Multiparty Homomorphic Encryption
(or: On Removing Setup in Multi-Key FHE)

Prabhanjan Ananth[1], Abhishek Jain[2], and Zhengzhong Jin[2]

[1]University of California Santa Barbara
[2]Johns Hopkins University

Abstract

The notion of threshold multi-key fully homomorphic encryption (TMK-FHE) [López-Alt, Tromer, Vaikuntanathan, STOC'12] was proposed as a generalization of fully homomorphic encryption to the multiparty setting. In a TMK-FHE scheme for n parties, each party can individually choose a key pair and use it to encrypt its own private input. Given n ciphertexts computed in this manner, the parties can homomorphically evaluate a circuit C over them to obtain a new ciphertext containing the output of C, which can then be decrypted via a threshold decryption protocol. The key efficiency property is that the size of the (evaluated) ciphertext is independent of the size of the circuit.

TMK-FHE with *one-round* threshold decryption, first constructed by Mukherjee and Wichs [Eurocrypt'16], has found several powerful applications in cryptography over the past few years. However, an important drawback of all such TMK-FHE schemes is that they require a common setup which results in applications in the *common random string* model.

To address this concern, we propose a notion of *multiparty homomorphic encryption* (MHE) that retains the communication efficiency property of TMK-FHE, but sacrifices on the efficiency of final decryption. Specifically, MHE is defined in a similar manner as TMK-FHE, except that the final output computation process performed locally by each party is "non-compact" in that we allow its computational complexity to depend on the size of the circuit. We observe that this relaxation does not have a significant bearing in many important applications of TMK-FHE.

Our main contribution is a construction of MHE from the learning with errors assumption in the *plain model*. Our scheme can be used to remove the setup in many applications of TMK-FHE. For example, it yields the first construction of low-communication reusable non-interactive MPC in the plain model. To obtain our result, we devise a recursive self-synthesis procedure to transform *any* "delayed-function" two-round MPC protocol into an MHE scheme.

1 Introduction

The emergence of fully homomorphic encryption (FHE) [29] over the last decade has revolutionized cryptography. Roughly speaking, FHE allows for homomorphically evaluating an arbitrary circuit over an encrypted plaintext such that the resulting ciphertext contains the output of the circuit evaluated over the plaintext. This remarkable feature has enabled numerous applications in cryptography over the years.

Since FHE is natively suited for tasks involving two parties, the notion of *threshold multi-key fully homomorphic encryption* (TMK-FHE) [36] was proposed as its natural extension to the multiparty setting. In a TMK-FHE scheme for n parties, each party can individually choose a key pair and use it to encrypt its own private input. Given n ciphertexts computed in this manner, the parties can homomorphically evaluate a circuit over them to obtain a new "multi-key ciphertext". This ciphertext, however, cannot be decrypted by any individual party; instead, the parties engage in a threshold decryption protocol, where at the end, each party can learn the output of the circuit over the inputs of the parties.

The efficiency properties of TMK-FHE are defined analogously to FHE, with the key requirement being that the size of the evaluated ciphertext must be independent of the size of the circuit. The security requirement is similar to secure multiparty computation (MPC) [43, 33], namely, even a majority coalition of up to $(n-1)$ parties should not be able to learn anything beyond their inputs and the output of the circuit. Indeed, TMK-FHE yields a natural design template for MPC: first the parties encrypt their inputs using their own keys. Later, when the parties agree upon a circuit C that they wish to compute, they homomorphically evaluate C over their set of encrypted inputs and then execute the threshold decryption protocol over the resulting ciphertext to learn the output of C.

The first generation of TMK-FHE schemes constructed by Lopez et. al. [36] required an interactive threshold decryption procedure that involved running a generic MPC protocol. Subsequently, building on Clear and McGoldrick [23], Mukherjee and Wichs [38] constructed the first TMK-FHE scheme with a *one-round* (a.k.a. non-interactive) decryption protocol based on the learning with errors (LWE) assumption. The non-interactive decryption ability has since turned out to be a game-changer; in particular, it has enabled numerous powerful applications including two-round MPC [38], homomorphic secret sharing [14, 15], spooky encryption [24], obfuscation and functional encryption combiners [5, 6], multiparty obfuscation [34], homomorphic time-lock puzzles [37, 17] and ad-hoc multi-input functional encryption [1].

On the use of Common Setup. All known TMK-FHE schemes with a one-round decryption protocol [38, 19, 39] require an initial setup process to sample common parameters that must be used by each party to compute its key. As a consequence, the aforementioned applications are achieved in the *common random string* (CRS) model.

A major open question in this area (stated explicitly in [38, 19, 21]) is whether it is possible to avoid the use of a common setup. The importance of this question stems from the fact that a positive resolution would eliminate the necessity of a CRS in the applications of TMK-FHE, thereby yielding schemes in the *plain model*.

Multiparty Homomorphic Encryption. To address the above concern, we study a notion of *multiparty homomorphic encryption* (MHE) – a variant of TMK-FHE that retains its key virtue of communication efficiency but sacrifices on the efficiency of final output computation step. Specifically, MHE is defined in a similar manner as TMK-FHE, except that the final output computation step performed locally by each party is "non-compact" in that we allow its computational complexity to depend on the size of the circuit. Syntactically, this is achieved by allowing the output computation algorithm to take as input the same circuit (say) C that was evaluated earlier. Crucially, however, we still require the size of the (evaluated) ciphertexts to be independent of size of the circuit.

A few remarks regarding the meaningfulness of this notion are in order:

- **Non-triviality:** We first observe that unlike the case of (single-key) FHE, allowing for non-compact output computation does *not* trivialize the notion of MHE. Indeed, in the case of FHE, a trivial scheme with non-compact output computation can be obtained via any public-key encryption scheme by simply considering a decryption process that first recovers the plaintext and then evaluates the circuit to compute the output. Such an approach, however, does not extend to the multiparty setting since it would violate the security requirement of MHE (defined similarly to TMK-FHE).

- **Applicability:** Second, allowing for non-compact output computation does not seem to adversely impact many important applications of TMK-FHE. For example, MHE still yields two-round MPC, with the minor difference that each party would need to perform work proportional to the circuit two times as opposed to once.

Crucially, we show that by allowing for this relaxation, we can eliminate the use of setup. Specifically, we show that MHE can be realized without any common setup, in the plain model.

1.1 Our Results

MHE from LWE. In this work, we study the notion of MHE as an alternative to TMK-FHE for cryptographic applications. Our main result is a construction of MHE in the plain model based on the LWE assumption.

Our approach differs significantly from [38] who build upon the Gentry et al. FHE scheme [30] to construct TMK-FHE and then use it to build two-round MPC. We take a "reverse approach" and provide a *generic transformation* from any delayed-function[1] two-round MPC protocol in the plain model to an MHE scheme.

Theorem 1.1 (Informal)**.** *Assuming LWE (with sub-exponential modulus-to-noise ratio), there exists a generic transformation from any delayed-function two-round semi-honest MPC in the plain model to MHE.*

Since delayed-function two-round MPC in the plain model is known from two-round oblivious transfer[2] [28, 10], which in turn can be realized based on LWE [40, 16], we obtain an MHE scheme based only on LWE.

Our Approach. A fundamental property of TMK-FHE is the ability to perform an unlimited number of homomorphic evaluations (of possibly different functions) on a tuple of ciphertexts. That is, given a tuple of ciphertexts $(c_1, \ldots, c_n)$ and functions $f_1, \ldots, f_k$ (for any polynomial k), it is possible to compute $C_{f_1}, \ldots, C_{f_k}$, where each C_{f_i} is obtained by homomorphically evaluating f_i over $(c_1, \ldots, c_n)$. This *reusability* property is crucial to many applications of TMK-FHE.

Our main technical contribution is a recursive *self-synthesis* procedure for achieving this property. Our approach bears resemblance to, and builds upon, several unrelated works dating as far back as the construction of pseudorandom functions from pseudorandom generators [31], as well as recent constructions of indistinguishability obfuscation from functional encryption [12, 7] (and even more recently, constructions of identity-based encryption [26, 18]). Indeed, the central goal underlying all of these works can be re-cast as achieving some form of reusability. We further elaborate on our approach in Section 1.2.

Applications. MHE can be plugged into many applications of TMK-FHE, to eliminate the use of CRS and obtain results in the plain model. In particular, by plugging in MHE in the aforementioned template for constructing MPC from TMK-FHE, we obtain a two-round MPC protocol in the plain model with the following two salient properties:

- The first round of the protocol, which only depends on the inputs of the parties, can be *reused* for an arbitrary number of computations. That is, after the completion of the first round, the parties can execute the second round multiple times, each time with a different circuit C_ℓ of their choice, to learn the output of C_ℓ over their fixed inputs.

[1] A two-round MPC protocol satisfies the delayed function property if the first round messages of the protocol are computed independently of the description (but not necessarily the size) of the function.

[2] While the original constructions of [28, 10] do not achieve the delayed-function property; as observed in [3, 4], they can be easily adapted to satisfy this property.

- The communication complexity of the protocol is independent of the circuit size (and only depends on the circuit depth).

Previously, such a protocol – obtained via TMK-FHE – was only known in the CRS model [38].

Benhamouda and Lin [11] investigated the problem of two-round reusable MPC (with circuit-size dependent communication) and give a construction for the same, in the plain model, based on bilinear maps.[3] Our construction is based on a different assumption, namely, LWE, and therefore also enjoys post-quantum security.

Future Directions. A few known applications of TMK-FHE [24, 14, 37] do rely on the compactness (and in some cases, "simplicity") of output reconstruction procedure achieved by the scheme of [38]. Our construction does not enjoy this property, and achieving it remains an important open problem for future.

1.2 Technical Overview

Towards constructing MHE, we consider a relaxed notion of MHE where the evaluation algorithm is allowed to be private; we call this notion pMHE. It turns out that if we have a pMHE scheme then we can combine this with any (single-key) leveled fully homomorphic encryption scheme to achieve MHE. We will show the transformation from pMHE to MHE in Section 3. Hence, we focus on constructing pMHE.

MHE with Private Evaluation (pMHE). An MHE scheme with private evaluation, associated with n parties, consists of the following algorithms:

- **Encryption**: The i^{th} party, for $i \in [n]$, on input x_i produces a ciphertext ct_i and secret key sk_i.
- **Evaluation**: The i^{th} party on input all the ciphertexts $\mathsf{ct}_1, \ldots, \mathsf{ct}_n$, secret key sk_i, circuit C, it evaluates ciphertexts to obtain a partial decrypted value p_i.
- **Final Decryption**: Given all the partial decrypted values $(p_1, \ldots, p_n)$ and the circuit C, we achieve the output $C(x_1, \ldots, x_n)$.

In terms of efficiency, we require that the size of the ciphertexts and the partial decrypted values do not depend on the size of the circuit. In more detail, we define the notion of succinctness as follows:

- The size of the ciphertext is polynomial in the security parameter λ, input length of C, namely $C.\mathsf{in}$, the output length of C, namely $C.\mathsf{out}$, and depth of C, namely $C.\mathsf{depth}$.
- The size of the partial decrypted values is polynomial in the security parameter λ, $C.\mathsf{in}, C.\mathsf{out}$ and $C.\mathsf{depth}$.

Ideally, we would like both the size of the ciphertexts and the size of the partial decrypted values to be also independent of the depth of the circuit; however note that even the problem of constructing single-key FHE schemes from learning with errors where the parameters do not grow with the depth of the circuit is still open.

Starting Point: Generic Two-Round Secure MPC. Towards constructing a multi-party homomorphic encryption scheme, we first identify a two round secure multiparty computation (MPC) protocol as a natural starting tool to construct a pMHE scheme. This construction is simple and can be described as follows:

[3]The authors communicated their result statement privately to us; however, a public version of their paper was not available at the time of writing of this paper. An online version of their paper can be found here [11].

- The i^{th} party, for $i \in [N]$, on input x_i produces the first round message $\mathsf{msg}_1^{(1)}$ of the MPC protocol along with the private state st_i. The ciphertext ct_i is set to be $\mathsf{msg}_i^{(1)}$ and the secret key sk_i is set to be the state st_i.
- The evaluation phase corresponds to the computation of second round messages. The i^{th} party on input all the ciphertexts $\mathsf{ct}_1, \ldots, \mathsf{ct}_N$, i.e., messages $\mathsf{msg}_1^{(1)}, \ldots, \mathsf{msg}_N^{(1)}$, circuit C and its secret key $\mathsf{sk}_i = st_i$, it produces the second round message $\mathsf{msg}_i^{(2)}$. We interpret $\mathsf{msg}_i^{(2)}$ as the partial decrypted value p_i.
- Finally, given all the partial decrypted values (aka second round messages), we can recover the output $C(x_1, \ldots, x_N)$.

While syntactically a two-round protocol does seem to yield a pMHE scheme as witnessed above, there are many reasons why the above scheme falls short of satisfying the properties of an pMHE scheme. We list some of the reasons below:

- *Non-Succinctness*: The size of the first and the second round messages in the MPC protocol, and thus the size of the ciphertexts and partial decrypted values, could grow polynomially in the size of the circuit.
- *Lack of delayed function property*: The first round messages in the MPC protocol could depend on the circuit being securely computed. This means the encryption procedure in the above scheme needs to take the circuit being evaluated as input.
- *Non-Reusability*: Even if the circuit, in the MPC protocol, is specified after the first message, it could be the case that the second round messages when issued for two different circuits will completely compromise the privacy of the inputs of the honest parties. That is, the first round message in the MPC protocol when reused for two different executions could leak the inputs of the honest parties. We call an MHE scheme that only allows for a single evaluation to be a one-time pMHE scheme.

Indeed, the current known two-round secure MPC protocols [28, 10] based on two-round oblivious transfer (and thus, learning with errors) have all the drawbacks mentioned above. It turns out that there is a simple modification to existing two-round protocols to achieve delayed-function property and thus for the rest of the overview, we only focus on handling the challenges of succinctness and reusability. We start by addressing the reusability property – the key technical challenge.

Reusability: Naive Solution. Our first attempt to build an pMHE scheme for a circuit class $\mathcal{C} = \{C_0, C_1\}$ that allows for only *two* decryption queries, denoted by TwoMHE, is as follows: we consider two instantiations of OneMHE, that we call OneMHE_0 and OneMHE_1.

- The i^{th} party, for $i \in [N]$, on input x_i, produces two ciphertexts ct_0^i and ct_1^i, where ct_0^i is computed by encrypting x_i using OneMHE_0 and ct_1 is computed by encrypting x_i using OneMHE_1.
- To evaluate a circuit C_b, for $b \in \{0, 1\}$, run the evaluation procedure of OneMHE_b to obtain the partial decrypted values.
- The final decryption on input C_b and partial decrypted values produces the output.

It is easy to see that the above scheme supports two decryption queries. While the above template can be generalized if $\mathcal{C}$ consists of polynomially many circuits; every circuit in $\mathcal{C}$ is associated with an instantiation of OneMHE. However, it is clear that this approach does not scale when $\mathcal{C}$ consists of exponentially many circuits.

Recursive Self-Synthesis. Instead of generating all the instantiations of OneMHE during the encryption phase, as is done in TwoMHE, our main insight is to instead defer the generation of the instantiations of OneMHE to the evaluation phase. The advantage of this is that during the evaluation phase, we know exactly which circuit is being evaluated and thus we can afford to be frugal and only generate the instantiations of OneMHE that are necessary, based on the description of this circuit. The idea of bootstrapping a "one-time" secure scheme into a "multi-time" secure scheme is not new and has been studied in different contexts in cryptography; be it the classical result on pseudorandom functions from pseudorandom generators [32] or the more recent results on indistinguishability from functional encryption [7, 13, 35] and constructions of identity-based encryption [26, 18, 25]. In particular, as we will see soon, our implementation of deferring the executions of OneMHE and only invoke the instantiations as needed bears some resemblance to techniques developed in these works, albeit in a very different context.

Illustration. Before explaining our approach to handle any polynomial number of decryption queries, we start with the same example as before: the goal is to build pMHE scheme for a circuit class $\mathcal{C} = \{C_0, C_1\}$ that allows for 2 decryption queries. The difference, however, is, unlike before, the approach we describe below will scale to exponentially many circuits.

We employ a tree-based approach to solve this problem.

- The tree associated with this scheme consists of three nodes: a single root and two leaves. The first leaf is associated with the circuit C_0 and the second leaf is associated with the circuit C_1.

- Denote the one-time pMHE scheme associated with the root to $\mathsf{OneMHE}_\perp$, with the left leaf to be OneMHE_0 and the right leaf node to be OneMHE_1.

- Denote $\widetilde{C}_{i,b}$ to be the garbling of a circuit that takes as input OneMHE_b ciphertexts of $x_1, \ldots, x_N$, performs evaluation of C using the i^{th} secret key associated with OneMHE_b and outputs the OneMHE_b partial decryption values.

- Finally, denote the circuit $C_\perp$ to be the circuit [4] that takes as input $(x_1, \ldots, x_N)$ and produces:
  - wire labels, associated with $\widetilde{C}_{i,0}$ for OneMHE_0 ciphertexts of x_i under the i^{th} party's secret key and,
  - wire labels, associated with $\widetilde{C}_{i,1}$, for OneMHE_1 ciphertexts of x_i under the i^{th} party's secret key.

Armed with the above notation, we present an overview of construction of a pMHE scheme for $\mathcal{C} = \{C_0, C_1\}$ allowing for 2 decryption queries as follows:

- The i^{th} party, for $i \in [N]$, on input x_i, produces the ciphertext $\mathsf{ct}^i_\perp$, where $\mathsf{ct}^i_\perp$ is computed by encrypting x_i using $\mathsf{OneMHE}_\perp$.

[4]We consider the setting where the circuit is randomized; this is without loss of generality since we can assume that the randomness for this circuit is supplied by the parties

- To evaluate a circuit C_b, for $b \in \{0,1\}$, the i^{th} party does the following:
  - First run the evaluation procedure of $\mathsf{OneMHE}_\perp$ on input circuit $C_\perp$ to obtain the i^{th} partial decrypted value associated with $\mathsf{OneMHE}_\perp$.
  - It computes a garbled circuit $\widetilde{C}_{i,b}$ as described above.

  Output the i^{th} partial decrypted value associated with $\mathsf{OneMHE}_\perp$ and $\widetilde{C}_{i,b}$.

- The final decryption algorithm takes as input the $\mathsf{OneMHE}_\perp$ partial decryption values from all the parties, garbled circuits $\widetilde{C}_{1,b}, \ldots, \widetilde{C}_{N,b}$, circuit C_b and performs the following operations:
  - It first runs the final decryption procedure of $\mathsf{OneMHE}_\perp$ to obtain the wire labels corresponding to all the garbled circuits $\widetilde{C}_{1,b}, \ldots, \widetilde{C}_{N,b}$.
  - It then evaluates all the garbled circuits to obtain the OneMHE_b partial decryption values.
  - Using the OneMHE_b partial decryption values, compute the final decryption procedure of OneMHE_b to obtain $C_b(x_1, \ldots, x_N)$.

Full-Fledged Tree-Based Approach. We can generalize the above approach to construct a pMHE scheme for any circuit class and that handles any polynomially many queries. If s is the maximum size of the circuit in the class of circuits, we consider a binary tree of depth s.

- Every edge in the tree is labeled. If an edge e is incident from the parent to its left child then label it with 0 and if e is incident from the parent to its right child then label it with 1.
- Every node in the tree is labeled. The label is the concatenation of all the edge labels on the path from the root to the node.
- Every leaf is associated with a circuit of size s.

With each node v, associate with v a new instantiation of a one-time pMHE scheme, that we denote by $\mathsf{OneMHE}_{\mathbf{l}(v)}$, where $\mathbf{l}(v)$ is the label associated with node v. If v is the root node $\mathbf{l}(v) = \perp$.

Informally, the encryption algorithm of pMHE generates $\mathsf{OneMHE}_\perp$ encryption of x_i under the i^{th} secret key. During the evaluation procedure, on input C, we generate s garbled circuits, one for every node on the path from the root to the leaf labeled with C. The role of these garbled circuits is to delegate the computation of the partial decrypted values to the final decryption phase. In more detail, the garbled circuit associated with the node v computes the $\mathsf{OneMHE}_{\mathbf{l}v||0}$ and $\mathsf{OneMHE}_{\mathbf{l}v||1}$ partial decrypted values and outputs the corresponding wire labels for the garbled circuits associated with its children.

During the final decryption, starting from the root node, each garbled circuit (of every party) is evaluated to obtain wire labels of the garbled circuit associated with the child node on the path from the root to the leaf labelled with C. Finally, the garbled circuit associated with the leaf labelled with C is then evaluated to obtain the OneMHE_C partial decrypted values. These partial decrypted values are then decoded to recover the final output $C(x_1, \ldots, x_N)$.

Efficiency Challenges. To argue that the above scheme is a pMHE scheme, we should at the very least argue that the encryption, evaluation and final decryption algorithms can be executed in polynomial time. Let us first argue that all the garbled circuits can be computed in polynomial time by the i^{th} party. The time to compute the garbled circuit associated with the root node is polynomial in the time to compute OneMHE_0 and OneMHE_1 ciphertexts. The runtime of OneMHE_0 (resp.,

OneMHE_1) ciphertext is polynomial in $|G_0|$ (resp., $|G_1|$). Moreover, $|G_0|$ itself grows polynomially in the time to compute the OneMHE_{00} and OneMHE_{01} ciphertexts; this is similarly also true for $|G_1|$.

However, continuing this way, we realize that the time to compute the first garbled circuit is *exponential* in s! Thus, the above scheme does not even have polynomial efficiency, let alone the stronger succinctness property we need.

Identifying the Necessary Efficiency for Recursion. To make the above recursion idea work, we impose a stringent efficiency constraint on the encryption complexity of OneMHE. In particular, we require three properties to hold:

1. The *running time* of the encryption circuit is a polynomial in the security parameter λ, the number of parties, the input size, output size, and the depth of the circuit.

2. The *depth* of the encryption circuit OneMHE grows polynomially in λ, and only logarithmically in the number of parties, input length, output length, and the depth of the circuit being evaluated.

3. The *size* of the ciphertext output by the encryption of OneMHE is a polynomial of λ, and depends logarithmically on the number of parties, the input length, output length, and the depth of the circuit being evaluated.

Put together, we refer to the above efficiency properties as *strong ciphertext succinctness*. It turns out that *if* we have an OneMHE scheme with strong ciphertext succinctness, then the resulting reusable pMHE scheme has polynomial efficiency and moreover, the ciphertext sizes in the resulting scheme are polynomial in the security parameter alone.[5]

But what about the size of partial decryption values? Unfortunately, it turns out that the reusable pMHE scheme obtained from the above process does not have succinct partial decryption values. In particular, the size of the partial decryption values grow proportional to the size of the circuit. We address this problem by applying a compiler that generically transforms a pMHE scheme with large partial decryption values into a scheme with succinct partial decryption values; that is, one that only grows proportional to the input, output lengths and the depth of the circuit being evaluated. Such compilers, that we refer to as *low communication* compilers were recently studied in the context of two-round secure MPC protocols [42, 2] and we adapt them to our setting. Once we apply such a compiler, we achieve our desired pMHE scheme that satisfies the required efficiency properties.

Achieving Strong Ciphertext Succinctness. The above discussion hinged on the fact that there *exists* a OneMHE scheme that has strong ciphertext succinctness property. We next show how to construct such a scheme in two steps:

- *Weak Ciphertext Succinctness* (Section 4.2): First, we relax the notion of strong ciphertext succinctness property to allow for the size of the ciphertexts, and the depth of OneMHE to be polynomial in λ, input, output lengths and the depth of the circuit. We call this relaxed version to be *weak ciphertext succinctness* property. We apply the low communication compiler

[5]An informed reader may wish to draw an analogy to recent works that devise recursive strategies to build indistinguishability obfuscation from functional encryption [7, 13, 35]. These works show that a functional encryption scheme with a sufficiently compact encryption procedure (roughly, where the complexity of encryption is sublinear in the size of the circuit) can be used to build an indistinguishability obfuscation scheme. In a similar vein, strong ciphertext succinctness can be seen as the necessary efficiency notion for driving the recursion in our setting without blowing up efficiency.

discussed above on a OneMHE scheme with no succinctness properties whatsoever (that can in turn be constructed using any delayed-function two-round semi-honest secure MPC) to obtain a OneMHE scheme satisfying weak ciphertext succinctness property. (Note that we apply this compiler not only in this step but also once more on top of the reusable pMHE scheme as discussed above.)

- *From Weak to Strong Ciphertext Succinctness* (Section 4.3): We then show how to transform a OneMHE scheme satisfying weak ciphertext succinctness property into one that satisfies strong ciphertext succinctness property. There are two differences between these two notions: (i) in the weak ciphertext succinctness property, the size of the ciphertexts could grow polynomially in λ, input, output lengths and depth of the circuit being evaluated whereas in the strong ciphertext succinctness property, the size of the ciphertexts grows only polynomially in the security parameter λ and, (ii) in the strong ciphertext succinctness property, there is an additional restriction on the depth of the encryption circuit to be polynomial in the security parameter.

  We tackle these differences one by one by employing two different tools:

  - We first use randomized encodings computable in NC^1 [8] to compress the depth of the encryption circuit in the OneMHE scheme with weak ciphertext succinctness.
  - We next use the recently studied notion of laconic oblivious transfer (LOT) [22] to compress the size of the ciphertexts. Roughly speaking, LOT allows a receiver to use a hash function to compress a large database $D \in \{0,1\}^*$ to a short digest d. A sender can then use the digest d to encrypt two messages m_0, m_1 under an index i to a ciphertext ct, such that the receiver can decrypt ct to learn $m_{D[i]}$, given the database D.

    To compress the size of the first round message, for each party, we view the first round message as the database D, and use the hash function to compress it to a short digest. Then in the second round, each party garbles the next round function, and encrypts both labels for each input wires using the digests. Each party also outputs D in the clear in the second round, so that all other parties can use it to decrypt the labels, and then evaluate the garbled circuit to obtain the second round messages of the underlying protocol.

Both of the above steps combined yields a OneMHE scheme satisfying strong ciphertext succinctness property which can then be bootstrapped to obtain a (reusable) pMHE scheme. Further, we note that the tools employed in all of these steps can be built from LWE.

Achieving Public Evaluation Using FHE. (Section 3.4) So far, we have seen how to achieve a reusable pMHE scheme, namely, an MHE scheme with private evaluation. We show how to achieve an MHE scheme, i.e., the one with public evaluation, from reusable pMHE scheme using any (single-key) leveled FHE scheme. Each party encrypts its secret key using FHE; that is the i^{th} party generates FHE public key-secret key pair $(\mathsf{pk}_i, \mathsf{sk}_i)$ and encrypts the i^{th} secret key using pk_i; call this $\mathsf{FHE.ct}_i$. The i^{th} party ciphertext of the MHE scheme $\mathsf{MHE.ct}_i$ now consists of the i^{th} party ciphertext of the pMHE scheme, namely $\mathsf{pMHE.ct}_i$, along with $\mathsf{FHE.ct}_i$. The public evaluation of MHE now consists of homomorphically evaluating the pMHE private evaluation circuit, with $(C, \mathsf{pMHE.ct}_1, \ldots, \mathsf{pMHE.ct}_N)$ hardwired, on the ciphertext $\mathsf{FHE.ct}_i$. Since this is performed for each party, there are N resulting FHE ciphertexts $(\widehat{\mathsf{FHE.ct}}_1, \ldots, \widehat{\mathsf{FHE.ct}}_N)$. During the partial decryption phase, the i^{th} party decrypts $\widehat{\mathsf{FHE.ct}}_i$ using sk_i to obtain the partial decryption value corresponds to the pMHE scheme. The final decryption of MHE is the same as the final decryption of pMHE.

Summary. We summarise below the steps involved in constructing MHE.

- **First Step: Delayed-function two-round secure MPC.** We start with a two-round secure MPC protocol satisfies delayed-function property: i.e., the functionality to be securely computed can be specified after the first round of the protocol. Such protocols exist from two-round semi-honest oblivious transfer [28, 10][6] and thus can be based on the hardness of learning with errors. This already yields a one-time pMHE scheme, albeit with no succinctness properties.

- **Second Step: Low-Communication Compiler.** (Section 4.2) Then we show how to apply a low-communication compiler on the above one-time pMHE scheme to obtain a one-time pMHE scheme satisfying weak ciphertext succinctness property. This compiler additionally assumes the hardness of learning with errors.

- **Third Step: From Weak to Strong Ciphertext Succinctness.** (Section 4.3) Assuming laconic oblivious transfer and randomized encodings computable in NC^1 – both of which can be instantiated from the hardness of learning with errors [22, 18, 27, 8, 9] – we show how to transform an pMHE scheme satisfying weak ciphertext succinctness property into one satisfying strong ciphertext succinctness property.

- **Fourth Step: One-Time pMHE to Reusable MHE scheme.** (Section 5) We then show how to generically bootstrap the one-time pMHE scheme satisfying strong ciphertext succinctness property to obtain a reusable pMHE scheme; this is the main technical contribution of our paper. The resulting scheme does not have succinct partial decryption values.

- **Fifth Step: Low-Communication Compiler (again!).** (Section 4.2) We then apply the low-communication compiler again to the (reusable) pMHE scheme obtained above to obtain a scheme that has succinct partial decryption values.

- **Final Step: From private evaluation to public evaluation.** (Section 3) We show how to use any (single-key) leveled fully homomorphic encryption scheme, that can be based on the hardness of learning with errors, to obtain a (reusable) MHE scheme, i.e., with public evaluation, starting from a pMHE scheme.

2 Preliminaries

We denote the security parameter by λ. We focus only on boolean circuits in this work. For any circuit C, let $C.\mathsf{in}, C.\mathsf{out}, C.\mathsf{depth}$ be the input length, output length and depth of the circuit C, respectively. Denote $C.\mathsf{params} = (C.\mathsf{in}, C.\mathsf{out}, C.\mathsf{depth})$.

For any totally ordered sets $S_1, S_2, \ldots, S_n$, and any tuple $(i_1^*, i_2^*, \ldots, i_n^*) \in S_1 \times S_2 \times \cdots \times S_n$, we use the notation $(i_1^*, i_2^*, \ldots, i_n^*) + 1$ (resp. $(i_1^*, i_2^*, \ldots, i_n^*) - 1$) to denote the lexicographical smallest (resp. biggest) element in $S_1 \times S_2 \times \cdots \times S_n$ that is lexicographical bigger (resp. smaller) than $(i_1^*, i_2^*, \ldots, i_n^*)$.

Pseudorandom Generators. We recall the definition of pseudorandom generators. A function $\mathsf{PRG}_\lambda : \{0,1\}^{\mathsf{PRG.in}_\lambda} \to \{0,1\}^{\mathsf{PRG.out}_\lambda}$ is a pseduorandom generator, if for any PPT distinguisher $\mathcal{D}$, there exits a negligible function $\nu(\lambda)$ such that

$$\left| \Pr\left[s \leftarrow \{0,1\}^{\mathsf{PRG.in}_\lambda} : \mathcal{D}(1^\lambda, \mathsf{PRG}_\lambda(s)) = 1 \right] - \Pr\left[u \leftarrow \{0,1\}^{\mathsf{PRG.out}_\lambda} : \mathcal{D}(1^\lambda, u) = 1 \right] \right| < \nu(\lambda)$$

[6]As stated, [28, 10] do not satisfies delayed-function property but a simple modification to these protocols yields the desired property.

Learning with Errors. We recall the learning with errors (LWE) distribution.

Definition 2.1 (LWE distribution)**.** *For a positive integer dimension n and modulo q, the LWE distribution $A_{\mathbf{s},\chi}$ is obtained by sampling $\mathbf{a} \leftarrow \mathbb{Z}_q^n$, and an error $e \leftarrow \chi$, then outputting $(\mathbf{a}, b = \mathbf{s}^T \cdot \mathbf{a} + e) \in \mathbb{Z}_q^n \times \mathbb{Z}_q$.*

Definition 2.2 (LWE problem)**.** *The decisional $\mathsf{LWE}_{n,m,q,\chi}$ problem is to distinguish the uniform distribution from the distribution $A_{\mathbf{s},\chi}$, where $\mathbf{s} \leftarrow \mathbb{Z}_q^n$, and the distinguisher is given m samples.*

Standard instantiation of LWE takes χ to be a *discrete Gaussian* distribution.

Definition 2.3 (LWE assumption)**.** *Let $n = n(\lambda), m = m(\lambda), q = q(\lambda)$ and $\chi = \chi(\lambda)$. The Learning with Error (LWE) assumption states that for any PPT distinguisher $\mathcal{D}$, there exits a negligible function $\nu(\lambda)$ such that*

$$|\Pr[\mathcal{D}(1^\lambda, (\mathbf{A}, \mathbf{s}^T\mathbf{A} + \mathbf{e})) = 1] - \Pr[\mathcal{D}(1^\lambda, (\mathbf{A}, \mathbf{u})) = 1]| < \nu(\lambda)$$

where $\mathbf{A} \leftarrow \mathbb{Z}_q^{n\times m}, \mathbf{s} \leftarrow \mathbb{Z}_q^n, \mathbf{u} \leftarrow \mathbb{Z}_q^m, \mathbf{e} \leftarrow \chi^m$.

2.1 Garbling Schemes

A garbling scheme [43] is a tuple of algorithms $(\mathsf{GC.Garble}, \mathsf{GC.Eval})$ defined as follows.

$\mathsf{GC.Garble}(1^\lambda, C, \mathsf{lab})$ On input the security parameter, a circuit C, and a set of labels $\mathsf{lab} = \{\mathsf{lab}_{i,b}\}_{i\in[C.\mathsf{in}],b\in\{0,1\}}$, where $\mathsf{lab}_{i,b} \in \{0,1\}^\lambda$, it outputs a garbled circuit $\widetilde{C}$.

$\mathsf{GC.Eval}(\widetilde{C}, \mathsf{lab})$ On input a garbled circuit $\widetilde{C}$ and a set of labels $\mathsf{lab} = \{\mathsf{lab}_i\}_{i\in[C.\mathsf{in}]}$, it outputs a value y.

We require the garbling scheme to satisfy the following properties.

Correctness For any circuit C, and any input $x \in \{0,1\}^{C.\mathsf{in}}$,

$$\Pr\left[\substack{\mathsf{lab}=\{\mathsf{lab}_{i,b}\}_{(i,b)\in[C.\mathsf{in}]\times\{0,1\}}\leftarrow\{0,1\}^{2\lambda C.\mathsf{in}},\\ \widetilde{C}\leftarrow\mathsf{GC.Garble}(1^\lambda,C,\mathsf{lab}),y\leftarrow\mathsf{GC.Eval}(\widetilde{C},(\mathsf{lab}_{i,x_i})_{i\in[C.\mathsf{in}]})} : y = C(x)\right] = 1$$

Selective Security For any PPT adversaries $(\mathcal{A}_1, \mathcal{A}_2)$, and any input x, there exits a simulator $(\mathsf{GC.Sim}_1, \mathsf{GC.Sim}_2)$, and a negligible function $\nu(\lambda)$ such that

$$\Bigg|\Pr\left[\substack{\mathsf{lab}\leftarrow\{0,1\}^{2\lambda C.\mathsf{in}},(C,\mathsf{st}_A)\leftarrow\mathcal{A}_1(1^\lambda,\mathsf{lab}_x)\\ \widetilde{C}\leftarrow\mathsf{GC.Garble}(1^\lambda,C,\mathsf{lab})} : \mathcal{A}_2(\mathsf{st}_A, \widetilde{C}) = 1\right] -$$
$$\Pr\left[\substack{(\mathsf{st}_S,\overline{\mathsf{lab}})\leftarrow\mathsf{GC.Sim}_1(1^\lambda,C.\mathsf{in}),(C,\mathsf{st}_A)\leftarrow\mathcal{A}_1(1^\lambda,\overline{\mathsf{lab}}),\\ \overline{C}\leftarrow\mathsf{GC.Sim}_2(\mathsf{st}_S,C(x))} : \mathcal{A}_2(\mathsf{st}_A, \overline{C}) = 1\right]\Bigg| < \nu(\lambda)$$

Theorem 2.4 ([43])**.** *There exists a garbling scheme for all poly-sized circuits from one-way functions.*

2.2 Randomized Encoding

A randomized encoding scheme [8] is a generalization of garbling schemes where the circuit and the input are encoded using one function. Let $\mathcal{F} = \{f_\lambda \mid f_\lambda : \{0,1\}^{f_\lambda.\mathsf{in}} \to \{0,1\}^{f_\lambda.\mathsf{out}}\}$ be a function family. We say $\hat{\mathcal{F}} = \{\hat{f}_\lambda \mid \hat{f}_\lambda : \{0,1\}^{\hat{f}_\lambda.\mathsf{in}_1} \times \{0,1\}^{\hat{f}_\lambda.\mathsf{in}_2} \to \{0,1\}^{\hat{f}_\lambda.\mathsf{out}}\}$ is a randomized encoding of $\mathcal{F}$, if it satisfies the following properties.

Correctness There exists a recover algorithm $\mathsf{RE.Recover}$ such that for any λ and $x \in \{0,1\}^{f_\lambda.\mathsf{in}}$,

$$\Pr\left[r \leftarrow \{0,1\}^{\hat{f}.\mathsf{in}_2} : \mathsf{RE.Recover}(\hat{f}_\lambda(x,r)) = f_\lambda(x)\right] = 1$$

Computational Privacy There exits a PPT simulator $\mathsf{RE.Sim}$ such that for any PPT distinguisher $\mathcal{D}$, there exits a negligible function $\nu(\lambda)$, for any $x \in \{0,1\}^{f_\lambda.\mathsf{in}}$,

$$\left| \Pr\left[r \leftarrow \{0,1\}^{\hat{f}_\lambda.\mathsf{in}_2} : \mathcal{D}(1^\lambda, \hat{f}_\lambda(x,r)) = 1\right] - \Pr\left[\mathcal{D}(1^\lambda, \mathsf{RE.Sim}(1^\lambda, f_\lambda(x))) = 1\right] \right| < \nu(\lambda)$$

Theorem 2.5 ([8, 9])**.** *Assuming the hardness of learning with errors, there exists a randomized encoding scheme computable in $\mathcal{NC}^1$ for every efficiently computable function.*

2.3 Laconic Oblivious Transfer

Informally, a laconic oblivious transfer is an interactive protocol with two parties (S, R). The receiver R has a database $D \in \{0,1\}^*$ as input. It then uses a deterministic hash function to hash the database, and obtains a digest. Next, the receiver sends the digest to the sender. The sender takes as input an index i, the digest, and two messages m_0, m_1, and computes a ciphertext ct. Any one who knows the database D can decrypt the ciphertext ct to $m_{D[i]}$. We give the formal definition in the following.

A laconic oblivious transfer (laconic OT) scheme [22] is a tuple of algorithms $(\mathsf{Gen}, \mathsf{Hash}, \mathsf{Enc}, \mathsf{Dec})$, which works as follows.

$\mathsf{Gen}(1^\lambda)$ On input security parameters, it outputs a *uniformly random common string* crs.

$\mathsf{Hash}(\mathsf{crs}, D)$ On input crs and a binary string $D \in \{0,1\}^*$, it outputs a digest digest.

$\mathsf{Enc}(\mathsf{crs}, \mathsf{digest}, i, m_0, m_1)$ On input crs, a digest, an index i, and two messages m_0, m_1, it outputs a ciphertext ct.

$\mathsf{Dec}(\mathsf{crs}, \mathsf{ct}, D)$ On input crs, a ciphertext ct, and a binary string D, it outputs a decrypted message m.

A laconic OT scheme satisfies the following properties.

Correctness For any D, index $i \in [|D|]$, and any m_0, m_1,

$$\Pr\left[\substack{\mathsf{crs} \leftarrow \mathsf{Gen}(1^\lambda), \mathsf{digest} \leftarrow \mathsf{Hash}(\mathsf{crs}, D) \\ \mathsf{ct} \leftarrow \mathsf{Enc}(\mathsf{crs}, \mathsf{digest}, i, m_0, m_1), m \leftarrow \mathsf{Dec}(\mathsf{crs}, \mathsf{ct}, D)} : m = m_{D_i}\right] = 1$$

Semi-Honest Sender-Privacy For any binary string D, any index $i \in [|D|]$, and any message m_0, m_1, and any PPT distinguisher $\mathcal{D}$, there exists a negligible $\nu(\lambda)$ such that

$$\begin{aligned} &|\Pr[\mathcal{D}(1^\lambda, \mathsf{crs}, \mathsf{digest}, \mathsf{Enc}(\mathsf{crs}, \mathsf{digest}, i, m_0, m_1)) = 1] - \\ &\Pr[\mathcal{D}(1^\lambda, \mathsf{crs}, \mathsf{digest}, \mathsf{Enc}(\mathsf{crs}, \mathsf{digest}, i, m_{D_i}, m_{D_i})) = 1]| < \nu(\lambda) \end{aligned}$$

where $\mathsf{crs} \leftarrow \mathsf{Gen}(1^\lambda), \mathsf{digest} \leftarrow \mathsf{Hash}(\mathsf{crs}, D)$.

Efficiency Size of digest is succinct i.e. $|\mathsf{digest}| = \mathsf{poly}(\lambda)$. The running time of Enc is $\mathsf{poly}(\lambda, \log|D|)$. The running time of Hash is $\mathsf{poly}(\lambda, |D|)$. The depth of Hash is bounded by $\mathsf{poly}(\lambda, \log|D|)$.

Theorem 2.6 ([22, 18, 27])**.** *Assuming the hardness of learning with errors, there exists a laconic oblivious transfer scheme.*

2.4 Laconic Function Evaluation

A laconic function evaluation (LFE) scheme [42] for a class of poly-sized circuits consists of four PPT algorithms $\mathsf{crsGen}, \mathsf{Compress}, \mathsf{Enc}, \mathsf{Dec}$ described below.

$\mathsf{crsGen}(1^\lambda, \mathsf{params})$ It takes as input the security parameter λ, circuit parameters params and outputs a uniformly random common string crs.

$\mathsf{Compress}(\mathsf{crs}, C)$ It takes as input the common random string crs, poly-sized circuit C and outputs a digest digest_C. This is a deterministic algorithm.

$\mathsf{Enc}(\mathsf{crs}, \mathsf{digest}_C, x)$ It takes as input the common random string crs, a digest digest_C, a message x and outputs a ciphertext ct.

$\mathsf{Dec}(\mathsf{crs}, C, \mathsf{ct})$ It takes as input the common random string crs, circuit C, ciphertext ct and outputs a message y.

Correctness. We require the following to hold:

$$\Pr\left[\begin{array}{c}\mathsf{crs}\leftarrow\mathsf{crsGen}(1^\lambda,\mathsf{params})\\ \mathsf{digest}_C\leftarrow\mathsf{Compress}(\mathsf{crs},C)\\ \mathsf{ct}\leftarrow\mathsf{Enc}(\mathsf{crs},\mathsf{digest}_C,x)\\ y\leftarrow\mathsf{Dec}(\mathsf{crs},C,\mathsf{ct})\end{array} : y = C(x)\right] = 1$$

Efficiency. The size of CRS should be polynomial in λ, the input, output lengths and the depth of C. The size of digest, namely digest_C, should be polynomial in λ, the input, output lengths and the depth of C. The size of the output of $\mathsf{Enc}(\mathsf{crs}, \mathsf{digest}_C)$ should be polynomial in λ, the input, output lengths and the depth of C.

Security. For every PPT adversary $\mathcal{A}$, input x, circuit C, there exists a PPT simulator Sim such that for every PPT distinguisher $\mathcal{D}$, the following holds:

$$\Bigg| \Pr_{\substack{\mathsf{crs}\leftarrow\mathsf{crsGen}(1^\lambda,\mathsf{params})\\ \mathsf{digest}_C\leftarrow\mathsf{Compress}(\mathsf{crs},C)}}\left[1 \leftarrow \mathcal{D}\left(1^\lambda, \mathsf{crs}, \mathsf{digest}_C, \mathsf{Enc}(\mathsf{crs}, \mathsf{digest}_C, x)\right)\right] - \Pr_{\substack{\mathsf{crs}\leftarrow\mathsf{crsGen}(1^\lambda,\mathsf{params})\\ \mathsf{digest}_C\leftarrow\mathsf{Compress}(\mathsf{crs},C)}}\left[1 \leftarrow \mathcal{D}\left(1^\lambda, \mathsf{crs}, \mathsf{digest}_C, \mathsf{Sim}(\mathsf{crs}, \mathsf{digest}_C, C(x))\right)\right]\Bigg| \le \mathsf{neg}(\lambda)$$

for some negligible function neg.

Remark 2.7. *A strong version of security, termed as adaptive security, was defined in [42]; for our construction, selective security suffices.*

Theorem 2.8 ([42])**.** *Assuming the hardness of learning with errors, there exists a laconic function evaluation protocol.*

3 Multiparty Homomorphic Encryption

We define the notion of multiparty homomorphic encryption (MHE) in this section. As mentioned earlier, this notion can be seen as a variant of threshold multikey homomorphic encryption [23, 38]; unlike threshold multikey FHE, this notion does not require a trusted setup, however, the final decryption phase needs to take as input the circuit being evaluated as input.

3.1 Definition

A multiparty homomorphic encryption is a tuple of algorithms $\mathsf{MHE} = (\mathsf{MHE.KeyGen}, \mathsf{MHE.Enc}, \mathsf{MHE.Eval}, \mathsf{MHE.PartDec}, \mathsf{MHE.FinDec})$, which are defined as follows.

$\mathsf{MHE.KeyGen}(1^\lambda, i)$ On input the security parameter λ, and an index $i \in [N]$, it outputs a public-key secret-key pair $(\mathsf{pk}_i, \mathsf{sk}_i)$ for the i-th party.

$\mathsf{MHE.Enc}(\mathsf{pk}_i, x_i)$ On input a public key pk_i of the i-th party, and a message x_i, it outputs a ciphertext ct_i.

$\mathsf{MHE.Eval}(C, (\mathsf{ct}_j)_{j\in[N]})$ On input the circuit C of size polynomial in λ and the ciphertexts $(\mathsf{ct}_j)_{j\in[N]}$, it outputs the evaluated ciphertext $\widehat{\mathsf{ct}}$.

$\mathsf{MHE.PartDec}(\mathsf{sk}_i, i, \widehat{\mathsf{ct}})$ On input the secret key sk_i of i^{th} party, the index i, and the evaluated ciphertext $\widehat{\mathsf{ct}}$, it outputs the partial decryption p_i of the i^{th} party.

$\mathsf{MHE.FinDec}(C, (p_j)_{j\in[N]})$ On input the circuit C, and all the partial decryptions $(p_j)_{j\in[N]}$, it outputs a value $y \in \{0,1\}^{C.\mathsf{out}}$.

We require that a MHE scheme satisfies the properties of correctness, succinctness and reusable simulation security.

Correctness. We require the following definition to hold.

Definition 3.1 (Correctness)**.** *A scheme* $(\mathsf{MHE.KeyGen}, \mathsf{MHE.Enc}, \mathsf{MHE.Eval}, \mathsf{MHE.PartDec}, \mathsf{MHE.FinDec})$ *is said to satisfy the correctness of an MHE scheme if for any inputs* $(x_i)_{i\in[N]}$*, and circuit* C*, the following holds:*

$$\Pr\left[\begin{array}{c} \forall i\in[N], (\mathsf{pk}_i,\mathsf{sk}_i)\leftarrow \mathsf{MHE.KeyGen}(1^\lambda,i) \\ \mathsf{ct}_i\leftarrow\mathsf{MHE.Enc}(\mathsf{pk}_i,x_i) \\ \widehat{\mathsf{ct}}\leftarrow\mathsf{MHE.Eval}(C,(\mathsf{ct}_j)_{j\in[N]}) \\ p_i\leftarrow\mathsf{MHE.PartDec}(\mathsf{sk}_i,i,\widehat{\mathsf{ct}}) \\ y\leftarrow\mathsf{MHE.FinDec}(C,(p_j)_{j\in[N]}) \end{array} : y = C(x_1,\ldots,x_N)\right] = 1$$

Succinctness. We require that the size of the ciphertexts and the partial decrypted values to be independent of the size of the circuit being evaluated. More formally,

Definition 3.2 (Succinctness)**.** *A scheme* $(\mathsf{MHE.KeyGen}, \mathsf{MHE.Enc}, \mathsf{MHE.Eval}, \mathsf{MHE.PartDec}, \mathsf{MHE.FinDec})$ *is said to satisfy the succinctness property of an MHE scheme if for any inputs* $(x_i)_{i\in[N]}$*, and circuit* C*, the following holds: for any inputs* $(x_i)_{i\in[N]}$*, and circuit* C*,*

- Succinctness of Ciphertext*: for* $j \in [N]$, $|\mathsf{ct}_j| = \mathsf{poly}(\lambda, |x_j|)$.
- Succinctness of Partial Decryptions*: for* $j \in [N]$, $|p_j| = \mathsf{poly}(\lambda, N, C.\mathsf{in}, C.\mathsf{out}, C.\mathsf{depth})$*, where* N *is the number of parties,* $C.\mathsf{in}$ *is the input length of the circuit being evaluated,* $C.\mathsf{out}$ *is the output length and* $C.\mathsf{depth}$ *is the depth of the circuit.*

where, for every $i \in [N]$, (i) $(\mathsf{pk}_i, \mathsf{sk}_i) \leftarrow \mathsf{MHE.KeyGen}(1^\lambda, i)$, (ii) $\mathsf{ct}_i \leftarrow \mathsf{MHE.Enc}(\mathsf{pk}_i, x_i)$, (iii) $\widehat{\mathsf{ct}} \leftarrow \mathsf{MHE.Eval}(C, (\mathsf{ct}_j)_{j\in[N]})$ and, (iv) $p_i \leftarrow \mathsf{MHE.PartDec}(\mathsf{sk}_i, i, \widehat{\mathsf{ct}})$.

Remark 3.3. *En route to constructing MHE schemes satisfying the above succinctness properties, we also consider MHE schemes that satisfy the correctness and security (stated next) properties but fail to satisfy the above succinctness definition. We refer to such schemes as non-succinct MHE schemes.*

3.2 Security

We define the security of MHE by real-ideal paradigm. We only consider the semi-honest security notion.

In the real world, the adversary is given the public key (pk_i) and ciphertext ct_i for each party, and the randomness generating the public keys and ciphertexts for the dishonest parties. The adversary is also given access to an oracle $\mathcal{O}$. Each time, the adversary can query the oracle $\mathcal{O}$ with a circuit C. The oracle $\mathcal{O}$ firstly evaluates C homomorphically over the ciphertexts $(\mathsf{ct}_i)_{i\in[N]}$, and obtains an evaluated ciphertext $\widehat{\mathsf{ct}}$. Then it outputs the partial decryption of $\widehat{\mathsf{ct}}$ for each honest party.

In the ideal world, the public keys and the ciphertexts for all parties, and the randomness for dishonest parties are obtained by the simulator $\mathsf{MHE.Sim}_1$. The adversary is also given access to an oracle $\mathcal{O}'$. For each query C made by the adversary, the oracle $\mathcal{O}'$ executes the *stateful* simulator $\mathsf{MHE.Sim}_2$ to obtain the simulated partial decryption $(p_i)_{i\in H}$. Then the oracle $\mathcal{O}'$ outputs $(p_i)_{i\in H}$.

Reusable Simulation Security. We define the real and ideal experiments below. The experiments are parameterized by adversary $\mathcal{A}$, PPT simulator $\mathsf{MHE.Sim}$ implemented as algorithms $(\mathsf{MHE.Sim}_1, \mathsf{MHE.Sim}_2)$ and $H \subseteq [N]$.

```
$\mathsf{Real}^{\mathcal{A}}(1^\lambda, (x_i)_{i\in[N]})$
for $i \in [N]$ do
    $r_i, r'_i \leftarrow \{0,1\}^*$
    $(\mathsf{pk}_i, \mathsf{sk}_i) = \mathsf{MHE.KeyGen}(1^\lambda, i; r_i)$
    $\mathsf{ct}_i = \mathsf{MHE.Enc}(\mathsf{pk}_i, x_i; r'_i)$
endfor
$\mathcal{A}^{\mathcal{O}(1^\lambda,\cdot)}(1^\lambda, (\mathsf{pk}_i, \mathsf{ct}_i)_{i\in[N]}, (x_i, r_i, r'_i)_{i\notin H})$
return $\mathsf{View}_{\mathcal{A}}$
```

```
$\mathsf{Ideal}^{\mathcal{A}}(1^\lambda, (x_i)_{i\in[N]})$
$(\mathsf{st}_S, (\overline{\mathsf{pk}_i}, \overline{\mathsf{ct}_i})_{i\in[N]}, (r_i, r'_i)_{i\in[N]\setminus H}) \leftarrow \mathsf{MHE.Sim}_1(1^\lambda, H, (x_i)_{i\notin H})$
$\mathcal{A}^{\mathcal{O}'(1^\lambda,\cdot)}(1^\lambda, (\overline{\mathsf{pk}_i}, \overline{\mathsf{ct}_i})_{i\in[N]}, (x_i, r_i, r'_i)_{i\notin H})$
return $\mathsf{View}_{\mathcal{A}}$
```

```
$\mathcal{O}(1^\lambda, C)$
$\widehat{\mathsf{ct}} \leftarrow \mathsf{MHE.Eval}(C, (\mathsf{ct}_j)_{j\in[N]})$
for $i \in H$ do
    $p_i \leftarrow \mathsf{MHE.PartDec}(\mathsf{sk}_i, i, \widehat{\mathsf{ct}})$
endfor
return $(p_i)_{i\in H}$
```

```
$\mathcal{O}'(1^\lambda, C)$
$(\mathsf{st}'_S, (p_i)_{i\in H}) \leftarrow \mathsf{MHE.Sim}_2(\mathsf{st}_S, C, C((x_i)_{i\in[N]}))$
Update $\mathsf{st}_S = \mathsf{st}'_S$
return $(p_i)_{i\in H}$
```

Definition 3.4. *A scheme* $(\mathsf{MHE.KeyGen}, \mathsf{MHE.Enc}, \mathsf{MHE.Eval}, \mathsf{MHE.PartDec}, \mathsf{MHE.FinDec})$ *is said to satisfy the reusable simulation security if the following holds: there exists two simulators* $(\mathsf{MHE.Sim}_1, \mathsf{MHE.Sim}_2)$ *such that for any PPT adversary $\mathcal{A}$, for any set of honest parties $H \subseteq [N]$, PPT distinguisher $\mathcal{D}$, and any messages $(x_i)_{i\in[N]}$, there exists a negligible function $\nu(\lambda)$ such that*

$$\left| \Pr\left[\mathcal{D}\left(1^\lambda, \mathsf{Real}^{\mathcal{A}}(1^\lambda, (x_i)_{i\in[N]})\right) = 1\right] - \Pr\left[\mathcal{D}\left(1^\lambda, \mathsf{Ideal}^{\mathcal{A}}(1^\lambda, (x_i)_{i\in[N]})\right) = 1\right] \right| < \nu(\lambda)$$

One-Time Simulation Security We say a multiparty homomorphic encryption scheme is a *one-time* multiparty homomorphic encryption scheme, if the security holds for all adversary $\mathcal{A}$ that only query the oracle at most once.

Remark. Definition 3.4 directly captures the reusability property implied by the definition of [38]. However, our definition is somewhat incomparable to [38] due to the following reasons: [38] give a one-time (semi-malicious) statistical simulation security definition for threshold decryption, which implies multi-use security via a standard hybrid argument. In contrast, Definition 3.4, which guarantees (semi-honest) computational security, is given directly for the multi-use setting. Second, [38] define security of threshold decryption only for $n-1$ corruptions[7] whereas our definition captures any dishonest majority.

3.3 pMHE: MHE with Private Evaluation

Towards achieving MHE, we first consider a relaxation of the notion of MHE where we allow the evaluation algorithm to be a private-key procedure. We call this notion *MHE with private evaluation*, denoted by pMHE.

A multiparty homomorphic encryption with private evaluation (pMHE) is a tuple of algorithms $(\mathsf{pMHE.Enc}, \mathsf{pMHE.PrivEval}, \mathsf{pMHE.FinDec})$, which are defined as follows.

$\mathsf{pMHE.Enc}(1^\lambda, C.\mathsf{params}, i, x_i)$ On input the security parameter λ, the parameters of a circuit C, $C.\mathsf{params} = (C.\mathsf{in}, C.\mathsf{out}, C.\mathsf{depth})$, an index i, and an input x_i, it outputs a ciphertext ct_i, and a partial decryption key sk_i.

$\mathsf{pMHE.PrivEval}(\mathsf{sk}_i, i, C, (\mathsf{ct}_j)_{j\in[N]})$ [8] On input the partial decryption key sk_i, an index i, a circuit C, and the ciphertexts $(\mathsf{ct}_j)_{j\in[N]}$, it outputs a partial decryption message p_i.

$\mathsf{pMHE.FinDec}(C, (p_j)_{j\in[N]})$ On input the circuit C and the partial decryptions $(p_j)_{j\in[N]}$, it outputs $y \in \{0,1\}^{C.\mathsf{out}}$.

Correctness For any input $(x_i)_{i\in[N]}$, and any circuit C, we have

$$\Pr\left[\begin{smallmatrix}\forall i\ (\mathsf{ct}_i,\mathsf{sk}_i)\leftarrow\mathsf{pMHE.Enc}(1^\lambda,C.\mathsf{params},i,x_i)\\ \forall i\ p_i\leftarrow\mathsf{pMHE.PrivEval}(\mathsf{sk}_i,i,C,(\mathsf{ct}_j)_{j\in[N]})\\ y\leftarrow\mathsf{pMHE.FinDec}(C,(p_j)_{j\in[N]})\end{smallmatrix} : y = C((x_i)_{i\in[N]})\right] = 1$$

Reusable (resp. One-Time) Simulation Security The experiments are parameterized by adversary $\mathcal{A}$, input $(x_i)_{i\in[N]}$, PPT simulator $\mathsf{MHE.Sim}$ implemented as algorithms $(\mathsf{MHE.Sim}_1, \mathsf{MHE.Sim}_2)$, and subsets of honest parties $H \subseteq [N]$.

[7]As such, counter-intuitively, additional work is required when using it in applications such as MPC, when less than $n-1$ parties may be corrupted. We refer the reader to [38] for details.

[8]In fact, $\mathsf{PrivEval}$ is a combination of private evaluation and partial decryption.

```
$\mathsf{Real}^{\mathcal{A}}(1^\lambda, (x_i)_{i\in[N]})$
for $i \in [N]$ do
    $r_i \leftarrow \{0,1\}^*$
    $(\mathsf{ct}_i, \mathsf{sk}_i) = \mathsf{MHE.Enc}(1^\lambda, C.\mathsf{params}, i, x_i; r_i)$
endfor
$\mathcal{A}^{\mathcal{O}(1^\lambda,\cdot)}(1^\lambda, (\mathsf{ct}_i)_{i\in[N]}, (x_i, r_i)_{i\notin H})$
return $\mathsf{View}_{\mathcal{A}}$
```

```
$\mathsf{Ideal}^{\mathcal{A}}(1^\lambda, (x_i)_{i\in[N]})$
$(\mathsf{st}_S, (\overline{\mathsf{ct}_i})_{i\in[N]}, (x_i, r_i)_{i\notin H}) \leftarrow \mathsf{MHE.Sim}_1(1^\lambda, H, (x_i)_{i\notin H})$
$\mathcal{A}^{\mathcal{O}'(1^\lambda,\cdot)}(1^\lambda, (\overline{\mathsf{ct}_i})_{i\in[N]}, (x_i, r_i)_{i\notin H})$
return $\mathsf{View}_{\mathcal{A}}$
```

```
$\mathcal{O}(1^\lambda, C)$
for $i \in H$ do
    $p_i \leftarrow \mathsf{MHE.PrivEval}(\mathsf{sk}_i, i, C, (\mathsf{ct}_j)_{j\in[N]})$
endfor
return $(p_i)_{i\in H}$
```

```
$\mathcal{O}'(1^\lambda, C)$
$(\mathsf{st}'_S, (p_i)_{i\in H}) \leftarrow \mathsf{MHE.Sim}_2(\mathsf{st}_S, C, C((x_i)_{i\in[N]}))$
Update $\mathsf{st}_S = \mathsf{st}'_S$
return $(p_i)_{i\in H}$
```

Succinctness We define the succinctness of ciphertext and partial decryption of pMHE in the same manner as in Definition 3.2.

3.4 MHE from pMHE and Fully Homomorphic Encryption

We show how to construct an MHE scheme from pMHE and a leveled fully homomorphic encryption scheme.

Theorem 3.5 (From pMHE to MHE)**.** *If there exits a reusable simulation secure pMHE scheme* pMHE *with succinctness property, and a (leveled) fully homomorphic encryption scheme* $\mathsf{FHE} = (\mathsf{FHE.KeyGen}, \mathsf{FHE.Enc}, \mathsf{FHE.Dec}, \mathsf{FHE.Eval})$*, then there exits a reusbale simulation secure MHE scheme* MHE *with succinctness property.*

Construction.

$C'_{i,[C,(\mathsf{ct}_j)_{j\in[N]}]}(\mathsf{pMHE.sk}_i)$ Execute $p_i = \mathsf{pMHE.PrivEval}(1^\lambda, \mathsf{pMHE.sk}_i, i, C, (\mathsf{ct}_j)_{j\in[N]})$.

Output p_i.

$\mathsf{MHE.KeyGen}(1^\lambda, i)$ Execute $(\mathsf{FHE.pk}_i, \mathsf{FHE.sk}_i) \leftarrow \mathsf{FHE.KeyGen}(1^\lambda, 1^{C'.\mathsf{depth}})$.

Let $\mathsf{pk}_i = \mathsf{FHE.pk}_i$, and $\mathsf{sk}_i = \mathsf{FHE.sk}_i$.

Output $(\mathsf{pk}_i, \mathsf{sk}_i)$.

$\mathsf{MHE.Enc}(\mathsf{pk}_i, x_i)$ Parse pk_i as $\mathsf{FHE.pk}_i$.

Execute $(\mathsf{pMHE.ct}_i, \mathsf{pMHE.sk}_i) \leftarrow \mathsf{pMHE.Enc}(1^\lambda, C.\mathsf{params}, i, x_i)$.

Execute $\mathsf{FHE.ct}_i \leftarrow \mathsf{FHE.Enc}(\mathsf{FHE.pk}_i, \mathsf{pMHE.sk}_i)$.

Output $\mathsf{ct}_i = (\mathsf{pMHE.ct}_i, \mathsf{FHE.ct}_i)$.

$\mathsf{MHE.Eval}(C, (\mathsf{ct}_j)_{j\in[N]})$ For each $j \in [N]$, parse ct_j as $(\mathsf{pMHE.ct}_j, \mathsf{FHE.ct}_j)$.

For each $i \in [N]$, execute $\widehat{\mathsf{ct}_i} \leftarrow \mathsf{FHE.Eval}(C'_{i,[C,(\mathsf{pMHE.ct}_j)_{j\in[N]}]}, \mathsf{FHE.ct}_i)$.

Output $(\widehat{\mathsf{ct}_i})_{i\in[N]}$.

$\mathsf{MHE.PartDec}(\mathsf{sk}_i, i, \widehat{\mathsf{ct}_i})$ Parse sk_i as $\mathsf{FHE.sk}_i$.

Execute $p_i \leftarrow \mathsf{FHE.Dec}(\mathsf{FHE.sk}_i, \widehat{\mathsf{ct}_i})$.

Output p_i.

$\mathsf{MHE.FinDec}(C, (p_j)_{j\in[N]})$ Execute $y \leftarrow \mathsf{pMHE.FinDec}(C, (p_j)_{j\in[N]})$.

Output y.

Proof. The correctness and succinctness follows from the correctness and succinctness of the pMHE scheme pMHE and FHE.

For simulation security, we build the following hybrids.

Hybrid_0 This hybrid is identical to the Real.

Hybrid_1 In this hybrid, we replace the oracle $\mathcal{O}(1^\lambda, C)$ with the following, which doesn't use the FHE secret keys $(\mathsf{FHE.sk}_i)_{i\in[N]}$.

Oracle $\mathcal{O}(1^\lambda, C)$ Execute $p_i \leftarrow \mathsf{pMHE.PrivEval}(\mathsf{pMHE.sk}_i, i, C, (\mathsf{pMHE.ct}_j)_{j\in[N]})$.
Output p_i.

$\mathsf{Hybrid}_{1.5}^{i^*}$ We replace the function $\mathsf{MHE.Enc}(\mathsf{pk}_i, x_i)$ with the following, which doesn't use pMHE secret keys for honest parties $(\mathsf{pMHE.sk}_i)_{i\in H}$.

$\mathsf{MHE.Enc}(\mathsf{pk}_i, x_i)$ Execute
$(\mathsf{pMHE.ct}_i, \mathsf{pMHE.sk}_i) \leftarrow \mathsf{pMHE.Enc}(1^\lambda, C.\mathsf{params}, i, m_i)$.
$\underline{\text{If } i \in H \text{ and } i \le i^*, \text{ execute } \mathsf{FHE.ct}_i \leftarrow \mathsf{FHE.Enc}(\mathsf{FHE.pk}_i, 0^{|\mathsf{pMHE.sk}_i|}).}$
Otherwise, execute $\mathsf{FHE.ct}_i \leftarrow \mathsf{FHE.Enc}(\mathsf{FHE.pk}_i, \mathsf{pMHE.sk}_i)$.
Output $\mathsf{ct}_i = (\mathsf{pMHE.ct}_i, \mathsf{FHE.ct}_i)$.

Hybrid_2 We replace the function $\mathsf{MHE.Enc}(\mathsf{pk}_i, x_i)$ with the following, which doesn't use pMHE secret keys for honest parties $(\mathsf{pMHE.sk}_i)_{i\in H}$.

$\mathsf{MHE.Enc}(\mathsf{pk}_i, x_i)$ Execute
$(\mathsf{pMHE.ct}_i, \mathsf{pMHE.sk}_i) \leftarrow \mathsf{pMHE.Enc}(1^\lambda, C.\mathsf{params}, i, m_i)$.
$\underline{\text{If } i \in H, \text{ execute } \mathsf{FHE.ct}_i \leftarrow \mathsf{FHE.Enc}(\mathsf{FHE.pk}_i, 0^{|\mathsf{pMHE.sk}_i|}).}$
Otherwise, execute $\mathsf{FHE.ct}_i \leftarrow \mathsf{FHE.Enc}(\mathsf{FHE.pk}_i, \mathsf{pMHE.sk}_i)$.
Output $\mathsf{ct}_i = (\mathsf{pMHE.ct}_i, \mathsf{FHE.ct}_i)$.

Ideal We replace the Hybrid_2 with the ideal world, where the simulators are defined as follows.

$\mathsf{MHE.Sim}_1(1^\lambda, H, (x_i)_{i\notin H})$ For each $i \in [N]$, randomly sample random coins r_i, r'_i.
execute $(\mathsf{FHE.pk}_i, \mathsf{FHE.sk}_i) = \mathsf{FHE.KeyGen}(1^\lambda; r_i)$.
$\underline{\text{Execute } (\mathsf{pMHE.st}_S, (\mathsf{pMHE.ct}_i)_{i\in[N]}, (\mathsf{pMHE}.r_i)_{i\notin H}) \leftarrow \mathsf{pMHE.Sim}_1(1^\lambda, H, (x_i)_{i\notin H}).}$
For each $i \in H$, execute $\mathsf{FHE.ct}_i \leftarrow \mathsf{FHE.Enc}(\mathsf{FHE.pk}_i, 0^{|\mathsf{pMHE.sk}_i|})$.
For each $i \notin H$, let $(\mathsf{pMHE.ct}_i, \mathsf{pMHE.sk}_i) = \mathsf{pMHE.Enc}(1^\lambda, x_i; \mathsf{pMHE}.r_i)$,
and $\mathsf{FHE.ct}_i = \mathsf{FHE.Enc}(\mathsf{FHE.pk}_i, \mathsf{pMHE.sk}_i; r'_i)$.
Let $\mathsf{ct}_i = (\mathsf{pMHE.ct}_i, \mathsf{FHE.ct}_i)$, and $\mathsf{st}_S = \mathsf{pMHE.st}_S$.
Output $(\mathsf{pMHE.st}_S, (\mathsf{ct}_i)_{i\in[N]}, (x_i, r_i, (\mathsf{pMHE}.r_i, r'_i))_{i\notin H})$.

$\mathsf{MHE.Sim}_2(\mathsf{st}_S, C, C((x_i)_{i\in[N]}))$ $\underline{\text{Execute } (p_i)_{i\in H} \leftarrow \mathsf{pMHE.Sim}_2(\mathsf{pMHE.st}_S, C, C((x_i)_{i\in[N]})).}$

Output $(\mathsf{st}_S, p_i)_{i\in H}$.

Lemma 3.6. $\mathsf{Hybrid}_0, \mathsf{Hybrid}_1$, *and* $\mathsf{Hybrid}^0_{1.5}$ *are identical. For any* $i^* \in [N]$*, and any PPT distinguisher* $\mathcal{D}$*, there exits a negligible function* $\nu(\lambda)$ *such that* $|\Pr[\mathcal{D}(1^\lambda, \mathsf{Hybrid}^{i^*-1}_{1.5}) = 1] - \Pr[\mathcal{D}(1^\lambda, \mathsf{Hybrid}^{i^*}_{1.5}) = 1]| < \nu(\lambda)$.

Proof. From correctness of the (leveled) FHE, Hybrid_0 and Hybrid_1 are identical. In $\mathsf{Hybrid}^{i^*}_{1.5}$, when $i^* = 0$, all $\mathsf{FHE.ct}_i$ are generated in the same manner as the real exeuction. Hence, Hybrid_1 and $\mathsf{Hybrid}^0_{1.5}$ are identical.

For any PPT adversary $\mathcal{A}$ and distinguisher $\mathcal{D}$, we build the following distinguisher $\mathcal{D}'$ breaking the ciphertext-indistinguishable security of FHE.

Adversary $\mathcal{D}'(1^\lambda, \mathsf{FHE.pk})$ For each $i \in [N]$, randomly sample $r_i, r'_i, \mathsf{pMHE}.r_i$, and

execute $(\mathsf{FHE.pk}_i, \mathsf{FHE.sk}_i) = \mathsf{FHE.KeyGen}(1^\lambda; r_i)$, and

execute $(\mathsf{pMHE.ct}_i, \mathsf{pMHE.sk}_i) = \mathsf{pMHE.Enc}(1^\lambda, C.\mathsf{params}, i, x_i; \mathsf{pMHE}.r_i)$.

For each $i \in H$ and $i < i^*$, execute $\mathsf{FHE.ct}_i \leftarrow \mathsf{FHE.Enc}(\mathsf{FHE.pk}_i, 0^{|\mathsf{pMHE.sk}_i|})$.

Query the challenger with plaintext $(0^{|\mathsf{pMHE.sk}_{i^*}|}, \mathsf{pMHE.sk}_{i^*})$.

Then the challenger sends a challenge ciphertext ct.

For $i = i^* \in H$, Let $\mathsf{FHE.pk}_{i^*} = \mathsf{FHE.pk}, \mathsf{FHE.ct}_{i^*} = \mathsf{ct}$.

For each $i \notin H$ or $i > i^*$, let $\mathsf{FHE.ct}_i = \mathsf{FHE.Enc}(1^\lambda, \mathsf{FHE.pk}_i; r'_i)$.

Let $\mathsf{pk}_i = \mathsf{FHE.pk}_i, \mathsf{ct}_i = (\mathsf{pMHE.ct}_i, \mathsf{FHE.ct}_i)$.

Execute $\mathcal{A}^{\mathcal{O}_\mathcal{A}(1^\lambda,\cdot)}(1^\lambda, (\mathsf{pk}_i, \mathsf{ct}_i)_{i\in[N]}, (x_i, r_i, (\mathsf{pMHE}.r_i, r'_i))_{i\notin H})$.

Let $b \leftarrow \mathcal{D}(1^\lambda, \mathsf{View}_\mathcal{A})$.

Output b.

Oracle $\mathcal{O}_\mathcal{A}(1^\lambda, C)$ For each $i \in H$, execute

$p_i = \mathsf{pMHE.PrivEval}(1^\lambda, \mathsf{pMHE.sk}_i, i, C, (\mathsf{pMHE.ct}_j)_{j\in[N]})$.

Output $(p_i)_{i\in H}$.

When the challenger ct is generated by $\mathsf{FHE.Enc}(\mathsf{FHE.pk}, 0^{|\mathsf{pMHE.sk}_{i^*}|})$, then the adversary $\mathcal{D}'$ simulates the environment of $\mathsf{Hybrid}^{i^*}_{1.5}$ for $\mathcal{A}$. Hence,

$$\Pr\left[\mathsf{ct} \leftarrow \mathsf{FHE.Enc}(\mathsf{FHE.pk}, 0^{|\mathsf{pMHE.sk}_{i^*}|}) : \mathcal{D}'(1^\lambda, \mathsf{FHE.pk}) = 1\right] = \Pr[\mathcal{D}(1^\lambda, \mathsf{Hybrid}^{i^*}_{1.5}) = 1] \quad (1)$$

When the challenger ct is generated by $\mathsf{FHE.Enc}(\mathsf{FHE.pk}, \mathsf{pMHE.sk}_{i^*})$, then the adversary $\mathcal{D}'$ simulates the environment of $\mathsf{Hybrid}^{i^*-1}_{1.5}$ for $\mathcal{A}$. Hence,

$$\Pr\left[\mathsf{ct} \leftarrow \mathsf{FHE.Enc}(\mathsf{FHE.pk}, \mathsf{pMHE.sk}_{i^*}) : \mathcal{D}'(1^\lambda, \mathsf{FHE.pk}) = 1\right] = \Pr[\mathcal{D}(1^\lambda, \mathsf{Hybrid}^{i^*-1}_{1.5}) = 1] \quad (2)$$

By the security of FHE, there exits a negligible function $\nu(\lambda)$ such that the difference of the left hand sides of Equation (1) and (2) is bounded by $\nu(\lambda)$. Hence, we have $|\Pr[\mathcal{D}(1^\lambda, \mathsf{Hybrid}^{i^*-1}_{1.5}) = 1] - \Pr[\mathcal{D}(1^\lambda, \mathsf{Hybrid}^{i^*}_{1.5}) = 1]| < \nu(\lambda)$. □

Lemma 3.7. $\mathsf{Hybrid}_{1.5}^{N}$ *is identical to* Hybrid_2. *For any PPT distinguisher $\mathcal{D}$, there exits a negligible function $\nu(\lambda)$ such that* $|\Pr[\mathcal{D}(1^\lambda, \mathsf{Hybrid}_2^{\mathcal{A}}) = 1] - \Pr[\mathcal{D}(1^\lambda, \mathsf{Ideal}^{\mathcal{A}}) = 1]| < \nu(\lambda)$.

Proof. When $i^* = N$, all $\mathsf{FHE.ct}_i$ are generated by encrypting $0^{|\mathsf{pMHE.sk}_i|}$. Hence, $\mathsf{Hybrid}_{1.5}^{N}$ is identical to Hybrid_2. For any PPT adversary $\mathcal{A}$, and distinguisher $\mathcal{D}$, we build the following adversary $\mathcal{A}'$ for pMHE.

Adversary $\mathcal{A}'^{\mathcal{O}_{\mathcal{A}'}}(1^\lambda, (\mathsf{pMHE.ct}_i)_{i\in[N]}, (x_i, \mathsf{pMHE}.r_i)_{i\notin H})$ For each $i \in [N]$, randomly sample r_i, r'_i.

Execute $(\mathsf{FHE.pk}_i, \mathsf{FHE.sk}_i) = \mathsf{FHE.KeyGen}(1^\lambda; r_i)$.

For each $i \notin H$, $(\mathsf{pMHE.ct}_i, \mathsf{pMHE.sk}_i) = \mathsf{pMHE.Enc}(1^\lambda, x_i; \mathsf{pMHE}.r_i)$, and let

$\mathsf{FHE.ct}_i = \mathsf{FHE.Enc}(\mathsf{FHE.pk}_i, \mathsf{pMHE.sk}_i; r'_i)$.

For each $i \in H$, let $\mathsf{FHE.ct}_i \leftarrow \mathsf{FHE.Enc}(\mathsf{FHE.pk}_i, 0^{|\mathsf{pMHE.sk}_i|})$.

For each $i \in [N]$, let $\mathsf{ct}_i = (\mathsf{pMHE.ct}_i, \mathsf{FHE.ct}_i)$.

Invoke $\mathcal{A}^{\mathcal{O}_{\mathcal{A}}}(1^\lambda, (\mathsf{ct}_i)_{i\in[N]}, (x_i, r_i, (\mathsf{pMHE}.r_i, r'_i))_{i\notin H})$.

Output $\mathsf{View}_{\mathcal{A}}$.

Oracle $\mathcal{O}_{\mathcal{A}}(1^\lambda, C)$ The adversary $\mathcal{A}'$ queries the oracle $\mathcal{O}_{\mathcal{A}'}(1^\lambda, \cdot)$ with C, and obtains $(p_i)_{i\in H}$.

Output $(p_i)_{i\in H}$.

When $\mathcal{A}'$ is interacting with Real world, it simulates the Hybrid_2 for $\mathcal{A}$. Hence,

$$\Pr\left[\mathcal{D}(1^\lambda, \mathsf{Real}^{\mathcal{A}'}) = 1\right] = \Pr\left[\mathcal{D}(1^\lambda, \mathsf{Hybrid}_2^{\mathcal{A}}) = 1\right]$$

When $\mathcal{A}'$ is interacting with Ideal world, it simulates the Ideal world for $\mathcal{A}$. Hence,

$$\Pr\left[\mathcal{D}(1^\lambda, \mathsf{Ideal}^{\mathcal{A}'}) = 1\right] = \Pr\left[\mathcal{D}(1^\lambda, \mathsf{Ideal}^{\mathcal{A}}) = 1\right]$$

Since the pMHE scheme is simulation secure, there exits a negligible function $\nu(\lambda)$ such that $|\Pr[\mathcal{D}(1^\lambda, \mathsf{Ideal}^{\mathcal{A}'}) = 1] - \Pr[\mathcal{D}(1^\lambda, \mathsf{Real}^{\mathcal{A}'}) = 1]| < \nu(\lambda)$.

Hence, we have $|\Pr[\mathcal{D}(1^\lambda, \mathsf{Hybrid}_2^{\mathcal{A}}) = 1] - \Pr[\mathcal{D}(1^\lambda, \mathsf{Ideal}^{\mathcal{A}}) = 1]| < \nu(\lambda)$. ☐

We finish the proof by combining Lemma 3.6 and Lemma 3.7. ☐

4 One-Time pMHE

In this section, we focus on constructing a one-time pMHE scheme; recall that a one-time pMHE scheme is one where the adversary is allowed to make only one decryption query. Later, we show how to bootstrap a one-time pMHE scheme into a reusable pMHE scheme; it turns out that the underlying one-time pMHE scheme needs to satisfy certain ciphertext succintness property in order for the bootstrapping step to work.

We first define two notions of ciphertext succinctness: weak ciphertext succinctness and strong ciphertext succinctness. We show how to achieve pMHE with strong ciphertext succinctness from pMHE with weak ciphertext succinctness assuming laconic oblivious transfer. Later in Section 5, we show how to achieve reusable pMHE scheme from a one-time pMHE scheme satisfying strong succinctness property.

4.1 Ciphertext Succinctness

We define two notions of ciphertext succinctness associated with a pMHE scheme.

Definition 4.1 (Weak Ciphertext Succinctness)**.** *A pMHE scheme* $(\mathsf{pMHE.Enc}, \mathsf{pMHE.PrivEval}, \mathsf{pMHE.FinDec})$ *is said to satisfy weak ciphertext succinctness property if it satisfies the correctness, reusable simulation security of a pMHE scheme and in addition, satisfies the following property:*

- *The* running time *of the* $\mathsf{pMHE.Enc}$ *circuit is* $\mathsf{poly}(\lambda, N, C.\mathsf{in}, C.\mathsf{out}, C.\mathsf{depth})$.

where N is the number of parties, and $(C.\mathsf{in}, C.\mathsf{out}, C.\mathsf{depth})$ are the parameters associated with the circuits being evaluated.

Definition 4.2 (Strong Ciphertext Succinctness)**.** *A pMHE scheme* $(\mathsf{pMHE.Enc}, \mathsf{pMHE.PrivEval}, \mathsf{pMHE.FinDec})$ *is said to satisfy weak ciphertext succinctness property if it satisfies the correctness, reusable simulation security of a pMHE scheme and in addition, satisfies the following properties:*

- *It satisfies weak ciphertext succinctness.*
- *The depth of the* $\mathsf{pMHE.Enc}$ *circuit is* $\mathsf{poly}(\lambda, \log N, \log(C.\mathsf{in}), \log(C.\mathsf{out}), \log(C.\mathsf{depth}))$.
- *The output length of the* $\mathsf{pMHE.Enc}$ *circuit is* $\mathsf{poly}(\lambda, \log N, \log(C.\mathsf{in}), \log(C.\mathsf{out}), \log(C.\mathsf{depth}))$.

where N is the number of parties, and $(C.\mathsf{in}, C.\mathsf{out}, C.\mathsf{depth})$ are the parameters associated with the circuits being evaluated.

Remark 4.3. *The weak ciphertext succinctness is weaker than the succinctness property (Definition 3.2) of a pMHE scheme. On the other hand, the strong ciphertext succinctness property is incomparable with the succinctness property of an MHE scheme; while there is no requirement on the size of the partial decryptions in the above definitions, there is a strict requirement on the complexity of the encryption procedure in the above definition as against a requirement on just the size of the ciphertexts as specified in the succinctness definition of MHE.*

4.2 One-Time pMHE with Weak Ciphertext Succinctness

We now show how to generically transform a non-succinct pMHE scheme into a succinct pMHE scheme. This, in particular, implies that the resulting pMHE scheme also achieves weak ciphertext succinctness property as defined above. Furthermore, the transformation preserves the number of queries the adversary can make to the decryption oracle. That is, if the underlying pMHE scheme is *reusable*, then so is the resulting scheme.

Theorem 4.4. *Assuming LWE, there exists a generic transformation from any non-succinct (Remark 3.3) one-time (resp., reusable) pMHE to a succinct (Definition 3.2) one-time (resp., reusable) pMHE scheme.*

Proof. Let NSpMHE be a non-succinct pMHE scheme. We show how to transform NSpMHE into a succinct pMHE scheme SpMHE. We use laconic function evaluation (Section 2.4) as an intermediate tool in this construction. Denote the algorithms associated with laconic function evaluation to be $\mathsf{LFE} = (\mathsf{LFE.crsGen}, \mathsf{LFE.Compress}, \mathsf{LFE.Enc}, \mathsf{LFE.Dec})$. Our construction, proceeds along the same lines as the construction of low-communication secure MPC in [42].[9]

[9]One can also use functional encryption combiners [2] to achieve the same result.

$\mathsf{SpMHE.Enc}(1^\lambda, C.\mathsf{params}, i, x_i)$: On input security parameter λ, circuit parameters $C.\mathsf{params}$, index $i \in [N]$, input x_i, compute $\mathsf{ct}_i^0 \leftarrow \mathsf{NSpMHE.Enc}(1^\lambda, G.\mathsf{params}, i, (x_i, r_i))$; where $G.\mathsf{params}$ are circuit parameters associated with the $\mathsf{LFE.Enc}$ circuit and r_i is uniformly chosen at random. Output the ciphertext ct_i^0. If $i = 1$, also additionally output crs, where $\mathsf{crs} \leftarrow \mathsf{LFE.crsGen}(1^\lambda, C.\mathsf{params})$.

$\mathsf{SpMHE}_{wk}.\mathsf{PrivEval}(1^\lambda, \mathsf{sk}_i, i, C, (\mathsf{ct}_j)_{j\in[N]})$ On input the security parameter λ, secret key sk_i, index $i \in [N]$, circuit C, ciphertexts $(\mathsf{ct}_j)_{j\in[N]}$, first compute $\mathsf{digest}_C \leftarrow \mathsf{Compress}(1^\lambda, C)$.[10] Then compute $\mathsf{NSpMHE.PrivEval}(1^\lambda, \mathsf{sk}_i, i, G, (\mathsf{ct}'_j)_{j\in[N]})$, where $\mathsf{ct}_1 = (\mathsf{ct}'_1, \mathsf{crs})$, for every $j \neq 1$, $\mathsf{ct}_j = \mathsf{ct}'_j$, and G takes as input $((x_1, r_1), \ldots, (x_N, r_N))$ and computes $p'_i \leftarrow \mathsf{LFE.Enc}(\mathsf{crs}, \mathsf{digest}_C, (x_1, \ldots, x_N); \oplus_{i\in[N]} r_i)$. Output the partial decryption $p_i = (\mathsf{crs}, p'_i)$.

$\mathsf{SpMHE.FinDec}(1^\lambda, C, (p_j)_{j\in[N]})$ On input security parameter λ, circuit C, partial decryptions $(p_j)_{j\in[N]}$, first compute $\mathsf{NSpMHE.FinDec}(1^\lambda, C, (p'_j)_{j\in[N]})$ to obtain $\mathsf{LFE.ct}$. Compute $\mathsf{LFE.Dec}(\mathsf{crs}, C, \mathsf{LFE.ct})$ to obtain the output y.

To argue succinctness, we show the following:

- First, we argue about the size of the ciphertexts output by $\mathsf{SpMHE.Enc}$. To do this, it suffices to separately bound the sizes of the CRS of the LFE scheme annd the size of the ciphertexts of NSpMHE scheme. From the efficiency property of the LFE scheme, it follows that the size of CRS is $\mathsf{poly}(\lambda, C.\mathsf{in}, C.\mathsf{out}, C.\mathsf{depth})$. The size of ciphertexts of NSpMHE is $\max_i(\mathsf{poly}(\lambda, |x_i|, |r_i|))$, which is $\mathsf{poly}(\lambda)$.
- Next, we argue about the size of the partial decryption values output by $\mathsf{SpMHE.PrivEval}$. Since we already established an upper bound on the size of the CRS output by SpMHE, it suffices to establish an upper bound on the ciphertexts of the LFE scheme. Again from the efficiency of the LFE scheme, we have that the size of the ciphertexts output by $\mathsf{LFE.Enc}$ is $\mathsf{poly}(\lambda, C.\mathsf{in}, C.\mathsf{out}, C.\mathsf{depth})$.

We omit the proof of security since it follows from [42]. Since laconic function evaluation can be based on the hardness of learning with errors [42], we have the theorem.

□

Corollary 4.5. *Assuming LWE, there is a generic transformation that converts any delayed-function semi-honest secure MPC into a one-time pMHE scheme satisfying weak ciphertext succinctness property.*

Proof. If we can show any delayed-function semi-honest secure MPC can be converted to a pMHE scheme, that is possibly non-succinct, then from Theorem 4.4, we can further convert it to a succinct pMHE scheme; and in particular, the resulting scheme satisfies the weak ciphertext succinctness property. This would finish the proof.

Now we describe how to convert any delayed-function semi-honest secure MPC to a pMHE scheme. Since this construction is simple, we only give a sketch of the construction below.

- The i^{th} party, for $i \in [N]$, on input x_i produces the first round message $\mathsf{msg}_1^{(1)}$ of the delayed-function MPC protocol along with the private state st_i. The ciphertext ct_i is set to be $\mathsf{msg}_i^{(1)}$ and the secret key sk_i is set to be the state st_i.

[10]Note that we crucially use the fact that $\mathsf{LFE.Compress}$ is a deterministic algorithm to argue that all parties will compute the same digest_C.

- The private evaluation phase corresponds to the computation of second round messages. The i^{th} party on input all the ciphertexts $\mathsf{ct}_1, \ldots, \mathsf{ct}_N$, i.e., messages $\mathsf{msg}_1^{(1)}, \ldots, \mathsf{msg}_N^{(1)}$, circuit C and its secret key $\mathsf{sk}_i = st_i$, it produces the second round message $\mathsf{msg}_i^{(2)}$. We interpret $\mathsf{msg}_i^{(2)}$ as the partial decrypted value p_i.
- Finally, given all the partial decrypted values (aka the second round messages), we can recover the output $C(x_1, \ldots, x_N)$.

$\square$

4.3 From Weak to Strong Ciphertext Succinctness

We show how to generically achieve strong ciphertext succinctness from weak succinctness assuming laconic OT and randomized encodings.

Lemma 4.6 (From Weak to Strong Ciphertext Succinctness)**.** *Assuming the existence of laconic oblivious transfer and randomized encodings computable in* NC^1*, there is a generic transformation that transforms a pMHE scheme* $\mathsf{pMHE}' = (\mathsf{pMHE}'.\mathsf{Enc}, \mathsf{pMHE}'.\mathsf{PrivEval}, \mathsf{pMHE}'.\mathsf{FinDec})$ *satisfying weak ciphertext succinctness, into a pMHE scheme* $\mathsf{pMHE} = (\mathsf{pMHE}.\mathsf{Enc}, \mathsf{pMHE}.\mathsf{PrivEval}, \mathsf{pMHE}.\mathsf{FinDec})$ *satisfying strong ciphertext succinctness.*

We construct the pMHE scheme pMHE as follows.

Construction.

$\mathsf{pMHE}.\mathsf{Enc}(1^\lambda, C.\mathsf{params}, i, x_i)$ Since the function $\mathsf{pMHE}'.\mathsf{Enc}$ outputs two parts: ct_i and sk_i, let $\mathsf{pMHE}'.\mathsf{Enc}_1$ be the function that outputs the first part ct_i.

Let $\widehat{\mathsf{pMHE}'.\mathsf{Enc}}_1$ be the randomized encoding of the function $\mathsf{pMHE}'.\mathsf{Enc}_1(\cdot, \cdot, \cdot, \cdot; \cdot)$.

Randomly sample random coins r_i for $\mathsf{pMHE}'.\mathsf{Enc}$.

Randomly sample $\widehat{r_i}$, and execute $\widehat{\mathsf{ct}_i} = \widehat{\mathsf{pMHE}'.\mathsf{Enc}}_1((1^\lambda, C.\mathsf{params}, i, x_i; r_i); \widehat{r_i})$.

Execute $\mathsf{LOT}.\mathsf{crs}_i \leftarrow \mathsf{LOT}.\mathsf{Gen}(1^\lambda)$, and let $\mathsf{digest}_i \leftarrow \mathsf{LOT}.\mathsf{Hash}(\mathsf{LOT}.\mathsf{crs}_i, \widehat{\mathsf{ct}_i})$.

Output $\mathsf{ct}_i = (\mathsf{LOT}.\mathsf{crs}_i, \mathsf{digest}_i)$, and $\mathsf{sk}_i = (\widehat{\mathsf{ct}_i}, C.\mathsf{params}, x_i; r_i)$.

$\mathsf{KG}_{[\mathsf{sk}'_i, C, r'_i]}((\widehat{\mathsf{ct}_j})_{j\in[N]})$ For each $j \in [N]$, execute $\mathsf{ct}_j \leftarrow \mathsf{RE}.\mathsf{Recover}(\widehat{\mathsf{ct}_j})$.

Execute $p'_i = \mathsf{pMHE}'.\mathsf{PrivEval}(\mathsf{sk}'_i, i, C, (\mathsf{ct}_j)_{j\in[N]}; r'_i)$.

Output p'_i.

$\mathsf{pMHE}.\mathsf{PrivEval}(\mathsf{sk}_i, i, C, (\mathsf{ct}_j)_{j\in[N]})$ Parse sk_i as $(\widehat{\mathsf{ct}_i}, C.\mathsf{params}, x_i; r_i)$.

Compute sk'_i from $\mathsf{pMHE}'.\mathsf{Enc}(1^\lambda, C.\mathsf{params}, x_i; r_i)$.

Sample random coins r'_i for $\mathsf{pMHE}'.\mathsf{PrivEval}$.

Randomly sample the labels lab_i, parse lab_i as $(\mathsf{lab}_{i,j,k,b})_{j\in[N],k\in[|\widehat{\mathsf{ct}_j}|],b\in\{0,1\}}$.

Execute $\widetilde{\mathsf{KG}_i} \leftarrow \mathsf{GC}.\mathsf{Garble}(1^\lambda, \mathsf{KG}_{[\mathsf{sk}'_i, C, r'_i]}, \mathsf{lab}_i)$.

For each $j \in [N]$, parse ct_j as $\mathsf{ct}_j = (\mathsf{LOT}.\mathsf{crs}_j, \mathsf{digest}_j)$.

For each $j \in [N], k \in [|\widehat{\mathsf{ct}_j}|]$, execute

$\mathsf{LOT}.\mathsf{ct}_{i,j,k} \leftarrow \mathsf{LOT}.\mathsf{Enc}(\mathsf{LOT}.\mathsf{crs}_j, \mathsf{digest}_j, k, \mathsf{lab}_{i,j,k,0}, \mathsf{lab}_{i,j,k,1})$.

Output $p_i = (\widetilde{\mathsf{KG}_i}, \widehat{\mathsf{ct}_i}, (\mathsf{LOT}.\mathsf{ct}_{i,j,k})_{j,k}, \mathsf{ct}_i)$.

$\mathsf{pMHE.FinDec}(C, (p_i)_{i\in[N]})$ For each $i \in [N]$, parse p_i as $(\widetilde{\mathsf{KG}}_i, \widehat{\mathsf{ct}_i}, (\mathsf{LOT.ct}_{i,j,k})_{j,k}, \mathsf{ct}_i)$, and parse ct_i as $(\mathsf{LOT.crs}_i, \mathsf{digest}_i)$.

For each $i, j \in [N]$, and $k \in [|\widehat{\mathsf{ct}_j}|]$, execute $\mathsf{lab}_{i,j,k} \leftarrow \mathsf{LOT.Dec}(\mathsf{LOT.crs}_j, \mathsf{LOT.ct}_{i,j,k}, \widehat{\mathsf{ct}_j})$.

For each $i \in [N]$, evaluate $p'_i \leftarrow \mathsf{GC.Eval}(\widetilde{\mathsf{KG}}_i, (\mathsf{lab}_{i,j,k})_{j,k})$, and $\mathsf{ct}_i \leftarrow \mathsf{RE.Recover}(\widehat{\mathsf{ct}}_i)$.

Finally execute $y \leftarrow \mathsf{pMHE}'.\mathsf{FinDec}(C, (p'_i)_{i\in[N]})$, and output y.

Lemma 4.7 (Correctness)**.** *The construction of* pMHE *is correct.*

Proof. For any input $(x_i)_{i\in[N]}$, any circuit C, and any $i \in [N]$, let $(\mathsf{ct}_i, \mathsf{sk}_i) \leftarrow \mathsf{pMHE.Enc}(1^\lambda, C.\mathsf{params}, i, x_i)$. For any $i \in [N]$, let $p_i \leftarrow \mathsf{pMHE.PrivEval}(\mathsf{sk}_i, i, C, (\mathsf{ct}_j)_{j\in[N]})$.

For each $i, j \in [N]$, $k \in [|\widehat{\mathsf{ct}_j}|]$, $\mathsf{lab}_{i,j,k} \leftarrow \mathsf{LOT.Dec}(\mathsf{LOT.crs}_j, \mathsf{LOT.ct}_{i,j,k}, \widehat{\mathsf{ct}_j})$.

From the correctness of the laconic OT, we have $\mathsf{lab}_{i,j,k} = \mathsf{lab}_{i,j,k,\widehat{\mathsf{ct}_j}[k]}$.

From the correctness of the garbling scheme, we have $p'_i = \mathsf{KG}_{[\mathsf{sk}'_i, C, r'_i]}((\widehat{\mathsf{ct}_j})_{j\in[N]})$.

In circuit KG, it first executes $\mathsf{ct}_j \leftarrow \mathsf{RE.Recover}(\widehat{\mathsf{ct}_j})$ for each $j \in [N]$. From the correctness of the randomized encoding scheme, we have $\mathsf{ct}_j = \mathsf{pMHE}'.\mathsf{Enc}_1(1^\lambda, C.\mathsf{params}, i, x_j; r_j)$. Finally, the circuit KG outputs $p'_i = \mathsf{pMHE}'.\mathsf{PrivEval}(\mathsf{sk}'_i, C, (\mathsf{ct}_j)_{j\in[N]}; r'_i)$. From the correctness of pMHE', since $y \leftarrow \mathsf{pMHE}'.\mathsf{FinDec}(C, (p'_i)_{i\in[N]})$, we derive that $y = C((x_i)_{i\in[N]})$. □

Lemma 4.8 (One-Time Simulation Security)**.** *The construction of* pMHE *is one-time simulation secure.*

Proof. For any adversary $\mathcal{A}$, and any subset $H \subseteq [N]$, and any input $(x_i)_{i\in[N]}$, we prove the Lemma by a series of hybrids.

Hybrid_0 This hybrid is identical to the real execution $\mathsf{Real}^{\mathcal{A}}(1^\lambda, (x)_{i\in[N]})$.

$\mathsf{Hybrid}_1^{i^*}$ In this hybrid, we replace the $\mathsf{pMHE.Enc}$ to the following procedure, and keep all other parts the same as Hybrid_0.

$\mathsf{pMHE.Enc}(1^\lambda, C.\mathsf{params}, i, x_i)$ Randomly sample random coins r_i for $\mathsf{pMHE}'.\mathsf{Enc}$.

$\underline{\text{If } i \le i^* \text{ and } i \in H, \text{ execute } \mathsf{ct}_i = \mathsf{pMHE}'.\mathsf{Enc}_1(1^\lambda, C.\mathsf{params}, i, x_i; r_i), \text{ and } \widehat{\mathsf{ct}_i} \leftarrow \mathsf{RE.Sim}(1^\lambda, \mathsf{ct}_i).}$

$\underline{\text{Otherwise, let } \widehat{\mathsf{ct}_i} \leftarrow \widehat{\mathsf{pMHE.Enc}'_1}(1^\lambda, C.\mathsf{params}, i, x_i, r_i).}$

Execute $\mathsf{LOT.crs}_i \leftarrow \mathsf{LOT.Gen}(1^\lambda)$, and let $\mathsf{digest}_i \leftarrow \mathsf{LOT.Hash}(\mathsf{LOT.crs}_i, \widehat{\mathsf{ct}_i})$.

Output $\mathsf{ct}_i = (\mathsf{LOT.crs}_i, \mathsf{digest}_i)$, and $\mathsf{sk}_i = (\widehat{\mathsf{ct}_i}, C.\mathsf{params}, x_i; r_i)$.

$\mathsf{Hybrid}_2^{(i^*,j^*,k^*)}$ In this hybrid, we replace the $\mathsf{pMHE.PrivEval}$ with the following procedure, and keep all other parts the same as Hybrid_1^N.

$\mathsf{pMHE.PrivEval}(1^\lambda, \mathsf{sk}_i, i, C, (\mathsf{ct}_j)_{j\in[N]})$ Parse sk_i as $(\widehat{\mathsf{ct}_i}, C.\mathsf{params}, x_i; r_i)$.

Compute sk'_i from $\mathsf{pMHE}'.\mathsf{Enc}(1^\lambda, C.\mathsf{params}, x_i; r_i)$.

Sample random coins r'_i for $\mathsf{pMHE}'.\mathsf{PrivEval}$.

Execute $\widetilde{\mathsf{KG}}_i \leftarrow \mathsf{GC.Garble}(1^\lambda, \mathsf{KG}_{[\mathsf{sk}'_i, C, r'_i]}, \mathsf{lab}_i)$.

For each $j \in [N]$, parse ct_j as $\mathsf{ct}_j = (\mathsf{LOT.crs}_j, \mathsf{digest}_j)$.

For each $j \in [N], k \in [|\widehat{\mathsf{ct}_j}|]$, $\underline{\text{if } (i,j,k) \le (i^*, j^*, k^*), \text{ let } b_{j,k} = \widehat{\mathsf{ct}_j}[k],}$

$\underline{\text{execute } \mathsf{LOT.ct}_{i,j,k} \leftarrow \mathsf{LOT.Enc}(\mathsf{LOT.crs}_j, \mathsf{digest}_j, k, \mathsf{lab}_{i,j,k,b_{j,k}}, \mathsf{lab}_{i,j,k,b_{j,k}}).}$

$\underline{\text{Otherwise, execute } \mathsf{LOT.ct}_{i,j,k} \leftarrow \mathsf{LOT.Enc}(\mathsf{LOT.crs}_j, \mathsf{digest}_j, k, \mathsf{lab}_{i,j,k,0}, \mathsf{lab}_{i,j,k,1}).}$

Output $p_i = (\widetilde{\mathsf{KG}}_i, \widehat{\mathsf{ct}_i}, (\mathsf{LOT.ct}_{i,j,k})_{j,k}, \mathsf{ct}_i)$.

$\mathsf{Hybrid}_3^{i^*}$ In this hybrid, we replace the $\mathsf{pMHE.PrivEval}$ with the following procedure, and keep all other parts the same as $\mathsf{Hybrid}_2^{N,N,|\widehat{\mathsf{ct}_N}|}$.

$\mathsf{pMHE.PrivEval}(1^\lambda, \mathsf{sk}_i, i, C, (\mathsf{ct}_j)_{j\in[N]})$ Parse sk_i as $(\widehat{\mathsf{ct}_i}, C.\mathsf{params}, x_i; r_i)$.

Compute sk'_i from $\mathsf{pMHE}'.\mathsf{Enc}(1^\lambda, C.\mathsf{params}, x_i; r_i)$.

Sample random coins r'_i for $\mathsf{pMHE}'.\mathsf{PrivEval}$.

If $i > i^*$, randomly sample labels lab_i, execute $\widetilde{\mathsf{KG}_i} \leftarrow \mathsf{GC.Garble}(1^\lambda, \mathsf{KG}, \mathsf{lab}_i)$.

Parse lab_i as $(\mathsf{lab}_{i,j,k,b})_{j\in[N],k\in[|\widehat{\mathsf{ct}_j}|],b\in\{0,1\}}$. Let $\mathsf{lab}'_{i,j,k} = \mathsf{lab}_{i,j,k,\widehat{\mathsf{ct}_j}[k]}$.

Otherwise, execute $\mathsf{lab}'_i \leftarrow \mathsf{GC.Sim}_1(1^\lambda, \mathsf{KG.in})$, and $\widetilde{\mathsf{KG}_i} \leftarrow \mathsf{GC.Sim}_2(1^\lambda, \mathsf{KG}((\mathsf{ct}_j)_{j\in[N]}), \mathsf{lab}'_i)$.

Parse lab'_i as $\mathsf{lab}'_{i,j,k}$.

For each $j \in [N]$, parse ct_j as $\mathsf{ct}_j = (\mathsf{LOT.crs}_j, \mathsf{digest}_j)$.

For each $j \in [N], k \in [|\widehat{\mathsf{ct}_j}|]$, let $b_{j,k} = \widehat{\mathsf{ct}_j}[k]$, and execute $\mathsf{LOT.ct}_{i,j,k} \leftarrow \mathsf{LOT.Enc}(\mathsf{LOT.crs}_j, \mathsf{digest}_j, k, \mathsf{lab}'_{i,j,k}, \mathsf{lab}'_{i,j,k})$.

Output $p_i = (\widetilde{\mathsf{KG}_i}, \widehat{\mathsf{ct}_i}, (\mathsf{LOT.ct}_{i,j,k})_{j,k}, \mathsf{ct}_i)$.

Hybrid_4 In this hybrid, we replace the $\mathsf{pMHE.PrivEval}$ with the following procedure, and keep all other parts the same as Hybrid_3^N.

$\mathsf{pMHE.PrivEval}(1^\lambda, \mathsf{sk}_i, i, C, (\mathsf{ct}_j)_{j\in[N]})$ Parse sk_i as $(\widehat{\mathsf{ct}_i}, C.\mathsf{params}, x_i; r_i)$.

Compute sk'_i from $\mathsf{pMHE}'.\mathsf{Enc}(1^\lambda, C.\mathsf{params}, x_i; r_i)$.

Sample random coins r'_i for $\mathsf{pMHE}'.\mathsf{PrivEval}$.

For each $j \in [N]$, let $\mathsf{ct}'_j \leftarrow \mathsf{RE.Recover}((\widehat{\mathsf{ct}_j})_{j\in[N]})$.

Let $p'_i = \mathsf{pMHE}'.\mathsf{PrivEval}(1^\lambda, \mathsf{sk}'_i, C, (\mathsf{ct}_j)_{j\in[N]}; r'_i)$.

Execute $\mathsf{lab}'_i \leftarrow \mathsf{GC.Sim}_1(1^\lambda, \mathsf{KG.in})$, and $\widetilde{\mathsf{KG}_i} \leftarrow \mathsf{GC.Sim}_2(1^\lambda, p'_i, \mathsf{lab}'_i)$.

Parse lab'_i as $(\mathsf{lab}'_{i,j,k})_{j\in[N],k\in[|\widehat{\mathsf{ct}_j}|]}$.

For each $j \in [N]$, parse ct_j as $\mathsf{ct}_j = (\mathsf{LOT.crs}_j, \mathsf{digest}_j)$.

For each $j \in [N], k \in [|\widehat{\mathsf{ct}_j}|]$, execute $\mathsf{LOT.ct}_{i,j,k} \leftarrow \mathsf{LOT.Enc}(\mathsf{LOT.crs}_j, \mathsf{digest}_j, k, \mathsf{lab}'_{i,j,k}, \mathsf{lab}'_{i,j,k})$.

Output $p_i = (\widetilde{\mathsf{KG}_i}, \widehat{\mathsf{ct}_i}, (\mathsf{LOT.ct}_{i,j,k})_{j,k}, \mathsf{ct}_i)$.

Ideal In this hybrid, we replace the $\mathsf{pMHE.Enc}$ and $\mathsf{pMHE.PrivEval}$ with the following simulators.

$\mathsf{pMHE.Sim}_1(1^\lambda, H, (x_i)_{i\in[N]\setminus H})$ Execute $(\mathsf{st}'_S, (\overline{\mathsf{ct}_i}')_{i\in[H]}, (r_i)_{i\in[N]\setminus H}) \leftarrow \mathsf{pMHE.Sim}_1(1^\lambda, H, (x_i)_{i\in[N]\setminus H})$.

For each $i \in [N]$, randomly sample $\widehat{r_i}$ for randomized encoding $\widehat{\mathsf{pMHE.Enc}}'_1$.

Randomly sample $\mathsf{LOT}.r_i$ for $\mathsf{LOT.Gen}$.

If $i \in H$, execute $\widehat{\mathsf{ct}_i} \leftarrow \mathsf{RE.Sim}(1^\lambda, \overline{\mathsf{ct}_i}')$.

Otherwise let $\widehat{\mathsf{ct}_i} = \widehat{\mathsf{pMHE}'.\mathsf{Enc}}_1(1^\lambda, C.\mathsf{params}, i, x_i, r_i; \widehat{r_i})$.

For each $i \in [N]$, execute $\mathsf{LOT.crs}_i \leftarrow \mathsf{LOT.Gen}(1^\lambda; \mathsf{LOT}.r_i)$.

Execute $\mathsf{digest}_i \leftarrow \mathsf{LOT.Hash}(\mathsf{LOT.crs}_i, \widehat{\mathsf{ct}_i})$.

Set $\overline{\mathsf{ct}_i} = (\mathsf{LOT.crs}_i, \mathsf{digest}_i)$, and $\mathsf{st}_S = (\mathsf{st}'_S, (\overline{\mathsf{ct}_i})_{i\in[N]}, (\widehat{\mathsf{ct}_i})_{i\in[N]})$.

Output $(\mathsf{st}_S, (\overline{\mathsf{ct}_i})_{i\in H}, (r_i, \widehat{r_i}, \mathsf{LOT}.r_i)_{i\in[N]\setminus H})$.

$\mathsf{pMHE.Sim}_2(\mathsf{st}_S, C, C(x_1, x_2, \ldots, x_n))$ Parse st_S as $(\mathsf{st}'_S, (\overline{\mathsf{ct}_i})_{i\in[N]}, (\widehat{\mathsf{ct}_i})_{i\in[N]})$.

Execute $(p_i)_{i\in H} \leftarrow \mathsf{pMHE}'.\mathsf{Sim}_2(\mathsf{sk}_S, C, C((x_i)_{i\in[N]}))$.

Execute $\mathsf{lab}'_i \leftarrow \mathsf{GC.Sim}_1(1^\lambda, \mathsf{KG.in})$, and $\widetilde{\mathsf{KG}_i} \leftarrow \mathsf{GC.Sim}_2(1^\lambda, p'_i, \mathsf{lab}'_i)$.

Parse lab'_i as $(\mathsf{lab}'_{i,j,k})_{j\in[N],k\in[|\widehat{\mathsf{ct}_j}|]}$.
For each $i \in [N]$, parse $\overline{\mathsf{ct}_i}$ as $\overline{\mathsf{ct}_i} = (\mathsf{LOT.crs}_i, \mathsf{digest}_i)$.
For each $j \in [N], k \in [|\widehat{\mathsf{ct}_j}|]$, let $b_{j,k} = \widehat{\mathsf{ct}_j}[k]$, and
execute $\mathsf{LOT.ct}_{i,j,k} \leftarrow \mathsf{LOT.Enc}(\mathsf{LOT.crs}_j, \mathsf{digest}_j, k, \mathsf{lab}'_{i,j,k}, \mathsf{lab}'_{i,j,k})$.
Set $p_i = (\widehat{\mathsf{KG}_i}, \widehat{\mathsf{ct}_i}, (\mathsf{LOT.ct}_{i,j,k})_{j,k}, \overline{\mathsf{ct}_i})$.
Output $(\mathsf{st}_S, p_i)_{i\in H}$.

Lemma 4.9. Hybrid_0 *and* Hybrid_1^0 *are identical. Moreover, for any* $i^* \in [N]$*, any PPT adversary* $\mathcal{A}$*, and distinguisher* $\mathcal{D}$*, there exists a negligible function* $\nu(\lambda)$ *such that* $|\Pr[\mathcal{D}(1^\lambda, \mathsf{Hybrid}_1^{i^*-1}) = 1] - \Pr[\mathcal{D}(1^\lambda, \mathsf{Hybrid}_1^{i^*}) = 1]| < \nu(\lambda)$.

Proof. If $i^* = 0$, then all $i \in [N]$ satisfy $i > i^*$. Hence, Hybrid_0 and Hybrid_1^0 are identical. The only difference between $\mathsf{Hybrid}_0^{i^*-1}$ and $\mathsf{Hybrid}_1^{i^*}$ is how $\widehat{\mathsf{ct}_{i^*}}$ is generated.

For any PPT adversary $\mathcal{A}$, distinguisher $\mathcal{D}$, any input $(x_i)_{i\in[N]}$, any random coins r_i, we construct a PPT adversary $\mathcal{D}'$ breaking the computational privacy of the randomized encoding $\widehat{\mathsf{pMHE.Enc}'_1}$.

Distinguisher $\mathcal{D}'(1^\lambda, \widehat{\mathsf{ct}})$ On input the security parameter λ and an encoding $\widehat{\mathsf{ct}}$, it simulates the environment for $\mathcal{A}$ by instantiating the function calls to $\mathsf{pMHE.Enc}$. Specifically,

$\mathsf{pMHE.Enc}(1^\lambda, C.\mathsf{params}, i, x_i)$ If $i < i^*$ and $i \in H$, then randomly sample random coins r_i,

and execute $\mathsf{ct}_i = \mathsf{pMHE}'.\mathsf{Enc}_1(1^\lambda, C.\mathsf{params}, i, x_i; r_i)$, and $\widehat{\mathsf{ct}_i} \leftarrow \mathsf{RE.Sim}(1^\lambda, \mathsf{ct}_i)$.

$\underline{\text{If } i = i^* \text{ and } i \in H, \text{ let } \widehat{\mathsf{ct}_{i^*}} = \widehat{\mathsf{ct}}.}$

If $i > i^*$ or $i \notin H$, randomly sample random coins $r_i, \widehat{r_i}$,

and execute $\widehat{\mathsf{ct}_i} = \widehat{\mathsf{pMHE.Enc}'_1}((1^\lambda, C.\mathsf{params}, i, x_i; r_i); \widehat{r_i})$.

Execute $\mathsf{LOT.crs}_i \leftarrow \mathsf{LOT.Gen}(1^\lambda)$, and let $\mathsf{digest}_i \leftarrow \mathsf{LOT.Hash}(\mathsf{LOT.crs}_i, \widehat{\mathsf{ct}_i})$.

Output $\mathsf{ct}_i = (\mathsf{LOT.crs}_i, \mathsf{digest}_i)$, and $\mathsf{sk}_i = (\widehat{\mathsf{ct}_i}, C.\mathsf{params}, x_i; r_i)$.

For each query made by $\mathcal{A}$, it executes $\mathsf{pMHE.PrivEval}$ in $\mathsf{Hybrid}_1^{i^*-1}$ and $\mathsf{Hybrid}_1^{i^*}$. Note that $\mathsf{pMHE.PrivEval}$ remains the same in both hybrids.

When $\widehat{\mathsf{ct}}$ is generated by $\widehat{\mathsf{pMHE.Enc}'_1}((1^\lambda, C.\mathsf{params}, i^*, x_i; r_i); \widehat{r_i})$ with random coins r_i and $\widehat{r_i}$, the distinguisher simulates the Hybrid^{i^*-1} for $\mathcal{A}$. Hence,

$$\Pr\left[\widehat{\mathsf{ct}} = \widehat{\mathsf{pMHE.Enc}'_1}((1^\lambda, C.\mathsf{params}, i^*, x_i; r_i); \widehat{r_i}) : \mathcal{D}'(1^\lambda, \widehat{\mathsf{ct}}) = 1\right] = \Pr[\mathcal{D}(1^\lambda, \mathsf{Hybrid}^{i^*-1}) = 1] \quad (3)$$

When $\widehat{\mathsf{ct}_{i^*}}$ is obtained by $\mathsf{RE.Sim}(1^\lambda, \mathsf{pMHE.Enc}'_1((1^\lambda, C.\mathsf{params}, i^*, x_i; r_i); \widehat{r_i}))$, the distinguisher simulates Hybrid^{i^*} for $\mathcal{A}$. Hence,

$$\Pr\left[\widehat{\mathsf{ct}} \leftarrow \mathsf{RE.Sim}(1^\lambda, \mathsf{pMHE.Enc}'_1((1^\lambda, C.\mathsf{params}, i^*, x_i; r_i); \widehat{r_i})) : \mathcal{D}'(1^\lambda, \widehat{\mathsf{ct}}) = 1\right] = \Pr[\mathcal{D}(1^\lambda, \mathsf{Hybrid}^{i^*}) = 1] \quad (4)$$

By the computational privacy of the randomized encoding, the difference on the left hand sides of the Equation (3) and (4) is bounded by a negligible function $\nu(\lambda)$. Hence, $|\Pr[\mathcal{D}(1^\lambda, \mathsf{Hybrid}^{i^*-1}) = 1] - \Pr[\mathcal{D}(1^\lambda, \mathsf{Hybrid}^{i^*}) = 1]| < \nu(\lambda)$. This finishes the proof. □

Recall that, for n totally ordered sets $S_1, S_2, \ldots, S_n$, and any tuple $(i_1^*, i_2^*, \ldots, i_n^*) \in S_1 \times S_2 \times \cdots \times S_n$, we use the notation $(i_1^*, i_2^*, \ldots, i_n^*)+1$ (resp. $(i_1^*, i_2^*, \ldots, i_n^*)-1$) to denote the lexicographical smallest (resp. biggest) element that is lexicographical bigger (resp. smaller) than $(i_1^*, i_2^*, \ldots, i_n^*)$.

Lemma 4.10. Hybrid_1^N *and* $\mathsf{Hybrid}_2^{(1,1,1)-1}$ *are identical. Moreover, for any* $(i^*, j^*, k^*) \in [N] \times [N] \times [|\widehat{\mathsf{ct}_{j^*}}|]$*, any PPT adversary* $\mathcal{A}$*, and any PPT distinguisher* $\mathcal{D}$*, there exists a negligible function* $\nu(\lambda)$ *such that* $|\Pr[\mathcal{D}(1^\lambda, \mathsf{Hybrid}_2^{(i^*,j^*,k^*)-1}) = 1] - \Pr[\mathcal{D}(1^\lambda, \mathsf{Hybrid}_2^{(i^*,j^*,k^*)}) = 1]| < \nu(\lambda)$.

Proof. When $(i^*, j^*, k^*) = (1,1,1) - 1$, the condition $(i,j,k) \leq (i^*, j^*, k^*)$ never holds. Hence, the hybrid $\mathsf{Hybrid}_2^{(1,1,1)-1}$ is identical to Hybrid_1^N. Now for any PPT adversary $\mathcal{A}$, any PPT distinguisher $\mathcal{D}$, we construct an adversary $\mathcal{D}'$ breaking the semi-honest sender privacy of laconic OT for each fixed $(\widehat{\mathsf{ct}_j})_{j \in [N]}$.

In the indistinguishability game of sender privacy of laconic OT, let the binary string $D = \widehat{\mathsf{ct}_{j^*}}$, and the index be k^*, and randomly sample m_0, m_1. We define the following functions.

$\mathsf{pMHE.Enc}_{[\mathsf{crs},\mathsf{digest}]}(1^\lambda, C.\mathsf{params}, i, x_i)$ If $i \neq j^*$, execute $\mathsf{LOT.crs}_i \leftarrow \mathsf{LOT.Gen}(1^\lambda)$,

and let $\mathsf{digest}_i \leftarrow \mathsf{LOT.Hash}(\mathsf{LOT.crs}_i, \widehat{\mathsf{ct}_i})$.

Otherwise, let $\mathsf{LOT.crs}_i = \mathsf{crs}$, and $\mathsf{digest}_i = \mathsf{digest}$.

Output $\mathsf{ct}_i = (\mathsf{LOT.crs}_i, \mathsf{digest}_i)$, and $\mathsf{sk}_i = (\widehat{\mathsf{ct}_i}, C.\mathsf{params}, x_i; r_i)$.

$\mathsf{pMHE.PrivEval}_{[\mathsf{ct}]}(1^\lambda, \mathsf{sk}_i, i, C, (\mathsf{ct}_j)_{j \in [N]})$ Parse sk_i as $(\widehat{\mathsf{ct}_i}, C.\mathsf{params}, x_i; r_i)$.

Compute sk'_i from $\mathsf{pMHE}'.\mathsf{Enc}(1^\lambda, C.\mathsf{params}, x_i; r_i)$.

Sample random coins r'_i for $\mathsf{pMHE}'.\mathsf{PrivEval}$.

Randomly sample labels lab_i, parse lab_i as $\{\mathsf{lab}_{i,j,k,b}\}_{i \in [N], j \in [N], k \in [|\hat{\mathsf{ct}_j}|], b \in \{0,1\}}$.

$\underline{\text{For each } b \in \{0,1\}, \text{ replace } \mathsf{lab}_{i^*,j^*,k^*,b} \text{ with } m_b.}$

Execute $\widetilde{\mathsf{KG}_i} \leftarrow \mathsf{GC.Garble}(1^\lambda, \mathsf{KG}, \mathsf{lab}_i)$.

For each $j \in [N]$, parse ct_j as $\mathsf{ct}_j = (\mathsf{LOT.crs}_j, \mathsf{digest}_j)$.

For each $j \in [N], k \in [|\widehat{\mathsf{ct}_j}|]$, if $(i,j,k) < (i^*, j^*, k^*)$, execute

$\mathsf{LOT.ct}_{i,j,k} \leftarrow \mathsf{LOT.Enc}(\mathsf{LOT.crs}_j, \mathsf{digest}_j, \mathsf{lab}_{i,j,k,b_{i,j,k}}, \mathsf{lab}_{i,j,k,b_{i,j,k}})$, where $b_{i,j,k} = \widehat{\mathsf{ct}_j}[k]$.

$\underline{\text{If } (i,j,k) = (i^*, j^*, k^*), \text{ let } \mathsf{LOT.ct}_{i,j,k} = \mathsf{ct}.}$

If $(i,j,k) > (i^*, j^*, k^*)$, execute $\mathsf{LOT.ct}_{i,j,k} \leftarrow \mathsf{LOT.Enc}(\mathsf{LOT.crs}_j, \mathsf{digest}_j, k, \mathsf{lab}_{i,j,k,0}, \mathsf{lab}_{i,j,k,1})$.

Output $p_i = (\widetilde{\mathsf{KG}}_i, \widehat{\mathsf{ct}_i}, (\mathsf{LOT.ct}_{i,j,k})_{j,k}, \mathsf{ct}_i)$.

The distinguisher $\mathcal{D}'$ is constructed as follows. We generate $\widehat{\mathsf{ct}_i}$ for each $i \in [N]$ in the same manner as in $\mathsf{Hybrid}_2^{(i^*,j^*,k^*)-1}$ and $\mathsf{Hybrid}_2^{(i^*,j^*,k^*)}$.

Distinguisher $\mathcal{D}'(1^\lambda, \mathsf{crs}, \mathsf{digest}, \mathsf{ct})$ For each $i \in [N]$, sample the random coins r_i for $\mathsf{pMHE.Enc}$.

Let $(\mathsf{ct}_i, \mathsf{sk}_i) = \mathsf{pMHE.Enc}_{[\mathsf{crs},\mathsf{digest}]}(1^\lambda, C.\mathsf{params}, i, x_i; r_i)$.

Execute $\mathcal{A}^{\mathcal{O}_\mathcal{A}(1^\lambda,\cdot)}(1^\lambda, (\mathsf{ct}_i)_{i \in [N]}, (x_i, r_i, \widehat{r_i}, \mathsf{LOT.crs}_i{}^{11})_{i \notin H})$, and let $b \leftarrow \mathcal{D}(1^\lambda, \mathsf{View}_\mathcal{A})$.

Output b.

Oracle $\mathcal{O}_\mathcal{A}(1^\lambda, C)$ For each $i \in H$, execute $p_i \leftarrow \mathsf{pMHE.PrivEval}_{[\mathsf{ct}]}(1^\lambda, \mathsf{sk}_i, i, C, (\mathsf{ct}_j)_{j \in [N]})$

Output $(p_i)_{i \in H}$.

[11] Note that the crs of laconic OT is an uniform random string. Hence, the random coin is the crs itself.

When $\mathsf{ct} \leftarrow \mathsf{LOT.Enc}(\mathsf{crs}, \mathsf{digest}, k^*, m_0, m_1)$, the distinguisher simulates the environment of $\mathsf{Hybrid}_2^{(i^*,j^*,k^*)-1}$ for $\mathcal{A}$ and $\mathcal{D}$. Hence,

$$\Pr\left[\mathsf{ct} \leftarrow \mathsf{LOT.Enc}(\mathsf{crs}, \mathsf{digest}, k^*, m_0, m_1) : \mathcal{D}(1^\lambda, \mathsf{crs}, \mathsf{digest}, \mathsf{ct}) = 1\right] = \Pr[\mathcal{D}(1^\lambda, \mathsf{Hybrid}_2^{(i^*,j^*,k^*)-1}) = 1] \tag{5}$$

When $\mathsf{ct} \leftarrow \mathsf{LOT.Enc}(\mathsf{crs}, \mathsf{digest}, k^*, m_{\widehat{\mathsf{ct}_{j^*}}[k^*]}, m_{\widehat{\mathsf{ct}_{j^*}}[k^*]})$, the distinguisher simulates the environment of $\mathsf{Hybrid}_2^{(i^*,j^*,k^*)}$ for $\mathcal{A}$ and $\mathcal{D}$. Hence,

$$\Pr\left[b = m_{\widehat{\mathsf{ct}_{j^*}}[k^*]}, \mathsf{ct} \leftarrow \mathsf{LOT.Enc}(\mathsf{crs}, \mathsf{digest}, k^*, m_b, m_b) : \mathcal{D}(1^\lambda, \mathsf{crs}, \mathsf{digest}, \mathsf{ct}) = 1\right] = \tag{6}$$

$$\Pr\left[\mathcal{D}(1^\lambda, \mathsf{Hybrid}_2^{(i^*,j^*,k^*)}) = 1\right] \tag{7}$$

By the semi-honest sender privacy of the laconic OT, the difference between left hand sides of the Equations (5) and (6) is bounded by a negligible function. Hence, there exits a negligible function $\nu(\lambda)$ such that $|\Pr[\mathcal{D}(1^\lambda, \mathsf{Hybrid}_2^{(i^*,j^*,k^*)-1}) = 1] - \Pr[\mathcal{D}(1^\lambda, \mathsf{Hybrid}_2^{(i^*,j^*,k^*)}) = 1]| < \nu(\lambda)$. $\square$

Lemma 4.11. $\mathsf{Hybrid}_2^{N,N,|\widehat{\mathsf{ct}_N}|}$ *is identical to* Hybrid_3^0. *Moreover, for any* $i^* \in [N]$*, any PPT adversary* $\mathcal{A}$*, and any PPT distinguisher* $\mathcal{D}$*, there exits a negligible function* $\nu(\lambda)$ *such that* $|\Pr[\mathcal{D}(1^\lambda, \mathsf{Hybrid}_3^{i^*-1}) = 1] - \Pr[\mathcal{D}(1^\lambda, \mathsf{Hybrid}_3^{i^*}) = 1]| < \nu(\lambda)$.

Proof. For any PPT adversary $\mathcal{A}$, and PPT distinguisher $\mathcal{D}$, we build the following adversary $\mathcal{A}' = (\mathcal{A}'_1, \mathcal{A}'_2)$ trying to break the garbling scheme.

$\mathcal{A}'(1^\lambda, \mathsf{lab}')$ For each $i \in [N]$, sample the random coins r_i for $\mathsf{pMHE.Enc}$.

Let $(\mathsf{ct}_i, \mathsf{sk}_i) = \mathsf{pMHE.Enc}(1^\lambda, C.\mathsf{params}, i, x_i; r_i)$.

Execute $\mathcal{A}^{\mathcal{O}_\mathcal{A}(1^\lambda,\cdot)}(1^\lambda, (\mathsf{ct}_i)_{i\in[N]}, (x_i, r_i)_{i\notin H})$, and $b \leftarrow \mathcal{D}(1^\lambda, \mathsf{View}_\mathcal{A})$.

Output b.

Oracle $\mathcal{O}_\mathcal{A}(1^\lambda, C)$ For each $i \in H$, execute $p_i \leftarrow \mathsf{pMHE.PrivEval}(1^\lambda, \mathsf{sk}_i, i, C, (\mathsf{ct}_j)_{j\in[N]})$.

Output $(p_i)_{i\in H}$

$\mathsf{pMHE.PrivEval}(1^\lambda, \mathsf{sk}_i, i, C, (\mathsf{ct}_j)_{j\in[N]})$ Parse sk_i as $(\widehat{\mathsf{ct}_i}, C.\mathsf{params}, x_i; r_i)$.

Compute sk'_i from $\mathsf{pMHE}'.\mathsf{Enc}(1^\lambda, C.\mathsf{params}, x_i; r_i)$.

Sample random coins r'_i for $\mathsf{pMHE}'.\mathsf{PrivEval}$.

If $i > i^*$, randomly sample labels lab_i, execute $\widetilde{\mathsf{KG}_i} \leftarrow \mathsf{GC.Garble}(1^\lambda, \mathsf{KG}, \mathsf{lab}_i)$.

Parse lab_i as $(\mathsf{lab}_{i,j,k,b})_{j\in[N],k\in[|\widehat{\mathsf{ct}_j}|],b\in\{0,1\}}$. Let $\mathsf{lab}'_{i,j,k} = \mathsf{lab}_{i,j,k,\widehat{\mathsf{ct}_j}[k]}$.

<u>If $i = i^*$, the adversary $\mathcal{A}'$ query the challenger with the circuit KG, and obtains $\widetilde{\mathsf{KG}}$.</u>

Denote $\widetilde{\mathsf{KG}_i} = \widetilde{\mathsf{KG}}$, and $\mathsf{lab}'_i = \mathsf{lab}'$. Parse lab'_i as $(\mathsf{lab}'_{i,j,k})_{j\in[N],k\in[|\widehat{\mathsf{ct}_j}|]}$.

If $i < i^*$, execute $\mathsf{lab}'_i \leftarrow \mathsf{GC.Sim}_1(1^\lambda, \mathsf{KG.in})$,

$\widetilde{\mathsf{KG}_i} \leftarrow \mathsf{GC.Sim}_2(1^\lambda, \mathsf{KG}((\mathsf{ct}_j)_{j\in[N]}), \mathsf{lab}'_i)$.

For each $j \in [N]$, parse ct_j as $\mathsf{ct}_j = (\mathsf{LOT.crs}_j, \mathsf{digest}_j)$.

For each $j \in [N], k \in [|\widehat{\mathsf{ct}_j}|]$, let $b_{j,k} = \widehat{\mathsf{ct}_j}[k]$, and

execute $\mathsf{LOT.ct}_{i,j,k} \leftarrow \mathsf{LOT.Enc}(\mathsf{LOT.crs}_j, \mathsf{digest}_j, \mathsf{lab}'_{i,j,k}, \mathsf{lab}'_{i,j,k})$.

Output $p_i = (\widetilde{\mathsf{KG}_i}, \widehat{\mathsf{ct}_i}, (\mathsf{LOT.ct}_{i,j,k})_{j,k}, \mathsf{ct}_i)$.

When $(\mathsf{lab}', \widetilde{\mathsf{KG}})$ is obtained by real execution, then the adversary $\mathcal{A}'$ simulates the environments of $\mathsf{Hybrid}_3^{i^*-1}$ for $\mathcal{A}$. Hence, $\Pr[\mathsf{Real}^{\mathcal{A}'} = 1] = \Pr[\mathcal{D}(1^\lambda, \mathsf{Hybrid}_3^{i^*-1}) = 1]$.

When $(\mathsf{lab}', \widetilde{\mathsf{KG}})$ is obtained from ideal execution, then the adversary $\mathcal{A}'$ simulates the enviroments of $\mathsf{Hybrid}_3^{i^*}$ for $\mathcal{A}$. Hence, $\Pr[\mathsf{Ideal}^{\mathcal{A}'} = 1] = \Pr[\mathcal{D}(1^\lambda, \mathsf{Hybrid}_3^{i^*}) = 1]$.

By the selective-security of the garbling scheme, there exists a negligible function that bound the difference of the left hand side.

Hence, there exits a negligible function $\nu(\lambda)$ such that $|\Pr[\mathcal{D}(1^\lambda, \mathsf{Hybrid}_3^{i^*-1}) = 1] - \Pr[\mathcal{D}(1^\lambda, \mathsf{Hybrid}_3^{i^*}) = 1]| < \nu(\lambda)$. □

Lemma 4.12. Hybrid_3^N *is identical to* Hybrid_4. *Moreover, for any PPT adversary* $\mathcal{A}$ *and distinguisher* $\mathcal{D}$, *there exits a negligible function* $\nu(\lambda)$ *such that* $|\Pr[\mathcal{D}(1^\lambda, \mathsf{Hybrid}_4^{\mathcal{A}}) = 1] - \Pr[\mathcal{D}(1^\lambda, \mathsf{Ideal}^{\mathcal{A}}) = 1]| < \nu(\lambda)$.

Proof. For any PPT adversary $\mathcal{A}$, we build the following adversary $\mathcal{A}'$ trying to break the scheme pMHE'.

Adversary $\mathcal{A}'^{\mathcal{O}_{\mathcal{A}'}}(1^\lambda, (\mathsf{ct}'_i)_{i\in[N]}, (x_i, r'_i)_{i\notin H})$ For $i \notin H$, randomly sample $k_i \leftarrow \{0,1\}^{\mathsf{PRG.in}}$,

Sample random coins $r'_i \leftarrow \{0,1\}^*$ for $\mathsf{pMHE.Enc}$, and $\widehat{r_i} \leftarrow \{0,1\}^*$ for randomized encoding, and $\mathsf{LOT}.r_i$ for $\mathsf{LOT.Gen}$.

For each $i \notin H$, execute $\widehat{\mathsf{ct}_i} = \widehat{\mathsf{pMHE}'.\mathsf{Enc}}_1((1^\lambda, C.\mathsf{params}, i, x_i; r'_i); \widehat{r_i})$.

For each $i \in H$, execute $\widehat{\mathsf{ct}_i} \leftarrow \mathsf{RE.Sim}(1^\lambda, \mathsf{ct}'_i)$.

For each $i \in [N]$, execute $\mathsf{LOT.crs}_i = \mathsf{LOT.Gen}(1^\lambda; \mathsf{LOT}.r_i)$, and let $\mathsf{digest}_i \leftarrow \mathsf{LOT.Hash}(\mathsf{LOT.crs}_i, \widehat{\mathsf{ct}_i})$.

Let $\mathsf{ct}_i = (\mathsf{LOT.crs}_i, \mathsf{digest}_i)$.

Execute $\mathcal{A}^{\mathcal{O}_{\mathcal{A}}}(1^\lambda, (\mathsf{ct}_i)_{i\in[N]}, (x_i, k_i, \widehat{r_i}, \mathsf{LOT}.r_i)_{i\notin H})$, and $b \leftarrow \mathcal{D}(1^\lambda, \mathsf{View}_{\mathcal{A}})$.

Output b.

Oracle $\mathcal{O}_{\mathcal{A}}(1^\lambda, C)$ The adversary $\mathcal{A}'$ queries the oracle $\mathcal{O}_{\mathcal{A}'}$ with circuit C, and obtains $(p'_i)_{i\in H}$.

Then for each $i \in H$, execute $\mathsf{lab}'_i \leftarrow \mathsf{GC.Sim}_1(1^\lambda, \mathsf{KG.in})$, and $\widetilde{\mathsf{KG}}_i \leftarrow \mathsf{GC.Sim}_2(1^\lambda, p'_i, \mathsf{lab}'_i)$.

Parse lab'_i as $(\mathsf{lab}'_{i,j,k})_{j\in[N], k\in[|\widehat{\mathsf{ct}_j}|]}$.

For each $i \in [N]$, parse $\overline{\mathsf{ct}_i}$ as $\overline{\mathsf{ct}_i} = (\mathsf{LOT.crs}_i, \mathsf{digest}_i)$.

For each $j \in [N], k \in [|\widehat{\mathsf{ct}_j}|]$, let $b_{j,k} = \widehat{\mathsf{ct}_j}[k]$, and

execute $\mathsf{LOT.ct}_{i,j,k} \leftarrow \mathsf{LOT.Enc}(\mathsf{LOT.crs}_j, \mathsf{digest}_j, \mathsf{lab}'_{i,j,k}, \mathsf{lab}'_{i,j,k})$.

Set $p_i = (\widetilde{\mathsf{KG}}_i, \widehat{\mathsf{ct}_i}, (\mathsf{LOT.ct}_{i,j,k})_{j,k}, \overline{\mathsf{ct}_i})$.

Output $(p_i)_{i\in H}$.

When $\mathcal{A}'^{\mathcal{O}_{\mathcal{A}'}}$ is interacting with Real, it simulates the environment of Hybrid_4 for $\mathcal{A}$. Hence, we have $\Pr[\mathsf{Real}^{\mathcal{A}'} = 1] = \Pr[\mathcal{D}(1^\lambda, \mathsf{Hybrid}_4^{\mathcal{A}}) = 1]$.

When $\mathcal{A}'^{\mathcal{O}_{\mathcal{A}'}}$ is interacting with Ideal, it simulates the environment of Ideal for $\mathcal{A}$. Hence, we have $\Pr[\mathsf{Ideal}^{\mathcal{A}'} = 1] = \Pr[\mathcal{D}(1^\lambda, \mathsf{Ideal}^{\mathcal{A}}) = 1]$.

By the one-time simulation security of pMHE', there exits a negligible function $\nu(\lambda)$ such that $|\Pr[\mathsf{Real}^{\mathcal{A}'} = 1] - \Pr[\mathsf{Ideal}^{\mathcal{A}'} = 1]| < \nu(\lambda)$. Hence, $|\Pr[\mathcal{D}(1^\lambda, \mathsf{Hybrid}_4^{\mathcal{A}}) = 1] - \Pr[\mathcal{D}(1^\lambda, \mathsf{Ideal}^{\mathcal{A}}) = 1]| < \nu(\lambda)$. □

Combining Lemma 4.9, 4.10, 4.11, and 4.12, we prove that $(\mathsf{pMHE.Sim}_1, \mathsf{pMHE.Sim}_2)$ is a simulator for pMHE. Hence, pMHE is (one-time) simulation secure. □

Lemma 4.13 (Strong Ciphertext Succinctness)**.** *If the underlying scheme* pMHE' *is weak ciphertext succinct, then the construction* pMHE *is strong ciphertext succinct.*

Proof. We prove that the properties of strong ciphertext succinctness are satisfied.

- The weak ciphertext succinctness of pMHE follows from the weak succinctness of pMHE', the efficiency of the randomized encoding, and the efficiency of LOT.

- The depth of the circuit $\mathsf{pMHE.Enc}$ is the depth of the randomized encoding $\widehat{\mathsf{pMHE}'.\mathsf{Enc}_1}$ adding the depth of $\mathsf{LOT.Hash}(\cdot,\cdot)$. The depth of $\widehat{\mathsf{pMHE}'.\mathsf{Enc}_1}$ is $\mathsf{poly}(\lambda, \log N, \log C.\mathsf{in}, \log C.\mathsf{out}, C.\mathsf{depth})$. The depth of $\mathsf{LOT.Hash}(\cdot,\cdot)$ is $\mathsf{poly}(\lambda, \log|\widehat{\mathsf{ct}_i}|) = \mathsf{poly}(\lambda, \log \text{Running time of } \mathsf{pMHE}'.\mathsf{Enc}_1)$ $= \mathsf{poly}(\lambda, \log N, \log C.\mathsf{in}, \log C.\mathsf{out}, \log C.\mathsf{depth})$. Hence, the depth of $\mathsf{pMHE.Enc}$ is $\mathsf{poly}(\lambda, \log N,$ $\log C.\mathsf{in}, \log C.\mathsf{out}, \log C.\mathsf{depth})$.

- The output length of $\mathsf{pMHE.Enc}$ is $|\mathsf{ct}_i| = |\mathsf{LOT.crs}_i| + |\mathsf{digest}_i| = \mathsf{poly}(\lambda)$.

□

5 Reusable pMHE from One-Time pMHE

In this section, we show how to bootstrap from a one-time pMHE with strong ciphertext succinctness property into a (possibly non-succinct) reusable pMHE scheme.

Lemma 5.1 (Bootstrap from One-Time Strong Ciphertext Succinctness Scheme to Reusable Scheme)**.** *Let* $\mathsf{pMHE}' = (\mathsf{pMHE}'.\mathsf{Enc}, \mathsf{pMHE}'.\mathsf{PrivEval}, \mathsf{pMHE}'.\mathsf{FinDec})$ *be a one-time strong ciphertext succinct pMHE scheme, and* $\mathsf{PRG} : \{0,1\}^{\mathsf{PRG.in}} \to \{0,1\}^{\mathsf{PRG.out}}$ *be a PRG. We can build a reusable strong ciphertext succinct pMHE scheme* $\mathsf{pMHE} = (\mathsf{pMHE.Enc}, \mathsf{pMHE.PrivEval}, \mathsf{pMHE.FinDec})$.

Construction.

$\mathsf{pMHE.Enc}(1^\lambda, C.\mathsf{params}, i, x_i)$ Randomly sample $k_i \leftarrow \{0,1\}^{\mathsf{PRG.in}}$, and random coins r_i.

Execute $(\mathsf{ct}'_i, \mathsf{sk}'_i) \leftarrow \mathsf{pMHE}'.\mathsf{Enc}(1^\lambda, \mathsf{NewEnc}^1.\mathsf{params}, i, (x_i, k_i))$.

We will specify the circuit NewEnc later. Recall that, for any circuit C, we denote $C.\mathsf{params}$ to be the tuple $(C.\mathsf{in}, C.\mathsf{out}, C.\mathsf{depth})$.

Set $\mathsf{ct}_i = \mathsf{ct}'_i$ and $\mathsf{sk}_i = (\mathsf{sk}'_i, (k_i, r_i))$. Output $(\mathsf{ct}_i, \mathsf{sk}_i)$.

$\mathsf{pMHE.PrivEval}(\mathsf{sk}_i, C, i, (\mathsf{ct}_j)_{j\in[N]})$ Parse sk_i as $(\mathsf{sk}'_i, (k_i, r_i))$.

Let id be the binary representation of the circuit C. Denote $|\mathsf{id}|$ as n.

For $t \in [n]$, let NewEnc^t be the following recursively defined circuits.

$\mathsf{NewEnc}^{n+1}((x_j, k_j)_{j\in[N]})$ Execute $y = C((x_j)_{j\in[N]})$.

Output y.

$\mathsf{NewEnc}^t((x_j, k_j)_{j\in[N]})$ For any $j \in [N]$, parse $\mathsf{PRG}(k_j)$ as $(\mathsf{lab}^{j,t,b}, k^t_{j,b}, r^t_{j,1,b}, r^t_{j,2,b}, r^t_{j,3,b})_{b\in\{0,1\}}$.

For any $j \in [N], b \in \{0,1\}$, execute
$(\mathsf{ct}_{j,b}, \mathsf{sk}_{j,b}) = \mathsf{pMHE}'.\mathsf{Enc}(1^\lambda, \mathsf{NewEnc}^{t+1}.\mathsf{params}, j, (x_j, k^t_{j,b}); r^t_{j,1,b})$.

For any $b \in \{0,1\}$, Let $\mathsf{ct}_b = (\mathsf{ct}_{j,b})_{j\in[N]}$.

Output $(\mathsf{lab}^{i,t,0}_{\mathsf{ct}_0}, \mathsf{lab}^{i,t,1}_{\mathsf{ct}_1})_{i\in[N]}$.

For $t \in [n]$, Boot^t is defined as follows.

$\mathsf{Boot}^t_{[\mathsf{sk}_i^t; r_i^t]}(\mathsf{ct}^t)$ Execute $p_i^t = \mathsf{pMHE}'.\mathsf{PrivEval}(\mathsf{sk}_i^t, \mathsf{NewEnc}^{t+1}, \mathsf{ct}^t; r_i^t)$.
Output p_i^t.

Execute $p_i^0 = \mathsf{pMHE}'.\mathsf{PrivEval}(\mathsf{sk}'_i, \mathsf{NewEnc}^1, (\mathsf{ct}_j)_{j\in[N]}; r_i)$.

Let $k_i^0 = k_i$.

For each $t = 1, 2, \ldots, n$,

Let $b = \mathsf{id}[t]$. Parse $\mathsf{PRG}(k_i^{t-1})$ as $(\mathsf{lab}^{i,t,b'}, k^t_{i,b'}, r^t_{i,1,b'}, r^t_{i,2,b'}, r^t_{i,3,b'})_{b'\in\{0,1\}}$

Compute sk_i^t from $\mathsf{pMHE}'.\mathsf{Enc}(1^\lambda, \mathsf{NewEnc}^t.\mathsf{params}, i, (x_i, k^t_{i,b}); r^t_{i,1,b})$.

Execute $\widetilde{\mathsf{Boot}_i^t} \leftarrow \mathsf{GC}.\mathsf{Garble}(1^\lambda, \mathsf{Boot}^t_{[\mathsf{sk}_i^t; r^t_{i,2,b}]}, \mathsf{lab}^{i,t,b}; r^t_{i,3,b})$.

Set $p_i = (p_i^0, (\widetilde{\mathsf{Boot}_i^t})_{t\in[n]}, \mathsf{ct}_i)$. Output p_i.

$\mathsf{pMHE}.\mathsf{FinDec}(C, (p_i)_{i\in[N]})$ Let id be the binary representation of C. Parse p_i as $(p_i^0, (\widetilde{\mathsf{Boot}_i^t})_{t\in[n]}, \mathsf{ct}_i)$.

For each $t = 1, 2, \ldots, n$,

Let $b = \mathsf{id}[t]$.

Execute $(\mathsf{lab}'^{i,t,0}, \mathsf{lab}'^{i,t,1})_{i\in[N]} \leftarrow \mathsf{pMHE}'.\mathsf{FinDec}(\mathsf{NewEnc}^t, (p_i^{t-1})_{i\in[N]})$.

For each $i \in [N]$, execute $p_i^t \leftarrow \mathsf{GC}.\mathsf{Eval}(1^\lambda, \widetilde{\mathsf{Boot}_i^t}, \mathsf{lab}'^{i,t,b})$.

Execute $y \leftarrow \mathsf{pMHE}'.\mathsf{FinDec}(\mathsf{NewEnc}^{n+1}, (p_i^n)_{i\in[N]})$.

Output y.

Lemma 5.2 (Correctness). *The construction of* pMHE *is correct.*

Proof. For any input $(x_i)_{i\in[N]}$, any circuit C, and any $i \in [N]$, let $(\mathsf{ct}_i, \mathsf{sk}_i) \leftarrow \mathsf{pMHE}.\mathsf{Enc}(1^\lambda, C.\mathsf{params}, i, x_i)$. Let $p_i = (p_i^0, (\widetilde{\mathsf{Boot}_i^t})_{t\in[n]}, \mathsf{ct}_i) \leftarrow \mathsf{pMHE}.\mathsf{PrivEval}(\mathsf{sk}_i, C, i, (\mathsf{ct}_j)_{j\in[N]})$.

Now we consider each step in $\mathsf{pMHE}.\mathsf{FinDec}(C, (p_i)_{i\in[N]})$. For each $t = 1, 2, \ldots, n$, we prove by induction the following claim.

Claim 5.3. *For any $t \in [n]$, we have*

- $(\mathsf{lab}'^{i,t,0}, \mathsf{lab}'^{i,t,1})_{i\in[N]} = (\mathsf{lab}^{i,t,0}_{\mathsf{ct}_0}, \mathsf{lab}^{i,t,1}_{\mathsf{ct}_1})_{i\in[N]}$, *where* $(\mathsf{lab}^{i,t,b})_{b\in\{0,1\}}, (\mathsf{ct}_b)_{b\in\{0,1\}}$ *are obtained by executing* $(\mathsf{lab}^{j,t,b}, k^t_{j,b}, r^t_{j,1,b}, r^t_{j,2,b}, r^t_{j,3,b})_{b\in\{0,1\}} = \mathsf{PRG}(k_j^{t-1})$ *for each* $j \in [N]$, *and for each* $b \in \{0,1\}$, *let* $(\mathsf{ct}_{j,b}, \mathsf{sk}_{j,b}) = \mathsf{pMHE}'.\mathsf{Enc}(1^\lambda, \mathsf{NewEnc}^{t+1}.\mathsf{params}, j, (x_j, k^t_{j,b}); r^t_{j,1,b})$, *and* $\mathsf{ct}_b = (\mathsf{ct}_{j,b})_{j\in[N]}$.
- *For any* $j \in [N]$, $k_j^0 = k_j$. $k_j^{t+1} = k^t_{j,\mathsf{id}[t]}$.
- *For any* $j \in [N]$, $p_i^t = \mathsf{pMHE}'.\mathsf{PrivEval}(\mathsf{sk}_i^t, \mathsf{NewEnc}^{t+1}, \mathsf{ct}_{\mathsf{id}[t]}; r^t_{i,2,\mathsf{id}[t]})$.

We prove the claim by induction on t. For the base case, we will show that the claim holds for $t = 1$. By the correctness of pMHE', we have that $(\mathsf{lab}'^{i,1,0}, \mathsf{lab}'^{i,1,1}) = (\mathsf{lab}^{i,1,0}_{\mathsf{ct}_0}, \mathsf{lab}^{i,1,1}_{\mathsf{ct}_1})$. For each $b \in \{0,1\}$, $\mathsf{ct}_b = (\mathsf{ct}_{j,b})_{j\in[N]}$, and $(\mathsf{ct}_{j,b}, \mathsf{sk}_{j,b}) = \mathsf{pMHE}'.\mathsf{Enc}(1^\lambda, \mathsf{NewEnc}^2.\mathsf{params}, j, (x_j, k^1_{j,b}); r^1_{j,1,b})$, where $(\mathsf{lab}^{j,t,b}, k^1_{j,b}, r^1_{j,1,b}, r^1_{j,2,b}, r^1_{j,3,b})_{b\in\{0,1\}} = \mathsf{PRG}(k_j)$. From the correctness of the garbling scheme, we have $p_i^1 = \mathsf{Boot}^1_{[\mathsf{sk}_i^1; r^1_{i,2,\mathsf{id}[1]}]}(\mathsf{ct}_{\mathsf{id}[1]}) = \mathsf{pMHE}'.\mathsf{PrivEval}(\mathsf{sk}_i, \mathsf{NewEnc}^2, \mathsf{ct}_{\mathsf{id}[1]}; r^1_{i,2,\mathsf{id}[1]})$.

Now we assume the claim holds for $t = t^* - 1$, and we now prove for the case of $t = t^*$. We have $p_i^{t^*-1} = \mathsf{pMHE'.PrivEval}(\mathsf{sk}_i^{t^*-1}, \mathsf{NewEnc}^{t^*}, \mathsf{ct}_{\mathsf{id}[t^*-1]}; r^1_{i,2,\mathsf{id}[t^*-1]})$. From the correctness of $\mathsf{pMHE'}$, we have $(\mathsf{lab}'^{i,t,0}, \mathsf{lab}'^{i,t,1})_{i\in[N]} = (\mathsf{lab}^{i,t,0}_{\mathsf{ct}_0}, \mathsf{lab}^{i,t,1}_{\mathsf{ct}_1})_{i\in[N]}$, where $\mathsf{ct}_b = (\mathsf{ct}_{j,b})_{j\in[N]}$, and $(\mathsf{ct}_{j,b}, \mathsf{sk}_{j,b}) = \mathsf{pMHE'.Enc}(1^\lambda, \mathsf{NewEnc}^{t+1}.\mathsf{params}, j, (x_j, k^t_{j,b}); r^t_{j,1,b})$, for all $j \in [N], b \in \{0,1\}$. We then finish proving the claim by the correctness of the garbling scheme.

Thus, the claim holds for any $t^* \in [n]$. Hence, $p_i^n = \mathsf{pMHE'.PrivEval}(\mathsf{sk}_i^n, \mathsf{NewEnc}^{n+1}, \mathsf{ct}_{\mathsf{id}[n]}; r^n_{i,2,\mathsf{id}[n]})$, and $\mathsf{ct}_{\mathsf{id}[n]}$ is obtained from $\mathsf{pMHE'.Enc}(1^\lambda, \mathsf{NewEnc}^{n+1}.\mathsf{params}, j, (x_j, k^n_{j,\mathsf{id}[n]})_{j\in[N]})$. From the correctness of $\mathsf{pMHE'}$, we have $y = \mathsf{NewEnc}^{n+1}((x_j, k^n_{j,\mathsf{id}[n]})_{j\in[N]}) = C((x_i)_{i\in[N]})$. □

Lemma 5.4 (Reusable Simulation Security)**.** *The construction of* pMHE *is reusable simulation secure.*

For any input $(x_i)_{i\in[N]}$, any set of honest parties $H \subseteq [N]$, and any PPT adversary $\mathcal{A}$ that queries the oracle $\mathcal{O}$ with at most $Q = Q(\lambda)$ times, we build the following hybrids.

Hybrid_0 This hybrid is identical to the real execution $\mathsf{Real}^{\mathcal{A}}(1^\lambda, (x_i)_{i\in[N]})$.

Hybrid_1 From Hybrid_0 to Hybrid_1, we replace $(\mathsf{ct}'_i)_{i\in[N]}$ and $(p_i^0)_{i\in H}$ with the simulating messages $(\overline{\mathsf{ct}_i}')_{i\in[N]}$ and $(\overline{p_i^0})_{i\in H}$ generated by the simulators of $\mathsf{pMHE'}$.

For each $i \in [N]$, randomly sample $k_i \leftarrow \{0,1\}^{\mathsf{PRG.in}}$.

For each $i \in [N]$, if $i \notin H$, sample random coins r_i, otherwise, let $r_i = \perp$.

Execute $(\mathsf{st}'_S, (\overline{\mathsf{ct}_i}')_{i\in[N]}, (\mathsf{pMHE'}.r_i)_{i\in[N]\setminus H}) \leftarrow \mathsf{pMHE'.Sim}_1(1^\lambda, H, (x_i, k_i)_{i\in[N]\setminus H})$.

Execute $(\mathsf{st}''_S, (\overline{p_i^0})_{i\in H}) \leftarrow \mathsf{pMHE'.Sim}_2(\mathsf{st}'_S, \mathsf{NewEnc}^1, \mathsf{NewEnc}^1((x_j, k_j)_{j\in[N]}))$.

Invoke $\mathcal{A}^{\mathcal{O}(1^\lambda,\cdot)}(1^\lambda, (\overline{\mathsf{ct}_i}')_{i\in[N]}, (x_j, k_j, r_j, \mathsf{pMHE'}.r_j)_{j\notin H})$.

Output $\mathsf{View}_{\mathcal{A}}$.

```
$\mathsf{pMHE.PrivEval}(1^\lambda, \mathsf{sk}_i, C, i, (\overline{\mathsf{ct}_j}')_{j\in[N]})$ Let $k_i^0 = k_i$.
    For each $t = 1, 2, \ldots, n$,
        Let $b = \mathsf{id}[t]$. Parse $\mathsf{PRG}(k_i^{t-1})$ as $(\mathsf{lab}^{i,t,b'}, k^t_{i,b'}, r^t_{i,1,b'}, r^t_{i,2,b'}, r^t_{i,3,b'})_{b'\in\{0,1\}}$.
        Compute $\mathsf{sk}_i^t$ from $\mathsf{pMHE'.Enc}(1^\lambda, \mathsf{NewEnc}^t.\mathsf{params}, i, (x_i, k^t_{i,b}); r^t_{i,1,b})$.
        Execute $\widetilde{\mathsf{Boot}_i^t} \leftarrow \mathsf{GC.Garble}(1^\lambda, \mathsf{Boot}^t_{[\mathsf{sk}_i^t; r^t_{i,2,b}]}, \mathsf{lab}^{i,t,b}; r^t_{i,3,b})$.
    Set $p_i = (\overline{p_i^0}, (\widetilde{\mathsf{Boot}_i^t})_{t\in[n]}, \overline{\mathsf{ct}_i}')$. Output $p_i$.
```

Hybrid_2 This hybrid is almost the same as Hybrid_1, except that $\mathsf{pMHE.PrivEval}$ is replaced with the following functions.

For each $i \in [N]$, randomly sample $k_i \leftarrow \{0,1\}^{\mathsf{PRG.in}}$,

and parse $\mathsf{PRG}(k_i)$ as $(\mathsf{lab}^{i,b}, k_{i,b}, r_{i,1,b}, r_{i,2,b}, r_{i,3,b})_{b\in\{0,1\}}$.

For each $i \in [N]$, if $i \notin H$, sample random coins r_i, otherwise, let $r_i = \perp$.

Execute $(\mathsf{st}'_S, (\overline{\mathsf{ct}_i}')_{i\in[N]}, (\mathsf{pMHE'}.r_i)_{i\in[N]\setminus H}) \leftarrow \mathsf{pMHE'.Sim}_1(1^\lambda, H, (x_i, k_i)_{i\in[N]\setminus H})$.

For any $j \in [N], b \in \{0,1\}$, execute $(\mathsf{ct}_{j,b}, \mathsf{sk}_{j,b}) = \mathsf{pMHE'.Enc}(1^\lambda, \mathsf{NewEnc}^2.\mathsf{params}, j, (x_j, k_{j,b}); r_{j,1,b})$.

For any $b \in \{0,1\}$, let $\mathsf{ct}_b = (\mathsf{ct}_{j,b})_{j\in[N]}$.

Execute $(\mathsf{st}''_S, (\overline{p_i^0})_{i\in H}) \leftarrow \mathsf{pMHE}'.\mathsf{Sim}_2(\mathsf{st}'_S, \mathsf{NewEnc}^1, ((\mathsf{lab}^{i,0}_{\mathsf{ct}_0})_{i\in[N]}, (\mathsf{lab}^{i,1}_{\mathsf{ct}_1})_{i\in[N]}))$.

Invoke $\mathcal{A}^{\mathcal{O}(1^\lambda,\cdot)}(1^\lambda, (\overline{\mathsf{ct}_i}')_{i\in[N]}, (x_j, k_j, r_j, \mathsf{pMHE}'.r_j)_{j\notin H})$.

Output $\mathsf{View}_{\mathcal{A}}$.

$\mathsf{pMHE.PrivEval}(1^\lambda, \mathsf{sk}_i, C, i, (\overline{\mathsf{ct}_j}')_{j\in[N]})$ Set $(\mathsf{lab}^{i,1,b}, k^1_{i,b}, r^1_{i,1,b}, r^1_{i,2,b}, r^1_{i,3,b})_{b\in\{0,1\}}$

to be $(\mathsf{lab}^{i,b}, k_{i,b}, r_{i,1,b}, r_{i,2,b}, r_{i,3,b})_{b\in\{0,1\}}$.

For each $t = 1, 2, \ldots, n$,

Let $b = \mathsf{id}[t]$.

Compute sk_i^t from $\mathsf{pMHE}'.\mathsf{Enc}(1^\lambda, \mathsf{NewEnc}^t.\mathsf{params}, i, (x_i, k^t_{i,b}); r^t_{i,1,b})$.

Execute $\widetilde{\mathsf{Boot}_i^t} \leftarrow \mathsf{GC.Garble}(1^\lambda, \mathsf{Boot}^t_{[\mathsf{sk}_i^t; r^t_{i,2,b}]}, \mathsf{lab}^{i,t,b}; r^t_{i,3,b})$.

Parse $\mathsf{PRG}(k^t_{i,b})$ as $(\mathsf{lab}^{i,t+1,b'}, k^{t+1}_{i,b'}, r^{t+1}_{i,1,b'}, r^{t+1}_{i,2,b'}, r^{t+1}_{i,3,b'})_{i\in H, b'\in\{0,1\}}$.

Set $p_i = (\overline{p_i^0}, (\widetilde{\mathsf{Boot}_i^t})_{t\in[n]}, \overline{\mathsf{ct}_i}')$. Output p_i.

$\mathsf{Hybrid}^{i^*}_{2.5}$ This hybrid is almost the same as Hybrid_2, except that we replace the output of the PRG with the uniform random string for each $i \in H$ one by one.

For each $i \in [N] \setminus H$, randomly sample $k_i \leftarrow \{0,1\}^{\mathsf{PRG.in}}$,

and parse $\mathsf{PRG}(k_i)$ as $(\mathsf{lab}^{i,b}, k_{i,b}, r_{i,1,b}, r_{i,2,b}, r_{i,3,b})_{b\in\{0,1\}}$.

For each $i \in H$, if $i < i^*$, then randomly sample $(\mathsf{lab}^{i,b}, k_{i,b}, r_{i,1,b}, r_{i,2,b}, r_{i,3,b})_{b\in\{0,1\}}$.

Otherwise parse $\mathsf{PRG}(k_i)$ as $(\mathsf{lab}^{i,b}, k_{i,b}, r_{i,1,b}, r_{i,2,b}, r_{i,3,b})_{b\in\{0,1\}}$.

Hybrid_3 This hybrid is essentially identical to $\mathsf{Hybrid}^{N+1}_{2.5}$.

For each $i \in [N] \setminus H$, randomly sample $k_i \leftarrow \{0,1\}^{\mathsf{PRG.in}}$,

and parse $\mathsf{PRG}(k_i)$ as $(\mathsf{lab}^{i,b}, k_{i,b}, r_{i,1,b}, r_{i,2,b}, r_{i,3,b})_{b\in\{0,1\}}$.

For each $i \in H$, randomly sample $(\mathsf{lab}^{i,b}, k_{i,b}, r_{i,1,b}, r_{i,2,b}, r_{i,3,b})_{b\in\{0,1\}}$.

$\mathsf{Hybrid}^{i^*,b^*}_{3.5}$ This hybrid is almost the same as Hybrid_3, except that we replace the labels of the garbled circuit with the labels simulated by $\mathsf{GC.Sim}_1$.

We maintain a set $T' \subseteq [N] \times \{0,1\}^*$. An element $(i,s) \in T'$, if at the tree node s, the garble circuit Boot^i_s for ith party already been generated by $\mathsf{GC.Sim}_1$, but haven't been used by $\mathsf{GC.Sim}_2$.

Initialize an empty set $T' = \phi$.

For each $i \in [N] \setminus H$, randomly sample $k_i \leftarrow \{0,1\}^{\mathsf{PRG.in}}$,

and parse $\mathsf{PRG}(k_i)$ as $(\mathsf{lab}^{i,b}, k_{i,b}, r_{i,1,b}, r_{i,2,b}, r_{i,3,b})_{b\in\{0,1\}}$.

Execute $(\mathsf{st}'_S, (\overline{\mathsf{ct}_i}')_{i\in[N]}, (\mathsf{pMHE}'.r_i)_{i\in[N]\setminus H}) \leftarrow \mathsf{pMHE}'.\mathsf{Sim}_1(1^\lambda, H, (x_i, k_i)_{i\in[N]\setminus H})$.

For each $i \in H$, if $(i,b) < (i^*, b^*)$, randomly sample $(k_{i,b}, r_{i,1,b}, r_{i,2,b})$.

Otherwise, randomly sample $(\mathsf{lab}^{i,b}, k_{i,b}, r_{i,1,b}, r_{i,2,b}, r_{i,3,b})$.

For each $i \in [N]$, if $i \notin H$, sample random coins r_i, otherwise, let $r_i = \perp$.

For any $j \in [N], b \in \{0,1\}$,

execute $(\mathsf{ct}_{j,b}, \mathsf{sk}_{j,b}) = \mathsf{pMHE}'.\mathsf{Enc}(1^\lambda, \mathsf{NewEnc}^2.\mathsf{params}, j, (x_j, k_{j,b}); r_{j,1,b})$.

For any $b \in \{0,1\}$, let $\mathsf{ct}_b = (\mathsf{ct}_{j,b})_{j\in[N]}$.

For any $b \in \{0,1\}, i \in H$, if $(i,b) < (i^*, b^*)$, execute

$(\mathsf{GC.st}_b^i, \mathsf{lab}'^{i,b}) \leftarrow \mathsf{GC.Sim}_1(1^\lambda, \mathsf{KG.in})$.

Update $T' = T' \cup \{(i,b)\}$.

Otherwise $\mathsf{lab}'^{i,b} = \mathsf{lab}^{i,b}_{\mathsf{ct}_b}$.

For any $i \in [N] \setminus H, b \in \{0,1\}$, let $\mathsf{lab}'^{i,b} = \mathsf{lab}^{i,b}_{\mathsf{ct}_b}$.

Execute $(\mathsf{st}''_S, (\overline{p_i^0})_{i\in H}) \leftarrow \mathsf{pMHE'.Sim}_2(\mathsf{st}'_S, \mathsf{NewEnc}^1, ((\mathsf{lab}'^{i,0})_{i\in[N]}, (\mathsf{lab}'^{i,1})_{i\in[N]}))$.

Invoke $\mathcal{A}^{\mathcal{O}(1^\lambda,\cdot)}(1^\lambda, (\overline{\mathsf{ct}_i}')_{i\in[N]}, (x_j, k_j, r_j, \mathsf{pMHE'}.r_j)_{j\notin H})$.

Output $\mathsf{View}_{\mathcal{A}}$.

$\mathsf{pMHE.PrivEval}(1^\lambda, \mathsf{sk}_i, C, i, (\overline{\mathsf{ct}_j}')_{j\in[N]})$ Let $b = \mathsf{id}[1]$.

    Set $(k^1_{i,b'}, r^1_{i,1,b'}, r^1_{i,2,b'})_{b'\in\{0,1\}}$ to be $(k_{i,b'}, r_{i,1,b'}, r_{i,2,b'})_{b'\in\{0,1\}}$.

    Compute sk_i^1 from $\mathsf{pMHE'.Enc}(1^\lambda, \mathsf{NewEnc}^1.\mathsf{params}, i, (x_i, k^1_{i,b}); r^1_{i,1,b})$.

    If $(i,b) < (i^*, b^*)$, and $(i,b) \notin T'$, then let $\widetilde{\mathsf{Boot}_i^1} = \mathsf{Boot}_b^i$.

    If $(i,b) < (i^*, b^*)$, and $(i,b) \in T'$, then execute

    $p_i^1 = \mathsf{pMHE'.PrivEval}(1^\lambda, \mathsf{sk}_i^1, \mathsf{NewEnc}^2, \mathsf{ct}_b; r^1_{i,2,b})$,

    $\widetilde{\mathsf{Boot}_i^1} \leftarrow \mathsf{GC.Sim}_2(\mathsf{GC.st}_b^i, p_i^1)$, and define $\mathsf{Boot}_b^i = \widetilde{\mathsf{Boot}_i^1}$, and update $T' = T' \setminus \{(i,b)\}$.

    If $(i,b) \geq (i^*, b^*)$, then execute $\widetilde{\mathsf{Boot}_i^1} = \mathsf{GC.Garble}(1^\lambda, \mathsf{Boot}^1_{[\mathsf{sk}_i^1; r^1_{i,2,b}]}; r^1_{i,3,b})$.

    Parse $\mathsf{PRG}(k^1_{i,b})$ as $(\mathsf{lab}^{i,2,b'}, k^2_{i,b'}, r^2_{i,1,b'}, r^2_{i,2,b'}, r^2_{i,3,b'})_{b'\in\{0,1\}}$.

    For $t = 2 \ldots n$, let $b = \mathsf{id}[t]$.

        Compute sk_i^t from $\mathsf{pMHE'.Enc}(1^\lambda, \mathsf{NewEnc}^t.\mathsf{params}, i, (x_i, k^t_{i,b}); r^t_{i,1,b})$.

        Then execute $\widetilde{\mathsf{Boot}_i^t} \leftarrow \mathsf{GC.Garble}(1^\lambda, \mathsf{Boot}^t_{[\mathsf{sk}_i^t; r^t_{i,2,b}]}, \mathsf{lab}^{i,t,b}; r^t_{i,3,b})$.

        Parse $\mathsf{PRG}(k^t_{i,b})$ as $(\mathsf{lab}^{i,t+1,b'}, k^{t+1}_{i,b'}, r^{t+1}_{i,1,b'}, r^{t+1}_{i,2,b'}, r^{t+1}_{i,3,b'})_{b'\in\{0,1\}}$.

    Set $p_i = (\overline{p_i^0}, (\widetilde{\mathsf{Boot}_i^t})_{t\in[n]}, \overline{\mathsf{ct}_i}')$. Output p_i.

Hybrid_4 This hybrid is essentially the same as $\mathsf{Hybrid}_{3.5}^{(N,1)+1}$.

Initialize an empty set $T' = \phi$.

For each $i \in [N] \setminus H$, randomly sample $k_i \leftarrow \{0,1\}^{\mathsf{PRG.in}}$,

and parse $\mathsf{PRG}(k_i)$ as $(\mathsf{lab}^{i,b}, k_{i,b}, r_{i,1,b}, r_{i,2,b}, r_{i,3,b})_{b\in\{0,1\}}$.

Execute $(\mathsf{st}'_S, (\overline{\mathsf{ct}_i}')_{i\in[N]}, (\mathsf{pMHE'}.r_i)_{i\in[N]\setminus H}) \leftarrow \mathsf{pMHE'.Sim}_1(1^\lambda, H, (x_i, k_i)_{i\in[N]\setminus H})$.

For each $i \in H$, randomly sample $(k_{i,b}, r_{i,1,b}, r_{i,2,b})_{b\in\{0,1\}}$.

For each $i \in [N]$, if $i \notin H$, sample random coins r_i, otherwise, let $r_i = \perp$.

For any $j \in [N], b \in \{0,1\}$, execute

$(\mathsf{ct}_{j,b}, \mathsf{sk}_{j,b}) = \mathsf{pMHE'.Enc}(1^\lambda, \mathsf{NewEnc}^2.\mathsf{params}, j, (x_j, k_{j,b}); r_{j,1,b})$.

For any $b \in \{0,1\}$, let $\mathsf{ct}_b = (\mathsf{ct}_{j,b})_{j\in[N]}$.

For any $b \in \{0,1\}, i \in H$, execute $(\mathsf{GC.st}_b^i, \mathsf{lab}'^{i,b}) \leftarrow \mathsf{GC.Sim}_1(1^\lambda, \mathsf{KG.in})$. Update $T' = T' \cup \{(i,b)\}$.

For any $i \in [N] \setminus H, b \in \{0,1\}$, let $\mathsf{lab}'^{i,b} = \mathsf{lab}^{i,b}_{\mathsf{ct}_b}$.

Execute $(\mathsf{st}''_S, (\overline{p_i^0})_{i\in H}) \leftarrow \mathsf{pMHE}'.\mathsf{Sim}_2(\mathsf{st}'_S, \mathsf{NewEnc}^1, ((\mathsf{lab}'^{i,0})_{i\in[N]}, (\mathsf{lab}'^{i,1})_{i\in[N]}))$.

Invoke $\mathcal{A}^{\mathcal{O}(1^\lambda,\cdot)}(1^\lambda, (\overline{\mathsf{ct}_i}')_{i\in[N]}, (x_j, k_j, r_j, \mathsf{pMHE}'.r_j)_{j\notin H})$.

Output $\mathsf{View}_{\mathcal{A}}$.

$\mathsf{pMHE.PrivEval}(1^\lambda, \mathsf{sk}_i, C, i, (\overline{\mathsf{ct}_j}')_{j\in[N]})$ Let $b = \mathsf{id}[1]$.

Set $(k^1_{i,b'}, r^1_{i,1,b'}, r^1_{i,2,b'})_{b'\in\{0,1\}}$ to be $(k_{i,b'}, r_{i,1,b'}, r_{i,2,b'})_{b'\in\{0,1\}}$.

Compute sk^1_i from $\mathsf{pMHE}'.\mathsf{Enc}(1^\lambda, \mathsf{NewEnc}^1.\mathsf{params}, i, (x_i, k^1_{i,b}); r^1_{i,1,b})$.

If $(i,b) \notin T'$, let $\widetilde{\mathsf{Boot}}^1_i = \mathsf{Boot}^i_b$.

If $(i,b) \in T'$, then execute $p^1_i = \mathsf{pMHE}'.\mathsf{PrivEval}(1^\lambda, \mathsf{sk}^1_i, \mathsf{NewEnc}^2, \mathsf{ct}_b; r^1_{i,2,b})$,

$\widetilde{\mathsf{Boot}}^1_i \leftarrow \mathsf{GC.Sim}_2(\mathsf{GC.st}^i_b, p^1_i)$, and define $\mathsf{Boot}^i_b = \widetilde{\mathsf{Boot}}^1_i$. Update $T' = T' \setminus \{(i,b)\}$.

Parse $\mathsf{PRG}(k^1_{i,b})$ as $(\mathsf{lab}^{i,2,b'}, k^2_{i,b'}, r^2_{i,1,b'}, r^2_{i,2,b'}, r^2_{i,3,b'})_{b'\in\{0,1\}}$.

For $t = 2 \ldots n$, let $b = \mathsf{id}[t]$.

Compute sk^t_i from $\mathsf{pMHE}'.\mathsf{Enc}(1^\lambda, \mathsf{NewEnc}^t.\mathsf{params}, i, (x_i, k^t_{i,b}); r^t_{i,1,b})$.

Then execute $\widetilde{\mathsf{Boot}}^t_i \leftarrow \mathsf{GC.Garble}(1^\lambda, \mathsf{Boot}^t_{[\mathsf{sk}^t_i; r^t_{i,2,b}]}, \mathsf{lab}^{i,t,b}; r^t_{i,3,b})$.

Parse $\mathsf{PRG}(k^t_{i,b})$ as $(\mathsf{lab}^{i,t+1,b'}, k^{t+1}_{i,b'}, r^{t+1}_{i,1,b'}, r^{t+1}_{i,2,b'}, r^{t+1}_{i,3,b'})_{b'\in\{0,1\}}$.

Set $p_i = (\overline{p_i^0}, (\widetilde{\mathsf{Boot}}^t_i)_{t\in[n]}, \overline{\mathsf{ct}_i}')$. Output p_i.

Hybrid_5 This hybrid is almost the same as Hybrid_4, except that we replace the $(\mathsf{ct}_b)_{b\in\{0,1\}}$ with $(\overline{\mathsf{ct}_b})_{b\in\{0,1\}}$ generated by the simulator $\mathsf{pMHE}'.\mathsf{Sim}_1$.

We maintain a set $T'' \subseteq \{0,1\}^*$. A string $s \in T''$, if at the tree node s, the first round message $\overline{\mathsf{ct}_s}$ has already been generated by $\mathsf{pMHE}'.\mathsf{Sim}_1$, but the corresponding $\mathsf{pMHE}'.\mathsf{Sim}_2$ hasn't been executed.

Initialize the empty sets $T', T'' = \phi$.

For each $i \in [N] \setminus H$, randomly sample $k_i \leftarrow \{0,1\}^{\mathsf{PRG.in}}$,

and parse $\mathsf{PRG}(k_i)$ as $(\mathsf{lab}^{i,b}, k_{i,b}, r_{i,1,b}, r_{i,2,b}, r_{i,3,b})_{b\in\{0,1\}}$.

Execute $(\mathsf{st}'_S, (\overline{\mathsf{ct}_i}')_{i\in[N]}, (\mathsf{pMHE}'.r_i)_{i\in[N]\setminus H}) \leftarrow \mathsf{pMHE}'.\mathsf{Sim}_1(1^\lambda, H, (x_i, k_i)_{i\in[N]\setminus H})$.

For each $i \in H, b \in \{0,1\}$, sample $(k_{i,b})_{b\in\{0,1\}} \leftarrow \{0,1\}^*$.

For any $b \in \{0,1\}$, execute $(\mathsf{st}'_b, \overline{\mathsf{ct}_b}, (r'_i)_{i\in[N]\setminus H}) \leftarrow \mathsf{pMHE}'.\mathsf{Sim}_1(1^\lambda, (x_j, k_{j,b})_{j\in[N]\setminus H})$.

Update $T'' = T'' \cup \{b\}$.

For any $b \in \{0,1\}, i \in H$, execute $(\mathsf{GC.st}^i_b, \mathsf{lab}'^{i,b}) \leftarrow \mathsf{GC.Sim}_1(1^\lambda, \mathsf{KG.in})$.

Update $T' = T' \cup \{(i,b)\}$.

For any $i \in [N] \setminus H, b \in \{0,1\}$, let $\mathsf{lab}'^{i,b} = \mathsf{lab}^{i,b}_{\overline{\mathsf{ct}_b}}$.

Execute $(\overline{p_i^0})_{i\in H} \leftarrow \mathsf{pMHE}'.\mathsf{Sim}_2(\mathsf{st}'_S, \mathsf{NewEnc}^1, ((\mathsf{lab}'^{i,0})_{i\in[N]}, (\mathsf{lab}'^{i,1})_{i\in[N]}))$.

Invoke $\mathcal{A}^{\mathcal{O}(1^\lambda,\cdot)}(1^\lambda, (\overline{\mathsf{ct}_i}')_{i\in[N]}, (x_j, k_j, r_j, \mathsf{pMHE}'.r_j)_{j\notin H})$.

Output $\mathsf{View}_{\mathcal{A}}$.

Oracle $\mathcal{O}(1^\lambda, C)$ Let $b = \mathsf{id}[1]$.

If $b \in T''$, then execute

$(\overline{p_i^b})_{i \in H} \leftarrow \mathsf{pMHE'.Sim}_2(\mathsf{st}'_b, \mathsf{NewEnc}^2, \mathsf{NewEnc}^2((x_j, k_{j,b})_{j \in [N]}))$.

Update $T'' = T'' \setminus \{b\}$.

Execute the following procedure for every $i \in H$ to obtain p_i. Output $(p_i)_{i \in H}$.

$\mathsf{pMHE.PrivEval}(1^\lambda, \mathsf{sk}_i, C, i, (\overline{\mathsf{ct}_j}')_{j \in [N]})$ Set $(k^1_{i,b})_{b \in \{0,1\}}$ to be $(k_{i,b})_{b \in \{0,1\}}$.

Let $b = \mathsf{id}[1]$.

If $(i, b) \notin T'$, let $\widetilde{\mathsf{Boot}^1_i} = \mathsf{Boot}^i_b$.

If $(i, b) \in T'$, then execute $\widetilde{\mathsf{Boot}^1_i} \leftarrow \mathsf{GC.Sim}_2(\mathsf{GC.st}^i_b, \overline{p^b_i})$, and define $\mathsf{Boot}^i_b = \widetilde{\mathsf{Boot}^1_i}$.

Update $T' = T' \setminus \{(i, b)\}$.

Parse $\mathsf{PRG}(k^1_{i,b})$ as $(\mathsf{lab}^{i,2,b'}, k^2_{i,b'}, r^2_{i,1,b'}, r^2_{i,2,b'}, r^2_{i,3,b})_{b' \in \{0,1\}}$.

For each $t = 2, \ldots, n$, let $b = \mathsf{id}[t]$.

Compute sk^t_i from $\mathsf{pMHE'.Enc}(1^\lambda, \mathsf{NewEnc}^t.\mathsf{params}, i, (x_i, k^t_{i,b}); r^t_{i,1,b})$.

Execute $\widetilde{\mathsf{Boot}^t_i} \leftarrow \mathsf{GC.Garble}(1^\lambda, \mathsf{Boot}^t_{[\mathsf{sk}^t_i; r^t_{i,2,b}]}, \mathsf{lab}^{i,t,b}; r^t_{i,3,b})$.

Parse $\mathsf{PRG}(k^t_{i,b})$ as $(\mathsf{lab}^{i,t+1,b'}, k^{t+1}_{i,b'}, r^{t+1}_{i,1,b'}, r^{t+1}_{i,2,b'}, r^{t+1}_{i,3,b})_{b' \in \{0,1\}}$.

Set $p_i = (\overline{p^0_i}, (\widetilde{\mathsf{Boot}^t_i})_{t \in [n]}, \overline{\mathsf{ct}_i}')$. Output p_i.

$\mathsf{Hybrid}_6^{q^*}$ This hybrid is almost the same as Hybrid_5, except that the oracle $\mathcal{O}$ is replaced with the following oracle.

We maintain a set $T \subseteq \{0,1\}^n$. An element $s \in T$, if the tree node s is accessed by one of the previous queries C. We initialize the empty set $T = \phi$.

Oracle $\mathcal{O}(1^\lambda, C)$

Let q be the number of times that $\mathcal{O}(1^\lambda, \cdot)$ is invoked.

Let id be the binary representation of C, and let $h = \max(0 \le h' \le n \mid \mathsf{id}[1 \ldots h'] \in T)$.

Case 1: $q < q^*$**.** In this case, we answer the query by simulation.

For each $t = h+1, h+2, \ldots n$,

Denote $\mathsf{id}' = \mathsf{id}[1 \ldots t]$, $\mathsf{id}'_0 = \mathsf{id}' \circ 0$, $\mathsf{id}'_1 = \mathsf{id}' \circ 1$.

Now we generate $(\overline{p^{\mathsf{id}'}_i})_{i \in H}$.

If $t = n$, then execute $(\overline{p^{\mathsf{id}'}_i})_{i \in H} \leftarrow \mathsf{pMHE'.Sim}_2(\mathsf{st}'_{\mathsf{id}'}, \mathsf{NewEnc}^{t+1}, C((x_i)_{i \in [N]}))$.

If $t < n$, for each $b' \in \{0,1\}$, execute $(\mathsf{st}'_{\mathsf{id}'_{b'}}, \overline{\mathsf{ct}_{b'}}, (r_{j,b'})_{j \notin H}) \leftarrow \mathsf{pMHE'.Sim}_1(1^\lambda, (x_j, k_{j,\mathsf{id}'_{b'}})_{j \notin H})$.

Update $T'' = T'' \cup \{\mathsf{id}'_{b'}\}$.

For each $i \in H, b' \in \{0,1\}$, execute $(\mathsf{GC.st}^i_{\mathsf{id}'_{b'}}, \mathsf{lab}'^{i,b'}) \leftarrow \mathsf{GC.Sim}_1(1^\lambda, \mathsf{KG.in})$.

Update $T' = T' \cup \{(i, \mathsf{id}'_{b'})\}$.

For each $i \in [N] \setminus H, b' \in \{0,1\}$, let $\mathsf{lab}'^{i,b'} = \mathsf{lab}^{i,\mathsf{id}'_{b'}}_{\overline{\mathsf{ct}_{b'}}}$.

Execute $(\overline{p^{\mathsf{id}'}_i})_{i \in H} \leftarrow \mathsf{pMHE'.Sim}_2(\mathsf{st}'_{\mathsf{id}'}, \mathsf{NewEnc}^{t+1}, ((\mathsf{lab}'^{i,0})_{i \in [N]}, (\mathsf{lab}'^{i,1})_{i \in [N]}))$.

For each $i \in H$, execute $\widetilde{\mathsf{Boot}_i^t} \leftarrow \mathsf{GC.Sim}_2(\mathsf{GC.st}^i_{\mathsf{id}'}, \overline{p_i^{\mathsf{id}'}})$, and set $\mathsf{Boot}^i_{\mathsf{id}'} = \widetilde{\mathsf{Boot}_i^t}$.

Update $T'' = T'' \setminus \{\mathsf{id}'\}$, $T' = T' \setminus \{(i, \mathsf{id}')\}$, and $T = T \cup \{\mathsf{id}'\}$.

Randomly sample $(k_{i,\mathsf{id}'_{b'}})_{i \in H, b' \in \{0,1\}} \leftarrow \{0,1\}^{\mathsf{PRG.in}}$.

For $t \in [h]$, we set $\widetilde{\mathsf{Boot}_i^t} = \mathsf{Boot}^i_{\mathsf{id}[1\dots t]}$. Set $p_i = (\overline{p_i^0}, (\widetilde{\mathsf{Boot}_i^t})_{t \in [n]}, \overline{\mathsf{ct}_i}')$. Output p_i.

Case 2: $q \geq q^*$**.** In this case, we answer the query by real execution.

Denote $b = \mathsf{id}[h+1]$, $\mathsf{id}' = \mathsf{id}[1 \dots h+1]$.

For each $i \in H$, set $k_{i,b}^{h+1}$ to be $k_{i,\mathsf{id}'}$.

If $\mathsf{id}' \notin T''$, let $\widetilde{\mathsf{Boot}_i^{h+1}} = \mathsf{Boot}^i_{\mathsf{id}'}$, for each $i \in H$.

If $\mathsf{id}' \in T''$, then execute

$(\overline{p_i^{\mathsf{id}'}})_{i \in H} \leftarrow \mathsf{pMHE'.Sim}_2(\mathsf{st}'_{\mathsf{id}'}, \mathsf{NewEnc}^{h+2}, \mathsf{NewEnc}^{h+2}((x_j, k_{j,\mathsf{id}'})_{j \in [N]}))$,

then let $\widetilde{\mathsf{Boot}_i^{h+1}} \leftarrow \mathsf{GC.Sim}_2(\mathsf{GC.st}^i_{\mathsf{id}'}, \overline{p_i^{\mathsf{id}'}})$ for every $i \in H$, and let $\mathsf{Boot}^i_{\mathsf{id}'} = \widetilde{\mathsf{Boot}_i^{h+1}}$.

Update $T' = T' \setminus \{(i, \mathsf{id}') \mid i \in H\}$, $T'' = T'' \setminus \{\mathsf{id}'\}$.

For each $i \in H$, parse $\mathsf{PRG}(k_{i,b}^{h+1})$ as $(\mathsf{lab}^{i,h+2,b'}, k_{i,b'}^{h+2}, r_{i,1,b'}^{h+2}, r_{i,2,b'}^{h+2}, r_{i,3,b'}^{h+2})_{b' \in \{0,1\}}$.

For each $t = h+2 \dots n$, and each $i \in H$, let $b = \mathsf{id}[t]$.

Compute sk_i^t from $\mathsf{pMHE'.Enc}(1^\lambda, \mathsf{NewEnc}^t.\mathsf{params}, i, (x_i, k_{i,b}^t); r_{i,1,b}^t)$.

Execute $\widetilde{\mathsf{Boot}_i^t} \leftarrow \mathsf{GC.Garble}(1^\lambda, \mathsf{Boot}^t_{[\mathsf{sk}_i^t; r_{i,2,b}^t]}, \mathsf{lab}^{i,t,b}; r_{i,3,b}^t)$.

Parse $\mathsf{PRG}(k_{i,b}^t)$ as $(\mathsf{lab}^{i,t+1,b'}, k_{i,b'}^{t+1}, r_{i,1,b'}^{t+1}, r_{i,2,b'}^{t+1}, r_{i,3,b'}^{t+1})_{b' \in \{0,1\}}$.

For each $t \in [h]$, we set $\widetilde{\mathsf{Boot}_i^t} = \mathsf{Boot}_{\mathsf{id}[1\dots t]}$. Set $p_i = (\overline{p_i^0}, (\widetilde{\mathsf{Boot}_i^t})_{t \in [n]}, \overline{\mathsf{ct}_i}')$. Output p_i.

$\mathsf{Hybrid}_7^{q^*,h^*}$ This hybrid is almost the same as Hybrid_6.

Oracle $\mathcal{O}(1^\lambda, C)$

Let q be the number of times that $\mathcal{O}$ is invoked.

Let id be the binary representation of C, and let $h = \max(0 \leq h' \leq n \mid \mathsf{id}[1 \dots h'] \in T)$.

Case 1: $q < q^*$**.** In this case, we answer the query by simulation.

For each $t = h+1, h+2, \dots n$,

Denote $\mathsf{id}' = \mathsf{id}[1 \dots t]$, $\mathsf{id}'_0 = \mathsf{id}' \circ 0$, $\mathsf{id}'_1 = \mathsf{id}' \circ 1$.

Now we generate $(\overline{p_i^{\mathsf{id}'}})_{i \in H}$.

If $t = n$, then execute $(\overline{p_i^{\mathsf{id}'}})_{i \in H} \leftarrow \mathsf{pMHE'.Sim}_2(\mathsf{st}'_{\mathsf{id}'}, \mathsf{NewEnc}^{t+1}, C((x_i)_{i \in [N]}))$.

If $t < n$, for each $b' \in \{0,1\}$, execute $(\mathsf{st}'_{\mathsf{id}'_{b'}}, \overline{\mathsf{ct}_{b'}}, (r_{j,b'})_{j \notin H}) \leftarrow \mathsf{pMHE'.Sim}_1(1^\lambda, (x_j, k_{j,\mathsf{id}'_{b'}})_{j \notin H})$.

Update $T'' = T'' \cup \{\mathsf{id}'_{b'}\}$.

For each $i \in H, b' \in \{0,1\}$, execute $(\mathsf{GC.st}^i_{\mathsf{id}'_{b'}}, \mathsf{lab}'^{i,b'}) \leftarrow \mathsf{GC.Sim}_1(1^\lambda, \mathsf{KG.in})$.

Update $T' = T' \cup \{(i, \mathsf{id}'_{b'})\}$.

For each $i \in [N] \setminus H, b' \in \{0,1\}$, let $\mathsf{lab}'^{i,b'} = \mathsf{lab}^{i,\mathsf{id}'_{b'}}_{\overline{\mathsf{ct}_{b'}}}$.

Execute $(\overline{p_i^{\mathsf{id}'}})_{i\in H} \leftarrow \mathsf{pMHE}'.\mathsf{Sim}_2(\mathsf{st}'_{\mathsf{id}'}, \mathsf{NewEnc}^{t+1}, ((\mathsf{lab}'^{i,0})_{i\in[N]}, (\mathsf{lab}'^{i,1})_{i\in[N]}))$.

For each $i \in H$, execute $\widetilde{\mathsf{Boot}_i^t} \leftarrow \mathsf{GC}.\mathsf{Sim}_2(\mathsf{GC}.\mathsf{st}^i_{\mathsf{id}'}, \overline{p_i^{\mathsf{id}'}})$, and set $\mathsf{Boot}^i_{\mathsf{id}'} = \widetilde{\mathsf{Boot}_i^t}$.

Update $T'' = T'' \setminus \{\mathsf{id}'\}$, $T' = T' \setminus \{(i, \mathsf{id}')\}$, and $T = T \cup \{\mathsf{id}'\}$.

Randomly sample $(k_{i,\mathsf{id}'_{b'}})_{i\in H, b'\in\{0,1\}} \leftarrow \{0,1\}^{\mathsf{PRG.in}}$.

For $t \in [h]$, we set $\widetilde{\mathsf{Boot}_i^t} = \mathsf{Boot}^i_{\mathsf{id}[1\ldots t]}$. Set $p_i = (\overline{p_i^0}, (\widetilde{\mathsf{Boot}_i^t})_{t\in[n]}, \overline{\mathsf{ct}_i}')$. Output p_i.

Case 2: $q = q^*$ In this case, we simulate the $\widetilde{\mathsf{Boot}_i^t}$ for $t = h+1, 2, \ldots, h^*$, and get the $\widetilde{\mathsf{Boot}_i^t}$ for $t = h^*+1, \ldots n$ from the real execution.

For each $t = h+1, \ldots, h^*$,

denote $b = \mathsf{id}[t]$, $\mathsf{id}' = \mathsf{id}[1\ldots t], \mathsf{id}'_0 = \mathsf{id}' \circ 0$, $\mathsf{id}'_1 = \mathsf{id}' \circ 1$.

Now we generate $(\overline{p_i^{\mathsf{id}'}})_{i\in H}$.

If $t = n$, then execute $(\overline{p_i^{\mathsf{id}'}})_{i\in H} \leftarrow \mathsf{pMHE}'.\mathsf{Sim}_2(\mathsf{st}'_{\mathsf{id}_b}, \mathsf{NewEnc}^{t+1}, C((x_i)_{i\in[N]}))$.

If $t < n$, for each $b' \in \{0,1\}$, execute $(\mathsf{st}'_{\mathsf{id}'_{b'}}, \overline{\mathsf{ct}_{b'}}, (r_{j,b'})_{j\notin H}) \leftarrow \mathsf{pMHE}'.\mathsf{Sim}_1(1^\lambda, (x_j, k_{j,\mathsf{id}'_{b'}})_{j\notin H})$.

Update $T'' = T'' \cup \{\mathsf{id}'_{b'}\}$.

For each $i \in H, b' \in \{0,1\}$, execute $(\mathsf{GC}.\mathsf{st}^i_{\mathsf{id}'_{b'}}, \mathsf{lab}'^{i,b'}) \leftarrow \mathsf{GC}.\mathsf{Sim}_1(1^\lambda, \mathsf{KG.in})$.

Update $T' = T' \cup \{(i, \mathsf{id}'_{b'})\}$.

For each $i \in [N] \setminus H, b' \in \{0,1\}$, let $\mathsf{lab}'^{i,b'} = \mathsf{lab}^{i,\mathsf{id}'_{b'}}_{\overline{\mathsf{ct}_{b'}}}$.

Execute $(\overline{p_i^{\mathsf{id}'}})_{i\in H} \leftarrow \mathsf{pMHE}'.\mathsf{Sim}_2(\mathsf{st}'_{\mathsf{id}'}, \mathsf{NewEnc}^{t+1}, ((\mathsf{lab}'^{i,0})_{i\in[N]}, (\mathsf{lab}'^{i,1})_{i\in[N]}))$.

For any $i \in H$, execute $\widetilde{\mathsf{Boot}_i^t} \leftarrow \mathsf{GC}.\mathsf{Sim}_2(\mathsf{GC}.\mathsf{st}^i_{\mathsf{id}'}, \overline{p_i^{\mathsf{id}'}})$, and set $\mathsf{Boot}^i_{\mathsf{id}'} = \widetilde{\mathsf{Boot}_i^t}$.

Update $T'' = T'' \setminus \{\mathsf{id}'\}$, $T' = T' \setminus \{(i, \mathsf{id}')\}$, and $T = T \cup \{\mathsf{id}'\}$.

Randomly sample $(k_{i,\mathsf{id}'_0}, k_{i,\mathsf{id}'_1})_{i\in H} \leftarrow \{0,1\}^{\mathsf{PRG.in}}$.

Denote $b = \mathsf{id}[h^*+1]$, $\mathsf{id}' = \mathsf{id}[1\ldots h^*+1], \mathsf{id}'_0 = \mathsf{id}' \circ 0$, $\mathsf{id}'_1 = \mathsf{id}' \circ 1$.

For each $i \in H$, set $k_{i,b}^{h^*+1} = k_{i,\mathsf{id}'}$.

If $\mathsf{id}' \notin T''$, let $\widetilde{\mathsf{Boot}_i^{h^*+1}} = \mathsf{Boot}^i_{\mathsf{id}'}$.

If $\mathsf{id}' \in T''$, then execute

$(\overline{p_i^{h^*+1}})_{i\in H} \leftarrow \mathsf{pMHE}'.\mathsf{Sim}_2(\mathsf{st}'_{\mathsf{id}'}, \mathsf{NewEnc}^{h^*+2}, \mathsf{NewEnc}^{h^*+2}((x_j, k_{j,\mathsf{id}'})_{j\in[N]}))$,

execute $\widetilde{\mathsf{Boot}_i^{h^*+1}} \leftarrow \mathsf{GC}.\mathsf{Sim}_2(\mathsf{GC}.\mathsf{st}^i_{\mathsf{id}'}, \overline{p_i^{h^*+1}})$ for every $i \in H$, and define $\mathsf{Boot}^i_{\mathsf{id}'} = \widetilde{\mathsf{Boot}_i^t}$.

Update $T' = T' \setminus \{(i, \mathsf{id}') \mid i \in H\}$, $T'' = T'' \setminus \{\mathsf{id}'\}$.

For each $i \in H$, parse $\mathsf{PRG}(k_{i,b}^{h^*+1})$ as $(\mathsf{lab}^{i,h^*+2,b'}, k_{i,b'}^{h^*+2}, r_{i,1,b'}^{h^*+2}, r_{i,2,b'}^{h^*+2}, r_{i,3,b'}^{h^*+2})_{b'\in\{0,1\}}$.

For each $t = h^*+2\ldots n$, and each $i \in H$, let $b = \mathsf{id}[t]$.

Compute sk_i^t from $\mathsf{pMHE}'.\mathsf{Enc}(1^\lambda, \mathsf{NewEnc}^t.\mathsf{params}, i, (x_i, k_{i,b}^t); r_{i,1,b}^t)$.

Execute $\widetilde{\mathsf{Boot}_i^t} \leftarrow \mathsf{GC.Garble}(1^\lambda, \mathsf{Boot}^t_{[\mathsf{sk}_i^t; r^t_{i,2,b}]}, \mathsf{lab}^{i,t,b}; r^t_{i,3,b})$.

Parse $\mathsf{PRG}(k^t_{i,b})$ as $(\mathsf{lab}^{i,t+1,b'}, k^{t+1}_{i,b'}, r^{t+1}_{i,1,b'}, r^{t+1}_{i,2,b'}, r^{t+1}_{i,3,b'})_{b' \in \{0,1\}}$.

For $t \in [h]$, we set $\widetilde{\mathsf{Boot}_i^t} = \mathsf{Boot}^i_{\mathsf{id}[1\dots t]}$. Set $p_i = (\overline{p_i^0}, (\widetilde{\mathsf{Boot}_i^t})_{t\in[n]}, \overline{\mathsf{ct}_i}')$. Output p_i.

Case 3: $q > q^*$**.** In this case, we answer the query by real execution.

Denote $b = \mathsf{id}[h+1]$, $\mathsf{id}' = \mathsf{id}[1\dots h+1]$.

For each $i \in H$, set $k^{h+1}_{i,b}$ to be $k_{i,\mathsf{id}'}$.

If $\mathsf{id}' \notin T''$, let $\widetilde{\mathsf{Boot}_i^{h+1}} = \mathsf{Boot}^i_{\mathsf{id}'}$, for each $i \in H$.

If $\mathsf{id}' \in T''$, then execute

$(\overline{p_i^{\mathsf{id}'}})_{i\in H} \leftarrow \mathsf{pMHE'.Sim}_2(\mathsf{st}'_{\mathsf{id}'}, \mathsf{NewEnc}^{h+2}, \mathsf{NewEnc}^{h+2}((x_j, k_{j,\mathsf{id}'})_{j\in[N]}))$,

then let $\widetilde{\mathsf{Boot}_i^{h+1}} \leftarrow \mathsf{GC.Sim}_2(\mathsf{GC.st}^i_{\mathsf{id}'}, \overline{p_i^{\mathsf{id}'}})$ for every $i \in H$, and let $\mathsf{Boot}^i_{\mathsf{id}'} = \widetilde{\mathsf{Boot}_i^{h+1}}$.

Update $T' = T' \setminus \{(i,\mathsf{id}') \mid i \in H\}$, $T'' = T'' \setminus \{\mathsf{id}'\}$.

For each $i \in H$, parse $\mathsf{PRG}(k^{h+1}_{i,b})$ as $(\mathsf{lab}^{i,h+2,b'}, k^{h+2}_{i,b'}, r^{h+2}_{i,1,b'}, r^{h+2}_{i,2,b'}, r^{h+2}_{i,3,b'})_{b'\in\{0,1\}}$.

For each $t = h+2\dots n$, and each $i \in H$, let $b = \mathsf{id}[t]$.

Compute sk_i^t from $\mathsf{pMHE'.Enc}(1^\lambda, \mathsf{NewEnc}^t.\mathsf{params}, i, (x_i, k^t_{i,b}); r^t_{i,1,b})$.

Execute $\widetilde{\mathsf{Boot}_i^t} \leftarrow \mathsf{GC.Garble}(1^\lambda, \mathsf{Boot}^t_{[\mathsf{sk}_i^t; r^t_{i,2,b}]}, \mathsf{lab}^{i,t,b}; r^t_{i,3,b})$.

Parse $\mathsf{PRG}(k^t_{i,b})$ as $(\mathsf{lab}^{i,t+1,b'}, k^{t+1}_{i,b'}, r^{t+1}_{i,1,b'}, r^{t+1}_{i,2,b'}, r^{t+1}_{i,3,b'})_{b'\in\{0,1\}}$.

For each $t \in [h]$, we set $\widetilde{\mathsf{Boot}_i^t} = \mathsf{Boot}^i_{\mathsf{id}[1\dots t]}$. Set $p_i = (\overline{p_i^0}, (\widetilde{\mathsf{Boot}_i^t})_{t\in[n]}, \overline{\mathsf{ct}_i}')$. Output p_i.

Ideal This hybrid is almost the same as Hybrid_5.

$\mathsf{pMHE.Sim}_1(1^\lambda, (x_i)_{i\notin H})$ For each $i \in [N]$, randomly sample $k_i \leftarrow \{0,1\}^{\mathsf{PRG.in}}$.

Execute $(\mathsf{st}'_S, (\overline{\mathsf{ct}_i}')_{i\in[N]}, (\mathsf{pMHE}'.r_i)_{i\in[N]\setminus H}) \leftarrow \mathsf{pMHE'.Sim}_1(1^\lambda, H, (x_i, k_i)_{i\in[N]\setminus H})$.

Execute $(\overline{p_i^0})_{i\in H} \leftarrow \mathsf{pMHE'.Sim}_2(\mathsf{st}'_S, \mathsf{NewEnc}^1, \mathsf{NewEnc}^1((x_j,k_j)_{j\in[N]})$.

Set $\mathsf{ct}_i = \mathsf{ct}'_i$ and $\mathsf{sk}_i = (\mathsf{sk}'_i, (k_i, \perp))$. Output $(\mathsf{ct}_i, \mathsf{sk}_i)$.

Initialize the empty sets $T, T', T'' = \phi$.

For each $i \in [N]\setminus H$, randomly sample $k_i \leftarrow \{0,1\}^{\mathsf{PRG.in}}$,

and parse $\mathsf{PRG}(k_i)$ as $(\mathsf{lab}^{i,b}, k_{i,b}, r_{i,1,b}, r_{i,2,b}, r_{i,3,b})_{b\in\{0,1\}}$.

Execute $(\mathsf{st}'_S, (\overline{\mathsf{ct}_i}')_{i\in[N]}, (\mathsf{pMHE}'.r_i)_{i\in[N]\setminus H}) \leftarrow \mathsf{pMHE'.Sim}_1(1^\lambda, H, (x_i, k_i)_{i\in[N]\setminus H})$.

For each $i \in H, b \in \{0,1\}$, sample $(k_{i,b})_{b\in\{0,1\}} \leftarrow \{0,1\}^*$.

For any $b \in \{0,1\}$, execute $(\mathsf{st}'_b, \overline{\mathsf{ct}_b}, (r'_i)_{i\in[N]\setminus H}) \leftarrow \mathsf{pMHE'.Sim}_1(1^\lambda, (x_j, k_{j,b})_{j\in[N]\setminus H})$.

Update $T'' = T'' \cup \{b\}$.

For any $b \in \{0,1\}, i \in H$, execute $(\mathsf{GC.st}^i_b, \mathsf{lab}'^{i,b}) \leftarrow \mathsf{GC.Sim}_1(1^\lambda, \mathsf{KG.in})$.

Update $T' = T' \cup \{(i,b)\}$.

For any $i \in [N]\setminus H, b \in \{0,1\}$, let $\mathsf{lab}'^{i,b} = \mathsf{lab}^{i,b}_{\overline{\mathsf{ct}_b}}$.

Execute $(\overline{p_i^0})_{i\in H} \leftarrow \mathsf{pMHE'.Sim}_2(\mathsf{st}'_S, \mathsf{NewEnc}^1, ((\mathsf{lab}'^{i,0})_{i\in[N]}, (\mathsf{lab}'^{i,1})_{i\in[N]}))$.

Let $\mathsf{st}_S = ((\overline{p_i^0})_{i\in H}, \mathsf{st}'_S, T, T', T'', (k_i)_{i\in H})$.

Output $(\mathsf{st}_S, \overline{(\mathsf{ct}_i)}_{i\in[N]}, (k_i, r_i, \mathsf{pMHE}'.r_i)_{i\notin H})$.

```
$\mathsf{pMHE.Sim}_2(\mathsf{st}_S, C, C((x_i)_{i\in[N]})$ Let $\mathsf{id}$ be the binary representation of $C$.
    let $h = \max(0 \le h' \le n \mid \mathsf{id}[1\ldots h'] \in T)$.
    For each $t = h+1\ldots n$,
        Denote $\mathsf{id}' = \mathsf{id}[1\ldots t]$, $\mathsf{id}'_0 = \mathsf{id}' \circ 0$, $\mathsf{id}'_1 = \mathsf{id}' \circ 1$.
        Now we generate $(\overline{p_i^{\mathsf{id}'}})_{i\in H}$.
        If $t = n$, then execute $(\overline{p_i^{\mathsf{id}'}})_{i\in H} \leftarrow \mathsf{pMHE}'.\mathsf{Sim}_2(\mathsf{st}'_{\mathsf{id}'}, \mathsf{NewEnc}^{t+1}, C((x_i)_{i\in[N]}))$.
        If $t < n$, for each $b' \in \{0,1\}$, execute $(\mathsf{st}'_{\mathsf{id}'_{b'}}, \overline{\mathsf{ct}_{b'}}, (r_{j,b'})_{j\notin H}) \leftarrow \mathsf{pMHE}'.\mathsf{Sim}_1(1^\lambda, (x_j, k_{j,\mathsf{id}'_{b'}})_{j\notin H})$.
        Update $T'' = T'' \cup \{\mathsf{id}'_{b'}\}$.
        For each $i \in H, b' \in \{0,1\}$, execute $(\mathsf{GC.st}^i_{\mathsf{id}_b\circ b'}, \mathsf{lab}'^{i,b'}) \leftarrow \mathsf{GC.Sim}_1(1^\lambda, \mathsf{KG.in})$.
        Update $T' = T' \cup \{(i, \mathsf{id}'_{b'})\}$.
        For each $i \in [N]\setminus H, b' \in \{0,1\}$, let $\mathsf{lab}'^{i,b'} = \mathsf{lab}^{i,\mathsf{id}'_{b'}}_{\overline{\mathsf{ct}_{b'}}}$.
        Execute $(\overline{p_i^{\mathsf{id}'}})_{i\in H} \leftarrow \mathsf{pMHE}'.\mathsf{Sim}_2(\mathsf{st}'_{\mathsf{id}'}, \mathsf{NewEnc}^{t+1}, ((\mathsf{lab}'^{i,0})_{i\in[N]}, (\mathsf{lab}'^{i,1})_{i\in[N]}))$.
        For each $i \in H$, execute $\widetilde{\mathsf{Boot}_i^t} \leftarrow \mathsf{GC.Sim}_2(\mathsf{GC.st}^i_{\mathsf{id}'}, \overline{p_i^{\mathsf{id}'}})$, and set $\mathsf{Boot}^i_{\mathsf{id}'} = \widetilde{\mathsf{Boot}_i^t}$.
        Update $T'' = T'' \setminus \{\mathsf{id}'\}$, $T' = T' \setminus \{(i,\mathsf{id}')\}$, and $T = T \cup \{\mathsf{id}'\}$.
        Randomly sample $(k_{i,\mathsf{id}'_{b'}})_{i\in H, b'\in\{0,1\}} \leftarrow \{0,1\}^{\mathsf{PRG.in}}$.
    For $t \in [h]$, we set $\widetilde{\mathsf{Boot}_i^t} = \mathsf{Boot}^i_{\mathsf{id}[1\ldots t]}$. Set $p_i = (\overline{p_i^0}, (\widetilde{\mathsf{Boot}_i^t})_{t\in[n]}, \overline{\mathsf{ct}_i}')$.
    Let $\mathsf{st}_S = ((\overline{p_i^0})_{i\in H}, \mathsf{st}'_S, T, T', T'', (k_i)_{i\in H})$
    Output $(\mathsf{st}_S, (p_i)_{i\in H})$.
```

Lemma 5.5. *For any PPT adversary $\mathcal{A}$, and any PPT distinguisher $\mathcal{D}$, there exits a negligible function $\nu(\lambda)$ such that* $|\Pr[\mathcal{D}(1^\lambda, \mathsf{Hybrid}_0^{\mathcal{A}}) = 1] - \Pr[\mathcal{D}(1^\lambda, \mathsf{Hybrid}_1^{\mathcal{A}}) = 1]| < \nu(\lambda)$.

Proof. For any honest party set $H \subseteq [N]$, any input $(x_i)_{i\in[N]}$, we build the following adversary $\mathcal{A}'$ for pMHE'.

Note that the adversary $\mathcal{A}'$ takes input $(1^\lambda, (\mathsf{ct}'_i)_{i\in[N]}, ((x_i,k_i), \mathsf{pMHE}.r_i)_{i\notin H})$, and can query the oracle $\mathcal{O}(1^\lambda,\cdot)$ at most once.

```
**Adversary** $\mathcal{A}'^{\mathcal{O}}(1^\lambda, (\mathsf{ct}'_i)_{i\in[N]}, ((x_i,k_i), \mathsf{pMHE}'.r'_i)_{i\notin H})$ For any $i \in [N]$, randomly sample $k_i \leftarrow \{0,1\}^{\mathsf{PRG.in}}$.
    For any $i \notin H$, randomly sample random coins $r_i$.
    Invoke the oracle $(\overline{p_i^0})_{i\in H} \leftarrow \mathcal{O}(1^\lambda, \mathsf{NewEnc}^1)$.
    Invoke $\mathcal{A}^{\mathcal{O}_{\mathcal{A}}(1^\lambda,\cdot)}(1^\lambda, (\mathsf{ct}'_i)_{i\in[N]}, (x_i, (k_i, r_i, \mathsf{pMHE}'.r'_i))_{i\notin H})$.
    Output $\mathsf{View}_{\mathcal{A}}$.

**Oracle** $\mathcal{O}_{\mathcal{A}}(1^\lambda, C)$ For each $i \in H$, $t = 1,2,\ldots,n$, set $k_i^0 = k_i$.
        Let $b = \mathsf{id}_t$. Parse $\mathsf{PRG}(k_i^{t-1})$ as $(\mathsf{lab}^{i,t,b'}, k^t_{i,b'}, r^t_{i,1,b'}, r^t_{i,2,b'}, r^t_{i,3,b'})_{b'\in\{0,1\}}$.
        Compute $\mathsf{sk}_i^t$ from $\mathsf{pMHE}'.\mathsf{Enc}(1^\lambda, \mathsf{NewEnc}^t.\mathsf{params}, i, (x_i, k^t_{i,b}); r^t_{i,1,b})$.
        Execute $\widetilde{\mathsf{Boot}_i^t} \leftarrow \mathsf{GC.Garble}(1^\lambda, \mathsf{Boot}^t_{[\mathsf{sk}_i^t; r^t_{i,2,b}]}, \mathsf{lab}^{i,t,b}; r^t_{i,3,b})$.
    Set $p_i = (\overline{p_i^0}, (\widetilde{\mathsf{Boot}_i^t})_{t\in[n]}, \mathsf{ct}'_i)$.
    Output $(p_i)_{i\in H}$.
```

For each $i \in [N]$, let k_i be uniform random string over $\{0,1\}^{\mathsf{PRG.in}}$.

If $((\mathsf{ct}'_i)_{i\in[N]}, ((x_i,k_i), \mathsf{pMHE}'.r'_i)_{i\notin H})$ is generated as in Real, then the adversary $\mathcal{A}'$ simulates the environment of Hybrid_0 for $\mathcal{A}$. Hence, $\Pr[\mathcal{D}(1^\lambda, \mathsf{Hybrid}_0^{\mathcal{A}}) = 1] = \Pr[\mathcal{D}(1^\lambda, \mathsf{Real}^{\mathcal{A}'}) = 1]$.

If $((\mathsf{ct}'_i)_{i\in[N]}, ((x_i,k_i), \mathsf{pMHE}'.r'_i)_{i\notin H})$ is generated as in Ideal, then the adversary $\mathcal{A}'$ simulates the environment of Hybrid_1 for $\mathcal{A}$. Hence, $\Pr[\mathcal{D}(1^\lambda, \mathsf{Hybrid}_1^{\mathcal{A}}) = 1] = \Pr[\mathcal{D}(1^\lambda, \mathsf{Ideal}^{\mathcal{A}'}) = 1]$.

Since pMHE' is one-time secure, there exits a negligible function $\nu(\lambda)$ such that $|\Pr[\mathcal{D}(1^\lambda, \mathsf{Real}^{\mathcal{A}'}) = 1] - \Pr[\mathcal{D}(1^\lambda, \mathsf{Ideal}^{\mathcal{A}'}) = 1]| < \nu(\lambda)$. Hence, we have $|\Pr[\mathcal{D}(1^\lambda, \mathsf{Hybrid}_0^{\mathcal{A}}) = 1] - \Pr[\mathcal{D}(1^\lambda, \mathsf{Hybrid}_1^{\mathcal{A}}) = 1]| < \nu(\lambda)$. $\square$

Note that Hybrid_1 and Hybrid_2 are essentially identical, since the only change is the expanding of $\mathsf{NewEnc}^1((x_i,k_i)_{i\in[N]})$ in Hybrid_2.

Lemma 5.6. *$\mathsf{Hybrid}_1, \mathsf{Hybrid}_2$ and $\mathsf{Hybrid}_{2.5}^1$ are identical. $\mathsf{Hybrid}_{2.5}^{N+1}$ is identical to Hybrid_3. Morever, for any $i^* \in [N]$, and any PPT adversary $\mathcal{A}$, and PPT distinguisher $\mathcal{D}$, there exits a negligible function $\nu(\lambda)$ such that $|\Pr[\mathcal{D}(1^\lambda, \mathsf{Hybrid}_{2.5}^{i^*}) = 1] - \Pr[\mathcal{D}(1^\lambda, \mathsf{Hybrid}_{2.5}^{i^*+1}) = 1]| < \nu(\lambda)$.*

Proof. Hybrid_2 is obtained by replacing the output of NewEnc in Hybrid_1. Hence, Hybrid_1 and Hybrid_2 are identical. When $i^* = 1$, $(\mathsf{lab}^{i,b}, k_{i,b}, r_{i,1,b}, r_{i,2,b}, r_{i,3,b})$ are generated by PRG for all $i \in [N]$. Hence, $\mathsf{Hybrid}_{2.5}^1$ is identical to Hybrid_2. When $i^* = N+1$, for every i, $(\mathsf{lab}^{i,b}, k_{i,b}, r_{i,1,b}, r_{i,2,b}, r_{i,3,b})_{b\in\{0,1\}}$ is generated randomly. Hence, $\mathsf{Hybrid}_{2.5}^{N+1}$ is identical to Hybrid_3.

Note that the only difference between $\mathsf{Hybrid}_{2.5}^{i^*}$ and $\mathsf{Hybrid}_{2.5}^{i^*+1}$ is that, in $\mathsf{Hybrid}_{2.5}^{i^*}$, $(\mathsf{lab}^{i^*,b}, k_{i^*,b}, r_{i^*,1,b}, r_{i^*,2,b}, r_{i^*,3,b})_{b\in\{0,1\}}$ is generated by PRG, while in $\mathsf{Hybrid}_{2.5}^{i^*+1}$, $(\mathsf{lab}^{i^*,b}, k_{i^*,b}, r_{i^*,1,b}, r_{i^*,2,b}, r_{i^*,3,b})_{b\in\{0,1\}}$ is generated randomly.

Now, for any adversary $\mathcal{A}$ for pMHE, we build the following distinguisher $\mathcal{D}'$ for the PRG.

Distinguisher $\mathcal{D}'(1^\lambda, v \in \{0,1\}^{\mathsf{PRG.out}})$ For each $i \in [N] \setminus H$, randomly sample $k_i \leftarrow \{0,1\}^{\mathsf{PRG.in}}$, and parse $\mathsf{PRG}(k_i)$ as $(\mathsf{lab}^{i,b}, k_{i,b}, r_{i,1,b}, r_{i,2,b}, r_{i,3,b})_{b\in\{0,1\}}$.

For each $i \in H$, if $i < i^*$, then randomly sample $(\mathsf{lab}^{i,b}, k_{i,b}, r_{i,1,b}, r_{i,2,b}, r_{i,3,b})_{b\in\{0,1\}}$.

If $i = i^*$, then parse v as $(\mathsf{lab}^{i,b}, k_{i,b}, r_{i,1,b}, r_{i,2,b}, r_{i,3,b})_{b\in\{0,1\}}$.

If $i > i^*$, then parse $\mathsf{PRG}(k_i)$ as $(\mathsf{lab}^{i,b}, k_{i,b}, r_{i,1,b}, r_{i,2,b}, r_{i,3,b})_{b\in\{0,1\}}$.

For each $i \in [N]$, if $i \notin H$, sample $r_i \leftarrow \{0,1\}^*$, otherwise, let $r_i = \perp$.

Execute $(\mathsf{st}'_S, (\overline{\mathsf{ct}_i}')_{i\in[N]}, (\mathsf{pMHE}'.r_i)_{i\in[N]\setminus H}) \leftarrow \mathsf{pMHE}'.\mathsf{Sim}_1(1^\lambda, H, (x_i,k_i)_{i\in[N]\setminus H})$.

For any $j \in [N], b \in \{0,1\}$, execute

$(\mathsf{ct}_{j,b}, \mathsf{sk}_{j,b}) = \mathsf{pMHE}'.\mathsf{Enc}(1^\lambda, \mathsf{NewEnc}^2.\mathsf{params}, j, (x_j, k_{j,b}); r_{j,1,b})$.

For any $b \in \{0,1\}$, let $\mathsf{ct}_b = (\mathsf{ct}_{j,b})_{j\in[N]}$.

Execute $(\overline{p_i^0})_{i\in H} \leftarrow \mathsf{pMHE}'.\mathsf{Sim}_2(\mathsf{st}'_S, \mathsf{NewEnc}^1, ((\mathsf{lab}^{i,0}_{\mathsf{ct}_0})_{i\in[N]}, (\mathsf{lab}^{i,1}_{\mathsf{ct}_1})_{i\in[N]}))$.

Execute $\mathcal{A}^{\mathcal{O}_\mathcal{A}}(1^\lambda, (\overline{\mathsf{ct}_i}')_{i\in[N]}, (x_i, k_i, r_i, \mathsf{pMHE}'.r_i)_{i\notin H})$, where the oracle $\mathcal{O}_\mathcal{A}$ is specified below.

Let $b \leftarrow \mathcal{D}(1^\lambda, \mathsf{View}_\mathcal{A})$.

Output b.

Oracle $\mathcal{O}_\mathcal{A}(1^\lambda, C)$ For each $i \in H$, execute $p_i \leftarrow \mathsf{pMHE.PrivEval}(1^\lambda, \perp, C, i, (\overline{\mathsf{ct}_j}')_{j\in[N]})$ where the function $\mathsf{pMHE.PrivEval}$ is the same as Hybrid_2.

Output $(p_i)_{i\in H}$.

When $v = \mathsf{PRG}(s)$, where $s \leftarrow \{0,1\}^{\mathsf{PRG.in}}$, the distinguisher simulates the $\mathsf{Hybrid}_{2.5}^{i^*}$ for $\mathcal{A}$. Hence, $\Pr[s \leftarrow \{0,1\}^{\mathsf{PRG.in}} : \mathcal{D}'(1^\lambda, \mathsf{PRG}(s)) = 1] = \Pr[\mathcal{D}(1^\lambda, \mathsf{Hybrid}_{2.5}^{i^*}) = 1]$.

When v is uniform random, the distinguisher simulates the $\mathsf{Hybrid}_{2.5}^{i^*+1}$ for $\mathcal{A}$. Hence, $\Pr[v \leftarrow \{0,1\}^{\mathsf{PRG.out}} : \mathcal{D}'(1^\lambda, v) = 1] = \Pr[\mathcal{D}(1^\lambda, \mathsf{Hybrid}_{2.5}^{i^*+1}) = 1]$.

From the security of the PRG, we derive that there exits a negligible function $\nu(\lambda)$ such that $|\Pr[\mathcal{D}(1^\lambda, \mathsf{Hybrid}_{2.5}^{i^*}) = 1] - \Pr[\mathcal{D}(1^\lambda, \mathsf{Hybrid}_{2.5}^{i^*+1}) = 1]| < \nu(\lambda)$. □

Lemma 5.7. Hybrid_3 *is identical to* $\mathsf{Hybrid}_{3.5}^{(1,1)}$. $\mathsf{Hybrid}_{3.5}^{(N,1)+1}$ *is identical to* Hybrid_4. *Moreover, for each* $(i^*, b^*) \in [N] \times \{0,1\}$*, any PPT adversary* $\mathcal{A}$*, any PPT distinguisher* $\mathcal{D}$*, there exits a negligible function* $\nu(\lambda)$ *such that* $|\Pr[\mathcal{D}(1^\lambda, \mathsf{Hybrid}_{3.5}^{(i^*,b^*)}) = 1] - \Pr[\mathcal{D}(1^\lambda, \mathsf{Hybrid}_{3.5}^{(i^*,b^*)+1}) = 1]| < \nu(\lambda)$.

Proof. The main difference between $\mathsf{Hybrid}_{3.5}^{(i^*,b^*)}$ and $\mathsf{Hybrid}_{3.5}^{(i^*,b^*)+1}$ is $(\mathsf{lab}'^{i^*,b^*}, \widetilde{\mathsf{Boot}_i^1})$. We obtain it from $\mathsf{GC.Garble}$ in $\mathsf{Hybrid}_{3.5}^{(i^*,b^*)}$, and obtain it from $\mathsf{GC.Sim}_2$ in $\mathsf{Hybrid}_{3.5}^{(i^*,b^*)+1}$.

Now we build an adversary $\mathcal{A}' = (\mathcal{A}'_1, \mathcal{A}'_2)$ breaking the garbling scheme for the input ct_{b^*} and the circuit $\mathsf{Boot}^t_{[\mathsf{sk}_i^t; r_{i,2,b}^t]}$.

$\mathcal{A}'(1^\lambda, \mathsf{lab})$ Initialize an empty set $T' = \phi$.

For each $i \in [N] \setminus H$, randomly sample $k_i \leftarrow \{0,1\}^{\mathsf{PRG.in}}$,

and parse $\mathsf{PRG}(k_i)$ as $(\mathsf{lab}^{i,b}, k_{i,b}, r_{i,1,b}, r_{i,2,b}, r_{i,3,b})_{b \in \{0,1\}}$.

Execute $(\mathsf{st}'_S, (\overline{\mathsf{ct}_i}')_{i \in [N]}, (\mathsf{pMHE}'.r_i)_{i \in [N] \setminus H}) \leftarrow \mathsf{pMHE}'.\mathsf{Sim}_1(1^\lambda, H, (x_i, k_i)_{i \in [N] \setminus H})$.

For each $i \in H$, if $(i,b) < (i^*, b^*)$, randomly sample $(k_{i,b}, r_{i,1,b}, r_{i,2,b})$.

Otherwise, randomly sample $(\mathsf{lab}^{i,b}, k_{i,b}, r_{i,1,b}, r_{i,2,b}, r_{i,3,b})$.

For each $i \in [N]$, if $i \notin H$, sample random coins r_i, otherwise, let $r_i = \perp$.

For any $j \in [N], b \in \{0,1\}$, execute

$(\mathsf{ct}_{j,b}, \mathsf{sk}_{j,b}) = \mathsf{pMHE}'.\mathsf{Enc}(1^\lambda, \mathsf{NewEnc}^2.\mathsf{params}, j, (x_j, k_{j,b}); r_{j,1,b})$.

For any $b \in \{0,1\}$, let $\mathsf{ct}_b = (\mathsf{ct}_{j,b})_{j \in [N]}$.

For any $b \in \{0,1\}, i \in H$, if $(i,b) < (i^*, b^*)$, execute $(\mathsf{GC.st}_b^i, \mathsf{lab}'^{i,b}) \leftarrow \mathsf{GC.Sim}_1(1^\lambda, \mathsf{KG.in})$, Update $T' = T' \cup \{(i,b)\}$.

<u>If $(i,b) = (i^*, b^*)$, let $\mathsf{lab}'^{i,b} = \mathsf{lab}$.</u>

<u>If $(i,b) > (i^*, b^*)$, execute $\mathsf{lab}'^{i,b} = \mathsf{lab}^{i,b}_{\mathsf{ct}_b}$.</u>

For any $i \in [N] \setminus H, b \in \{0,1\}$, let $\mathsf{lab}'^{i,b} = \mathsf{lab}^{i,b}_{\mathsf{ct}_b}$.

Execute $(\mathsf{st}''_S, (\overline{p_i^0})_{i \in H}) \leftarrow \mathsf{pMHE}'.\mathsf{Sim}_2(\mathsf{st}'_S, \mathsf{NewEnc}^1, ((\mathsf{lab}'^{i,0})_{i \in [N]}, (\mathsf{lab}'^{i,1})_{i \in [N]}))$.

Invoke $\mathcal{A}^{\mathcal{O}(1^\lambda, \cdot)}(1^\lambda, (\overline{\mathsf{ct}_i}')_{i \in [N]}, (x_j, k_j, r_j, \mathsf{pMHE}'.r_j)_{j \notin H})$.

Output $\mathcal{D}(1^\lambda, \mathsf{View}_{\mathcal{A}})$.

$\mathsf{pMHE.PrivEval}(1^\lambda, \mathsf{sk}_i, C, i, (\overline{\mathsf{ct}_j}')_{j \in [N]})$ Let $b = \mathsf{id}[1]$.

Set $(k_{i,b'}^1, r_{i,1,b'}^1, r_{i,2,b'}^1)_{b' \in \{0,1\}}$ to be $(k_{i,b'}, r_{i,1,b'}, r_{i,2,b'})_{b' \in \{0,1\}}$.

Compute sk_i^1 from $\mathsf{pMHE}'.\mathsf{Enc}(1^\lambda, \mathsf{NewEnc}^1.\mathsf{params}, i, (x_i, k_{i,b}^1); r_{i,1,b}^1)$.

If $(i,b) < (i^*, b^*)$, and $(i,b) \notin T'$, then let $\widetilde{\mathsf{Boot}_i^1} = \mathsf{Boot}_b^i$.

If $(i,b) < (i^*, b^*)$, and $(i,b) \in T'$, then execute

$p_i^1 = \mathsf{pMHE}'.\mathsf{PrivEval}(1^\lambda, \mathsf{sk}_i^1, \mathsf{NewEnc}^2, \mathsf{ct}_b; r_{i,2,b}^1)$,

$\widetilde{\mathsf{Boot}_i^1} \leftarrow \mathsf{GC}.\mathsf{Sim}_2(\mathsf{GC}.\mathsf{st}_b^i, p_i^1)$, and define $\mathsf{Boot}_b^i = \widetilde{\mathsf{Boot}_i^1}$, and update $T' = T' \setminus \{(i,b)\}$.

$\underline{\text{If } (i,b) = (i^*, b^*), \text{ and } (i,b) \notin T', \text{ let } \widetilde{\mathsf{Boot}_i^1} = \mathsf{Boot}_b^i}$.

$\underline{\text{If } (i,b) = (i^*, b^*), \text{ and } (i,b) \in T',}$

$\underline{\text{query the challenger of } \mathcal{A}' \text{ with circuit } \mathsf{Boot}^1_{[\mathsf{sk}_i^1; r_{i,2,b}^1]} \text{ to obtain } \widetilde{\mathsf{Boot}_i^1}, \text{ and set } \mathsf{Boot}_b^i = \widetilde{\mathsf{Boot}_i^1}.}$

$\underline{\text{If } (i,b) > (i^*, b^*)}$, then execute $\widetilde{\mathsf{Boot}_i^1} = \mathsf{GC}.\mathsf{Garble}(1^\lambda, \mathsf{Boot}^1_{[\mathsf{sk}_i^1; r_{i,2,b}^1]}; r_{i,3,b}^1)$.

Parse $\mathsf{PRG}(k_{i,b}^1)$ as $(\mathsf{lab}^{i,2,b'}, k_{i,b'}^2, r_{i,1,b'}^2, r_{i,2,b'}^2, r_{i,3,b'}^2)_{b' \in \{0,1\}}$.

For $t = 2 \dots n$, let $b = \mathsf{id}[t]$.

  Compute sk_i^t from $\mathsf{pMHE}'.\mathsf{Enc}(1^\lambda, \mathsf{NewEnc}^t.\mathsf{params}, i, (x_i, k_{i,b}^t); r_{i,1,b}^t)$.

  Then execute $\widetilde{\mathsf{Boot}_i^t} \leftarrow \mathsf{GC}.\mathsf{Garble}(1^\lambda, \mathsf{Boot}^t_{[\mathsf{sk}_i^t; r_{i,2,b}^t]}, \mathsf{lab}^{i,t,b}; r_{i,3,b}^t)$.

  Parse $\mathsf{PRG}(k_{i,b}^t)$ as $(\mathsf{lab}^{i,t+1,b'}, k_{i,b'}^{t+1}, r_{i,1,b'}^{t+1}, r_{i,2,b'}^{t+1}, r_{i,3,b'}^{t+1})_{b' \in \{0,1\}}$.

Set $p_i = (\overline{p_i^0}, (\widetilde{\mathsf{Boot}_i^t})_{t \in [n]}, \overline{\mathsf{ct}_i}')$. Output p_i.

If $(\mathsf{lab}_{\mathsf{ct}_{b^*}}, \widetilde{\mathsf{Boot}_{i^*}^1})$ is obtained from the real execution Real, then the adversary $\mathcal{A}'$ simulates the environment of $\mathsf{Hybrid}_{3.5}^{(i^*,b^*)}$ for $\mathcal{A}$. Hence, $\Pr[\mathsf{Real}^{\mathcal{A}'} = 1] = \Pr[\mathcal{D}(1^\lambda, \mathsf{Hybrid}_{3.5}^{i^*,b^*}) = 1]$.

If $(\mathsf{lab}_{\mathsf{ct}_{b^*}}, \widetilde{\mathsf{Boot}_{i^*}^1})$ is obtained from the ideal execution Ideal, then the adversary $\mathcal{A}'$ simulates the environment of $\mathsf{Hybrid}_{3.5}^{(i^*,b^*)+1}$ for $\mathcal{A}$. Hence, $\Pr[\mathsf{Ideal}^{\mathcal{A}'} = 1] = \Pr[\mathcal{D}(1^\lambda, \mathsf{Hybrid}_{3.5}^{(i^*,b^*)+1}) = 1]$.

Since we the garbling scheme is selective secure, there exits a negligible function $\nu(\lambda)$ such that $|\Pr[\mathsf{Real}^{\mathcal{A}'} = 1] - \Pr[\mathsf{Ideal}^{\mathcal{A}'} = 1]| < \nu(\lambda)$. Hence, we have $|\Pr[\mathcal{D}(1^\lambda, \mathsf{Hybrid}_{3.5}^{(i^*,b^*)}) = 1] - \Pr[\mathcal{D}(1^\lambda, \mathsf{Hybrid}_{3.5}^{(i^*,b^*)+1}) = 1]| < \nu(\lambda)$. □

Lemma 5.8. *For any PPT adversary $\mathcal{A}$, and any PPT distinguisher $\mathcal{D}$, there exists a negligible function $\nu(\lambda)$ such that $|\Pr[\mathcal{D}(1^\lambda, \mathsf{Hybrid}_4) = 1] - \Pr[\mathcal{D}(1^\lambda, \mathsf{Hybrid}_5) = 1]| < \nu(\lambda)$.*

Proof. The proof follows the same idea from Lemma 5.5. □

Lemma 5.9. Hybrid_5 *is identical to* Hybrid_6^1. Hybrid_6^{Q+1} *is identical to* Ideal. *Moreover, for any $q^* \in [Q]$, any PPT adversary $\mathcal{A}$, and any PPT distinguisher $\mathcal{D}$, there exists a negligible function $\nu(\lambda)$ such that $|\Pr[\mathcal{D}(1^\lambda, \mathsf{Hybrid}_6^{q^*}) = 1] - \Pr[\mathcal{D}(1^\lambda, \mathsf{Hybrid}_6^{q^*+1}) = 1]| < \nu(\lambda)$.*

Proof. We prove that for any (q^*, h^*), there exits a negligible function $\nu(\lambda)$ such that $|\Pr[\mathcal{D}(1^\lambda, \mathsf{Hybrid}_7^{(q^*,h^*)}) = 1] - \Pr[\mathcal{D}(1^\lambda, \mathsf{Hybrid}_7^{(q^*,h^*)+1}) = 1]| < \nu(\lambda)$.

The proof follows the same strategy as Lemma 5.5, and Lemma 5.6. □

Proof of Lemma 5.4. Combining Lemma 5.5, Lemma 5.6, Lemma 5.8, and Lemma 5.9, we finish the proof. □

Lemma 5.10 (Efficiency)**.** *If the underlying pMHE* pMHE' *is strong ciphertext succinct, then the construction of* pMHE *runs in polynomial time and is weak ciphertext succinct.*

Proof. Since the scheme pMHE' is strong ciphertext succinct, for each ct_i, we have that the running time of pMHE' is $\mathsf{poly}(\lambda, N, C.\mathsf{in}, C.\mathsf{out}, C.\mathsf{depth})$, and the depth of $\mathsf{pMHE}'.\mathsf{Enc}$ is $\mathsf{poly}(\lambda, \log N, \log C.\mathsf{in}, \log C.\mathsf{out}, \log C.\mathsf{depth})$.

Now, we check each requirement for weak ciphertext succinctness.

The running time of $\mathsf{pMHE.Enc}$ is $\mathsf{poly}(\lambda, N, \mathsf{NewEnc}^1.\mathsf{in}, \mathsf{NewEnc}^1.\mathsf{out}, \mathsf{NewEnc}^1.\mathsf{depth})$.

For $\mathsf{NewEnc}^t.\mathsf{in}$, we have $\mathsf{NewEnc}^t.\mathsf{in} = \mathsf{poly}(\lambda, N, C.\mathsf{in})$. For $\mathsf{NewEnc}^t.\mathsf{out}$, we have $\mathsf{NewEnc}^t.\mathsf{out} = \mathsf{poly}(\lambda, N, C.\mathsf{in}, C.\mathsf{out}, C.\mathsf{depth})$. Now we only need to bound $\mathsf{NewEnc}^1.\mathsf{depth}$.

For any $t \in [n]$, we have

$$\begin{aligned}\mathsf{NewEnc}^t.\mathsf{depth} =& \mathsf{PRG.depth} + \mathsf{pMHE}'.\mathsf{Enc.depth} + O(1)\\ =& \mathsf{poly}(\lambda, \log N) + \mathsf{poly}(\lambda, \log N, \log \mathsf{NewEnc}^{t+1}.\mathsf{in}, \log \mathsf{NewEnc}^{t+1}.\mathsf{out}, \log \mathsf{NewEnc}^{t+1}.\mathsf{depth})\\ =& \mathsf{poly}(\lambda, \log N, \log \mathsf{NewEnc}^{t+1}.\mathsf{in}, \log \mathsf{NewEnc}^{t+1}.\mathsf{out}, \log \mathsf{NewEnc}^{t+1}.\mathsf{depth})\\ =& \mathsf{poly}(\lambda, \log N, \log C.\mathsf{in}, \log C.\mathsf{out}, \log C.\mathsf{depth}, \log \mathsf{NewEnc}^{t+1}.\mathsf{depth})\end{aligned}$$

For $t = n+1$, we have $\mathsf{NewEnc}^t.\mathsf{in} = C.\mathsf{in}, \mathsf{NewEnc}^t.\mathsf{out} = C.\mathsf{out}, \mathsf{NewEnc}^t.\mathsf{depth} = C.\mathsf{depth}$.

Claim 5.11. *There exits two constants c' and c_0 such that for any $\lambda > c_0$, for all $t \in [n]$, $\mathsf{NewEnc}^t.\mathsf{depth} < \lambda^{c'} \cdot N^{c'} \cdot (C.\mathsf{in})^{c'} \cdot (C.\mathsf{out})^{c'} \cdot (C.\mathsf{depth})^{c'}$.*

Proof. There exits a $c \geq 1$ such that $\mathsf{NewEnc}^t.\mathsf{depth} < \lambda^c \cdot \log^c N \cdot \log^c(C.\mathsf{in}) \cdot \log^c(C.\mathsf{out}) \cdot \log^c(C.\mathsf{depth}) \cdot \log^c(\mathsf{NewEnc}^{t+1}.\mathsf{depth})$.

Set $c' = c+1$. We prove the claim by induction on t. For $t = n+1$, as $c' > 1$, the theorem clearly holds.

Now we assume the claim holds for $t = t^* + 1$, we prove the claim for $t = t^*$. By the induction assumption, we have

$$\begin{aligned}\mathsf{NewEnc}^{t^*}.\mathsf{depth} &< \lambda^c \cdot \log^c N \cdot \log^c(C.\mathsf{in}) \cdot \log^c(C.\mathsf{out}) \cdot \log^c(C.\mathsf{depth}) \cdot \log^c(\mathsf{NewEnc}^{t^*+1}.\mathsf{depth})\\ &< \lambda^c \cdot N^c \cdot (C.\mathsf{in})^c \cdot (C.\mathsf{out})^c \cdot (C.\mathsf{depth})^c \cdot \log^c(\mathsf{NewEnc}^{t+1}.\mathsf{depth})\\ &< \lambda^c \cdot N^c \cdot (C.\mathsf{in})^c \cdot (C.\mathsf{out})^c \cdot (C.\mathsf{depth})^c \cdot (c+1)^c \cdot \log^c(\lambda \cdot N \cdot C.\mathsf{in} \cdot C.\mathsf{out} \cdot C.\mathsf{depth})\\ &< \lambda^{c'} \cdot N^{c'} \cdot (C.\mathsf{in})^{c'} \cdot (C.\mathsf{out})^{c'} \cdot (C.\mathsf{depth})^{c'}\end{aligned}$$

The last equality holds if $\lambda > c_0$, for some $c_0 > 0$. Thus, the claim holds for $t = t^*$. By induction, the claim holds for all $t \in [n]$. □

By the claim, we derive that the running time of $\mathsf{pMHE.Enc}$ is a polynomial of $\lambda, N, C.\mathsf{in}, C.\mathsf{out}, C.\mathsf{depth}$. □

Reusability Compiler for MPC. Similar to above, we can obtain an analogous reusability compiler for MPC in the semi-honest setting. A reusable MPC is a delayed-function two-round MPC protocol where the first round messages of the MPC protocol can be reused for secure computation of multiple functionalities.

Theorem 5.12 (Semi-Honest Reusability Compiler)**.** *Let Π be a two-round (non-reusable) delayed-function semi-honest MPC protocol for f with the following property:*

- *The time complexity to compute the first round message is polynomial in λ, input, output length of f.*
- *The output length of the first round message is a fixed polynomial in λ and the number of parties.*

Then there exists a two-round delayed-function semi-honest MPC protocol, where the first round messages of the parties are reusable.

The proof of the above theorem is essentially the same as Lemma 5.1. For completeness sake, we give an overview here. The idea is to allow the parties to recursively generate two fresh first round messages of Π, and use one leaf of the recursive tree to do the actual multi-party computation. To realize this idea, we need to use garbled circuit to 'delay' the computation to the second round.

We sketch the construction of the reusable MPC. Denote the input to the i-th party as x_i.

First Round Each party generates a first round message of Π, with input $(x_i)_{i\in[N]}$.

Second Round The second round message consists of a second round message p_i of Π, and several garbled circuits $(\widetilde{C_{i,j}})_{j\in[|C|]}$, where each $\widetilde{C_{i,j}}$ is a garbling of the circuit $C_{i,j}$. The circuit $C_{i,j}$ takes as input the first round messages of all parties, computes the partial decryption of a function f_j. The function f_j is specified as follows: it takes $(x_i)_{i\in[N]}$ as input, computes *two* fresh first round messages of the input $(x_i)_{i\in[N]}$. Then output the labels of the fresh first round messages, where the labels is used to garble the circuit $(C_{i,j+1})_{i\in[N]}$. Note that all the randomness used is obtained by applying a PRG on a random seed.

Publicly Recover Output Given the first and the second round message, use the garbled circuits to compute the second round messages of Π, and finally use the leaf node to do the computation, and thus obtains $f(x_1, \dots, x_n)$.

The proofs of correctness, simulation security, and efficiency are essentially the same as Lemma 5.2, Lemma 5.4, and Lemma 5.10. In a subsequent work, we build on the above theorem to construct semi-malicious reusable MPC from LWE [41].

6 Main Result: (Reusable) MHE

We now show how to achieve our main result of (reusable) multiparty homomorphic encryption scheme.

From delayed-function 2-round MPC to one-time pMHE with weak ciphertext succincntess. We first transform any delayed-function two-round secure MPC protocol into a pMHE scheme satisfying weak ciphertext succinctness property. The following was proven in Section 4.2.

Lemma 6.1. *Assuming the existence of laconic function evaluation, there exists a generic transformation from any non-succinct pMHE scheme into a succinct pMHE scheme. Moreover, the transformation preserves the number of decryption queries made by the adversary.*

The work of [42] presented a construction of laconic function evaluation assuming the hardness of learning with errors. Moreover, as shown in Corollary 4.5, a delayed-function two-round secure MPC yields a non-succinct one-time pMHE scheme. Thus, we have the following corollary.

Corollary 6.2. *Assuming the hardness of learning with errors, there exists a generic transformation from any delayed-function two-round secure MPC protocol into a one-time succinct pMHE scheme; and in particular, the resulting scheme satisfies weak ciphertext succinctness property.*

From weak to strong ciphertext succinctness. We then show how to transform a one-time pMHE scheme satisfying weak ciphertext succinctness into a scheme that satisfies strong ciphertext succinctness property. The following was proven in Section 4.3.

Lemma 6.3. *Assuming the existence of laconic oblivious transfer and randomized encodings computable in* NC^1*, there exists a generic transformation from any one-time pMHE scheme satisfying weak ciphertext succinctness property into a one-time pMHE scheme satisfying strong ciphertext succinctness property.*

Laconic oblivious transfer can be based on the hardness of learning with errors [22, 18, 27]. Randomized encodings in NC^1 can also be based on the hardness of learning with errors [8, 9]. Thus, we have the following corollary.

Corollary 6.4. *Assuming the hardness of learning with errors, there exists a generic transformation from any one-time pMHE scheme satisfying weak ciphertext succinctness property into a one-time pMHE scheme satisfying strong ciphertext succinctness property.*

From one-time pMHE with strong ciphertext succinctness to (reusable) non-succinct pMHE. Next, we show how to construct a (reusable) pMHE scheme from one-time pMHE satisfying strong ciphertext succinctness property. However, the resulting reusable MHE scheme is non-succinct; i.e., it does not satisfy Definition 3.2. The following was proven in Section 5.

Lemma 6.5. *Assuming one-way functions, there exists a generic transformation from any one-time pMHE scheme satisfying strong ciphertext succinctness property into a non-succinct reusable pMHE scheme.*

From non-succinct (reusable) pMHE to succinct (reusable) pMHE. We next show how to construct a succinct (reusable) pMHE scheme from a non-succinct (reusable) pMHE scheme. We invoke Lemma 6.1 once more to obtain the following lemma.

Lemma 6.6. *Assuming the hardness of learning with errors, there exists a generic transformation from a (reusable) non-succinct pMHE scheme into a (reusable) succinct pMHE scheme.*

From (reusable) pMHE to (reusable) MHE. We show how to construct a (reusable) MHE scheme from a (reusable) pMHE scheme. This was proven in Section 3.4.

Lemma 6.7. *Assuming the existence of leveled fully homomorphic encryption, there is a generic transformation from a (reusable) pMHE scheme into a (reusable) MHE scheme.*

Since level fully homomorphic encryption can be instantiated from the hardness of learning with errors [20], we have the following corollary.

Corollary 6.8. *Assuming the hardness of learning with errors, there is a generic transformation from a (reusable) pMHE scheme into a (reusable) MHE scheme.*

Main Theorem. Combining corollaries 6.2, 6.4, 6.8 and lemmas 6.5, 6.6 and the fact that we can achieve a delayed-function two-round semi-honest secure MPC from hardness of learning with errors [28, 10, 40], we have the following main theorem.

Theorem 6.9 (Main Theorem)**.** *Assuming the hardness of learning with errors, there exists a multiparty homomorphic encryption scheme.*

References

[1] Agrawal, S., Clear, M., Frieder, O., Garg, S., O'Neill, A., Thaler, J.: Ad hoc multi-input functional encryption. In: Vidick, T. (ed.) ITCS 2020. vol. 151, pp. 40:1–40:41. LIPIcs, Seattle, WA, USA (Jan 12–14, 2020). https://doi.org/10.4230/LIPIcs.ITCS.2020.40

[2] Ananth, P., Badrinarayanan, S., Jain, A., Manohar, N., Sahai, A.: From FE combiners to secure MPC and back. In: Hofheinz, D., Rosen, A. (eds.) TCC 2019, Part I. LNCS, vol. 11891, pp. 199–228. Springer, Heidelberg, Germany, Nuremberg, Germany (Dec 1–5, 2019). https://doi.org/10.1007/978-3-030-36030-6_9

[3] Ananth, P., Choudhuri, A.R., Goel, A., Jain, A.: Round-optimal secure multiparty computation with honest majority. In: Shacham, H., Boldyreva, A. (eds.) CRYPTO 2018, Part II. LNCS, vol. 10992, pp. 395–424. Springer, Heidelberg, Germany, Santa Barbara, CA, USA (Aug 19–23, 2018). https://doi.org/10.1007/978-3-319-96881-0_14

[4] Ananth, P., Choudhuri, A.R., Goel, A., Jain, A.: Two round information-theoretic MPC with malicious security. In: Ishai, Y., Rijmen, V. (eds.) EUROCRYPT 2019, Part II. LNCS, vol. 11477, pp. 532–561. Springer, Heidelberg, Germany, Darmstadt, Germany (May 19–23, 2019). https://doi.org/10.1007/978-3-030-17656-3_19

[5] Ananth, P., Jain, A., Naor, M., Sahai, A., Yogev, E.: Universal constructions and robust combiners for indistinguishability obfuscation and witness encryption. In: Robshaw, M., Katz, J. (eds.) CRYPTO 2016, Part II. LNCS, vol. 9815, pp. 491–520. Springer, Heidelberg, Germany, Santa Barbara, CA, USA (Aug 14–18, 2016). https://doi.org/10.1007/978-3-662-53008-5_17

[6] Ananth, P., Jain, A., Sahai, A.: Robust transforming combiners from indistinguishability obfuscation to functional encryption. In: Coron, J., Nielsen, J.B. (eds.) EUROCRYPT 2017, Part I. LNCS, vol. 10210, pp. 91–121. Springer, Heidelberg, Germany, Paris, France (Apr 30 – May 4, 2017). https://doi.org/10.1007/978-3-319-56620-7_4

[7] Ananth, P., Jain, A.: Indistinguishability obfuscation from compact functional encryption. In: Gennaro, R., Robshaw, M.J.B. (eds.) CRYPTO 2015, Part I. LNCS, vol. 9215, pp. 308–326. Springer, Heidelberg, Germany, Santa Barbara, CA, USA (Aug 16–20, 2015). https://doi.org/10.1007/978-3-662-47989-6_15

[8] Applebaum, B., Ishai, Y., Kushilevitz, E.: Computationally private randomizing polynomials and their applications. computational complexity **15**(2), 115–162 (2006)

[9] Banerjee, A., Peikert, C., Rosen, A.: Pseudorandom functions and lattices. In: Pointcheval, D., Johansson, T. (eds.) EUROCRYPT 2012. LNCS, vol. 7237, pp. 719–737. Springer, Heidelberg, Germany, Cambridge, UK (Apr 15–19, 2012). https://doi.org/10.1007/978-3-642-29011-4_42

[10] Benhamouda, F., Lin, H.: k-round multiparty computation from k-round oblivious transfer via garbled interactive circuits. In: Nielsen, J.B., Rijmen, V. (eds.) EUROCRYPT 2018, Part II. LNCS, vol. 10821, pp. 500–532. Springer, Heidelberg, Germany, Tel Aviv, Israel (Apr 29 – May 3, 2018). https://doi.org/10.1007/978-3-319-78375-8_17

[11] Benhamouda, F., Lin, H.: Multiparty reusable non-interactive secure computation. Cryptology ePrint Archive, Report 2020/221 (2020), https://eprint.iacr.org/2020/221

[12] Bitansky, N., Vaikuntanathan, V.: Indistinguishability obfuscation from functional encryption. In: Guruswami, V. (ed.) 56th FOCS. pp. 171–190. IEEE Computer Society Press, Berkeley, CA, USA (Oct 17–20, 2015). https://doi.org/10.1109/FOCS.2015.20

[13] Bitansky, N., Vaikuntanathan, V.: Indistinguishability obfuscation from functional encryption. Journal of the ACM (JACM) **65**(6), 39 (2018)

[14] Boyle, E., Gilboa, N., Ishai, Y.: Breaking the circuit size barrier for secure computation under DDH. In: Robshaw, M., Katz, J. (eds.) CRYPTO 2016, Part I. LNCS, vol. 9814, pp. 509–539. Springer, Heidelberg, Germany, Santa Barbara, CA, USA (Aug 14–18, 2016). https://doi.org/10.1007/978-3-662-53018-4_19

[15] Boyle, E., Gilboa, N., Ishai, Y.: Group-based secure computation: Optimizing rounds, communication, and computation. In: Coron, J., Nielsen, J.B. (eds.) EUROCRYPT 2017, Part II. LNCS, vol. 10211, pp. 163–193. Springer, Heidelberg, Germany, Paris, France (Apr 30 – May 4, 2017). https://doi.org/10.1007/978-3-319-56614-6_6

[16] Brakerski, Z., Döttling, N.: Two-message statistically sender-private OT from LWE. In: Beimel, A., Dziembowski, S. (eds.) TCC 2018, Part II. LNCS, vol. 11240, pp. 370–390. Springer, Heidelberg, Germany, Panaji, India (Nov 11–14, 2018). https://doi.org/10.1007/978-3-030-03810-6_14

[17] Brakerski, Z., Döttling, N., Garg, S., Malavolta, G.: Leveraging linear decryption: Rate-1 fully-homomorphic encryption and time-lock puzzles. In: Hofheinz, D., Rosen, A. (eds.) TCC 2019, Part II. LNCS, vol. 11892, pp. 407–437. Springer, Heidelberg, Germany, Nuremberg, Germany (Dec 1–5, 2019). https://doi.org/10.1007/978-3-030-36033-7_16

[18] Brakerski, Z., Lombardi, A., Segev, G., Vaikuntanathan, V.: Anonymous IBE, leakage resilience and circular security from new assumptions. In: Nielsen, J.B., Rijmen, V. (eds.) EUROCRYPT 2018, Part I. LNCS, vol. 10820, pp. 535–564. Springer, Heidelberg, Germany, Tel Aviv, Israel (Apr 29 – May 3, 2018). https://doi.org/10.1007/978-3-319-78381-9_20

[19] Brakerski, Z., Perlman, R.: Lattice-based fully dynamic multi-key FHE with short ciphertexts. In: Robshaw, M., Katz, J. (eds.) CRYPTO 2016, Part I. LNCS, vol. 9814, pp. 190–213. Springer, Heidelberg, Germany, Santa Barbara, CA, USA (Aug 14–18, 2016). https://doi.org/10.1007/978-3-662-53018-4_8

[20] Brakerski, Z., Vaikuntanathan, V.: Efficient fully homomorphic encryption from (standard) lwe. SIAM Journal on Computing **43**(2), 831–871 (2014)

[21] Chen, H., Chillotti, I., Song, Y.: Multi-key homomorphic encryption from TFHE. In: Galbraith, S.D., Moriai, S. (eds.) ASIACRYPT 2019, Part II. LNCS, vol. 11922, pp. 446–472. Springer, Heidelberg, Germany, Kobe, Japan (Dec 8–12, 2019). https://doi.org/10.1007/978-3-030-34621-8_16

[22] Cho, C., Döttling, N., Garg, S., Gupta, D., Miao, P., Polychroniadou, A.: Laconic oblivious transfer and its applications. In: Katz, J., Shacham, H. (eds.) CRYPTO 2017, Part II. LNCS, vol. 10402, pp. 33–65. Springer, Heidelberg, Germany, Santa Barbara, CA, USA (Aug 20–24, 2017). https://doi.org/10.1007/978-3-319-63715-0_2

[23] Clear, M., McGoldrick, C.: Multi-identity and multi-key leveled FHE from learning with errors. In: Gennaro, R., Robshaw, M.J.B. (eds.) CRYPTO 2015, Part II. LNCS, vol. 9216,

pp. 630–656. Springer, Heidelberg, Germany, Santa Barbara, CA, USA (Aug 16–20, 2015). https://doi.org/10.1007/978-3-662-48000-7_31

[24] Dodis, Y., Halevi, S., Rothblum, R.D., Wichs, D.: Spooky encryption and its applications. In: Robshaw, M., Katz, J. (eds.) CRYPTO 2016, Part III. LNCS, vol. 9816, pp. 93–122. Springer, Heidelberg, Germany, Santa Barbara, CA, USA (Aug 14–18, 2016). https://doi.org/10.1007/978-3-662-53015-3_4

[25] Döttling, N., Garg, S.: From selective IBE to full IBE and selective HIBE. In: Kalai, Y., Reyzin, L. (eds.) TCC 2017, Part I. LNCS, vol. 10677, pp. 372–408. Springer, Heidelberg, Germany, Baltimore, MD, USA (Nov 12–15, 2017). https://doi.org/10.1007/978-3-319-70500-2_13

[26] Döttling, N., Garg, S.: Identity-based encryption from the Diffie-Hellman assumption. In: Katz, J., Shacham, H. (eds.) CRYPTO 2017, Part I. LNCS, vol. 10401, pp. 537–569. Springer, Heidelberg, Germany, Santa Barbara, CA, USA (Aug 20–24, 2017). https://doi.org/10.1007/978-3-319-63688-7_18

[27] Döttling, N., Garg, S., Hajiabadi, M., Masny, D.: New constructions of identity-based and key-dependent message secure encryption schemes. In: Abdalla, M., Dahab, R. (eds.) PKC 2018, Part I. LNCS, vol. 10769, pp. 3–31. Springer, Heidelberg, Germany, Rio de Janeiro, Brazil (Mar 25–29, 2018). https://doi.org/10.1007/978-3-319-76578-5_1

[28] Garg, S., Srinivasan, A.: Two-round multiparty secure computation from minimal assumptions. In: Nielsen, J.B., Rijmen, V. (eds.) EUROCRYPT 2018, Part II. LNCS, vol. 10821, pp. 468–499. Springer, Heidelberg, Germany, Tel Aviv, Israel (Apr 29 – May 3, 2018). https://doi.org/10.1007/978-3-319-78375-8_16

[29] Gentry, C.: Fully homomorphic encryption using ideal lattices. In: Mitzenmacher, M. (ed.) 41st ACM STOC. pp. 169–178. ACM Press, Bethesda, MD, USA (May 31 – Jun 2, 2009). https://doi.org/10.1145/1536414.1536440

[30] Gentry, C., Sahai, A., Waters, B.: Homomorphic encryption from learning with errors: Conceptually-simpler, asymptotically-faster, attribute-based. In: Canetti, R., Garay, J.A. (eds.) CRYPTO 2013, Part I. LNCS, vol. 8042, pp. 75–92. Springer, Heidelberg, Germany, Santa Barbara, CA, USA (Aug 18–22, 2013). https://doi.org/10.1007/978-3-642-40041-4_5

[31] Goldreich, O., Goldwasser, S., Micali, S.: How to construct random functions (extended abstract). In: 25th FOCS. pp. 464–479. IEEE Computer Society Press, Singer Island, Florida (Oct 24–26, 1984). https://doi.org/10.1109/SFCS.1984.715949

[32] Goldreich, O., Goldwasser, S., Micali, S.: How to construct random functions. Journal of the ACM (JACM) **33**(4), 792–807 (1986)

[33] Goldreich, O., Micali, S., Wigderson, A.: How to play any mental game or A completeness theorem for protocols with honest majority. In: Aho, A. (ed.) 19th ACM STOC. pp. 218–229. ACM Press, New York City, NY, USA (May 25–27, 1987). https://doi.org/10.1145/28395.28420

[34] Halevi, S., Ishai, Y., Jain, A., Komargodski, I., Sahai, A., Yogev, E.: Non-interactive multiparty computation without correlated randomness. In: Takagi, T., Peyrin, T. (eds.) ASIACRYPT 2017, Part III. LNCS, vol. 10626, pp. 181–211. Springer, Heidelberg, Germany, Hong Kong, China (Dec 3–7, 2017). https://doi.org/10.1007/978-3-319-70700-6_7

[35] Lin, H., Pass, R., Seth, K., Telang, S.: Output-compressing randomized encodings and applications. In: Kushilevitz, E., Malkin, T. (eds.) TCC 2016-A, Part I. LNCS, vol. 9562, pp. 96–124. Springer, Heidelberg, Germany, Tel Aviv, Israel (Jan 10–13, 2016). https://doi.org/10.1007/978-3-662-49096-9_5

[36] López-Alt, A., Tromer, E., Vaikuntanathan, V.: On-the-fly multiparty computation on the cloud via multikey fully homomorphic encryption. In: Karloff, H.J., Pitassi, T. (eds.) 44th ACM STOC. pp. 1219–1234. ACM Press, New York, NY, USA (May 19–22, 2012). https://doi.org/10.1145/2213977.2214086

[37] Malavolta, G., Thyagarajan, S.A.K.: Homomorphic time-lock puzzles and applications. In: Boldyreva, A., Micciancio, D. (eds.) CRYPTO 2019, Part I. LNCS, vol. 11692, pp. 620–649. Springer, Heidelberg, Germany, Santa Barbara, CA, USA (Aug 18–22, 2019). https://doi.org/10.1007/978-3-030-26948-7_22

[38] Mukherjee, P., Wichs, D.: Two round multiparty computation via multi-key FHE. In: Fischlin, M., Coron, J.S. (eds.) EUROCRYPT 2016, Part II. LNCS, vol. 9666, pp. 735–763. Springer, Heidelberg, Germany, Vienna, Austria (May 8–12, 2016). https://doi.org/10.1007/978-3-662-49896-5_26

[39] Peikert, C., Shiehian, S.: Multi-key FHE from LWE, revisited. In: Hirt, M., Smith, A.D. (eds.) TCC 2016-B, Part II. LNCS, vol. 9986, pp. 217–238. Springer, Heidelberg, Germany, Beijing, China (Oct 31 – Nov 3, 2016). https://doi.org/10.1007/978-3-662-53644-5_9

[40] Peikert, C., Vaikuntanathan, V., Waters, B.: A framework for efficient and composable oblivious transfer. In: Wagner, D. (ed.) CRYPTO 2008. LNCS, vol. 5157, pp. 554–571. Springer, Heidelberg, Germany, Santa Barbara, CA, USA (Aug 17–21, 2008). https://doi.org/10.1007/978-3-540-85174-5_31

[41] Prabhanjan Ananth, Abhishek Jain, Z.J.G.M.: Multikey fhe in the plain model. Cryptology ePrint Archive, Report 2020/180 (2020), https://eprint.iacr.org/2020/180

[42] Quach, W., Wee, H., Wichs, D.: Laconic function evaluation and applications. In: Thorup, M. (ed.) 59th FOCS. pp. 859–870. IEEE Computer Society Press, Paris, France (Oct 7–9, 2018). https://doi.org/10.1109/FOCS.2018.00086

[43] Yao, A.C.C.: How to generate and exchange secrets (extended abstract). In: 27th FOCS. pp. 162–167. IEEE Computer Society Press, Toronto, Ontario, Canada (Oct 27–29, 1986). https://doi.org/10.1109/SFCS.1986.25