be a PRF (Definition 12).

1. H_1 first emulates $\mathcal{F}_{\mathsf{Setup}}$ and does the following:
- generate $(\mathsf{sk}, \mathsf{vk}) \leftarrow \mathcal{S}.\mathsf{Gen}(1^\kappa)$ and let $n = |\text{PRF key}| + |\text{DEHE key}| + |\mathsf{sk}|$
- compute $(\{\mathsf{ek}^*_{i+1,b,k,k'}\}_{S^{i+1}_{b,k'} \in \mathcal{C}_{i+1}, k \in K_i, k' \in K_{i+1}, i \in [\ell-1]}, \{\mathsf{ek}^*_{1,b,k}\}_{S^1_{b,k} \in \mathcal{C}_1, k \in K_1}, \mathsf{ct}^*_\alpha, \mathsf{ct}^*_{\mathsf{k}}, \mathsf{ct}^*_{\mathsf{sk}}, \{\mathsf{ct}^*_{j,0}, \mathsf{ct}^*_{j,1}\}_{j \in [\ell_1]}) \leftarrow \mathsf{Sim}_{\mathsf{DEHE}}(1^\kappa, n + 2\ell_1, 6\ell)$
- $\forall j \in [\ell_1], b \in \{0,1\}$, compute $\sigma^{\mathsf{ct}*}_{b,j} \leftarrow \mathcal{S}.\mathsf{Sign}(\mathsf{sk}, \mathsf{ct}^*_{j,b})$
- $\forall S^1_{b,k} \in \mathcal{C}_1$, sample $\{\langle \mathsf{ek}^1_{b,k} \rangle_j\}_{j \in [\ell_1]}$ st $\mathsf{ek}^*_{1,b,k} = \mathsf{Reconstruct}(\{\langle \mathsf{ek}^1_{b,k} \rangle_j\}_{j \in [\ell_1]})$, sample $\{\langle \mathsf{ek}^1_{b',k} \rangle_j\}_{j \in [\ell_1], k \in K_1}$ uniformly at random for every $S^1_{b',k} \in \mathcal{H}_1$
- for $b' \in \{0,1\}, k \in K_1, j \in [\ell_1]$, compute $\sigma_{b',k,j} = \mathcal{S}.\mathsf{Sign}(\mathsf{sk}, \langle \mathsf{ek}^1_{b',k} \rangle_j)$
- $\mathcal{A}$ receives, for each corrupt $C_j \in \mathcal{C}_0$, the following tuple from $\mathsf{Sim}_{\mathsf{GOD}}$:

$$\{b' : \mathsf{ct}^*_{j,b'}, \sigma^{\mathsf{ct}*}_{j,b'}\}_{b' \in \{0,1\}}, \mathsf{vk}, \mathsf{ct}^*_\alpha, \mathsf{ct}^*_{\mathsf{k}}, \mathsf{ct}^*_{\mathsf{sk}}, \{\langle \mathsf{ek}^1_{b',k} \rangle_j, \sigma_{b',k,j}\}_{k \in K_1, b' \in \{0,1\}}$$

2. H_1, emulating the first round of π_{GOD}, does the following:
- for each $C_j \in \mathcal{H}_0$, sample $x_j \leftarrow \{0,1\}$ and send to $\mathcal{A}$ the tuple: $\{\langle \mathsf{ek}^1_{b,k} \rangle_j, \sigma_{b,k,j}\}_{b \in \{0,1\}, k \in K_1}, \mathsf{ct}^*_{j,x_j}, \sigma^{\mathsf{ct}*}_{j,x_j}, \mathsf{vk}, \mathsf{ct}^*_\alpha, \mathsf{ct}^*_{\mathsf{k}}, \mathsf{ct}^*_{\mathsf{sk}}$
- **Input Extraction:** On behalf of every $S^1_{b'} \in \mathcal{H}_1$, H_1 accepts from $\mathcal{A}$, for each $C_j \in \mathcal{C}_0$ the tuple, $(\mathsf{ct}_{j,x_j}, \sigma^{\mathsf{ct}}_{j,x_j})$. Defines $y^*_0, \mathsf{ct}^*_{y'_0} = \emptyset$. for each $C_j \in \mathcal{C}_0$:
  - if $\mathsf{ct}_{j,x_j} \notin \{\mathsf{ct}^*_{j,0}, \mathsf{ct}^*_{j,1}\}$, then set $x_j = \mathsf{default}[j]$ and add $\{j : x_j\}$ to y^*_0.
  - else, set $x_j = b$ if it received $\mathsf{ct}^*_{j,b}$. Add $\{j : \mathsf{ct}^*_{j,b}\}$ to $\mathsf{ct}^*_{y'_0}$.

  Finally, send $\{x_j\}_{C_j \in \mathcal{C}_0}$ as the adversary's input to $\mathcal{F}_{\mathsf{GOD}}$
- H_1 (emulating protocol Π_1) internally computes, $(\{[\mathsf{ek}'_{0,k,k'}]_b, [\sigma^{\mathsf{ek}}_{0,k,k'}]_b, [\mathsf{ek}'_{1,k,k'}]_b, [\sigma^{\mathsf{ek}}_{1,k,k'}]_b\}_{k' \in K_{i+1}}, [y_1]_{b,k}, [\sigma_1]_{b,k}) \leftarrow \mathsf{DEHE.Eval}(b, \mathsf{ek}^*_{1,b,k}, (\mathsf{ct}^*_\alpha, \mathsf{ct}^*_{\mathsf{k}}, \mathsf{ct}^*_{\mathsf{sk}}, \mathsf{ct}^*_{y'_0}), P_{1,k}(1, k, y^*_0))$ as the corrupt $S^1_{b,k}$ would.

3. H_1 samples $\{y_i\}_{i \in [\ell-1]}$ uniformly at random and does the following:
- for $i \in [\ell-1]$, compute $\sigma_i = \mathcal{S}.\mathsf{Sign}(\mathsf{sk}, y_i)$ and for every $S^{i+1}_{b,k'} \in \mathcal{C}_{i+1}, k \in K_i, k' \in K_{i+1}$, compute $\sigma^{\mathsf{ek}}_{i+1,b,k,k'} = \mathcal{S}.\mathsf{Sign}(\mathsf{sk}, \mathsf{ek}^*_{i+1,b,k,k'})$.

Claim. Assuming $\mathcal{S}$ is a digital signature scheme (Definition 14) satisfying unforgeability, $\mathsf{H}_0 \overset{c}{\approx} \mathsf{H}_1$.

Proof. Recall that the hybrid experiment H_0 behaves the same way as $\mathsf{Sim}_{\mathsf{GOD}}$ (Algorithms 39 and 40) and H_1 (Algorithms 41 and 42) differs only in that the consistency checks that H_1 performs on behalf of the honest parties involve verifying signatures on messages, even if they do not match the internally computed messages by the hybrid experiment. In contrast, in H_0, any execution that involves a corrupt party in the computation layer aborts when the adversary's messages are not the same as that the simulation computes. The only cases where H_0 aborts but H_1 does not would correspond to an instance where the adversary has forged a digital signature on a message of its choice, and has provided correct additive shares of the same, causing it to verify correctly on the honest party's end.

We can show that both experiments will produce exactly the same view, except with negligible probability by reduction to the unforgeability property of the underlying signature scheme $\mathcal{S}$ (Definition 14). Consider for the sake of contradiction that there exists a PPT adversary $\mathsf{D}_{\mathsf{GOD}}$ that can, with non-negligible probability ϵ, produce a view distributed as in H_1 but not in H_0. We show that if this were possible, then $\mathsf{D}_{\mathsf{GOD}}$ can be used in a black-box way by a PPT adversary $\mathsf{D}_{\mathcal{S}}$ to violate the unforgeability property of the signature scheme $\mathcal{S}$. This distinguisher $\mathsf{D}_{\mathcal{S}}$ would work as follows:

Figure 42 Hybrid 1 – continued

Let $\mathsf{DEHE} = (\mathsf{KeyGen}, \mathsf{Gen}, \mathsf{Enc}, \mathsf{Eval})$ be a two-party DEHE scheme (Definition 1). Let $\mathcal{S}$ be a signature scheme (Definition 14) and F be a PRF (Definition 12).

4. H_1, $\forall i \in [\ell - 1]$, emulating π_{GOD}'s i^{th} computation round does the following:

- for each thread $k \in \mathsf{Corrupt}[i]$, having a corrupt party $S^i_{b,k}$, do the following:
  - compute $[y_i]_{\neg b,k} = y_i - [y_i]_{b,k}$ and $[\sigma_i]_{\neg b,k} = \sigma_i - [\sigma_i]_{b,k}$
  - for every $k' \in \mathsf{Corrupt}[i+1]$, with corrupt party $S^{i+1}_{b',k'}$, compute $[\mathsf{ek}_{b',k,k'}]_{\neg b} = \mathsf{ek}^*_{i+1,b',k,k'} - [\mathsf{ek}'_{b',k,k'}]_b$ and $[\sigma^{\mathsf{ek}}_{b',k,k'}]_{\neg b} = \sigma_{i,b',k,k'} - [\sigma^{\mathsf{ek}}_{b',k,k'}]_b$
  - abort the thread if any **consistency checks** given in Π_i fail
- for each thread $k \in \mathsf{Honest}[i]$, do the following:
  - sample $([y_i]_{0,k}, [y_i]_{1,k})$ such that $[y_i]_{0,k} + [y_i]_{1,k} = y_i$.
  - for $b \in \{0,1\}$, compute signatures $[\sigma_i]_{b,k} = \mathcal{S}.\mathsf{Sign}(\mathsf{sk}, [y_i]_{b,k})$
  - for every $k' \in \mathsf{Corrupt}[i+1]$, with corrupt party $S^{i+1}_{b',k'}$, sample $([\mathsf{ek}_{b',k,k'}]_0, [\mathsf{ek}'_{b',k,k'}]_1)$ such that $[\mathsf{ek}'_{b',k,k'}]_0 + [\mathsf{ek}'_{b',k,k'}]_1 = \mathsf{ek}^*_{i+1,b',k,k'}$, and for $b \in \{0,1\}$, compute signatures $[\sigma^{\mathsf{ek}}_{b',k,k'}]_b = \mathcal{S}.\mathsf{Sign}(\mathsf{sk}, [\mathsf{ek}'_{b',k,k'}]_b)$
  - abort this thread if any **consistency checks** given in Π_i fail
- send the following to $\mathcal{A}$:
  - Corresponding to every honest thread $k \in \mathsf{Honest}[i]$, send $\{[y_i]_{b,k}, [\sigma_i]_{b,k}, [\mathsf{ek}'_{b',k,k'}]_b, [\sigma^{\mathsf{ek}}_{b',k,k'}]_b\}_{b\in\{0,1\}}$.
  - Corresponding to every corrupt thread $k \in \mathsf{Corrupt}[i]$, with corrupt party $S^i_{b,k}$, send $([y_i]_{\neg b,k}, [\sigma_i]_{\neg b,k}, \{[\mathsf{ek}_{b',k'}]_{\neg b}, [\sigma^{\mathsf{ek}}_{b',k'}]_{\neg b}\}_{S^{i+1}_{b',k'} \in \mathcal{C}_{i+1}})$.
  - Additionally send $(\mathsf{vk}, \mathsf{ct}^*_\alpha, \mathsf{ct}^*_{\mathsf{k}}, \mathsf{ct}^*_{\mathsf{sk}})$.
- H_1 accepts $([y_i]_{b,k}, [\sigma_i]_{b,k}, [\mathsf{ek}'_{\neg b',k'}]_b, [\sigma^{\mathsf{ek}}_{\neg b',k'}]_b, \mathsf{vk}, \mathsf{ct}^*_\alpha, \mathsf{ct}^*_{\mathsf{k}}, \mathsf{ct}^*_{\mathsf{sk}})$ from $\mathcal{A}$ for each corrupt $S^i_{b,k}, k \in \mathsf{Corrupt}[i]$.
- Choose $k \in K_i$ to be lexicographically the smallest thread that has not been aborted and use values corresponding to this thread for layer $i+1$.
- H_1 (emulating protocol Π_{i+1}) computes, $(\{[\mathsf{ek}'_{0,k,k'}]_b, [\sigma^{\mathsf{ek}}_{0,k,k'}]_b, [\mathsf{ek}'_{1,k,k'}]_b, [\sigma^{\mathsf{ek}}_{1,k,k'}]_b\}_{k'\in K_{i+2}}, [y_{i+1}]_{b,k}, [\sigma_{i+1}]_{b,k}) \leftarrow \mathsf{DEHE.Eval}(b, \mathsf{ek}_{b,k}, (\mathsf{ct}_\alpha, \mathsf{ct}_{\mathsf{k}}, \mathsf{ct}_{\mathsf{sk}}), P_{i+1,k}(i+1, k, y_i))$ as the corrupt party $S^{i+1}_{b,k}$ would.

5. Before the final round, H_1 receives from $\mathcal{F}_{\mathsf{GOD}}$ the functionality output $f(x)$.

- set $y_\ell = f(x)$ and $\sigma_\ell = \mathcal{S}.\mathsf{Sign}(\mathsf{sk}, y_\ell)$
- for each thread $k \in \mathsf{Corrupt}[\ell]$, having a corrupt party $S^\ell_{b,k}$, do the following:
  - compute $[y_\ell]_{\neg b,k} = y_\ell - [y_\ell]_{b,k}$ and $[\sigma_\ell]_{\neg b,k} = \sigma_\ell - [\sigma_\ell]_{b,k}$
- for each thread $k \in \mathsf{Honest}[\ell]$, do the following:
  - sample $[y_\ell]_{0,k}, [y_\ell]_{1,k}$ such that $[y_\ell]_{0,k} + [y_\ell]_{1,k} = y_\ell$.
  - for $b \in \{0,1\}$, compute signatures $[\sigma_\ell]_{b,k} = \mathcal{S}.\mathsf{Sign}(\mathsf{sk}, [y_\ell]_{b,k})$.
- send the following to $\mathcal{A}$:
  - Corresponding to every honest thread $k \in \mathsf{Honest}[\ell]$, send $\{[y_\ell]_{b,k}, [\sigma_\ell]_{b,k}\}_{b\in\{0,1\}}$. Additionally send vk.
  - Corresponding to every corrupt thread $k \in \mathsf{Corrupt}[\ell]$, with corrupt party $S^i_{b,k}$, send $([y_\ell]_{\neg b,k}, [\sigma_\ell]_{\neg b,k})$.
- $\forall P^{\ell+1}_j \in \mathcal{H}_{\ell+1}$, H_1 accepts $(\{[y_\ell]_{b,k}, [\sigma_\ell]_{b,k}\}_{S^\ell_{b,k}\in\mathcal{C}_\ell, k\in\mathsf{Corrupt}[\ell]})$ from $\mathcal{A}$, abort this thread if **consistency checks** given in Π_ℓ fail

- $\mathsf{D}_{\mathcal{S}}$ has a verification key vk and oracle access to a signing algorithm $\mathcal{O}_{\mathcal{S}}$ parameterized with the corresponding secret signing key sk. It uses this to sign messages and starts interacting with $\mathsf{D}_{\mathsf{GOD}}$ exactly as given in the simulation H_0 in Algorithms 39 and 40.
- If for any ($\mathcal{F}_{\mathsf{Setup}}$ or DEHE output) message that $\mathsf{D}_{\mathsf{GOD}}$ sends, it is not the case that it equals the message that $\mathsf{D}_{\mathcal{S}}$ has internally computed, reconstruct the value m and corresponding signature σ using both the honest and corrupt party's additive

shares. If $\top \leftarrow \mathcal{S}.\mathsf{Verify}(\mathsf{vk}, m, \sigma)$, output (m, σ) and continue the protocol as in H_1 (Algorithms 41 and 42). Otherwise, abort the protocol.

Note that in the above, if the execution corresponding to the corrupt party has aborted without $\mathsf{D}_{\mathcal{S}}$ outputting a forgery (m, σ), the view of $\mathsf{D}_{\mathsf{GOD}}$ is the same as that of H_0. Otherwise, it is the same as that in the experiment H_1. The advantage of $\mathsf{D}_{\mathcal{S}}$ is the same as that of $\mathsf{D}_{\mathsf{GOD}}$ not inducing the view of H_0, which is non-negligible by assumption. However, since $\mathcal{S}$ is an unforgeable digital signature scheme (Definition 14), no such $\mathsf{D}_{\mathcal{S}}$ can exist, yielding a contradiction.

Figure 43 Hybrid Experiment H_2

Let $\mathsf{DEHE} = (\mathsf{KeyGen}, \mathsf{Gen}, \mathsf{Enc}, \mathsf{Eval})$ be a two-party DEHE scheme (Definition 1). Let $\mathcal{S}$ be a signature scheme (Definition 14) and F be a PRF (Definition 12).

1. H_2 first emulates $\mathcal{F}_{\mathsf{Setup}}$ and does the following:
- generate $(\mathsf{sk}, \mathsf{vk}) \leftarrow \mathcal{S}.\mathsf{Gen}(1^\kappa)$ and let $n = |\text{PRF key}| + |\text{DEHE key}| + |\mathsf{sk}|$
- compute $(\{\mathsf{ek}^*_{i+1,b,k,k'}\}_{S^{i+1}_{b,k'} \in \mathcal{C}_{i+1}, k \in K_i, k' \in K_{i+1}, i \in [\ell-1]}, \{\mathsf{ek}^*_{1,b,k}\}_{S^1_{b,k} \in \mathcal{C}_1, k \in K_1}, \mathsf{ct}^*_\alpha, \mathsf{ct}^*_{\mathsf{k}}, \mathsf{ct}^*_{\mathsf{sk}}, \{\mathsf{ct}^*_{j,0}, \mathsf{ct}^*_{j,1}\}_{j \in [\ell_1]}) \leftarrow \mathsf{Sim}_{\mathsf{DEHE}}(1^\kappa, n + 2\ell_1, 6\ell)$
- $\forall j \in [\ell_1], b \in \{0,1\}$, compute $\sigma^{\mathsf{ct}*}_{b,j} \leftarrow \mathcal{S}.\mathsf{Sign}(\mathsf{sk}, \mathsf{ct}^*_{j,b})$
- $\forall S^1_{b,k} \in \mathcal{C}_1$, sample $\{\langle \mathsf{ek}^1_{b,k} \rangle_j\}_{j \in [\ell_1]}$ st $\mathsf{ek}^*_{1,b,k} = \mathsf{Reconstruct}(\{\langle \mathsf{ek}^1_{b,k} \rangle_j\}_{j \in [\ell_1]})$, sample $\{\langle \mathsf{ek}^1_{b',k} \rangle_j\}_{j \in [\ell_1], k \in K_1}$ uniformly at random for every $S^1_{b',k} \in \mathcal{H}_1$
- for $b' \in \{0,1\}, k \in K_1, j \in [\ell_1]$, compute $\sigma_{b',k,j} = \mathcal{S}.\mathsf{Sign}(\mathsf{sk}, \langle \mathsf{ek}^1_{b',k} \rangle_j)$
- $\mathcal{A}$ receives, for each corrupt $C_j \in \mathcal{C}_0$, the following tuple from $\mathsf{Sim}_{\mathsf{GOD}}$:

$$\{b' : \mathsf{ct}^*_{j,b'}, \sigma^{\mathsf{ct}*}_{j,b'}\}_{b' \in \{0,1\}}, \mathsf{vk}, \mathsf{ct}^*_\alpha, \mathsf{ct}^*_{\mathsf{k}}, \mathsf{ct}^*_{\mathsf{sk}}, \{\langle \mathsf{ek}^1_{b',k} \rangle_j, \sigma_{b',k,j}\}_{k \in K_1, b' \in \{0,1\}}$$

2. H_2, emulating the first round of π_{GOD}, does the following:
- for each $C_j \in \mathcal{H}_0$, for each $C_j \in \mathcal{H}_0$, using honest party's input $x_j \in \{0,1\}$, and send to $\mathcal{A}$ the tuple: $\{\langle \mathsf{ek}^1_{b,k} \rangle_j, \sigma_{b,k,j}\}_{b \in \{0,1\}, k \in K_1}, \mathsf{ct}^*_{j,x_j}, \sigma^{\mathsf{ct}*}_{j,x_j}, \mathsf{vk}, \mathsf{ct}^*_\alpha, \mathsf{ct}^*_{\mathsf{k}}, \mathsf{ct}^*_{\mathsf{sk}}$
- **Input Extraction:** On behalf of every $S^1_{b'} \in \mathcal{H}_1$, H_2 accepts from $\mathcal{A}$, for each $C_j \in \mathcal{C}_0$ the tuple, $(\mathsf{ct}_{j,x_j}, \sigma^{\mathsf{ct}}_{j,x_j})$. Defines $y^*_0, \mathsf{ct}^*_{y'_0} = \emptyset$. for each $C_j \in \mathcal{C}_0$:
  - if $\mathsf{ct}_{j,x_j} \notin \{\mathsf{ct}^*_{j,0}, \mathsf{ct}^*_{j,1}\}$, then set $x_j = \mathsf{default}[j]$ and add $\{j : x_j\}$ to y^*_0.
  - else, set $x_j = b$ if it received $\mathsf{ct}^*_{j,b}$. Add $\{j : \mathsf{ct}^*_{j,b}\}$ to $\mathsf{ct}^*_{y'_0}$.

  Finally, send $\{x_j\}_{C_j \in \mathcal{C}_0}$ as the adversary's input to $\mathcal{F}_{\mathsf{GOD}}$
- H_2 (emulating protocol Π_1) internally computes, $(\{[\mathsf{ek}'_{0,k,k'}]_b, [\sigma^{\mathsf{ek}}_{0,k,k'}]_b, [\mathsf{ek}'_{1,k,k'}]_b, [\sigma^{\mathsf{ek}}_{1,k,k'}]_b\}_{k' \in K_{i+1}}, [y_1]_{b,k}, [\sigma_1]_{b,k}) \leftarrow \mathsf{DEHE.Eval}(b, \mathsf{ek}^*_{1,b,k}, (\mathsf{ct}^*_\alpha, \mathsf{ct}^*_{\mathsf{k}}, \mathsf{ct}^*_{\mathsf{sk}}, \mathsf{ct}^*_{y'_0}), P_{1,k}(1, k, y^*_0))$ as the corrupt $S^1_{b,k}$ would.

3. H_2 samples $\{r_i\}_{i \in [\ell-1]}$ uniformly at random and does the following:
- for $i \in [\ell-1]$, compute $y_i = \begin{cases} \mathbf{C}_i(y_{i-1}) + r_i & \text{if } i = 1 \\ \mathbf{C}_i(y_{i-1} - r_{i-1}) + r_i & \text{if } i < \ell \\ \mathbf{C}_i(y_{i-1} - r_{i-1}) & \text{otherwise} \end{cases}$
- for $i \in [\ell-1]$, compute $\sigma_i = \mathcal{S}.\mathsf{Sign}(\mathsf{sk}, y_i)$ and for every $S^{i+1}_{b,k'} \in \mathcal{C}_{i+1}, k \in K_i, k' \in K_{i+1}$, compute $\sigma^{\mathsf{ek}}_{i+1,b,k,k'} = \mathcal{S}.\mathsf{Sign}(\mathsf{sk}, \mathsf{ek}^*_{i+1,b,k,k'})$.

Claim. Assuming DEHE is a secure two-server DEHE scheme (Definition 1) satisfying multi-instance selective security, $\mathsf{H}_2 \overset{\mathrm{c}}{\approx} \mathsf{H}_3$.

Proof. Consider for the sake of contradiction that there exists a PPT distinguisher $\mathsf{D}_{\mathsf{GOD}}$ that can differentiate with non-negligible probability ϵ between the view pro-

Figure 44 Hybrid Experiment H_2 – continued

Let $\mathsf{DEHE} = (\mathsf{KeyGen}, \mathsf{Gen}, \mathsf{Enc}, \mathsf{Eval})$ be a two-party DEHE scheme (Definition 1). Let $\mathcal{S}$ be a signature scheme (Definition 14) and F be a PRF (Definition 12).

4. H_2, $\forall i \in [\ell-1]$, emulating π_{GOD}'s i^{th} computation round does the following:

- for each thread $k \in \mathsf{Corrupt}[i]$, having a corrupt party $S^i_{b,k}$, do the following:
  - compute $[y_i]_{\neg b,k} = y_i - [y_i]_{b,k}$ and $[\sigma_i]_{\neg b,k} = \sigma_i - [\sigma_i]_{b,k}$
  - for every $k' \in \mathsf{Corrupt}[i+1]$, with corrupt party $S^{i+1}_{b',k'}$, compute $[\mathsf{ek}_{b',k,k'}]_{\neg b} = \mathsf{ek}^*_{i+1,b',k,k'} - [\mathsf{ek}'_{b',k,k'}]_b$ and $[\sigma^{\mathsf{ek}}_{b',k,k'}]_{\neg b} = \sigma_{i,b',k,k'} - [\sigma^{\mathsf{ek}}_{b',k,k'}]_b$
  - abort the thread if any **consistency checks** given in Π_i fail
- for each thread $k \in \mathsf{Honest}[i]$, do the following:
  - sample $([y_i]_{0,k}, [y_i]_{1,k})$ such that $[y_i]_{0,k} + [y_i]_{1,k} = y_i$.
  - for $b \in \{0,1\}$, compute signatures $[\sigma_i]_{b,k} = \mathcal{S}.\mathsf{Sign}(\mathsf{sk}, [y_i]_{b,k})$
  - for every $k' \in \mathsf{Corrupt}[i+1]$, with corrupt party $S^{i+1}_{b',k'}$, sample $([\mathsf{ek}_{b',k,k'}]_0, [\mathsf{ek}'_{b',k,k'}]_1)$ such that $[\mathsf{ek}'_{b',k,k'}]_0 + [\mathsf{ek}'_{b',k,k'}]_1 = \mathsf{ek}^*_{i+1,b',k,k'}$, and for $b \in \{0,1\}$, compute signatures $[\sigma^{\mathsf{ek}}_{b',k,k'}]_b = \mathcal{S}.\mathsf{Sign}(\mathsf{sk}, [\mathsf{ek}'_{b',k,k'}]_b)$
  - abort this thread if any **consistency checks** given in Π_i fail
- send the following to $\mathcal{A}$:
  - Corresponding to every honest thread $k \in \mathsf{Honest}[i]$, send $\{[y_i]_{b,k}, [\sigma_i]_{b,k}, [\mathsf{ek}'_{b',k,k'}]_b, [\sigma^{\mathsf{ek}}_{b',k,k'}]_b\}_{b\in\{0,1\}}$.
  - Corresponding to every corrupt thread $k \in \mathsf{Corrupt}[i]$, with corrupt party $S^i_{b,k}$, send $([y_i]_{\neg b,k}, [\sigma_i]_{\neg b,k}, \{[\mathsf{ek}_{b',k'}]_{\neg b}, [\sigma^{\mathsf{ek}}_{b',k'}]_{\neg b}\}_{S^{i+1}_{b',k'} \in \mathcal{C}_{i+1}})$.
  - Additionally send $(\mathsf{vk}, \mathsf{ct}^*_\alpha, \mathsf{ct}^*_{\mathsf{k}}, \mathsf{ct}^*_{\mathsf{sk}})$.
- H_2 accepts $([y_i]_{b,k}, [\sigma_i]_{b,k}, [\mathsf{ek}'_{\neg b',k'}]_b, [\sigma^{\mathsf{ek}}_{\neg b',k'}]_b, \mathsf{vk}, \mathsf{ct}^*_\alpha, \mathsf{ct}^*_{\mathsf{k}}, \mathsf{ct}^*_{\mathsf{sk}})$ from $\mathcal{A}$ for each corrupt $S^i_{b,k}, k \in \mathsf{Corrupt}[i]$.
- Choose $k \in K_i$ to be lexicographically the smallest thread that has not been aborted and use values corresponding to this thread for layer $i+1$.
- H_2 (emulating protocol Π_{i+1}) computes, $(\{[\mathsf{ek}'_{0,k,k'}]_b, [\sigma^{\mathsf{ek}}_{0,k,k'}]_b, [\mathsf{ek}'_{1,k,k'}]_b, [\sigma^{\mathsf{ek}}_{1,k,k'}]_b\}_{k'\in K_{i+2}}, [y_{i+1}]_{b,k}, [\sigma_{i+1}]_{b,k}) \leftarrow \mathsf{DEHE.Eval}(b, \mathsf{ek}_{b,k}, (\mathsf{ct}_\alpha, \mathsf{ct}_{\mathsf{k}}, \mathsf{ct}_{\mathsf{sk}}), P_{i+1,k}(i+1, k, y_i))$ as the corrupt party $S^{i+1}_{b,k}$ would.

5. Before the final round, H_2 receives from $\mathcal{F}_{\mathsf{GOD}}$ the functionality output $f(x)$.

- set $y_\ell = f(x)$ and $\sigma_\ell = \mathcal{S}.\mathsf{Sign}(\mathsf{sk}, y_\ell)$
- for each thread $k \in \mathsf{Corrupt}[\ell]$, having a corrupt party $S^\ell_{b,k}$, do the following:
  - compute $[y_\ell]_{\neg b,k} = y_\ell - [y_\ell]_{b,k}$ and $[\sigma_\ell]_{\neg b,k} = \sigma_\ell - [\sigma_\ell]_{b,k}$
- for each thread $k \in \mathsf{Honest}[\ell]$, do the following:
  - sample $[y_\ell]_{0,k}, [y_\ell]_{1,k}$ such that $[y_\ell]_{0,k} + [y_\ell]_{1,k} = y_\ell$.
  - for $b \in \{0,1\}$, compute signatures $[\sigma_\ell]_{b,k} = \mathcal{S}.\mathsf{Sign}(\mathsf{sk}, [y_\ell]_{b,k})$.
- send the following to $\mathcal{A}$:
  - Corresponding to every honest thread $k \in \mathsf{Honest}[\ell]$, send $\{[y_\ell]_{b,k}, [\sigma_\ell]_{b,k}\}_{b\in\{0,1\}}$. Additionally send vk.
  - Corresponding to every corrupt thread $k \in \mathsf{Corrupt}[\ell]$, with corrupt party $S^i_{b,k}$, send $([y_\ell]_{\neg b,k}, [\sigma_\ell]_{\neg b,k})$.
- $\forall P^{\ell+1}_j \in \mathcal{H}_{\ell+1}$, H_2 accepts $(\{[y_\ell]_{b,k}, [\sigma_\ell]_{b,k}\}_{S^\ell_{b,k}\in\mathcal{C}_\ell, k\in\mathsf{Corrupt}[\ell]})$ from $\mathcal{A}$, abort this thread if **consistency checks** given in Π_ℓ fail

duced by the hybrid experiments H_2 and H_3. Then this can be used in a black-box way by a PPT distinguisher $\mathsf{D}_{\mathsf{DEHE}}$ to violate the multi-instance selective security of DEHE for $m = 6\ell$, with the same advantage. This distinguisher $\mathsf{D}_{\mathsf{DEHE}}$ would work as follows:

- $\mathsf{D}_{\mathsf{DEHE}}$ gives a challenger 6ℓ and the size of the input $n + 2\ell_1$ (as defined in Algorithms 39 and 40) and receives the tuple $(\{\mathsf{ek}^*_{i+1,b,k,k'}\}_{S^{i+1}_{b,k'}\in\mathcal{C}_{i+1}, k\in K_i, k'\in K_{i+1}, i\in[\ell-1]}, \{\mathsf{ek}^*_{1,b,k}\}_{S^1_{b,k}\in\mathcal{C}_1, k\in K_1}, \mathsf{ct}^*_\alpha, \mathsf{ct}^*_{\mathsf{k}}, \mathsf{ct}^*_{\mathsf{sk}}, \{\mathsf{ct}^*_{j,0}, \mathsf{ct}^*_{j,1}\}_{j\in[\ell_1]})$.

– $\mathsf{D}_{\mathsf{DEHE}}$ then uses this tuple to complete the interaction in the hybrid experiment exactly as in H_2.
– Note that if $(\{\mathsf{ek}^*_{i+1,b,k,k'}\}_{S^{i+1}_{b,k'}\in\mathcal{C}_{i+1},k\in K_i,k'\in K_{i+1},i\in[\ell-1]}, \{\mathsf{ek}^*_{1,b,k}\}_{S^1_{b,k}\in\mathcal{C}_1,k\in K_1}, \mathsf{ct}^*_\alpha, \mathsf{ct}^*_{\mathsf{k}}, \mathsf{ct}^*_{\mathsf{sk}}, \{\mathsf{ct}^*_{j,0}, \mathsf{ct}^*_{j,1}\}_{j\in[\ell_1]})$ $\mathsf{Sim}_{\mathsf{DEHE}}$, the view produced would be identically distributed as H_2. Otherwise, it would be the same as that of H_3.
– Finally, $\mathsf{D}_{\mathsf{DEHE}}$ sends this view to $\mathsf{D}_{\mathsf{GOD}}$ and outputs whatever it would output.

Note that in the above, the advantage of $\mathsf{D}_{\mathsf{DEHE}}$ is the same as that of $\mathsf{D}_{\mathsf{GOD}}$, which is non-negligible. However, since DEHE is a secure DEHE scheme (Definition 1), no such $\mathsf{D}_{\mathsf{DEHE}}$ can exist, yielding a contradiction.

Figure 45 Hybrid Experiment H_3

Let $\mathsf{DEHE} = (\mathsf{KeyGen}, \mathsf{Gen}, \mathsf{Enc}, \mathsf{Eval})$ be a two-party DEHE scheme (Definition 1). Let $\mathcal{S}$ be a signature scheme (Definition 14) and F be a PRF (Definition 12).

1. H_3 first emulates $\mathcal{F}_{\mathsf{Setup}}$ and does the following:
– generate $(\mathsf{sk}, \mathsf{vk}) \leftarrow \mathcal{S}.\mathsf{Gen}(1^\kappa)$ and sample a PRF key $\alpha \leftarrow \{0,1\}^\kappa$
– generate $\mathsf{k} \leftarrow \mathsf{DEHE.KeyGen}(1^\kappa)$, for $k \in K_1$, $(\mathsf{ek}^1_{0,k}, \mathsf{ek}^i_{1,k}) \leftarrow \mathsf{DEHE.Gen}(\mathsf{k})$, and for $i \in [\ell-1]$, $k \in K_i$, $k' \in K_{i+1}$, $(\mathsf{ek}^i_{0,k,k'}, \mathsf{ek}^i_{1,k,k'}) \leftarrow \mathsf{DEHE.Gen}(\mathsf{k})$
– compute $(\mathsf{ct}^*_\alpha, \mathsf{ct}^*_{\mathsf{k}}, \mathsf{ct}^*_{\mathsf{sk}}) \leftarrow \mathsf{DEHE.Enc}(\mathsf{k}, (\alpha, \mathsf{k}, \mathsf{sk}))$
– $\forall j \in [\ell_1]$, $b \in \{0,1\}$, compute $\mathsf{ct}^*_{j,b} \leftarrow \mathsf{DEHE.Enc}(\mathsf{k}, b)$ and $\sigma^{\mathsf{ct}*}_{b,j} \leftarrow \mathcal{S}.\mathsf{Sign}(\mathsf{sk}, \mathsf{ct}^*_{j,b})$
– for $b \in \{0,1\}, k \in K_1, \{\langle \mathsf{ek}^1_{b,k}\rangle_j\}_{j\in[\ell_1]}$ st $\mathsf{ek}^1_{b,k} = \mathsf{Reconstruct}(\{\langle \mathsf{ek}^1_{b,k}\rangle_j\}_{j\in[\ell_1]}$
– for $b' \in \{0,1\}, k \in K_1, j \in [\ell_1]$, compute $\sigma_{b',k,j} = \mathcal{S}.\mathsf{Sign}(\mathsf{sk}, \langle \mathsf{ek}^1_{b',k}\rangle_j)$
– for $S^1_{b,k} \in \mathcal{C}_1, k \in K_1$, set $\mathsf{ek}^*_{1,b,k} = \mathsf{ek}^1_{b,k}$ and for all $i \in [\ell-1], S^{i+1}_{b,k'} \in \mathcal{C}_{i+1}, k \in K_i, k' \in K_{i+1}$, set $\mathsf{ek}^*_{i+1,b,k,k'} = \mathsf{ek}^{i+1}_{b,k,k'}$
– $\mathcal{A}$ receives, for each corrupt $C_j \in \mathcal{C}_0$, the following tuple:

$$\{b' : \mathsf{ct}^*_{j,b'}, \sigma^{\mathsf{ct}*}_{j,b'}\}_{b'\in\{0,1\}}, \mathsf{vk}, \mathsf{ct}^*_\alpha, \mathsf{ct}^*_{\mathsf{k}}, \mathsf{ct}^*_{\mathsf{sk}}, \{\langle \mathsf{ek}^1_{b'}\rangle_j, \sigma_{b',j}\}_{b'\in\{0,1\}}$$

2. H_3, emulating the first round of π_{GOD}, does the following:
– for each $C_j \in \mathcal{H}_0$, for each $C_j \in \mathcal{H}_0$, using honest party's input $x_j \in \{0,1\}$, and send to $\mathcal{A}$ the tuple:
$\{\langle \mathsf{ek}^1_{b,k}\rangle_j, \sigma_{b,k,j}\}_{b\in\{0,1\},k\in K_1}, \mathsf{ct}^*_{j,x_j}, \sigma^{\mathsf{ct}*}_{j,x_j}, \mathsf{vk}, \mathsf{ct}^*_\alpha, \mathsf{ct}^*_{\mathsf{k}}, \mathsf{ct}^*_{\mathsf{sk}}$
– **Input Extraction:** On behalf of every $S^1_{b'} \in \mathcal{H}_1$, H_2 accepts from $\mathcal{A}$, for each $C_j \in \mathcal{C}_0$ the tuple, $(\mathsf{ct}_{j,x_j}, \sigma^{\mathsf{ct}}_{j,x_j})$. Defines $y^*_0, \mathsf{ct}^*_{y'_0} = \emptyset$. for each $C_j \in \mathcal{C}_0$:
  • if $\mathsf{ct}_{j,x_j} \notin \{\mathsf{ct}^*_{j,0}, \mathsf{ct}^*_{j,1}\}$, then set $x_j = \mathsf{default}[j]$ and add $\{j : x_j\}$ to y^*_0.
  • else, set $x_j = b$ if it received $\mathsf{ct}^*_{j,b}$. Add $\{j : \mathsf{ct}^*_{j,b}\}$ to $\mathsf{ct}^*_{y'_0}$.
  Finally, send $\{x_j\}_{C_j\in\mathcal{C}_0}$ as the adversary's input to $\mathcal{F}_{\mathsf{GOD}}$
– H_2 (emulating protocol Π_1) internally computes, $(\{[\mathsf{ek}'_{0,k,k'}]_b, [\sigma^{\mathsf{ek}}_{0,k,k'}]_b, [\mathsf{ek}'_{1,k,k'}]_b, [\sigma^{\mathsf{ek}}_{1,k,k'}]_b\}_{k'\in K_{i+1}}, [y_1]_{b,k}, [\sigma_1]_{b,k}) \leftarrow \mathsf{DEHE.Eval}(b, \mathsf{ek}^*_{1,b,k}, (\mathsf{ct}^*_\alpha, \mathsf{ct}^*_{\mathsf{k}}, \mathsf{ct}^*_{\mathsf{sk}}, \mathsf{ct}^*_{y'_0}), P_{1,k}(1,k,y^*_0))$ as the corrupt $S^1_{b,k}$ would.

3. H_2 samples $\{r_i\}_{i\in[\ell-1]}$ uniformly at random and does the following:
– for $i \in [\ell-1]$, compute $y_i = \begin{cases} \mathbf{C}_i(y_{i-1}) + r_i & \text{if } i = 1 \\ \mathbf{C}_i(y_{i-1} - r_{i-1}) + r_i & \text{if } i < \ell \\ \mathbf{C}_i(y_{i-1} - r_{i-1}) & \text{otherwise} \end{cases}$
– for $i \in [\ell-1]$, compute $\sigma_i = \mathcal{S}.\mathsf{Sign}(\mathsf{sk}, y_i)$ and for every $S^{i+1}_{b,k'} \in \mathcal{C}_{i+1}, k \in K_i, k' \in K_{i+1}$, compute $\sigma^{\mathsf{ek}}_{i+1,b,k,k'} = \mathcal{S}.\mathsf{Sign}(\mathsf{sk}, \mathsf{ek}^*_{i+1,b,k,k'})$.

Claim. Assuming F is a PRF (Definition 12), $\mathsf{H}_3 \overset{\mathrm{c}}{\approx} \mathsf{H}_4$.

Figure 46 Hybrid Experiment H_3 – continued

Let $\mathsf{DEHE} = (\mathsf{KeyGen}, \mathsf{Gen}, \mathsf{Enc}, \mathsf{Eval})$ be a two-party DEHE scheme (Definition 1). Let $\mathcal{S}$ be a signature scheme (Definition 14) and F be a PRF (Definition 12).

4. H_3, $\forall i \in [\ell - 1]$, emulating π_{GOD}'s i^{th} computation round does the following:

- for each thread $k \in \mathsf{Corrupt}[i]$, having a corrupt party $S^i_{b,k}$, do the following:
  - compute $[y_i]_{\neg b,k} = y_i - [y_i]_{b,k}$ and $[\sigma_i]_{\neg b,k} = \sigma_i - [\sigma_i]_{b,k}$
  - for every $k' \in \mathsf{Corrupt}[i+1]$, with corrupt party $S^{i+1}_{b',k'}$, compute $[\mathsf{ek}_{b',k,k'}]_{\neg b} = \mathsf{ek}^*_{i+1,b',k,k'} - [\mathsf{ek}'_{b',k,k'}]_b$ and $[\sigma^{\mathsf{ek}}_{b',k,k'}]_{\neg b} = \sigma_{i,b',k,k'} - [\sigma^{\mathsf{ek}}_{b',k,k'}]_b$
  - abort the thread if any **consistency checks** given in Π_i fail
- for each thread $k \in \mathsf{Honest}[i]$, do the following:
  - sample $([y_i]_{0,k}, [y_i]_{1,k})$ such that $[y_i]_{0,k} + [y_i]_{1,k} = y_i$.
  - for $b \in \{0,1\}$, compute signatures $[\sigma_i]_{b,k} = \mathcal{S}.\mathsf{Sign}(\mathsf{sk}, [y_i]_{b,k})$
  - for every $k' \in \mathsf{Corrupt}[i+1]$, with corrupt party $S^{i+1}_{b',k'}$, sample $([\mathsf{ek}_{b',k,k'}]_0, [\mathsf{ek}'_{b',k,k'}]_1)$ such that $[\mathsf{ek}'_{b',k,k'}]_0 + [\mathsf{ek}'_{b',k,k'}]_1 = \mathsf{ek}^*_{i+1,b',k,k'}$, and for $b \in \{0,1\}$, compute signatures $[\sigma^{\mathsf{ek}}_{b',k,k'}]_b = \mathcal{S}.\mathsf{Sign}(\mathsf{sk}, [\mathsf{ek}'_{b',k,k'}]_b)$
  - abort this thread if any **consistency checks** given in Π_i fail
- send the following to $\mathcal{A}$:
  - Corresponding to every honest thread $k \in \mathsf{Honest}[i]$, send $\{[y_i]_{b,k}, [\sigma_i]_{b,k}, [\mathsf{ek}'_{b',k,k'}]_b, [\sigma^{\mathsf{ek}}_{b',k,k'}]_b\}_{b\in\{0,1\}}$.
  - Corresponding to every corrupt thread $k \in \mathsf{Corrupt}[i]$, with corrupt party $S^i_{b,k}$, send $([y_i]_{\neg b,k}, [\sigma_i]_{\neg b,k}, \{[\mathsf{ek}_{b',k'}]_{\neg b}, [\sigma^{\mathsf{ek}}_{b',k'}]_{\neg b}\}_{S^{i+1}_{b',k'} \in \mathcal{C}_{i+1}})$.
  - Additionally send $(\mathsf{vk}, \mathsf{ct}^*_\alpha, \mathsf{ct}^*_{\mathsf{k}}, \mathsf{ct}^*_{\mathsf{sk}})$.
- H_3 accepts $([y_i]_{b,k}, [\sigma_i]_{b,k}, [\mathsf{ek}'_{\neg b',k'}]_b, [\sigma^{\mathsf{ek}}_{\neg b',k'}]_b, \mathsf{vk}, \mathsf{ct}^*_\alpha, \mathsf{ct}^*_{\mathsf{k}}, \mathsf{ct}^*_{\mathsf{sk}})$ from $\mathcal{A}$ for each corrupt $S^i_{b,k}, k \in \mathsf{Corrupt}[i]$.
- Choose $k \in K_i$ to be lexicographically the smallest thread that has not been aborted and use values corresponding to this thread for layer $i+1$.
- H_3 (emulating protocol Π_{i+1}) computes, $(\{[\mathsf{ek}'_{0,k,k'}]_b, [\sigma^{\mathsf{ek}}_{0,k,k'}]_b, [\mathsf{ek}'_{1,k,k'}]_b, [\sigma^{\mathsf{ek}}_{1,k,k'}]_b\}_{k'\in K_{i+2}}, [y_{i+1}]_{b,k}, [\sigma_{i+1}]_{b,k}) \leftarrow \mathsf{DEHE.Eval}(b, \mathsf{ek}_{b,k}, (\mathsf{ct}_\alpha, \mathsf{ct}_{\mathsf{k}}, \mathsf{ct}_{\mathsf{sk}}), P_{i+1,k}(i+1, k, y_i))$ as the corrupt party $S^{i+1}_{b,k}$ would.

5. Before the final round, H_3 receives from $\mathcal{F}_{\mathsf{GOD}}$ the functionality output $f(x)$.

- set $y_\ell = f(x)$ and $\sigma_\ell = \mathcal{S}.\mathsf{Sign}(\mathsf{sk}, y_\ell)$
- for each thread $k \in \mathsf{Corrupt}[\ell]$, having a corrupt party $S^\ell_{b,k}$, do the following:
  - compute $[y_\ell]_{\neg b,k} = y_\ell - [y_\ell]_{b,k}$ and $[\sigma_\ell]_{\neg b,k} = \sigma_\ell - [\sigma_\ell]_{b,k}$
- for each thread $k \in \mathsf{Honest}[\ell]$, do the following:
  - sample $[y_\ell]_{0,k}, [y_\ell]_{1,k}$ such that $[y_\ell]_{0,k} + [y_\ell]_{1,k} = y_\ell$.
  - for $b \in \{0,1\}$, compute signatures $[\sigma_\ell]_{b,k} = \mathcal{S}.\mathsf{Sign}(\mathsf{sk}, [y_\ell]_{b,k})$.
- send the following to $\mathcal{A}$:
  - Corresponding to every honest thread $k \in \mathsf{Honest}[\ell]$, send $\{[y_\ell]_{b,k}, [\sigma_\ell]_{b,k}\}_{b\in\{0,1\}}$. Additionally send vk.
  - Corresponding to every corrupt thread $k \in \mathsf{Corrupt}[\ell]$, with corrupt party $S^i_{b,k}$, send $([y_\ell]_{\neg b,k}, [\sigma_\ell]_{\neg b,k})$.
- $\forall P^{\ell+1}_j \in \mathcal{H}_{\ell+1}$, H_3 accepts $(\{[y_\ell]_{b,k}, [\sigma_\ell]_{b,k}\}_{S^\ell_{b,k} \in \mathcal{C}_\ell, k \in \mathsf{Corrupt}[\ell]})$ from $\mathcal{A}$, abort this thread if **consistency checks** given in Π_ℓ fail

Proof. Consider for the sake of contradiction that there exists a PPT distinguisher $\mathsf{D}_{\mathsf{GOD}}$ that can differentiate with non-negligible probability ϵ between the view produced by the hybrid experiments H_3 and H_4. Then this can be used in a black-box way by a PPT distinguisher $\mathsf{D}_{\mathsf{PRF}}$ to distinguish between having received oracle access to a PRF F keyed with a random key α, and a random function R with the same advantage. This distinguisher $\mathsf{D}_{\mathsf{PRF}}$ would work as follows:

- Let $\mathcal{O}$ be the oracle that $\mathsf{D}_{\mathsf{PRF}}$ has access to.

– for each $i \in [4\ell]$, $\mathsf{D}_{\mathsf{PRF}}$ queries $\mathcal{O}(i,1)$ and $\mathcal{O}(i,2)$, and uses its output to generate the rest of the view exactly as in H_3.
– Note that if $\mathcal{O} = R$, the view produced would be identically distributed as H_3. Otherwise, when $\mathcal{O} = F_\alpha$, this would be the same as that of H_4.
– Finally, $\mathsf{D}_{\mathsf{PRF}}$ sends this view to $\mathsf{D}_{\mathsf{GOD}}$ and outputs whatever it would output.

Note that in the above, the advantage of $\mathsf{D}_{\mathsf{PRF}}$ is the same as that of $\mathsf{D}_{\mathsf{GOD}}$, which is non-negligible. However, since F is a secure pseudo-random function (Definition 12), no such $\mathsf{D}_{\mathsf{PRF}}$ can exist, yielding a contradiction.