Delegating RAM Computations with Adaptive Soundness and Privacy

Prabhanjan Ananth[*] Yu-Chi Chen[†] Kai-Min Chung[†] Huijia Lin[‡] Wei-Kai Lin[†]

November 7, 2015

Abstract

We consider the problem of delegating RAM computations over persistent databases: A user wishes to delegate a sequence of computations over a database to a server, where each compuation may read and modify the database and the modifications persist between computations. For the efficiency of the server, it is important that computations are modeled as RAM programs, for their runtime may be sub-linear in the size of the database.

Two security needs arise in this context: Ensuring *Intergrity*, by designing means for the server to compute short proofs that allows the user to efficiently verify the correctness of the server computation, and *privacy*, providing means for the user to hide his private databases and programs from a malicious server. In this work, we aim to address both security needs, especially in the stringent, *adaptive*, setting, where the sequence of RAM computations are (potentially) chosen adaptively by a malicious server depending on the messages from an honest user.

To this end, we construct the first RAM delegation scheme achieving both *adaptive integrity* (a.k.a. soundness) and *adaptive privacy*, assuming the existence of indistinguishability obfuscation for circuits and a variant of the two-to-one somewhere perfectly binding hash [Okamoto et al. ASIACRYPT'15] (the latter can be based on the decisional Diffie-Hellman assumption). Prior works focused either only on adaptive soundness [Kalai and Paneth, ePrint'15] or on the weaker variant, selective soundness and privacy [Chen et al. ITCS'16, Canetti and Holmgren ITCS'16].

At a high-level, our result is obtained by applying a generic "*security lifting technique*" to the delegation scheme of Chen et al. and its proof of selective soundness and privacy. The security lifting technique formalizes an abstract framework of selective security proofs, and generically "lifts" such proofs into proofs of adaptive security. We believe that this technique can potentially be applied to other cryptographic schemes and is of independent interest.

[*]University of California Los Angeles and Center for Encrypted Functionalities. Email: `prabhanjan@cs.ucla.edu`.

[†]Academia Sinica, Taiwan. Email: `wycchen,kmchung,wklin@iis.sinica.edu.tw`

[‡]University of California Santa Barbara. Email: `rachel.lin@cs.ucsb.edu`.

Contents

1 Introduction

In the era of cloud computing, it is of growing popularity for users to outsource both their databases and computations to the cloud. When the databases are large, it is important that the delegated computations are modeled as RAM programs for efficiency (as computations maybe sub-linear), and that the state of a database is kept persistently across multiple (sequential) computations to support continuous updates to the database. In such a paradigm, it is imperative to address two security concerns: *Integrity* – ensuring that the cloud performs the computations correctly, and *Privacy* – information of users' private databases and programs is hidden from the cloud. In this work, we focus on designing *RAM delegation schemes* ensuring both integrity and privacy. Consider the following setting:

Private RAM Delegation Initially, to outsource her database DB, a user encodes the database using a secret key sk, and sends the encoding $\hat{DB}$ to the cloud. Later, whenever the user wishes to delegate a computation over the database, represented as a RAM program M, it encodes M using sk, producing an encoded program $\hat{M}$. Given $\hat{DB}$ and $\hat{M}$, the cloud runs an evaluation algorithm to obtain an encoded output $\hat{y}$, on the way updating the encoded database; for the user to verify the correctness of the output, the server additionally generates a proof π. Finally, upon receiving the tuple $(\hat{y}, \pi)$, the user verifies the proof and recovers the output y in the clear. The user can continue to delegate multiple computations.

Adaptive v.s. Selective Security Two "levels" of security exist for delegation schemes: The, *weaker*, selective security (referring to integrity and/or privacy) provides guarantees only in the restricted setting where all delegated computations are chosen statically, whereas, the, *stronger*, adaptive security allows them to be chosen adaptively, each (potentially) depending on the encodings of the database and previously chosen programs. Clearly, adaptive security is more natural and desirable in the context of cloud computing, especially for these applications where a large database is processed and outsourced once and many computations over the database are delegated over time.

State-of-the-art Significant progress has been made towards secure RAM delegation. Following the studies on succinct randomized encodings for Turing machines and RAM [BGL$^+$15, CHJV15, KLW15], Chen et al. [CCC$^+$16] and Canetti and Holmgren [CH16] constructed efficient RAM delegation schemes that achieve both *selective privacy* and *selective integrity*, assuming indistinguishability obfuscation ($\mathsf{i}\mathcal{O}$) and one-way functions. Very recently, Kalai and Paneth [KP15], building upon the seminal result of [KRR14], constructed the first RAM delegation scheme with *adaptive integrity*, based on super-polynomial hardness of the LWE assumption. However, their solution does not achieve privacy.

Summarizing the state-of-the-art, however, known constructions either focus only on adaptive integrity, without privacy, or achieve both integrity and privacy, but only the weaker selective version. This raises the question:

Can we have a RAM delegation scheme with adaptive integrity and privacy?

We answer this question affirmatively, by constructing such a scheme based on $\mathsf{i}\mathcal{O}$ and DDH.

1.1 Our Contributions

Our main result is the following theorem.

Theorem 1 (informal). *Assuming DDH and the existence of* $i\mathcal{O}$ *for all polynomial size circuits, there exists an efficient RAM delegation scheme with persistent database, with adaptive privacy and adaptive soundness (a.k.a. integrity).*

The efficiency of our RAM delegation scheme matches that of the selectively secure scheme of [CCC+16, CH16]: To outsource a database DB of size n, the user encodes the database in time linear in the database size, $n \operatorname{poly}(\lambda)$ (where λ is the security parameter), and the server merely stores the encoded database. To delegate the computation of a RAM program M, with l-bit outputs and time and space complexity T and S, the user encodes the program in time linear in the output length and polynomial in the program description size $l \times \operatorname{poly}(|M|, \lambda)$, independent of the complexity of the RAM program. At the other side, the evaluation time and space complexity of the cloud, or server, scales linearly with the complexity of the RAM programs, that is, $T \operatorname{poly}(\lambda)$ and $S \operatorname{poly}(\lambda)$ respectively. Finally, the user verifies the proof from the server and recovers the output in the clear in time $l \times \operatorname{poly}(\lambda)$.

We remark that the efficiency of our scheme is *nearly optimal*: The complexity of the user and the server are essentially the same as that of an *insecure* scheme (where the user simply sends the database and programs in the clear, and does not verify the correctness of the server computation), up to a multiplicative $\operatorname{poly}(\lambda)$ overhead at the server, and a $\operatorname{poly}(|M|, \lambda)$ overhead at the user. [1] In particular, if the runtime of a delegated RAM program is sub-linear in the database size, the server evaluation time is also sub-linear, which is crucial for server efficiency.

Our Ideas in a Nut Shell Our main theorem is established by showing that, in fact, the selectively secure RAM delegation scheme of Chen et al. [CCC+16] (CCC+), with slight modification is already secure against adaptive adversaries. Hence, our main technical contribution lies in the proof of adaptive security. In particular, instead of repeating and modifying the, quite complicated, CCC+ proof, our approach is abstract and general: We characterize a framework of proofs for showing the indistinguishability of two arbitrary experiments against selective adversaries that have certain special structure and properties. We then show that *any* proof of selective security that fits into the framework—called a "nice" proof—can be turned into a proof of adaptive security in a *generic way*. Therefore, by verifying that the CCC+ scheme (with a slight modification) admits such a "nice" proof, it follows from our abstract framework that it is also adaptively secure.

Our technique provides a semi-generic approach for "lifting" an indistinguishability proof for selective adversaries to a proof for adaptive adversaries: Simply verify whether the original proof satisfy the properties of a "nice" proof and then apply our result in a black-box. We believe that this technique can potentially be applied to other cryptographic schemes and is of independent interests.

[1] We believe that the polynomial dependency on the program description size can be further reduced to linear dependency, using techniques in the recent work of [AJS15].

Extension to Parallel RAM (PRAM) Delegation. While the RAM model captures sublinear time computation on the database, it does not support parallel accesses to the database. Chen et al. [CCC+16] also presented a delegation scheme for *parallel RAM computations*, with selective soundness and privacy. By applying our general technique, we can also lift the selective security of their PRAM delegation scheme to adaptive security, obtaining an adaptively secure PRAM delegation scheme.

Comparison to Turing Machine Based Delegation Schemes. We note that there are prior works that generically achieve adaptive security from selective security in the context of circuits [ABSV15] which was later generalized to the context of Turing machines [AS16]. One of the important techniques underlying these schemes is a refreshing mechanism that "refreshes" the entire database for every program encoding issued. This means that the runtime of every program encoding is proportional to the database size. However, applying this technique in the context of RAMs would destroy the sub-linear time efficiency that we crucially aim for.

1.2 Technical Overview

We now explain our technique in more detail, starting with the abstract proof framework, and then application to the CCC+ scheme and proof.

1.2.1 An Abstract Framework

We focus on the general task of proving the indistinguishability of two cryptographic experiments, referred to as the real experiments $Real_0$ and $Real_1$, where each experiment is defined by a challenger that interacts with an adversary. An unrestricted *adaptive adversary* can choose every messages it sends adaptively based on the messages it receives from the challenger, whereas a *selective adversary* is restricted to make certain choices statically at the beginning of its execution. (For example, in the context of delegating RAM computations, a selective adversary decides on the database DB and all RAM programs M_i's to be delegated statically).

A common proof paradigm We start with describing a common proof paradigm for showing the indistinguishability of two experiments:

- First, construct a sequence of hybrid experiments $H_0, \cdots, H_\ell$, that starts from one real experiment (i.e., $H_0 = Real_0$), and gradually morphs through intermediate hybrids H_i's into the other (i.e., $H_\ell = Real_1$).
- Second, show that every pair of neighboring hybrids H_i, H_{i+1} is indistinguishable via a *straight-line black-box* reduction R_i based on some computational assumption, (for instance, the pseudo-randomness of a Puncturable Pseudo-Random Function (PPRF) [BW13, BGI14, KPTZ13], and the indistinguishability of $i\mathcal{O}$ for circuits [BGI+01, GGH+13]).

Then, by standard hybrid arguments, it follows that the real experiments are indistinguishable. This proof paradigm applies to both selective and adaptive adversaries. But when considering only selective adversaries, the task of constructing the hybrids and reductions are (potentially) easier for twofold reasons:

1. Since selective adversaries make certain choices statically at the beginning (e.g., the database DB and RAM programs M_i's to be delegated), the challenger CH_i of an hybrid experiment H_i may generate or simulate messages to the adversary, depending on this choice (e.g., in the proof of the CCC+ scheme, some hybrids simulates the encoding of a RAM program depending on DB and all M_i's). Such hybrids can syntactically only interact with selective adversaries.
2. The reduction only need to work with selective adversaries (i.e., "transform" the advantage of a selective adversary in distinguishable the pair of hybrids into an advantage in breaking the corresponding assumption).

Therefore, to "lift" a proof for selective adversaries (in the above framework) to one also for adaptive adversaries, we need to remove the above two restrictions.

Generalized cryptographic experiments With respect to the first restriction, we propose a generalization of classical (cryptographic) experiments so that experiments can always be executed with both selective and adaptive adversaries, or even adversaries with more fine-grained levels of selectivity. Towards this, we differentiate the information that an adversary "commits" to at the beginning of its execution, from the information that a challenger depends on for generating its messages (such as, DB and M_i's or nothing).

- The information that an adversary A commits to is written to a *special output tape*, as opposed to sending to the challenger (e.g., in the context of RAM delegation, a selective adversary writes DB and M_i's, while an adaptive adversary writes nothing).
- The information that a challenger CH depends on is defined by a function G, and upon A writing α to its *special output tape*, it receives $G(\alpha)$; this mechanism is part of the generalized experiment, which A is oblivious to.

The function G could vary from being a null function that always outputs "nothing" (i.e., an empty string), meaning that the challenger CH is *oblivious* of the adversary's choice α, to the identify function, meaning that CH depends on α entirely, or to a function that outputs some "partial information" of α, (e.g., consider G that parses α as DB and M_i's, and outputs the address in DB that the k^{th} program M_k accesses at its j^{th} step, for some fixed k and j). As we will see later, experiments depending on such partial information are crucial for our proof—we call them *G-semi-oblivious* experiment; experiments in the first and second cases are also called *oblivious* or *non-oblivious* experiments.

Importantly, in generalized experiments, no matter what information (i.e., G) the challenger depends on, it can always interact with arbitrary adversaries, selective or adaptive. Thus, when (re)formulating the sequence of hybrids in a proof for selective adversaries as generalized experiments, we can consider their executions with adaptive adversaries.

Upgrading the reductions It remains to show that neighboring (generalized) hybrids, H_i and H_{i+1}, are indistinguishable to also adaptive adversaries, from which indistinguishability of real experiments against adaptive adversaries follows. But, we cannot directly apply the reductions R_i given in the proof for selective adversaries. Instead, we characterize hybrids and reductions with certain special properties, so that the reductions can be "upgraded" into ones that work with adaptive adversaries.

<u>Warm-up case</u> Consider first a simple case where two neighboring hybrids H_i and H_{i+1} are *oblivious* (i.e., their function G_i's are empty). It is well known that if an adversary distinguishes two hybrids with advantage 2γ, then it wins a "guessing game" with advantage γ (where after interacting with a randomly chosen challenger, CH_i or CH_{i+1}, it guesses correctly which one it has talked to with probability $1/2+\gamma$). We show that if the reduction R_i satisfies the following *statistical emulation property*, then it works directly with adaptive adversaries. Roughly speaking, R_i leverages adversaries' advantage by interacting with them in a straight-line black-box manner, and emulating every message in the "guessing game" statistically. More precisely, for every (prefix of) transcript $\rho = (m_1, a_1, m_2, a_2, \cdots)$ of message exchanges, the distribution of the next message generated by R_i conditioned on ρ appearing before (i.e., R_i receiving messages $m_1, m_2, \cdots$ and producing messages $a_1, a_2, \cdots$), is statistically close to the distribution of the next message in the "guessing game" conditioned on the same transcript ρ appearing.

In the literature, security reductions often emulate for an adversary statistically or even perfectly the game in which it has an advantage. As such, the reductions essentially do not differentiate selective or adaptive adversaries, and merely leverage their advantages in a universal way. Hence, we can show that this syntactical property implies that the reduction also works with adaptive adversaries.

<u>Intermediate case</u> We relax the restriction in the warm-up case and consider hybrids that depend on some "small" partial information, that is, H_i and H_{i+1} are respectively G_i- and G_{i+1}-semi-oblivious, with *G_i and G_{i+1} having polynomial sized ranges.* In this case, we first generalize the above statistical emulation property of R_i to the context of semi-oblivious experiments, and similarly show that such reductions work directly with a class of *semi-selective adversaries* that determines the partial information that H_i and H_{i+1} depend on (i.e., the outputs of G_i and G_{i+1}) statically.

Our next key observation is that the only difference between semi-selective and adaptive adversaries is that the latter chooses even the partial information adaptively. However, since G_i and G_{i+1} have only polynomial-sized ranges, the adaptive choice of an adaptive adversary can be guessed ahead of time with inverse polynomial probabilities. In other words, every adaptive adversary can be turned into a semi-adaptive adversary, with only a (multiplicative) polynomial factor decrease in the advantage (by guessing the choice of the adaptive adversary ahead of time and aborting in the end if the guess is incorrect); then the reduction R_i can be applied to deduce that the advantage of the adaptive adversary is also negligible. (Alternatively, we can construct a wrapper reduction running R_i internally, but works with adaptive adversaries.)

The above technique is, in fact, simply complexity leveraging. However, it is applied to a single step in the proof, to "upgrade" the indistinguishability of two neighboring hybrids to be resilient to adaptive adversaries—we call this technique *local complexity leveraging*. The benefit of local complexity leveraging is that the security loss is only related to the "size" of the information that the hybrids depend on, rather than the "size" of the global information a selective adversary chooses; yet, by applying it to every step in the proof, it achieves the same effect as global complexity leveraging.

Technicalities We point out that several technicalities arise when applying the local complexity leveraging. First, each hybrid H_i needs to satisfy an additional property called G_i*-hiding*, that is, the challenger's messages depend on, but computationally hides the output of G_i. Roughly speaking, this is needed to ensure that when executing H_i with a random output of G_i, the choice made by the adaptive adversary is computationally independent of the random output, and hence agrees with the random output with inverse polynomial probability. Second, considering the indistinguishability of neighboring hybrids is only accurate when the number of hybrids is a constant. Our formal proof considers polynomial number of hybrids, and proves via contradiction to local a sequences of neighboring hybrids that are distinguished by an adversary. This also leads to more complication, as the indistinguishability of different pairs of hybrids in the sequence may be reduced to different assumptions. We handle this depending on the fact that there is a constant number of underlying assumptions.

1.2.2 The CCC+ Scheme and Its Nice Proof

CCC+ proposed a selectively secure RAM delegation scheme, satisfying *input privacy* and *program privacy*, in the persistent database setting. It was shown in previous works [AIK10, GHRW14] that *output privacy* and *soundness* follow from input and program privacy using standard black-box transformations. So we focus our attention on only input and program privacy.

We now show how CCC+ scheme can be used to instantiate the abstract framework discussed earlier in this Section. Since the CCC+ scheme is quite involved, we only provide the relevant details of CCC+ and refer the reader to their paper for a thorough discussion.

There are two main components in CCC+. The first component is *storage* that maintains information about the database and the second component is the *machine* component that involves executing instructions of the delegated RAM. For every RAM delegated, there will be a separate *machine component*. Both the storage and the machine components are built on heavy machinery. We present below the three main building blocks, that are imported from Koppula et al. [KLW15]. The components also use additional tools such as indistinguishability obfuscation, puncturable PRFs and standard encryption schemes.

- *Positional Accumulators*: This primitive offers a mechanism of producing a short value, called *accumulator*, that commits to a large storage. Further, accumulators should also be updatable - if a small portion of storage changes then only a correspondingly small change is required to update the accumulator value. While there exist cryptographic hash functions that already achieve this goal; in the security proof, accumulators also allow for programming the parameters with respect to a particular location in such a way that the accumulator uniquely determines the value at that location. However, to program the parameters, it is also necessary that we need to know ahead of time all the changes the storage undergoes, which reflects the evolution of computation, since its initialization. We emphasize that the programming part occurs only in the security proof.

Henceforth, we refer to the setting when the accumulator parameters are programmed to be ENFORCE-MODE and the setting when it is not programmed to be REAL-MODE.

- *Iterators*: This tool offers a method of binding the current state of RAM. As in the case of positional accumulators, iterators exist in two modes, namely, real-mode and enforce-mode. And also, the enforce mode involves programming the parameters as a function of the entire computation until that point. However, unlike positional accumulators, fresh instantiations of iterators can be dynamically chosen once the computation path is decided. We refer to the relevant technical sections for more details.

- *Splittable signatures*: A splittable signature scheme allows for splitting of the signing key into two keys where one key is used to sign only messages in a set S and the other key is only used to sign messages in $\overline{S}$ (complement of S). We only consider the case when the set S contains one message. Correspondingly, the verification key can also be split into two keys, with one key used to verify signatures on messages in set S and the other used to verify signatures on messages in $\overline{S}$.

We now focus on the security proof of CCC+. We denote the set of hybrids in CCC+ to be $H_1, \ldots, H_\ell$. Correspondingly, we denote the reductions that argue indistinguishability of H_i and H_{i+1} to be R_i. We consider the following two cases depending on the type of neighboring hybrids H_i and H_{i+1}:

1. POSITIONAL ACCUMULATOR IS IN REAL-MODE IN BOTH H_i AND $H_i i + 1$:- In this case, the indistinguishability of H_i and H_{i+1} is argued either using iterators, splittable signatures, indistinguishability obfuscation or other standard cryptographic primitives. In the corresponding reduction R_i, we note that the information G_i written by the adversary to the special output tape is null. And hence, the reduction R_i corresponds to the *oblivious* experiment.

2. POSITIONAL ACCUMULATOR IS IN ENFORCE-MODE IN EITHER H_i AND H_{i+1}:- Here, the adversary is supposed to declare all its inputs in the beginning of experiment. The reason being that in the enforce-mode, the accumulator parameters need to be programmed. As remarked earlier, programming the parameters is possible only with the knowledge of the entire computation.

As seen from the above description, the second case is problematic for us since the information to be declared by the adversary in the beginning of the experiment is too long. Hence, we need to think of alternate variants to positional accumulators where the enforce-mode can be implemented without the knowledge of the computation history.

History-less Accumulators. To this end, we introduce a primitive called *history-less accumulators*. As the name is suggestive, in this primitive, programming the parameters requires only the location being information-theoretically bound to be known ahead of time. And note that the location can be represented using only logarithmic bits and satisfies the sizing requirements. That is, the G_i to be declared at the beginning of the experiment is now short and so the corresponding reduction now corresponds to *semi-oblivious* experiment. By plugging this into the CCC+ scheme, we obtain a security proof, where every reduction is either oblivious or semi-oblivious.

All is left is to construct the primitive of history-less accumulators. To achieve this goal, we consider the definition of Okamoto et al.'s [OPWW15] two-to-one hash. OPWW showed how to construct positional accumulators from two-to-one hash. We augment the definition of OPWW with an additional property and then we show how to adapt OPWW's transformation to obtain history-less accumulators. Finally, we adopt the OPWW's DDH-based construction of two-to-one hash to also satisfy our additional property.

1.3 Other Related Works

The notion of (one-time, non-succinct) garbled RAM was introduced by the work of Lu and Ostrovsky [LO13], and since then, a sequence of works [GHL+14, GLOS15] have led to a black-box construction based on one-way functions, due to Garg, Lu, and Ostrovsky [GLO15]. However, the garbled program size here is proportional to the worst-case time complexity of the RAM program, so this notion does not imply a RAM delegation scheme. The work of Gentry, Halevi, Raykova, and Wichs [GHRW14] showed how to make such garbled RAMs reusable based on various notions of obfuscations (with efficiency trade-offs), and constructed the first RAM delegation schemes in a (weaker) offline/online setting. Succinct garbled RAM was first studied by [BGL+15, CHJV15], where in their solutions, the garbled program size depends on the space complexity of the RAM program, but does not depend on its time complexity. This implies delegation for space-bounded RAM computations. Finally, as mentioned, the works of [CH16, CCC+16] constructed fully succinct garbled RAM, and [CCC+16] additionally gives the first fully succinct garbled PRAM.

1.4 Organization

We first describe the preliminaries in Section 2. In Section 3, we present our abstract proof framework. The formal definition of adaptive delegation for RAMs is presented in Section 4. Towards achieving adaptive RAM delegation, we first present the construction of history-less accumulators in Section 5. Then we show how to utilize this to construct Adaptive CiO for RAMs with persistent database in Section 6. In the next step we show how to achieve adaptive garbled RAM with persistent database in Section 7. In the final step, we give a generic transformation from adaptive GRAM to adaptive delegation in Section 8.

2 Preliminaries

We denote the security parameter by λ. We assume familiarity of the reader with standard cryptographic assumptions.

2.1 Indistinguishability Obfuscation

The notion of indistinguishability obfuscation (iO), first conceived by Barak et al. [BGI+01], guarantees that the obfuscation of two circuits are computationally indistinguishable as long as they both are equivalent circuits, i.e., the output of both the circuits are the same on every input. Formally,

Definition 1 (Indistinguishability Obfuscator (iO) for Circuits)**.** *A uniform PPT algorithm* $\mathsf{i}\mathcal{O}$ *is called an indistinguishability obfuscator for a circuit family* $\{\mathcal{C}_\lambda\}_{\lambda\in\mathbb{N}}$*, where* $\mathcal{C}_\lambda$ *consists of circuits* C *of the form* $C : \{0,1\}^{\mathsf{in}} \to \{0,1\}$ *with* $\mathsf{in} = \mathsf{in}(\lambda)$*, if the following holds:*

- **Completeness:** *For every* $\lambda \in \mathbb{N}$*, every* $C \in \mathcal{C}_\lambda$*, every input* $x \in \{0,1\}^{\mathsf{in}}$*, we have that*

$$\mathsf{Pr}\left[C'(x) = C(x) \ : \ C' \leftarrow \mathsf{i}\mathcal{O}(\lambda, C)\right] = 1$$

- **Indistinguishability:** *For any PPT distinguisher* D*, there exists a negligible function* $\mathsf{negl}(\cdot)$ *such that the following holds: for all sufficiently large* $\lambda \in \mathbb{N}$*, for all pairs of circuits* $C_0, C_1 \in \mathcal{C}_\lambda$ *such that* $C_0(x) = C_1(x)$ *for all inputs* $x \in \{0,1\}^{\mathsf{in}}$*, we have:*

$$\Big|\mathsf{Pr}\left[D(\lambda, \mathsf{i}\mathcal{O}(\lambda, C_0)) = 1\right] - \mathsf{Pr}[D(\lambda, \mathsf{i}\mathcal{O}(\lambda, C_1)) = 1]\Big| \leq \mathsf{negl}(\lambda)$$

2.2 Puncturable Pseudorandom Functions

A pseudorandom function family $\mathcal{F}$ consisting of functions of the form $\mathsf{PRF}_K(\cdot)$, that is defined over input space $\{0,1\}^{\eta(\lambda)}$, output space $\{0,1\}^{\chi(\lambda)}$ and key K in the key space $\mathcal{K}$, is said to be a *secure puncturable PRF family* if there exists a PPT algorithm $\mathsf{Puncture}$ that satisfies the following properties:

- **Functionality preserved under puncturing.** $\mathsf{Puncture}$ takes as input a PRF key K, sampled from $\mathcal{K}$, and an input $x \in \{0,1\}^{\eta(\lambda)}$ and outputs K_x such that for all $x' \neq x$, $\mathsf{PRF}_{K_x}(x') = \mathsf{PRF}_K(x')$.
- **Pseudorandom at punctured points.** For every PPT adversary $(\mathcal{A}_1, \mathcal{A}_2)$ such that $\mathcal{A}_1(1^\lambda)$ outputs an input $x \in \{0,1\}^{\eta(\lambda)}$, consider an experiment where $K \overset{\$}{\leftarrow} \mathcal{K}$ and $K_x \leftarrow \mathsf{Puncture}(K, x)$. Then for all sufficiently large $\lambda \in \mathbb{N}$, for a negligible function μ,

$$\big|\mathsf{Pr}[\mathcal{A}_2(K_x, x, \mathsf{PRF}_K(x)) = 1] - Pr[\mathcal{A}_2(K_x, x, U_{\chi(\lambda)}) = 1]\big| \leq \mu(\lambda)$$

  where $U_{\chi(\lambda)}$ is a string drawn uniformly at random from $\{0,1\}^{\chi(\lambda)}$.

As observed by [BW13, BGI14, KPTZ13], the GGM construction [GGM86] of PRFs from one-way functions yields puncturable PRFs.

Theorem 2 ([GGM86,BW13,BGI14,KPTZ13]). *If μ-secure one-way functions*[2] *exist, then for all polynomials $\eta(\lambda)$ and $\chi(\lambda)$, there exists a μ-secure puncturable PRF family that maps $\eta(\lambda)$ bits to $\chi(\lambda)$ bits.*

2.3 Tools of [KLW15]

We recall the tools used in the work of Chen et. al. [CCC+16], inherited from Koppula et al. [KLW15]. We start with the definition of the iterators and splittable signature schemes in Section 2.3.1 and 2.3.2. Then, we propose a variant of positional accumulators, termed as *history-less accumulators* in Section 5.

2.3.1 Iterators

In this subsection, we now describe the notion of cryptographic iterators. As remarked earlier, iterators essentially consist of states that are updated on the basis of the messages received. We describe its syntax below.

Syntax Let ℓ be any polynomial. An iterator $\mathsf{PP}_{\mathsf{Itr}}$ with message space $\mathsf{Msg}_\lambda = \{0,1\}^{\ell(\lambda)}$ and state space St_λ consists of three algorithms - $\mathsf{SetupItr}$, $\mathsf{ItrEnforce}$ and $\mathsf{Iterate}$ defined below.

$\mathsf{SetupItr}(1^\lambda, T)$ The setup algorithm takes as input the security parameter λ (in unary), and an integer bound T (in binary) on the number of iterations. It outputs public parameters $\mathsf{PP}_{\mathsf{Itr}}$ and an initial state $v_0 \in \mathsf{St}_\lambda$.

$\mathsf{ItrEnforce}(1^\lambda, T, \vec{m} = (m_1, \dots, m_k))$ The enforced setup algorithm takes as input the security parameter λ (in unary), an integer bound T (in binary) and k messages $(m_1, \dots, m_k)$, where each $m_i \in \{0,1\}^{\ell(\lambda)}$ and k is some polynomial in λ. It outputs public parameters $\mathsf{PP}_{\mathsf{Itr}}$ and a state $v_0 \in \mathsf{St}$.

$\mathsf{Iterate}(\mathsf{PP}_{\mathsf{Itr}}, v_{\mathsf{in}}, m)$ The iterate algorithm takes as input the public parameters $\mathsf{PP}_{\mathsf{Itr}}$, a state v_{in}, and a message $m \in \{0,1\}^{\ell(\lambda)}$. It outputs a state $v_{\mathsf{out}} \in \mathsf{St}_\lambda$.

For simplicity of notation, the dependence of ℓ on λ will not be explicitly mentioned. Also, for any integer $k \leq T$, we will use the notation $\mathsf{Iterate}^k(\mathsf{PP}_{\mathsf{Itr}}, v_0, (m_1, \dots, m_k))$ to denote $\mathsf{Iterate}(\mathsf{PP}_{\mathsf{Itr}}, v_{k-1}, m_k)$, where $v_j = \mathsf{Iterate}(\mathsf{PP}_{\mathsf{Itr}}, v_{j-1}, m_j)$ for all $1 \leq j \leq k-1$.

Security. Let $\mathsf{Itr} = \{\mathsf{SetupItr}, \mathsf{ItrEnforce}, \mathsf{Iterate}\}$, be an iterator scheme with message space Msg_λ and state space St_λ. We require the following notions of security.

Definition 2 (Indistinguishability of Setup). *An iterator $\mathsf{Itr} = \{\mathsf{SetupItr}, \mathsf{ItrEnforce}, \mathsf{Iterate}\}$ is said to satisfy indistinguishability of Setup phase if any PPT adversary $\mathcal{A}$'s advantage in the security game* ***Exp-Setup-Itr***$(1^\lambda, \mathsf{Itr}, \mathcal{A})$ *is at most negligible in λ, where* ***Exp-Setup-Itr*** *is defined as follows.*

[2]We say that a one-way function family is μ-secure if the probability of inverting a one-way function, that is sampled from the family, is at most $\mu(\lambda)$.

Exp-Setup-Itr$(1^\lambda, \mathsf{Itr}, \mathcal{A})$

- *The adversary $\mathcal{A}$ chooses a bound $N \in \Theta(2^\lambda)$ and sends it to the challenger.*
- *$\mathcal{A}$ sends $\vec{m}$ to the challenger, where $\vec{m} = (m_1, \ldots, m_k) \in (\mathsf{Msg}_\lambda)^k$.*
- *The challenger chooses a bit b. If $b = 0$, the challenger outputs $(\mathsf{PP}_{\mathsf{Itr}}, v_0) \leftarrow \mathsf{SetupItr}(1^\lambda, T)$. Else, it outputs $(\mathsf{PP}_{\mathsf{Itr}}, v_0) \leftarrow \mathsf{ItrEnforce}(1^\lambda, T, \vec{m})$.*
- *$\mathcal{A}$ sends a bit b'.*

$\mathcal{A}$ wins the security game if $b = b'$.

Definition 3 (Enforcing)**.** *Consider any $\lambda \in \mathbb{N}, T \in \Theta(2^\lambda), \vec{m} = (m_1, \ldots, m_k) \in (\mathsf{Msg}_\lambda)^k$. Let $(\mathsf{PP}_{\mathsf{Itr}}, v_0) \leftarrow \mathsf{ItrEnforce}(1^\lambda, T, \vec{m})$ and $v_j = \mathsf{Iterate}(\mathsf{PP}_{\mathsf{Itr}}, v_{j-1}, m_j)$ for all $j \in [k]$. Then, $\mathsf{Itr} = \{\mathsf{SetupItr}, \mathsf{ItrEnforce}, \mathsf{Iterate}\}$ is said to be* enforcing *if*

$$v_k = \mathsf{Iterate}(\mathsf{PP}_{\mathsf{Itr}}, v', m') \Rightarrow (v', m') = (v_{k-1}, m_k).$$

Note that this is an information-theoretic property.

2.3.2 Splittable Signatures

We describe the syntax of the splittable signatures scheme below.

Syntax A splittable signature scheme $\mathsf{SplScheme}$ for message space Msg consists of the following algorithms:

$\mathsf{SetupSpl}(1^\lambda)$ The setup algorithm is a randomized algorithm that takes as input the security parameter λ and outputs a signing key SK, a verification key VK and *reject-verification key* VK_{rej}.

$\mathsf{SignSpl}(\mathsf{SK}, m)$ The signing algorithm is a deterministic algorithm that takes as input a signing key SK and a message $m \in \mathsf{Msg}$. It outputs a signature σ.

$\mathsf{VerSpl}(\mathsf{VK}, m, \sigma)$ The verification algorithm is a deterministic algorithm that takes as input a verification key VK, signature σ and a message m. It outputs either 0 or 1.

$\mathsf{SplitSpl}(\mathsf{SK}, m^*)$ The splitting algorithm is randomized. It takes as input a secret key SK and a message $m^* \in \mathsf{Msg}$. It outputs a signature $\sigma_{\text{one}} = \mathsf{SignSpl}(\mathsf{SK}, m^*)$, a one-message verification key VK_{one}, an all-but-one signing key SK_{abo} and an all-but-one verification key VK_{abo}.

$\mathsf{SignSplAbo}(\mathsf{SK}_{\text{abo}}, m)$ The all-but-one signing algorithm is deterministic. It takes as input an all-but-one signing key SK_{abo} and a message m, and outputs a signature σ.

Correctness Let $m^* \in \mathsf{Msg}$ be any message. Let $(\mathsf{SK}, \mathsf{VK}, \mathsf{VK}_{\text{rej}}) \leftarrow \mathsf{SetupSpl}(1^\lambda)$ and $(\sigma_{\text{one}}, \mathsf{VK}_{\text{one}}, \mathsf{SK}_{\text{abo}}, \mathsf{VK}_{\text{abo}}) \leftarrow \mathsf{SplitSpl}(\mathsf{SK}, m^*)$. Then, we require the following correctness properties:

1. For all $m \in \mathsf{Msg}$, $\mathsf{VerSpl}(\mathsf{VK}, m, \mathsf{SignSpl}(\mathsf{SK}, m)) = 1$.
2. For all $m \in \mathsf{Msg}, m \neq m^*$, $\mathsf{SignSpl}(\mathsf{SK}, m) = \mathsf{SignSplAbo}(\mathsf{SK}_{\text{abo}}, m)$.

3. For all σ, $\mathsf{VerSpl}(\mathsf{VK}_{\text{one}}, m^*, \sigma) = \mathsf{VerSpl}(\mathsf{VK}, m^*, \sigma)$.
4. For all $m \neq m^*$ and σ, $\mathsf{VerSpl}(\mathsf{VK}, m, \sigma) = \mathsf{VerSpl}(\mathsf{VK}_{\text{abo}}, m, \sigma)$.
5. For all $m \neq m^*$ and σ, $\mathsf{VerSpl}(\mathsf{VK}_{\text{one}}, m, \sigma) = 0$.
6. For all σ, $\mathsf{VerSpl}(\mathsf{VK}_{\text{abo}}, m^*, \sigma) = 0$.
7. For all σ and all $m \in \mathsf{Msg}$, $\mathsf{VerSpl}(\mathsf{VK}_{\text{rej}}, m, \sigma) = 0$.

Security. We will now define the security notions for splittable signature schemes. Each security notion is defined in terms of a security game between a challenger and an adversary $\mathcal{A}$.

Definition 4 (VK_{rej} indistinguishability)**.** *A splittable signature scheme* Spl *is said to be* VK_{rej} *indistinguishable if any PPT adversary* $\mathcal{A}$ *has negligible advantage in the following security game:*

Exp-$\mathsf{VK}_{\text{rej}}(1^\lambda, \mathsf{Spl}, \mathcal{A})$

- *The challenger computes* $(\mathsf{SK}, \mathsf{VK}, \mathsf{VK}_{\text{rej}}) \leftarrow \mathsf{SetupSpl}$*. It chooses a bit* $b \in \{0,1\}$*. If* $b = 0$*, the challenger sends* VK *to* $\mathcal{A}$*. Else, it sends* VK_{rej} *to* $\mathcal{A}$*.*
- $\mathcal{A}$ *sends a bit* b'*.*

$\mathcal{A}$ *wins if* $b = b'$*.*

Definition 5 (VK_{one} indistinguishability)**.** *A splittable signature scheme* Spl *is said to be* VK_{one} *indistinguishable if any PPT adversary* $\mathcal{A}$ *has negligible advantage in the following security game:*

Exp-$\mathsf{VK}_{\text{one}}(1^\lambda, \mathsf{Spl}, \mathcal{A})$

- $\mathcal{A}$ *sends a message* $m^* \mathcal{M}_\lambda$*.*
- *The challenger computes* $(\mathsf{SK}, \mathsf{VK}, \mathsf{VK}_{\text{rej}}) \leftarrow \mathsf{SetupSpl}$*, and computes* $(\sigma_{\text{one}}, \mathsf{VK}_{\text{one}}, \mathsf{SK}_{\text{abo}}, \mathsf{VK}_{\text{abo}}) \leftarrow \mathsf{SignSpl}(\mathsf{SK}, m^*)$*. It chooses a bit* $b \in \{0,1\}$*. If* $b = 0$*, the challenger sends* $(\sigma_{\text{one}}, \mathsf{VK}_{\text{one}})$ *to* $\mathcal{A}$*. Else, it sends* $(\sigma_{\text{one}}, \mathsf{VK})$ *to* $\mathcal{A}$*.*
- $\mathcal{A}$ *sends a bit* b'*.*

$\mathcal{A}$ *wins if* $b = b'$*.*

Definition 6 (VK_{abo} indistinguishability)**.** *A splittable signature scheme* Spl *is said to be* VK_{abo} *indistinguishable if any PPT adversary* $\mathcal{A}$ *has negligible advantage in the following security game:*

Exp-$\mathsf{VK}_{\text{abo}}(1^\lambda, \mathsf{Spl}, \mathcal{A})$

- $\mathcal{A}$ *sends a message* $m^* \in \mathcal{M}_\lambda$*.*
- *The challenger computes* $(\mathsf{SK}, \mathsf{VK}, \mathsf{VK}_{\text{rej}}) \leftarrow \mathsf{SetupSpl}$*, and computes* $(\sigma_{\text{one}}, \mathsf{VK}_{\text{one}}, \mathsf{SK}_{\text{abo}}, \mathsf{VK}_{\text{abo}}) \leftarrow \mathsf{SignSpl}(\mathsf{SK}, m^*)$*. It chooses a bit* $b \in \{0,1\}$*. If* $b = 0$*, the challenger sends* $(\mathsf{SK}_{\text{abo}}, \mathsf{VK}_{\text{abo}})$ *to* $\mathcal{A}$*. Else, it sends* $(\mathsf{SK}_{\text{abo}}, \mathsf{VK})$ *to* $\mathcal{A}$*.*
- $\mathcal{A}$ *sends a bit* b'*.*

$\mathcal{A}$ *wins if* $b = b'$*.*

Definition 7 (Splitting indistinguishability)**.** *A splittable signature scheme* Spl *is said to be splitting indistinguishable if any PPT adversary* $\mathcal{A}$ *has negligible advantage in the following security game:*

Exp-$\mathsf{VK}_{\rm abo}(1^\lambda, \mathsf{Spl}, \mathcal{A})$

- $\mathcal{A}$ *sends a message* $m^* \in \mathcal{M}_\lambda$.
- *The challenger computes* $(\mathsf{SK}, \mathsf{VK}, \mathsf{VK}_{\rm rej}) \leftarrow \mathsf{SetupSpl}(1^\lambda)$, $(\mathsf{SK}', \mathsf{VK}', \mathsf{VK}'_{\rm rej}) \leftarrow \mathsf{SetupSpl}(1^\lambda)$, *and computes* $(\sigma_{\rm one}, \mathsf{VK}_{\rm one}, \mathsf{SK}_{\rm abo}, \mathsf{VK}_{\rm abo}) \leftarrow \mathsf{SignSpl}(\mathsf{SK}, m^*)$, $(\sigma'_{\rm one}, \mathsf{VK}'_{\rm one}, \mathsf{SK}'_{\rm abo}, \mathsf{VK}'_{\rm abo}) \leftarrow \mathsf{SignSpl}(\mathsf{SK}', m^*)$. *It chooses a bit* $b \in \{0,1\}$. *If* $b = 0$, *the challenger sends* $(\sigma_{\rm one}, \mathsf{VK}_{\rm one}, \mathsf{SK}_{\rm abo}, \mathsf{VK}_{\rm abo})$ *to* $\mathcal{A}$. *Else, it sends* $(\sigma'_{\rm one}, \mathsf{VK}'_{\rm one}, \mathsf{SK}_{\rm abo}, \mathsf{VK}_{\rm abo})$ *to* $\mathcal{A}$.
- $\mathcal{A}$ *sends a bit* b'.

$\mathcal{A}$ *wins if* $b = b'$.

2.4 RAM Computation

A single-program RAM computation Π is specified by a program P and an initial memory mem^0. The evaluator prepares the initial state st^0 and mem^0, and converts P to a stateful algorithm F. At each time t, F is executed with the state $\mathsf{st}^{\mathsf{in}}$ provided by the previous time step and a read $a^{\mathsf{in}}_{\mathtt{A}\leftarrow\mathtt{M}}$ from the memory as input, and outputs a new state $\mathsf{st}^{\mathsf{out}}$ for the next step and a memory access command $a^{\mathsf{out}}_{\mathtt{M}\leftarrow\mathtt{A}}$. Formally, it is written as $(\mathsf{st}^{\mathsf{out}}, a^{\mathsf{out}}_{\mathtt{M}\leftarrow\mathtt{A}}) \leftarrow F(\mathsf{st}^{\mathsf{in}}, a^{\mathsf{in}}_{\mathtt{A}\leftarrow\mathtt{M}})$ where an access denoted by $a = (I, b)$ includes a location and a value. In the following context, we sometimes interchangeably use program P or stateful algorithm F to denote the same RAM program.

Let us consider the persistent database setting. A multiple-program RAM computation is specified by a sequence of programs $\{P_i\}_{i=1}^{l}$ and an initial memory $\mathsf{mem}^{0,0}$. As above, the evaluator prepares initial state and memory, converts each P_i to the stateful algorithm F_i, and then runs these algorithms with intended order. In particular, each F_i at the beginning will use the current memory left by the termination of F_{i-1}.

3 Abstract Proof

In this section, we present an abstract framework of proofs for showing the indistinguishability of two cryptographic experiments agaisnt selective adversaries, and show how to use a proofs in this framework to show indistinguishability against even adaptive adversaries.

Notations We denote a non-uniform interactive (Turing) machine as a family of Turning machines $\mathcal{M} = \{M_\lambda\}$, one per *first input* length $\lambda \in \mathbb{N}$ (with the non-uniform advice hardwired in). The run time of $\mathcal{M}$ is measured with respect to λ. We use the convention that ensembles are denoted using capital letters $\mathcal{A}, \mathcal{B}, \cdots$ in calligraphic typeface, while elements in the ensembles are denoted using the same capital letter in standard math font $A, B, \cdots$. For any two interactive Turing machines A and B, we write $A \leftrightarrow B$ as the compound machine that on input x internally runs the interaction between A and B with common input x

and outputs what A outputs in the end. Similarly, for two non-uniform interactive Turing machines $\mathcal{A}$ and $\mathcal{B}$, we write $\mathcal{A} \leftrightarrow \mathcal{B}$ to denote the family of machines $\{A_\lambda \leftrightarrow B_\lambda\}$.

For two probability distributions D_1 and D_2 with the same support S, we denote by $\Delta(\mathsf{D}_1, \mathsf{D}_2)$ the statistical distance between then, defined as $\Delta(\mathsf{D}_1, \mathsf{D}_2) = 1/2\Sigma_{s\in S}| \Pr[s' \overset{\$}{\leftarrow} \mathsf{D}_1 \ : \ s' = s] - \Pr[s' \overset{\$}{\leftarrow} \mathsf{D}_2 \ : \ s' = s]|$.

3.1 Cryptographic Experiments and Games

We define standard cryptographic experiments and games between two parties, a challenger CH and an adversary A. The challenger defines the procedure and output of the experiment (or game), whereas the adversary can be any probabilistic interactive machine.

Definition 8 (Canonical Experiments). *A canonical experiment between two probabilistic interactive machines, the* challenger CH *and the* adversary A*, with security parameter* $\lambda \in \mathbb{N}$*, denoted as* $\mathsf{Exp}(\lambda, CH, A)$*, has the following form:*

- *CH and A receive common input 1^λ, and interact with each other.*
- *After the interaction, A writes an output γ on its output tape. In case A aborts before writing to its output tape, its output is set to $\perp$.*
- *CH additionally receives the output of A (receiving $\perp$ if A aborts), and outputs a bit b indicating accept or reject. (CH never aborts.)*

We say A wins *whenever CH outputs 1 in the above experiment.*

A canonical game (CH, τ) has additionally a threshold $\tau \in [0, 1)$. We say A has advantage γ if A wins in with probability $\tau + \gamma$ in $\mathsf{Exp}(\lambda, CH, A)$.

For machine $\star \in \{CH, A\}$, we denote by $\mathsf{Out}_\star(\lambda, CH, A)$ and $\mathsf{View}_\star(\lambda, (CH, A)$ the random variables describing the output and view of machine $\star$ in $\mathsf{Exp}(\lambda, CH, A)$.

Definition 9 (Cryptographic Experiments and Games). *A cryptographic experiment is defined by an ensemble of PPT challengers $\mathcal{CH} = \{CH_\lambda\}$. And a* cryptographic game $(\mathcal{CH}, \tau)$ *has additionally a threshold $\tau \in [0, 1)$. We say that a non-uniform adversary $\mathcal{A} = \{A_\lambda\}$ wins the cryptographic game with* advantage $\mathsf{Advt}(\star)$*, if for every $\lambda \in \mathbb{N}$, its advantage in $\mathsf{Exp}(\lambda, CH_\lambda, A_\lambda)$ is $\tau + \mathsf{Advt}(\lambda)$.*

Definition 10 (Intractability Assumptions). *An intractiablity assumption $(\mathcal{CH}, \tau)$ is the same as a cryptographic game, but with* potentially unbounded *challengers. It states that the advantage of every non-uniform* PPT *adversary $\mathcal{A}$ is negligible.*

For most cryptographic primitives, the security of an implementation of it can be modeled as an intractability assumption. For instance, in the security game of a puncturable PRF, the challenger upon receiving an input i from the adversary, sends back a key K_{-i} punctured at input i, together with either a random string or the pseudo-random output of the PRF on input i, decided by a random coin toss b. The challenger outputs 1 whenever A sends back $b' = b$, and the threshold of the game is 1/2. Here, the challenger is efficient, while for some other primitives, the challenger is unbounded. In the security game of an $\mathsf{i}\mathcal{O}$ scheme for circuits, the challenger upon receiving two circuits C_1 and C_2 first checks whether C_1 and C_2 are functionally equivalent (using unbounded computation power), and if so, it sends the adversary the obfuscation of a randomly picked circuit and checks whether the adversary guesses correctly the obfuscation of which circuit is sent. The threshold is again $1/2$.

3.2 Generalized Cryptographic Game

In the literature, experiments (or games) for selective security and adaptive security are often defined separately: In the former, the challenger requires the adversary to choose certain information at the beginning of the interaction, whereas in the latter, the challenger does not require such information.

We generalize stardard cryptographic experiments so that the same experiment can work with both selective and adaptive adversaries. This is achieve by separating information necessary for the execution of the challenger and information an adversary chooses statically, which can be viewed as a property of the adversary. More specifically, we consider adversaries that have a *special output tape*, and write information α it chooses statically at the beginning of the execution on it; and only the necessary information specified by a function, $G(\alpha)$, is sent to the challenger. This allows executing the same experiment with adversaries capturing different levels of selectivity/adaptivity.

Definition 11 (Generalized Experiments and Games)**.** *A* generalized experiment *between a challenger CH and an adversary A* with respect to a function G*, with security parameter $\lambda \in \mathbb{N}$, denoted as $\mathsf{Exp}(\lambda, CH, G, A)$, has the following form:*

1. *The adversary A on input 1^λ writes on its special output tape string α at the beginning of its execution, called the initial choice of A, and then proceeds as a normal probabilistic interactive machine. (α is set to the empty string ε if A does not write on the special output tape at the beginning.)*
2. *Let $A[G]$ denote the adversary that on input 1^λ runs A with the same security parameter internally; upon A writing α on its special output tape, it sends out message $m_1 = G(\alpha)$, and later forwards messages A sends, $m_2, m_3, \cdots$*
3. *The generalized experiment proceeds as a standard experiment between CH and $A[G]$, $\mathsf{Exp}(\lambda, CH, A[G])$.*

We say that A wins *whenever CH outputs* 1*.*

Furthermore, for any function $F : \{0,1\}^* \to \{0,1\}^*$, we say that A is F-selective in $\mathsf{Exp}(\lambda, CH, G, A)$, if it holds with probability 1 that either A aborts or its initial choice α and messages it sends satisfy that $F(\alpha) = F(m_2, m_3, \cdots)$. We say that A is adaptive, in the case that F is a constant function.

Similar to before, we denote by $\mathsf{Out}_\star(\lambda, CH, G, A)$ and $\mathsf{View}_\star(\lambda, CH, G, A)$ the random variables describing the output and view of machine $\star \in \{CH, A\}$ in $\mathsf{Exp}(\lambda, CH, G, A)$. In this work, we restrict our attention to these function G that are efficiently computable, as well as, *reversely computable*, meaning that given a value y in the domain of G, there is an efficient procedure that can output an input x such that $G(x) = y$. Below, we assume implicitly that functions in a generalized experiment is efficiently and reversely computable.

Definition 12 (Generalized Cryptographic Experiments and $\mathcal{F}$-Selective Adversaries)**.** *A generalized cryptographic experiment is a tuple* $(\mathcal{CH}, \mathcal{G})$*, where* $\mathcal{CH}$ *is an ensemble of PPT challengers* $\{CH_\lambda\}$ *and* $\mathcal{G}$ *is an ensemble of efficiently computable functions* $\{G_\lambda\}$*. Furthermore, for any an ensemble of functions* $\mathcal{F} = \{F_\lambda\}$ *mapping* $\{0,1\}^*$ *to* $\{0,1\}^*$*, we say that a non-uniform adversary* $\mathcal{A}$ *is* $\mathcal{F}$*-selective in cryptographic experiments* $(\mathcal{CH}, \mathcal{G})$ *if for every* $\lambda \in \mathbb{N}$*,* A_λ *is* F_λ*-selective in experiment* $\mathsf{Exp}(\lambda, CH_\lambda, G_\lambda, A_\lambda)$*.*

Similar to Definition 9, a generalized cryptographic experiment can be extended to a *generalized cryptographic game* $(\mathcal{CH}, \mathcal{G}, \tau)$ by adding an additional threshold $\tau \in [0, 1)$, where the advantage of any non-uniform probabilistic adversary $\mathcal{A}$ is defined identically as before.

Let us consider some examples of classes of adversary with different levels of selectivity.

- In one extreme, a *completely selective adversary* can choose all messages at the beginning of the execution and simply replay them later; this corresponds to F_λ being the identity function.
- Selective security for most cryptographic primitives (e.g., public-key encryption, or RAM delegation) considers an adverary that chooses certain "challenges messages" ahead of time (e.g., the pair of strings to be encrypted in the CPA security game, or the database and sequence of RAMs to be delegated in the security game of a RAM delegation scheme); this corresponds to a projection function $F_\lambda(\rho)$ that outputs only the challenge messages from the transcript ρ.
- Another type of adversaries that we consider later are $\mathcal{G}$-selective adversaries, who commits to the bare minimal information required by CH_λ statically, and other adaptively — we call them *semi-adaptive adversaries.*
- At the other end of the spectrum are *fully adaptive adversaries* who do not commit to any information at the beginning, and choose all messages adaptively, that is, F_λ is a constant function.

3.3 "Nice" Reductions

Standard straight-line black-box security reduction from a cryptographic game to an intractability assumption is a PPT machine R that interacts simultaneously with an adversary and the challenger of the assumption. The correctness of the reduction requires that if the adversary wins the game with certain advantage γ, then reduction (together with the adversary) wins the assumption with some related, potentially smaller, advantage γ'; the decrease in advantage is often referred to as the *security loss.* Since our generalized cryptographic games can be viewed as standard cryptographic games with adversaries of the form $\mathcal{A}[\mathcal{G}] = \{A_\lambda[G_\lambda]\}$, the standard notion of reductions extends naturally, by letting the reductions interact with adversaries of the form $\mathcal{A}[\mathcal{G}]$. Recall that $X \leftrightarrow Y$ denotes the compound machine running X and Y internally and outputting what X outputs. Formally,

Definition 13 (Reductions)**.** *A probabilistic interactive machine R is a (*straight-line black-box*) reduction from a generalized game (CH, G, τ) to a (canonical) game (CH', τ') for security parameter λ, if it has the following syntax:*

- *Syntax: On common input 1^λ, R interacts with CH' and an adversary $A[G]$ simultaneously in a straight-line—referred to as "left" and "right" interactions respectively. The left interaction proceeds identically to the experiment $\mathsf{Exp}(\lambda, CH', R \leftrightarrow A[G])$, and the right to experiment $\mathsf{Exp}(\lambda, CH' \leftrightarrow R, A[G])$.*

We say that R is (η, δ)-correct with respect to a family of adversaries $\mathcal{C}$, if

- *(η, δ)-Correctness: For every adversary $A \in \mathcal{C}$ with advantage γ in winning the generalized game (CH, G, τ) with security parameter λ, $\mathcal{R} \leftrightarrow A[G]$ wins the game $(\mathcal{CH}, \tau')$ with advantage $\gamma' = \eta(\gamma) - \delta(\lambda)$ (for the same security parameter).*

The above functions η and δ determines the security loss related to applying the reduction R. η captures how much the advantage of the reduction (together with the adverary) scales down compared to the advantage of the adversary, whereas, δ captures an additive decrease depending on the security parameter. We believe that using these two functions, one can describe the security loss of most (security) reductions, where often η is a polynomial while δ is negligible.

Definition 14. *A (*straight-line black-box*) reduction from an ensemble of generalized cryptographic game $(\mathcal{CH}, \mathcal{G}, \tau)$ to an intractability assumption $(\mathcal{CH}', \tau')$ is an ensemble of PPT reductions $\mathcal{R} = \{R_\lambda\}$ from game $(CH_\lambda, G_\lambda, \tau)$ to (CH'_λ, τ') (for security parameter λ). Moreover, we say that $\mathcal{R}$ is* correct*, if there is a polynomial η and a negligible function δ, such that, R_λ is (η, δ)-correct for every λ.*

Next, we describe a syntactical property of a reduction w.r.t. a function μ, and show that reductions satisfying this property is automatically (η, δ)-correct for a large class of adversaries, where η is the identify function, and δ is polynomial in μ. At a high-level, for any reduction R from (CH, G, τ) to (CH', τ), the syntactical property requires that R (together with the challenger CH of the assumption) generates messages and output that are statistically close to the messages and output of the challenger CH' of the game, *at every step*.

More precisely, let $\rho = (m_1, a_1, m_2, a_2, \cdots, m_t, a_t)$ denote a transcript of messages and outputs in the interaction between CH and an adversary (or in the interaction between $CH' \leftrightarrow R$ and an adversary) where $\vec{m} = m_1, m_2, \cdots, m_{t-1}$ and m_t correspond to the messages and output of the adversary ($m_t = \perp$ if the adversary aborts) and $\vec{a} = a_1, a_2, \cdots, a_{t-1}$ and a_t corresponds to the messages and output of CH (or $CH' \leftrightarrow R$). A transcript ρ possibly appears in an interaction with CH (or $CH' \leftrightarrow R$) if when receiving $\vec{m}$, CH (or $CH' \leftrightarrow R$) generates $\vec{a}$ with non-zero probability. The syntactical property requires that for every prefix of a transcript that possibly appear in both interaction with CH and interaction with $CH' \leftrightarrow R$, the distributions of the next message or output generated by CH and that by $CH' \leftrightarrow R$ are statistically close. In fact, for our purpose later, it suffices to consider the prefixes of transcripts that are *G-consistent*: A transcript ρ is G-consistent if $\vec{m}$ satisfies that either

$m_t = \perp$ or $m_1 = G(m_2, m_3, \cdots, m_{t-1})$; in other words, ρ could be generated by a G-selective adversary.

Definition 15 (Nice Reductions)**.** *We say that a reduction R from a generalized game (CH, G, τ) to a (canonical) game (CH', τ) (with the same threshold) for security parameter λ is μ-*nice*, if it satisfies the following property:*

- *$\mu(\lambda)$-statistical emulation for G-consistent transcripts:*
  For every prefix $\rho = (m_1, a_1, m_2, a_2, \cdots, m_{\ell-1}, a_{\ell-1}, m_\ell)$ of a G-consistent transcript of messages that possibly appears in interaction with both CH and $CH' \leftrightarrow R$, the following two distributions are $\mu(\lambda)$-close:

$$\Delta(\mathsf{D}_{CH' \leftrightarrow R}(\lambda, \rho),\ \mathsf{D}_{CH}(\lambda, \rho)) \le \mu(\lambda)$$

  where $\mathsf{D}_M(\lambda, \rho)$ for $M = CH' \leftrightarrow R$ or CH is the distribution of the next message or output *a_ℓ generated by $M(1^\lambda)$ after receiving messages $\vec{m}$ in ρ, and conditioned on $M(1^\lambda)$ having generated $\vec{a}$ in ρ.*

Next, we show that when a reduction is μ-nice, it is also automatically correct for *all G-selective interactive machines*, with only a *small additive security loss* related to the statistical distance μ and the run-time t of the reduction.

Lemma 1. *Every μ-nice reduction R from (CH, G, τ) to (CH', τ) for security parameter λ is (η, δ)-correct for the class of G-selective (potentially unbounded) interactive machines, where η is the identify function $\eta(\gamma) = \gamma$, and $\delta(\lambda) = t(\lambda)\mu(\lambda)$, where $t(\lambda)$ is an upper bound on the run-time of R.*

Proof of Lemma 1. Let λ be the security parameter under consideration, $t = t(\lambda)$ and $\mu = \mu(\lambda)$. Fix any G-selective interactive machine A that wins the generalized game (CH, G, τ) with advantage γ, that is, in experiment $\mathsf{Exp}(\lambda, CH, A[G])$, the adversary $A[G]$ makes CH output 1 with probability $\tau + \gamma$. It is without loss of generality to assume that A is deterministic: If A is randomized, there must exist a random tape r such that when using r, A achieves advantage $\tau + \gamma$. Then we can construct another *deterministic* machine A' with r hardwired in and proceeds as A does using that random tape; clearly, A' achieves advantage $\tau + \gamma$.

Our goal is showing that R when interacting with $A[G]$, breaks the assumption (CH', τ') with advantage greater than or equal to $\gamma - t\mu$. Suppose this is not true, that is, in experiment $\mathsf{Exp}(\lambda, CH' \leftrightarrow R, A[G])$, $CH' \leftrightarrow R$ outputs 1 with probability less than $\tau + \gamma - t\mu$. We derive a contradiction below.

Below, for simplicity of notation, we set $M_0 = CH$ and $M_1 = CH' \leftrightarrow R$. The two experiments under consideration are $E_0 = \mathsf{Exp}(\lambda^*, M_0, A[G])$ and $E_1 = \mathsf{Exp}(\lambda^*, M_1, A[G])$. Consider the interaction between M_b and $A[G]$ in experiment E_b. Let $\mathsf{Trans}_{b,i}$ denote the distribution of length-i prefix ρ of the transcript of messages and outputs in E_b. If i is even, $\rho = m_1, a_1, \cdots, m_{i/2}, a_{i/2}$; if i is odd, $\rho = m_1, a_1, \cdots, a_{(i-1)/2}, m_{(i-1)/2+1}$. We assume that for a transcript of length smaller than i, its length-i prefix is the transcript appended with $\top$

to length i. Since R runs at most t steps, $\mathsf{Trans}_{b,t}$ is the distribution of the full transcript in E_b. Since A is G-selective, every ρ in the support of $\mathsf{Trans}_{b,t}$ is G-consistent.

We denote by d_i the statistical distance between $\mathsf{Trans}_{0,i}$ and $\mathsf{Trans}_{1,i}$.

$$d_i = \Delta(\mathsf{Trans}_{0,i},\ \mathsf{Trans}_{1,i})$$

Clearly $d_0 = 0$ and $d_i \leq d_{i+1}$. Our premise and hypothesis state that the probabilities that M_0 and M_1 outputs 1 in E_0 and E_1 differ by more than $t\mu$, and hence, $d_t > t\mu$. Then, there must be an index i, such that, the gap between the statistical distances d_i and d_{i+1} is more than μ, that is, $d_{i+1} > d_i + \mu$

We first observe that if i is even and the $i+1^{\text{th}}$ message is from $A[G]$, then $d_{i+1} = d_i$. Since $A[G]$ is deterministic, the $i+1^{\text{th}}$ message is determined by the previous i messages, and thus $d_{i+1} \leq d_i$. Combining with the fact that $d_i \leq d_{i+1}$, we have $d_i = d_{i+1}$. If i is odd and the $i+1^{\text{th}}$ message is from M_b, we show that by the statistical emulation property of R, the gap between d_i and d_{i+1} is no larger than μ, which gives a contradiction. Below, we prove this fact.

Let Γ_b be the set of *length-i* prefix in the support of $\mathsf{Trans}_{b,i+1}$ (i.e., the set of length-i prefix that appear with non-zero probability in E_b), and let Γ be their intersection $\Gamma_0 \cap \Gamma_1$ (i.e., the set of length-i prefix that appear with non-zero probability in both E_0 and E_1). The statistical distance d_i can be divided into three parts,

$$\begin{aligned}
2d_{i+1} &= p + p_0 + p_1 \\
p &= \Sigma_{\rho\in\Gamma}\ \Sigma_a \left|\mathsf{Trans}_{0,i+1}(\rho||a) - \mathsf{Trans}_{1,i+1}(\rho||a)\right| \\
p_b &= \Sigma_{\rho\in\Gamma_b-\Gamma}\ \Sigma_a\ \mathsf{Trans}_{b,i+1}(\rho||a)
\end{aligned}$$

where $\mathsf{Trans}_{b,i+1}(\rho||a)$ is the probability that prefix $\rho||a$ (of length $(i+1)$) appears according to distribution $\mathsf{Trans}_{b,i+1}$. The probability p_b is exactly the probability that some prefix $\rho \in \Gamma_b - \Gamma$ appear in experiment E_b.

$$p_b = \Pr[\text{Some } \rho \in \Gamma_b - \Gamma \text{ appears in } E_b]$$

We can further expand the first term p,

$$p = \Sigma_{\rho\in\Gamma}\ \Sigma_a \left| \mathsf{D}_{M_0}[\lambda^*, \rho](a) \times \Pr[\rho \in \Gamma \text{ appears in } E_0] \right. \\ \left. - \mathsf{D}_{M_1}[\lambda^*, \rho](a) \times \Pr[\rho \in \Gamma \text{ appears in } E_1] \right|$$

Where $\mathsf{D}_{M_b}[\lambda^*, \rho](a)$ is the probability that M_b generates a as the next message or output conditioned on prefix ρ occurring in E_b. By the statistical emulation property of R, for every prefix ρ of a G-consistent transcript that appears in both E_0 and E_1, the distance between these two conditional distributions $\mathsf{D}_{M_0}[\lambda^*, \rho]$ and $\mathsf{D}_{M_1}[\lambda^*, \rho]$ is bounded by μ. Then,

$$\begin{aligned}
p \leq \Sigma_{\rho\in\Gamma}\ \Sigma_a\ \Big(\Big| & \mathsf{D}_{M_0}[\lambda^*, \rho](a) \times \Pr[\rho \in \Gamma \text{ appears in } E_0] \\
& - \mathsf{D}_{M_1}[\lambda^*, \rho](a) \times \Pr[\rho \in \Gamma \text{ appears in } E_0]\Big| \\
& + \Big|\mathsf{D}_{M_1}[\lambda^*, \rho](a) \times \Pr[\rho \in \Gamma \text{ appears in } E_0] \\
& \qquad - \mathsf{D}_{M_1}[\lambda^*, \rho](a) \times \Pr[\rho \in \Gamma \text{ appears in } E_1]\Big|\Big)
\end{aligned}$$

We note that the first two lines above correspond to the following:

$$\begin{aligned}
& \Sigma_{\rho\in\Gamma}\ \Sigma_a\ \Big|\mathsf{D}_{M_0}[\lambda^*, \rho](a) \times \Pr[\rho \in \Gamma \text{ appears in } E_0] - \mathsf{D}_{M_1}[\lambda^*, \rho](a) \times \Pr[\rho \in \Gamma \text{ appears in } E_0]\Big| \\
= \ & \Sigma_{\rho\in\Gamma}\ \Pr[\rho \in \Gamma \text{ appears in } E_0] \times \Big(\Sigma_a\ \Big|\mathsf{D}_{M_0}[\lambda^*, \rho](a) - \mathsf{D}_{M_1}[\lambda^*, \rho](a)\Big|\Big) \\
\leq \ & \Sigma_{\rho\in\Gamma}\ \Pr[\rho \in \Gamma \text{ appears in } E_0] \times\ 2\mu \\
\leq \ & 2\mu
\end{aligned}$$

The last two lines above correspond to the following:

$$\begin{aligned}
q &= \Sigma_{\rho\in\Gamma}\ \Sigma_a\ \Big|\mathsf{D}_{M_1}[\lambda^*, \rho](a) \times \Pr[\rho \in \Gamma \text{ appears in } E_0] - \mathsf{D}_{M_1}[\lambda^*, \rho](a) \times \Pr[\rho \in \Gamma \text{ appears in } E_1]\Big| \\
&= \Sigma_{\rho\in\Gamma}\ \Big|\Pr[\rho \in \Gamma \text{ appears in } E_0] - \Pr[\rho \in \Gamma \text{ appears in } E_1]\Big| \times \Big(\Sigma_a\ \mathsf{D}_{M_1}[\lambda^*, \rho](a)\Big) \\
&= \Sigma_{\rho\in\Gamma}\ \Big|\Pr[\rho \in \Gamma \text{ appears in } E_0] - \Pr[\rho \in \Gamma \text{ appears in } E_1]\Big|
\end{aligned}$$

Therefore, $p \leq 2\mu + q$ and $2d_{i+1} \leq 2\mu + p_0 + p_1 + q$. However, note that $p_0 + p_1 + q$ is bounded by twice the statistical distance $2d_i$ between the distributions of length-i prefixes. Thus, we get that $d_{i+1} \leq d_i + \mu$, which is a contradiction. □

3.4 Nice Proof

In this section, we characterize an abstract framework of proofs—called "nice" proofs—for showing the indistinguishability of two ensembles of (standard) cryptographic experiments $\mathcal{RL}_0$ and $\mathcal{RL}_1$. Roughly speaking, a standard indistinguishability proof consists of a sequence of hybrid experiments and shows that neighboring hybrids are indistinguishable via a reduction to a intractability assumption. We say that a proof is "nice" if the hybrid experiments, formalized as generalized cryptographic experiments, and the reductions have certain special properties. We show in the next section that these special properties allow for "lifting a nice proof" to prove security against adaptive adversaries. Below, we start with defining indistinguishability of generalized cryptographic experiments against $\mathcal{F}$-selective adversaries.

Indistinguishability Against $\mathcal{F}$-Selective Adversaries: Consider two arbitrary ensembles of generalized cryptographic experiments $(\mathcal{CH}_0, \mathcal{G}_0)$ and $(\mathcal{CH}_1, \mathcal{G}_1)$. These experiments can be run with adversaries with different levels of selectivity: For an ensemble $\mathcal{F}$ of efficiently computable functions, let $\mathcal{C}_{\mathcal{F}}$ denote the class of non-uniform PPT adversaries $\mathcal{A}$ that are $\mathcal{F}$-selective in $(\mathcal{CH}_b, \mathcal{G}_b)$ for both $b = 0, 1$. Indistinguishability of $(\mathcal{CH}_0, \mathcal{G}_0)$ and $(\mathcal{CH}_1, \mathcal{G}_1)$, against (efficient) $\mathcal{F}$-selective adversaries is defined as:

Definition 16 (Indistinguishability against $\mathcal{F}$-selective adversaries)**.** *Let $\mathcal{F}$ be an ensemble of efficiently computable functions. We say that two ensembles of generalized cryptographic experiments $(\mathcal{CH}_0, \mathcal{G}_0)$ and $(\mathcal{CH}_1, \mathcal{G}_1)$ are indistinguishable to $\mathcal{F}$-selective adversaries, if for any adversary $\mathcal{A}$ in $\mathcal{C}_F$, it holds that, for every $\lambda \in \mathbb{N}$, $CH_b = CH_{b,\lambda}$, $A = A_\lambda$, $G_b = G_{b,\lambda}$,*

$$|\Pr[\mathsf{Out}_A(\lambda, CH_0, G_0, A) = 1] - \Pr[\mathsf{Out}_A(\lambda, CH_1, G_1, A) = 1]| \leq \mathsf{negl}(\lambda)$$

Moreover, we say that $(\mathcal{CH}_0, \mathcal{G}_0)$ and $(\mathcal{CH}_1, \mathcal{G}_1)$ are indistinguishable to adaptive adversaries if they are indistinguishable to $\mathcal{F}$-selective adversaries, where $\mathcal{F}$ is a constant function.

It follows from classical argument that the above definition is equivalent to requiring that no (efficient) $\mathcal{F}$-selective adversaries can win a "guessing" game where the adversary participates in one of the two experiments chosen at random and tries to guess which experiment it is in.

Claim 1. *Definition 16 is equivalent to requiring that for any adversary $\mathcal{A}$ in $\mathcal{C}_F$, its advantage in the following generalized cryptographic game $(\mathcal{D}, \mathcal{G}_0||\mathcal{G}_1, 1/2)$ is negligible, where $\mathcal{G}_0||\mathcal{G}_1 = \{G_{0,\lambda}||G_{1,\lambda}\}$ are the concatenations of functions $G_{0,\lambda}$ and $G_{1,\lambda}$, and the challenger $\mathcal{D} = \{D_\lambda[CH_{0,\lambda}, CH_{1,\lambda}]\}$ proceeds as follows: For every security parameter $\lambda \in \mathbb{N}$, $D = D_\lambda[CH_{0,\lambda}, CH_{1,\lambda}]$, $G_b = G_{b,\lambda}$, $CH_b = CH_{b,\lambda}$, in experiment $\mathsf{Exp}(\lambda, D, G_0||G_1, \star)$,*

- *D tosses a random bit $b \xleftarrow{\$} \{0, 1\}$.*
- *Upon receiving $g_0||g_1$ (corresponding to $g_d = G_d(\alpha)$ for $d = 0, 1$ where α is the initial choice of the adversary), D internally runs challenger CH_b by feeding it g_b and forwarding messages to and from CH_b.*
- *If the adversary aborts, D output 0. Otherwise, upon receiving the adversary's output bit b', it output 1 if and only if $b = b'$.*

Abstract Proof Framework of Selective Security Consider two ensembles of standard cryptographic experiments $\mathcal{RL}_0$ and $\mathcal{RL}_1$. They are special cases of generalized cryptographic experiments with a function $\mathsf{null} : \{0, 1\}^* \to \{\varepsilon\}$ that always outputs the empty string, that is, $(\mathcal{RL}_0, \mathsf{null})$ and $(\mathcal{RL}_1, \mathsf{null})$; we refer to them as the "real" experiments.

Consider proving the indistinguishability of $\mathcal{RL}_0, \mathsf{null})$ and $(\mathcal{RL}_1, \mathsf{null})$ against $\mathcal{F}$-selective adversaries. Below we first describe a common high-level structure shared by most proof of indistinguishability in the literature, and then highlight the special feature of a "nice" proof, which allows for lifting the proof to work with adaptive adversaries. Below, we use the notation $\mathcal{X}_c$ to denote an ensemble of the form $\{X_{c,\lambda}\}$, and the notation $\mathcal{X}_I$ with a function

I, as the ensemble $\{X_{I(\lambda),\lambda}\}$; we further slightly abuse the notation to write $\mathcal{X}_{I+1}$ as the ensemble with function $I'(\lambda) = I(\lambda) + 1$, and

1. **Security based on intractability assumptions:** The indistinguishability is based on a set β of intractability assumptions $\{(\mathcal{CH}^\kappa, \tau^\kappa)\}$, where $\kappa \in [\beta]$.

2. **Security via hybrid argument:** The proof involves a sequence of *polynomial* number $\ell(\star)$ of hybrid experiments. More precisely, for every $\lambda \in \mathbb{N}$, there is a sequence of $\ell(\lambda) + 1$ hybrid (generalized) experiments $(H_{0,\lambda}, G_{0,\lambda}), \cdots (H_{\ell(\lambda),\lambda}, G_{\ell(\lambda),\lambda})$, such that, the "end" experiments matches the real experiments, that is,
$$(\mathcal{H}_0, \mathcal{G}_0) = (\{H_{0,\lambda}\}, \{G_{0,\lambda}\}) = (\mathcal{RL}_0, \mathsf{null})$$
$$(\mathcal{H}_\ell, \mathcal{G}_\ell) = (\{H_{\ell(\lambda),\lambda}\}, \{G_{\ell(\lambda),\lambda}\}) = (\mathcal{RL}_1, \mathsf{null}) ,$$
and an $\mathcal{F}$-selective adversary $\mathcal{A}$ in the real experiments $\{(\mathcal{RL}_b, \mathcal{U}_b)\}$ is also an $\mathcal{F}$-selective adversary in all ensembles of intermediate hybrids $(\mathcal{H}_I, \mathcal{G}_I) = (\{H_{I(\lambda),\lambda}\}, \{G_{I(\lambda),\lambda}\})$ for $0 \le I(\lambda) \le \ell(\lambda)$. (This usually holds as hybrids have the same interface as the real experiments when interacting with the adversary.)

3. **Indistinguishability of neighboring hybrids via reductions:** Neighboring hybrids are shown indistinguishable to $\mathcal{F}$-selective adversaries via a reduction to one of the β intractability assumptions. More precisely, for every λ and $0 \le i < \ell(\lambda)$, there is a reduction $R_{i,\lambda}$ from the guessing experiment $(D_{i,\lambda} = \mathcal{D}[H_{i,\lambda}, H_{i+1,\lambda}], G_{i,\lambda}||G_{i+1,\lambda})$ corresponding to distinguishing hybrids $H_{i,\lambda}, H_{i+1,\lambda}$, to one of the assumptions $CH_{i,\lambda} = CH_{\kappa,\lambda}$ for some $\kappa \in [\beta]$, such that,
   - Correctness w.r.t. $\mathcal{F}$-selective adversaries: There is a polynomial η and negligible function δ, such that, for every i, λ, $R_{i,\lambda}$ is (η, δ)-correct for every F_λ-selective adversary.

Given the above high-level structure, indistinguishability between $(\mathcal{RL}_0, \mathsf{null})$ and $(\mathcal{RL}_1, \mathsf{null})$ against $\mathcal{F}$-selective adversaries follows from standard arguments: Suppose for contradiction that there is a $\mathcal{F}$-selective adversary $\mathcal{A}$ that distinguishes the real experiments with some inverse polynomial probability $1/p(\lambda)$ for infinitely many $\lambda \in \mathbb{N}$. Then, the adversary $\mathcal{A}$ must distinguish a sequence of intermediate neighboring hybrids, $(\mathcal{H}_I, \mathcal{G}_I)$ and $(\mathcal{H}_{I+1}, \mathcal{G}_{I+1})$, for some $0 \le I(\lambda) < \ell(\lambda)$, with probability $1/q(\lambda) = 1/p(\lambda)\ell(\lambda)$ for infinitely many λ. For every such λ and $i = I(\lambda)$, by the η, δ-correctness of the reduction $R_{i,\lambda}$, $R_{i,\lambda} \leftrightarrow A_\lambda$ wins challenger $CH_{i,\lambda}$ with advantage $\eta(1/q(\lambda)) - \delta(\lambda)$. Since this happens for infinitely many λ and there are only a constant number of assumptions, there is an assumption $\kappa \in [\beta]$, such that, the ensemble of non-uniform machines $\mathcal{R}_I \leftrightarrow \mathcal{A}$ wins $\mathcal{CH}_\kappa$ with advantage $\eta(1/q(\lambda)) - \delta(\lambda)$ for an infinitely many λ. Since η is a polynomial and δ is negligible, this gives a contradiction.

The above proof framework considers only $\mathcal{F}$-selective adversaries. Towards our end goal of showing indistinguishability against adaptive adversaries, we consider "nice" proofs that follow this framework, but with two additional features.

3'. **Indistinguishability to semi-adaptive adversaries:** Neighboring hybrids are shown indistinguishable not only to $\mathcal{F}$-selective adversaries, but also, to *semi-adaptive adversaries.* More precisely, a semi-adaptive adversary distinguishing hybrids $H_{i,\lambda}$ and $H_{i+1,\lambda}$

is a $(G_{i,\lambda}||G_{i+1,\lambda})$-selective adversary in the guessing experiment $(D_{i,\lambda}, G_{i,\lambda}||G_{i+1,\lambda})$. A "nice" proof show that:

- Correctness w.r.t. semi-adaptive adversaries: There is a polynomial η and negligible function δ, such that, for every i, λ, $R_{i,\lambda}$ is (η, δ)-correct for every $(G_{i,\lambda}||G_{i+1,\lambda})$-selective adversary.

4. G_i **with polynomial-sized ranges:** There is a polynomial L, such that, the size of the range of every $G_{i,\lambda}$ function is bounded by $L(\lambda)$.

5. Hiding to adaptive adversaries G_i**:** The hybrids in a "nice" proof has the following additional property: In hybrid $(H_{i,\lambda}, G_{i,\lambda})$, the challenger's messages "hide" the $g = G_{i,\lambda}(\alpha)$ it receives at the beginning (recall α is the initial choice of the adversary), More precisely, if the challenger receives 0 instead (for simplicity, assume that 0 is in the range of $G_{i,\lambda}$), the challenger's messages remain indistinguishable to semi-selective, that is, $G_{i,\lambda}$-selective adversaries.

- G-Hiding: For any function I, experiments $(\mathcal{H}_I, \mathcal{G}_I)$ and $(\mathcal{H}_I, \mathbf{0})$ are indistinguishable to $\mathcal{G}_I$-selective adversaries, where $\mathbf{0}$ is the constant zero function.

In summary,

Definition 17 (Nice Indistinguishability Proof)**.** *A "nice" proof for the indistinguishability of two real experiments* $(\mathcal{RL}_0, \mathsf{null})$ *and* $(\mathcal{RL}_1, \mathsf{null})$ *is one that satisfy properties 1, 2, 3', 4 and 5 described above.*

Recall that Lemma 1 shows that nice reductions are automatically correct for all semi-adaptive adversaries. Therefore, property 3' is implied by the following stronger property 3":

3". Nice Reductions: The reductions for showing the indistinguishabilty of neighboring hybrids are "nice". More precisely,

- Nice Reductions: There is a polynomial t and a negligible function μ, such that, for every i, λ, the reduction $R_{i,\lambda}$ from the guessing experiment $(D_{i,\lambda}, G_{i,\lambda}||G_{i+1,\lambda})$ to assumption experiment (CH_i, τ_i) runs for at most $t(\lambda)$ steps and is μ-nice.

By Lemma 1, this implies that for every i, λ, $R_{i,\lambda}$ is (η, δ)-correct for the identity function η and $\delta(\lambda) = t(\lambda)\mu(\lambda)$, which is negligible.

Therefore,

Claim 2. *Every proof for the indistinguishability of two real experiments* $(\mathcal{RL}_0, \mathsf{null})$ *and* $(\mathcal{RL}_1, \mathsf{null})$ *satisfying properties 1, 2, 3", 4 and 5 is a "nice" proof.*

Later, we will show that the proof of the selectively secure RAM delegation scheme of [CCC+16] satisfy these properties and hence is a "nice" proof.

Finally, we remark that a "nice" proof not only applies to $\mathcal{F}$-selective adversaries, but also to the stronger semi-adaptive adversaries; but, it does not apply to adaptive adversaries. Nevertheless, in the next section, we show how to use a nice proof to show indistinguishability against adaptive adversaries in a generic way—we refer to this step as "lifting the nice proof".

3.5 Indistinguishability to Adaptive Adversaries

We show that if two ensembles of standard experiments $\mathcal{RL}_0$ and $\mathcal{RL}_1$ admit a "nice proof", then they are automatically indistinguishable to adaptive adversaries.

Theorem 3. *A "nice" proof for the indistinguishability of two ensembles of experiments $\mathcal{RL}_0$ and $\mathcal{RL}_1$ implies that these experiments are indistinguishable against adaptive adversaries.*

Proof. For every $\lambda \in \mathbb{N}$, let $(H_{0,\lambda}, G_{0,\lambda}), \cdots (H_{\ell(\lambda),\lambda}, G_{\ell(\lambda),\lambda})$ be the sequence of $\ell(\lambda) + 1$ hybrid (generalized) experiments considered in the "nice" proof, and $R_{i,\lambda}$ the reduction from the guessing game $(D_{i,\lambda} = \mathcal{D}[H_{i,\lambda}, H_{i+1,\lambda}], G_{i,\lambda}||G_{i+1,\lambda}, 1/2)$ to assumption $(CH_{i,\lambda}, \tau_{i,\lambda}) = (CH^{\kappa}_{\lambda}, \tau^{\kappa})$ for some $\kappa \in [\beta]$; $R_{i,\lambda}$ is (η, δ)-correct for all $G_{i,\lambda}||G_{i+1,\lambda}$-selective adversaries.

Suppose for contradiction that $(\mathcal{RL}_0, \mathsf{null})$ and $(\mathcal{RL}_1, \mathsf{null})$ is not indistinguishable to the class of non-uniform PPT *adaptive* adversaries. That is, there is a non-uniform PPT adaptive adversary $\mathcal{A} = \{A_\lambda\}$, and a polynomial p, such that for infinitely many λ,

$$|\Pr[\mathsf{Out}_{A_\lambda}(\lambda, Real_{0,\lambda}, \mathsf{null}, A_\lambda) = 1] - \Pr[\mathsf{Out}_{A_\lambda}(\lambda, Real_{1,\lambda}, \mathsf{null}, A_\lambda) = 1]| \geq 1/p(\lambda) \ .$$

We derive a contradiction below. For simplicity of notation, we suppress subscription of λ in the rest of the proof.

For every λ, consider the same sequence of hybrids $H_0, \cdots, H_\ell$, for $\ell = \ell(\lambda)$. But, instead of consider directly the output of the adaptive adversary A in the hybrid experiment H_i for $0 \leq i \leq \ell$, we consider running a *wrapper* adversary A'_{G_i} in H_i, where A'_X for an efficiently computable and reversely computable function X, on input 1^λ, runs A internally as follow:

- When A writes initial choice α on its special output tape, A'_X ignores α. Instead, it samples a random x from the range of function X, and finds an α' such that $X(\alpha') = x$, and writes α' on its own special output tape.
- It forwards messages to and from A externally.
- A'_X aborts whenever A aborts.
- If A does not abort and outputs bit b, A'_X checks whether its initial random guess x matches the output of function X applied to the transcript of messages ρ that A sends, $x = X(\rho)$. If it is the case, then A'_X outputs b as well. Otherwise, it aborts and writes $\perp$ on its special output tape; we denote this event err.

We remark that since A'_X aborts whenever its initial guess x does not match the messages A sends w.r.t. X, A'_X is a X-selective adversary. Furthermore if X is the function null, then A'_X would always guess correctly and event err never occurs, in which case, the output of A'_X is identical to that of A.

Given the wrapper adversaries, consider running A'_{G_i} in the experiment (H_i, G_i) and the probability p_i that A'_{G_i} outputs 1 conditioned even err does not occur:

$$p_i = \Pr\left[\mathsf{Out}_{A'_{G_i}}(\lambda, H_i, G_i, A'_{G_i}) = 1 \mid \neg\mathsf{err}\right]$$

Since (H_0, G_0) and (H_ℓ, G_ℓ) are exactly the real experiments $(Real_0, \mathsf{null})$ and $(Real_1, \mathsf{null})$ respectively. We have that

$$p_0 = \Pr\left[\mathsf{Out}_A(\lambda, Real_0, \mathsf{null}, A) = 1\right]$$
$$p_\ell = \Pr\left[\mathsf{Out}_A(\lambda, Real_1, \mathsf{null}, A) = 1\right]$$

By the contradiction hypothesis, for an infinitely many λ, p_0 and p_ℓ differ by $1/p(\lambda)$. For every such λ, there is an index $0 \le i < \ell$, such that, p_i and p_{i+1} differ by at least $1/p(\lambda)\ell(\lambda)$.

Towards deriving a contradiction, we want to apply the reduction R_i from the guessing experiment $(D_i, G_i||G_{i+1}, 1/2)$ to some assumption (CH_i, τ_i). However, the reduction is only (η, δ)-correct for $G_i||G_{i+1}$-selective adversaries. We observe that the distribution of output of A'_{G_i} in (H_i, G_i) (respectively, $A'_{G_{i+1}}$ in (H_{i+1}, G_{i+1})) is identical to that of $A'_{G_i||G_{i+1}}$ conditioned on err not occurring (in both experiments), that is,

$$p_i = \Pr\left[\mathsf{Out}_{A'_{G_i||G_{i+1}}}(\lambda, H_i, G_i, A'_{G_i||G_{i+1}}) = 1 \mid \neg\mathsf{err}\right]$$
$$p_{i+1} = \Pr\left[\mathsf{Out}_{A'_{G_i||G_{i+1}}}(\lambda, H_{i+1}, G_i, A'_{G_i||G_{i+1}}) = 1 \mid \neg\mathsf{err}\right]$$

By construction, $A'_{G_i||G_{i+1}}$ guesses $g_i||g_{i+1}$ at the beginning of the execution and checks in the end whether the messages from A matches its guess w.r.t. $G_i||G_{i+1}$. In (H_i, G_i), the challenger does not use g_{i+1}, hence the view of A emulated in $A'_{G_i||G_{i+1}}$ is information theoretically independent of g_{i+1}. Therefore, its probability of outputting 1 conditioned on both g_i and g_{i+1} guessed correctly is identically to that conditioned on only g_i is guessed correctly, which proves the first equation above. The second equation follows from the same proof.

Therefore, for infinitely many λ, adversary $A'_{G_i||G_{i+1}}$ distinguishes (H_i, G_i) and (H_{i+1}, G_{i+1}) with probability $1/p(\lambda)\ell(\lambda)$, conditioned on event err no occurring. Let I be the mapping from λ to such an index $0 \le i < \ell$ ($I(\lambda)$ maps to 0 when the aforementioned condition does not hold for this λ). Next we show that

Claim 3. *Let I be any function satisfying that $0 \le I(\lambda) < \ell(\lambda)$ for every $\lambda \in \mathbb{N}$. For every sufficiently large $\lambda \in \mathbb{N}$ and $i = I(\lambda)$, the probabilities that event* err *does not occur in experiments* $\mathsf{Exp}(\lambda, H_i, G_i, A'_{G_i||G_{i+1}})$ *and* $\mathsf{Exp}(\lambda, H_{i+1}, G_{i+1}, A'_{G_i||G_{i+1}})$ *are at least* $1/2(L(\lambda))^2$.

Proof. We give proof w.r.t. hybrid H_i. The case w.r.t. hybrid H_{i+1} follows syntactically from the same proof. For every $\lambda \in Nat$ and $i = I(\lambda)$, in $\mathsf{Exp}(\lambda, H_i, G_i, A'_{G_i||G_{i+1}})$, event err occurs when the strings g_i and g_{i+1} that $A'_{G_i||G_{i+1}}$ samples at random from the ranges of G_i and G_{i+1} do not match the transcript ρ of messages that A in $A'_{G_i||G_{i+1}}$ sends, that is, $G_i(\rho) \neq g_i$ or $G_{i+1} \neq g_{i+1}$. We first argue that the probability that $G_{i+1} = g_{i+1}$ is at least $1/L(\lambda)$. This is because, the execution of the challenger H_i in the experiment is independent of g_{i+1}, and hence the view of A in $A'_{G_i||G_{i+1}}$ is information theoretically independent of g_{i+1}. Therefore, the probability that the randomly chosen g_{i+1} matches $\mathcal{G}_{i+1}(\rho)$ is at least $1/L(\lambda)$.

Next, we show that the probability that for every sufficiently large λ, the probability that $G_i(\rho) = g_i$ is at least $1/2L(\lambda)$. Suppose for contradiction, that there are infinitely many

λ, such that this probability is less than $1/2L(\lambda)$. The G-hiding property of the hybrids states that the ensembles of experiments $(\mathcal{H}_I, \mathcal{G}_I)$ and $(\mathcal{H}_I, \mathcal{Z})$ are indistinguishable to all $\mathcal{G}_I$-selective adversaries, where $\mathcal{Z}$ is the ensemble of constant zero function $Z(x) = 0$ for all $x \in \{0,1\}^*$. Since for every λ, $A'_{G_i||G_{i+1}}$ is $G_i||G_{i+1}$-selective, and hence also G_i-selective, the hiding property applies to $A'_{G_i||G_{i+1}}$. Thus, for infinitely many λ, in the experiment $\mathsf{Exp}(\lambda, H_i, Z, A'_{G_i||G_{i+1}})$, the probability that $G_i(\rho) \neq g_i$ is less than $1/2L(\lambda) + \mathsf{negl}(\lambda) < 1/L(\lambda)$. However, in this experiment, the execution of the challenger no longer depends on the guess g_i, and hence the view of A in $A'_{G_i||G_{i+1}}$ is information theoretically independent of g_i. By the same argument above, the probability $g_i \neq G_i(\rho)$ is at least $1/L(\lambda)$, which gives a contradiction.

Finally, since g_i and g_{i+1} are sampled independently, the probability that err does not occur in experiment $\mathsf{Exp}(\lambda, H_i, Z, A'_{G_i||G_{i+1}})$ is thus at least $1/2(L(\lambda))^2$. □

By construction, whenever err occurs, A' aborts; therefore,

$$\begin{aligned}
&\Pr\left[\mathsf{Out}_{A'_{G_i||G_{i+1}}}(\lambda, H_i, G_i, A'_{G_i||G_{i+1}}) = 1\right] \\
=&\Pr\left[\mathsf{Out}_{A'_{G_i||G_{i+1}}}(\lambda, H_i, G_i, A'_{G_i||G_{i+1}}) \mid \neg\mathsf{err}\right] \times \Pr\left[\neg\mathsf{err} \text{ in } \mathsf{Exp}(\lambda, H_i, Z, A'_{G_i||G_{i+1}})\right] \\
\geq&\frac{p_i}{2L^2(\lambda)}
\end{aligned}$$

Similarly, the probability that A' outputs 1 in $\mathsf{Exp}(\lambda, H_{i+1}, Z, A'_{G_i||G_{i+1}})$ is no smaller than $p_{i+1}/2(L(\lambda))^2$.

Therefore, for infinitely many λ, the probabilities that A' outputs in 1 in the i^{th} and $i+1^{\text{th}}$ hybrid experiments differ by at least $1/q(\lambda) = 1/2p(\lambda)\ell(\lambda)(L(\lambda))^2$. For every such λ, it follows from standard arguments that $A'_{G_i||G_{i+1}}$ wins the guessing game $(D_i, G_i||G_{i+1}, 1/2)$ with advantage $1/q(\lambda) = 1/2p(\lambda)\ell(\lambda)$. Since $A'_{G_i||G_{i+1}}$ is $(G_i||G_{i+1})$-selective, by the (η, δ) correctness of reduction R_i, the compound machine $M_i = R_i \leftrightarrow A'_{G_i||G_{i+1}}[G_i||G_{i+1}]$ wins the assumption game (CH_i, τ_i) with advantage $\eta(1/q(\lambda)) - \delta(\lambda)$, which is non-negligible. Furthermore, as there are only a constant number of assumptions, there exists one assumption $\kappa \in [\beta]$, such that for infinitely many λ, M_i wins the assumption game (CH^κ, τ^κ) with non-negligible advantage, which gives a contradiction.

□

4 Adaptive Delegation for RAM computation

In this section, we introduce the notion of adaptive delegation for RAM computation ($\mathcal{DEL}$). In a $\mathcal{DEL}$ scheme, a client outsources the database encoding and then generates a sequence of program encodings. The server will evaluate those program encodings with intended order on the database encoding left over by the previous one. For security, we focus on *full privacy* where the server learns nothing about the database, delegated programs, and its outputs. Simultaneously, $\mathcal{DEL}$ is required to provide *soundness* where the client has to receive the correct output encoding from each program and current database.

4.1 Definition

Definition 18 ($\mathcal{DEL}$ with Persistent Database)**.** *A $\mathcal{DEL}$ scheme with persistent database, consists of algorithms $\mathcal{DEL} = \mathcal{DEL}.\{\mathsf{DBDel}, \mathsf{PDel}, \mathsf{Eval}, \mathsf{Ver}, \mathsf{Dec}\}$, is described below. Let* sid *be the program session identity where $1 \leq \mathsf{sid} \leq l$. We associate $\mathcal{DEL}$ with a class of programs $\mathcal{P}$.*

- $\mathcal{DEL}.\mathsf{DBDel}(1^\lambda, \mathsf{mem}^0, S) \rightarrow (\widetilde{\mathsf{mem}}^1, \mathsf{sk})$*: The database delegation algorithm* DBDel *is a randomized algorithm which takes as input the security parameter 1^λ, database mem^0, and a space bound S. It outputs a garbled database $\widetilde{\mathsf{mem}}^1$ and a secret key* sk*.*
- $\mathcal{DEL}.\mathsf{PDel}(1^\lambda, \mathsf{sk}, \mathsf{sid}, P_{\mathsf{sid}}) \rightarrow \widetilde{P}_{\mathsf{sid}}$*: The algorithm* PDel *is a randomized algorithm which takes as input the security parameter 1^λ, the secret key* sk*, the session ID* sid *and a description of a* RAM *program $P_{\mathsf{sid}} \in \mathcal{P}$. It outputs a program encoding $\widetilde{P}_{\mathsf{sid}}$.*
- $\mathcal{DEL}.\mathsf{Eval}(1^\lambda, T, S, \widetilde{P}_{\mathsf{sid}}, \widetilde{\mathsf{mem}}^{\mathsf{sid}}) \rightarrow (c_{\mathsf{sid}}, \sigma_{\mathsf{sid}}, \widetilde{\mathsf{mem}}^{\mathsf{sid}+1})$*: The evaluating algorithm* Eval *is a deterministic algorithm which takes as input the security parameter 1^λ, time bound T and space bound S, a garbled program $\widetilde{P}_{\mathsf{sid}}$, and the database $\widetilde{\mathsf{mem}}^{\mathsf{sid}}$. It outputs $(c_{\mathsf{sid}}, \sigma_{\mathsf{sid}}, \widetilde{\mathsf{mem}}^{\mathsf{sid}+1})$ or $\perp$, where c_{sid} is the encoding of the output y_{sid}, σ_{sid} is a proof of c_{sid}, and $(y_{\mathsf{sid}}, \mathsf{mem}^{\mathsf{sid}+1}) = P_{\mathsf{sid}}(\mathsf{mem}^{\mathsf{sid}})$.*
- $\mathcal{DEL}.\mathsf{Ver}(1^\lambda, \mathsf{sk}, c_{\mathsf{sid}}, \sigma_{\mathsf{sid}}) \rightarrow b_{\mathsf{sid}} \in \{0,1\}$*: The verification algorithm returns $b_{\mathsf{sid}} = 1$ if σ_{sid} is a valid proof for c_{sid}, or returns $b_{\mathsf{sid}} = 0$ if not.*
- $\mathcal{DEL}.\mathsf{Dec}(1^\lambda, \mathsf{sk}, c_{\mathsf{sid}}) \rightarrow y_{\mathsf{sid}}$*: The decoding algorithm* Dec *is a deterministic algorithm which takes as input the security parameter 1^λ, secret key* sk*, output encoding c_{sid}. It outputs y_{sid} by decoding c_{sid} with* sk*.*

Correctness. *A delegation scheme $\mathcal{DEL}$ is said to be* correct *if both verification and decryption are correct: for all $\mathsf{mem}^0 \in \{0,1\}^{\mathrm{poly}(\lambda)}, 1 \leq \mathsf{sid} \leq \ell, P_{\mathsf{sid}} \in \mathcal{P}$*

$$\begin{aligned}\Pr[&(\widetilde{\mathsf{mem}}^1, \mathsf{sk}) \leftarrow \mathcal{DEL}.\mathsf{DBDel}(1^\lambda, \mathsf{mem}^0, S); \widetilde{P}_{\mathsf{sid}} \leftarrow \mathcal{DEL}.\mathsf{PDel}(1^\lambda, \mathsf{sk}, \mathsf{sid}, P_{\mathsf{sid}});\\ &(c_{\mathsf{sid}}, \sigma_{\mathsf{sid}}, \widetilde{\mathsf{mem}}^{\mathsf{sid}+1}) \leftarrow \mathcal{DEL}.\mathsf{Eval}(1^\lambda, T, S, \widetilde{P}_{\mathsf{sid}}, \widetilde{\mathsf{mem}}^{\mathsf{sid}});\\ &b_{\mathsf{sid}} = \mathcal{DEL}.\mathsf{Ver}(1^\lambda, \mathsf{sk}, c_{\mathsf{sid}}, \sigma_{\mathsf{sid}}); y_{\mathsf{sid}} = \mathcal{DEL}.\mathsf{Dec}(1^\lambda, \mathsf{sk}, c_{\mathsf{sid}});\\ &(y'_{\mathsf{sid}}, \mathsf{mem}^{\mathsf{sid}+1}) \leftarrow P_{\mathsf{sid}}(\mathsf{mem}^{\mathsf{sid}}) \ : \ (y_{\mathsf{sid}} = y'_{\mathsf{sid}} \wedge b_{\mathsf{sid}} = 1) \ \forall \mathsf{sid}, 1 \leq \mathsf{sid} \leq l] = 1.\end{aligned}$$

Adaptive Security (full privacy). *A delegation scheme $\mathcal{DEL} = \mathcal{DEL}.\{\mathsf{DBDel}, \mathsf{PDel}, \mathsf{Eval}, \mathsf{Ver}, \mathsf{Dec}\}$ with persistent database is said to be* adaptively secure *if for all sufficiently large $\lambda \in \mathbb{N}$, for all total round $l \in \mathrm{poly}(\lambda)$, time bound T, space bound S, for every interactive PPT adversary $\mathcal{A}$, there exists an interactive PPT simulator $\mathcal{S}$ such that $\mathcal{A}$'s advantage in the following security game Exp-Del-Privacy$(1^\lambda, \mathcal{DEL}, \mathcal{A}, \mathcal{S})$ is at most negligible in λ.*

Exp-Del-Privacy$(1^\lambda, \mathcal{DEL}, \mathcal{A}, \mathcal{S})$

1. *The challenger $\mathcal{C}$ chooses a bit $b \in \{0,1\}$.*
2. *$\mathcal{A}$ chooses and sends database mem^0 to challenger $\mathcal{C}$.*

3. *If $b = 0$, challenger $\mathcal{C}$ computes* $(\widetilde{\mathsf{mem}}^1, \mathsf{sk}) \leftarrow \mathcal{DEL}.\mathsf{DBDel}(1^\lambda, \mathsf{mem}^0, S)$. *Otherwise, $\mathcal{C}$ simulates* $(\widetilde{\mathsf{mem}}^1, \mathsf{sk}) \leftarrow \mathcal{S}(1^\lambda, |\mathsf{mem}^0|)$, *where* $|\mathsf{mem}^0|$ *is the length of* mem^0. *$\mathcal{C}$ sends* $\widetilde{\mathsf{mem}}^1$ *back to $\mathcal{A}$.*

4. *For each round* sid *from 1 to l,*

   (a) $\mathcal{A}$ chooses and sends program P_{sid} to $\mathcal{C}$.

   (b) If $b = 0$, challenger $\mathcal{C}$ sends $\widetilde{P}_{\mathsf{sid}} \leftarrow \mathcal{DEL}.\mathsf{PDel}(1^\lambda, \mathsf{sk}, \mathsf{sid}, P_{\mathsf{sid}})$ *to $\mathcal{A}$. Otherwise, $\mathcal{C}$ simulates and sends* $\widetilde{P}_{\mathsf{sid}} \leftarrow \mathcal{S}(1^\lambda, \mathsf{sk}, \mathsf{sid}, 1^{|P_{\mathsf{sid}}|}, 1^{|c_{\mathsf{sid}}|}, T, S)$ *to $\mathcal{A}$.*

5. *$\mathcal{A}$ outputs a bit b'. $\mathcal{A}$ wins the security game if $b = b'$.*

We notice that an unrestricted adaptive adversary *can adaptively choose RAM programs P_i depending on the program encodings it receives, whereas a restricted* selective adversary *can only make the choice of programs statically at the beginning of the execution.*

Adaptive Soundness. *A delegation scheme $\mathcal{DEL}$ is said to be* adaptively sound *if for all sufficiently large $\lambda \in \mathbb{N}$, for all total round $l \in \text{poly}(\lambda)$, time bound T, space bound S, there exists an interactive PPT adversary $\mathcal{A}$, such that the probability of $\mathcal{A}$ win in the following security game* $\mathsf{\textit{Exp-Del-Soundness}}(1^\lambda, \mathcal{DEL}, \mathcal{A})$ *is at most negligible in λ.*

$\mathsf{\textit{Exp-Del-Soundness}}(1^\lambda, \mathcal{DEL}, \mathcal{A})$

1. *$\mathcal{A}$ chooses and sends database* mem^0 *to challenger $\mathcal{C}$.*

2. *The challenger $\mathcal{C}$ computes* $(\widetilde{\mathsf{mem}}^1, \mathsf{sk}) \leftarrow \mathcal{DEL}.\mathsf{DBDel}(1^\lambda, \mathsf{mem}^0, S)$. *$\mathcal{C}$ sends* $\widetilde{\mathsf{mem}}^1$ *back to $\mathcal{A}$.*

3. *For each round* sid *from 1 to l,*

   (a) $\mathcal{A}$ chooses and sends program P_{sid} to $\mathcal{C}$.

   (b) $\mathcal{C}$ sends $\widetilde{P}_{\mathsf{sid}} \leftarrow \mathcal{DEL}.\mathsf{PDel}(1^\lambda, \mathsf{sk}, \mathsf{sid}, P_{\mathsf{sid}})$ *to $\mathcal{A}$.*

4. *$\mathcal{A}$ outputs a triplet (k, c^*_k, σ^*_k). $\mathcal{A}$ wins the security game if* $1 \leftarrow \mathcal{DEL}.\mathsf{Ver}(1^\lambda, \mathsf{sk}, c^*_k, \sigma^*_k)$ *and $c^*_k \neq c_k$ for the k-th round.*

Efficiency. *For all session* sid, *we require* DBDel *and* PDel *runs in time $\tilde{O}(|\mathsf{mem}^0|)$ and $\tilde{O}(\text{poly}(|P_{\mathsf{sid}}|))$, and efficient* Eval *runs in time $\tilde{O}(t^*_{\mathsf{sid}})$. In addition, the length of $c_{\mathsf{sid}}, \sigma_{\mathsf{sid}}$ should depend only on $|y_{\mathsf{sid}}|$, and both* Ver *and* Dec *run in time $O(\text{poly}\,|y_{\mathsf{sid}}|)$.*

4.2 Construction of Adaptive RAM Delegation: Roadmap

Building towards adaptive RAM delegation, we first construct a new primitive, called adaptive computation-trace indistinguishability obfuscation ($\mathsf{Ci}\mathcal{O}$) with persistent database. We first define this primitive in Section 6. This notion is a strengthening of the notion of selective $\mathsf{Ci}\mathcal{O}$, introduced by [CCC+16], defined in CCC+. Then we show that by modifying the selective $\mathsf{Ci}\mathcal{O}$ scheme of CCC+, we achieve adaptive $\mathsf{Ci}\mathcal{O}$. A crucial change we make to the selective $\mathsf{Ci}\mathcal{O}$ scheme is that we replace the tool of positional accumulators with *history-less* *accumulators*; a notion we introduce in Section 5. Arguing that the modified selective $\mathsf{Ci}\mathcal{O}$ scheme implies adaptive $\mathsf{Ci}\mathcal{O}$ is quite involved: we first look at the sequence of reductions used by CCC+ in proving the security of selective $\mathsf{Ci}\mathcal{O}$ and then we cast these reductions in the abstract framework introduced in Section 3. This helps us in boosting reductions defined w.r.t semi-adaptive adversaries into reductions defined w.r.t adaptive adversaries. This would then imply that the modified CCC+ scheme is adaptively secure.

In the next step we consider the concept of adaptive garbled RAM in the persistent database setting. The definition of this primitive is provided in Section 7. We then show, in Section 7.2, how to achieve adaptive garbled RAM from adaptive $\mathsf{Ci}\mathcal{O}$. In the final step, we give a generic transformation to achieve adaptive RAM delegation from adaptive garbled RAM in the persistent setting. This is demonstrated in Section 8.

Extension to adaptive PRAM delegation. We adopt the strategy of CCC+ [CCC+16] to realize the adaptive PRAM delegation with persistent database. We provide a brief overview as to how to make this extension work. At first, we convert our adaptive $\mathsf{Ci}\mathcal{O}$ for RAM and branch-and-combine technique of [CCC+16] into the adaptive $\mathsf{Ci}\mathcal{O}$ for PRAM. Then, we construct our adaptive PRAM garbling from adaptive $\mathsf{Ci}\mathcal{O}$. Finally, the adaptive PRAM delegation is achieved with full privacy and soundness by applying the generic transformation. In this delegation scheme, the database delegation time (resp., program delegation time) depends only on database size (resp., program description size). The server's complexity matches the PRAM complexity of the computation up to polynomial factor of program description size.

5 History-less Accumulators

We introduce the notion of history-less accumulators. We explain later, the main difference between history-less accumulators and the accumulator scheme of KLW. We then demonstrate a construction of history-less accumulators based on decisional Diffie-Hellman (DDH) assumption. This construction, which follows along the footsteps of DDH-based construction of accumulators of Okamoto et al. [OPWW15], is divided into two main steps:

1. OPWW introduced the notion of two-to-one somewhere statistically binding hash. We consider a variant of this notion, called a extended two-to-one SPB hash, where SPB stands for somewhere perfectly binding hash. A two-to-one hash of OPWW is a hash function that takes two blocks of equal length as input and outputs a value whose length is slightly larger than that of a single block. The somewhere statistical binding property states that the hash function can be programmed such that the hash output statistically determines one of the input blocks. In the extended version, we need a stronger property that the hash output uniquely determines one of the input blocks. More importantly, we need this additional property, termed as *uniqueness of root*: suppose a hash output, h, uniquely determines the first block, say $\mathbf{x}_A$. Let the second block be $\mathbf{x}_B$. If there exists $\mathbf{x}'_B$ such that the hash output of $\mathbf{x}_A$ and $\mathbf{x}'_B$ is also h then for every $\mathbf{x}'_A$, we have that the hash output of $(\mathbf{x}'_A, \mathbf{x}_B)$ to be the same as the hash output of $(\mathbf{x}'_A, \mathbf{x}'_B)$. This

will come in handy in proving the write-enforcing property of the final accumulators scheme.

2. In the next step, we show how to go from extended two-to-one SPB hash to history-less accumulators. This construction is identical to the OPWW transformation from two-to-one hash to positional accumulators. The only difference is in the analysis – we show the uniqueness of root property of extended two-to-one SPB hash implies the write-enforcing property of history-less accumulators.

5.1 Definition

We now formally define the notion of history-less accumulators. The main difference between the positional accumulators of KLW and history-less accumulators is the following: in KLW, the "enforcing" parameters take as input the special index which is information-theoretically bound and also the history of computation till that point. In our case, the enforcing parameters only take as input the special index. We formally describe the scheme below.

A history-less accumulator, $\mathsf{hAcc} = \mathsf{hAcc}.\{\mathsf{Setup}, \mathsf{EnforceRead}, \mathsf{EnforceWrite}, \mathsf{PrepRead}, \mathsf{PrepWrite}, \mathsf{VerifyRead}, \mathsf{WriteStore}, \mathsf{Update}\}$ be an accumulator with message space $\mathcal{M}_\lambda$ for message space Msg_λ consists of the following algorithms.

$\mathsf{SetupAcc}(1^\lambda, T) \rightarrow \mathsf{PP}_{\mathsf{Acc}}, w_0, store_0$ The setup algorithm takes as input a security parameter λ in unary and an integer T in binary representing the maximum number of values that can stored. It outputs public parameters $\mathsf{PP}_{\mathsf{Acc}}$, an initial accumulator value w_0, and an initial storage value $store_0$.

$\mathsf{EnforceRead}(1^\lambda, T, \text{INDEX}^*) \rightarrow \mathsf{PP}_{\mathsf{Acc}}, w_0, store_0$ The setup enforce read algorithm takes as input a security parameter λ in unary, an integer T in binary representing the maximum number of values that can be stored, and an additional INDEX* between 0 and $T-1$. It outputs public parameters $\mathsf{PP}_{\mathsf{Acc}}$, an initial accumulator value w_0, and an initial storage value $store_0$.

$\mathsf{EnforceWrite}(1^\lambda, T, \text{INDEX}^*) \rightarrow \mathsf{PP}_{\mathsf{Acc}}, w_0, store_0$ The setup enforce write algorithm takes as input a security parameter λ in unary, an integer T in binary representing the maximum number of values that can be stored, and an INDEX* between 0 and $T-1$. It outputs public parameters $\mathsf{PP}_{\mathsf{Acc}}$, an initial accumulator value w_0, and an initial storage value $store_0$.

$\mathsf{PrepRead}(\mathsf{PP}_{\mathsf{Acc}}, store_{in}, \text{INDEX}) \rightarrow m, \pi$ The prep-read algorithm takes as input the public parameters $\mathsf{PP}_{\mathsf{Acc}}$, a storage value $store_{in}$, and an index between 0 and $T-1$. It outputs a symbol m (that can be ϵ) and a value π.

$\mathsf{PrepWrite}(\mathsf{PP}_{\mathsf{Acc}}, store_{in}, \text{INDEX}) \rightarrow aux$ The prep-write algorithm takes as input the public parameters $\mathsf{PP}_{\mathsf{Acc}}$, a storage value $store_{in}$, and an index between 0 and $T-1$. It outputs an auxiliary value aux.

$\mathsf{VerifyRead}(\mathsf{PP}_{\mathsf{Acc}}, w_{in}, m_{read}, \text{INDEX}, \pi) \to \{True, False\}$ The verify-read algorithm takes as input the public parameters $\mathsf{PP}_{\mathsf{Acc}}$, an accumulator value w_{in}, a symbol, m_{read}, an index between 0 and $T-1$, and a value π. It outputs $True$ or $False$.

$\mathsf{WriteStore}(\mathsf{PP}_{\mathsf{Acc}}, store_{in}, \text{INDEX}, m) \to store_{out}$ The write-store algorithm takes in the public parameters, a storage value $store_{in}$, an index between 0 and $T-1$, and a symbol m. It outputs a storage value $store_{out}$.

$\mathsf{Update}(\mathsf{PP}_{\mathsf{Acc}}, w_{in}, m_{write}, \text{INDEX}, aux) \to w_{out}$ **or** $Reject$ The update algorithm takes in the public parameters $\mathsf{PP}_{\mathsf{Acc}}$, an accumulator value w_{in}, a symbol m_{write}, and index between 0 and $T-1$, and an auxiliary value aux. It outputs an accumulator value w_{out} or $Reject$.

Except the algorithms EnforceRead and EnforceWrite, the rest of the algorithms described above are identical to the corresponding algorithms of the positional accumulator schemes.

Correctness. We consider any sequence $(m_1, \text{INDEX}_1), \ldots, (m_k, \text{INDEX}_k)$ of symbols $m_1, \ldots, m_k$ and indices $\text{INDEX}_1, \ldots, \text{INDEX}_k$ each between 0 and $T-1$. We fix any $\mathsf{PP}_{\mathsf{Acc}}, w_0, store_0 \leftarrow \mathsf{SetupAcc}(1^\lambda, T)$. For j from 1 to k, we define $store_j$ iteratively as $store_j := \mathsf{WriteStore}(\mathsf{PP}_{\mathsf{Acc}}, store_{j-1}, \text{INDEX}_j, m_j)$. We similarly define aux_j and w_j iteratively as $aux_j := \mathsf{PrepWrite}(\mathsf{PP}_{\mathsf{Acc}}, store_{j-1}, \text{INDEX}_j)$ and $w_j := Update(\mathsf{PP}_{\mathsf{Acc}}, w_{j-1}, m_j, \text{INDEX}_j, aux_j)$. Note that the algorithms other than SetupAcc are deterministic, so these definitions fix precise values, not random values (conditioned on the fixed starting values $\mathsf{PP}_{\mathsf{Acc}}, w_0, store_0$).

Security. Let $\mathsf{hAcc} = \mathsf{hAcc}.\{\mathsf{Setup}, \mathsf{EnforceRead}, \mathsf{EnforceWrite}, \mathsf{PrepRead}, \mathsf{PrepWrite}, \mathsf{VerifyRead}, \mathsf{WriteStore}, \mathsf{Update}\}$ be an accumulator with message space $\mathcal{M}_\lambda$.

We define the following security notions, which are a natural adaptation of the security properties in the positional accumulators scheme to the history-less setting.

Definition 19 (Indistinguishability of Read-Setup)**.** *A history-less accumulator* hAcc *is said to satisfy indistinguishability of Read-Setup phase if any PPT adversary $\mathcal{A}$'s advantage in the security game* ***Exp-Setup-Read***$(1^\lambda, \mathsf{hAcc}, \mathcal{A})$ *at most is negligible in λ, where* ***Exp-Setup-Read*** *is defined as follows.*

Exp-Setup-Read$(1^\lambda, \mathsf{hAcc}, \mathcal{A})$

- *The adversary $\mathcal{A}$ chooses a bound $S \in \Theta(2^\lambda)$ and sends it to challenger.*
- *$\mathcal{A}$ sends an index* $\text{INDEX}^* \in \{0, \ldots, S-1\}$.
- *The challenger chooses a bit b. If $b = 0$, the challenger outputs* $(\mathsf{PP}_{\mathsf{hAcc}}, w_0, store_0) \leftarrow \mathsf{hAcc}.\mathsf{Setup}(1^\lambda, S)$. *Else, it outputs* $(\mathsf{PP}_{\mathsf{hAcc}}, w_0, store_0) \leftarrow \mathsf{hAcc}.\mathsf{EnforceRead}(1^\lambda, S, \text{INDEX}^*)$.
- *$\mathcal{A}$ sends a bit b'.*

$\mathcal{A}$ wins the security game if $b = b'$.

Definition 20 (Indistinguishability of Write-Setup)**.** *A history-less accumulator* hAcc *is said to satisfy indistinguishability of Write-Setup phase if any PPT adversary* $\mathcal{A}$*'s advantage in the security game* $\boldsymbol{\mathit{Exp\text{-}Setup\text{-}Write}}(1^\lambda, \mathsf{hAcc}, \mathcal{A})$ *at most is negligible in* λ*, where* ***Exp-Setup-Write*** *is defined as follows.*

$\boldsymbol{\mathit{Exp\text{-}Setup\text{-}Write}}(1^\lambda, \mathsf{hAcc}, \mathcal{A})$

- *The adversary* $\mathcal{A}$ *chooses a bound* $S \in \Theta(2^\lambda)$ *and sends it to challenger.*
- $\mathcal{A}$ *sends an index* $\mathrm{INDEX}^* \in \{0, \ldots, S-1\}$.
- *The challenger chooses a bit* b*. If* $b = 0$*, the challenger outputs* $(\mathsf{PP}_{\mathsf{hAcc}}, w_0, store_0) \leftarrow \mathsf{hAcc.Setup}(1^\lambda, S)$*. Else, it outputs* $(\mathsf{PP}_{\mathsf{hAcc}}, w_0, store_0) \leftarrow \mathsf{hAcc.EnforceWrite}(1^\lambda, S, \mathrm{INDEX}^*)$.
- $\mathcal{A}$ *sends a bit* b'.

$\mathcal{A}$ *wins the security game if* $b = b'$.

Definition 21 (Read-Enforcing)**.** *Consider any* $\lambda \in \mathbb{N}, S \in \Theta(2^\lambda)$*,* $m_1, \ldots, m_k \in \mathcal{M}_\lambda$*, any* $\mathrm{INDEX}_1, \ldots, \mathrm{INDEX}_k \in \{0, \ldots, S-1\}$*, and any* $\mathrm{INDEX}^* \in \{0, \ldots, S-1\}$.

Let $(\mathsf{PP}_{\mathsf{hAcc}}, w_0, store_0) \leftarrow \mathsf{hAcc.EnforceRead}(1^\lambda, S, \mathrm{INDEX}^*)$.

For all $j \in [k]$*, we define* $store_j$ *iteratively as* $store_j := \mathsf{WriteStore}(\mathsf{PP}_{\mathsf{hAcc}}, store_{j-1}, \mathrm{INDEX}_j, m_j)$.

We similarly define aux_j *and* w_j *iteratively as* $aux_j := \mathsf{PrepWrite}(\mathsf{PP}_{\mathsf{hAcc}}, store_{j-1}, \mathrm{INDEX}_j)$ *and* $w_j := \mathsf{Update}(\mathsf{PP}_{\mathsf{hAcc}}, w_{j-1}, m_j, \mathrm{INDEX}_j, aux_j)$.

Then, hAcc *is said to be* read-enforcing *if* $\mathsf{VerifyRead}(\mathsf{PP}_{\mathsf{hAcc}}, w_k, m, \mathrm{INDEX}^*, \pi) = 1$*, then either* $\mathrm{INDEX}^* \notin \{\mathrm{INDEX}_1, \ldots, \mathrm{INDEX}_k\}$ *and* $m = \emptyset$*, or* $m = m_i$ *for the largest* $i \in [k]$ *such that* $\mathrm{INDEX}_i = \mathrm{INDEX}^*$.

Note that this is an information-theoretic property. We are requiring that for all other symbols m*, values of* π *that would cause* $\mathsf{VerifyRead}$ *to output* 1 *at* INDEX^* *do not exist.*

Definition 22 (Write-Enforcing)**.** *Consider any* $\lambda \in \mathbb{N}, S \in \Theta(2^\lambda)$*,* $m_1, \ldots, m_k \in \mathcal{M}_\lambda$*,* $\mathrm{INDEX}_1, \ldots, \mathrm{INDEX}_k \in \{0, \ldots, S-1\}$.

Let $(\mathsf{PP}_{\mathsf{hAcc}}, w_0, store_0) \leftarrow \mathsf{hAcc.EnforceWrite}(1^\lambda, S, \mathrm{INDEX}_k)$.

For all $j \in [k]$*, we define* $store_j$ *iteratively as* $store_j := \mathsf{WriteStore}(\mathsf{PP}_{\mathsf{hAcc}}, store_{j-1}, \mathrm{INDEX}_j, m_j)$.

We similarly define aux_j *and* w_j *iteratively as* $aux_j := \mathsf{PrepWrite}(\mathsf{PP}_{\mathsf{hAcc}}, store_{j-1}, \mathrm{INDEX}_j)$ *and* $w_j := \mathsf{Update}(\mathsf{PP}_{\mathsf{hAcc}}, w_{j-1}, m_j, \mathrm{INDEX}_j, aux_j)$.

Then, hAcc *is said to be* write-enforcing *if* $\mathsf{Update}(\mathsf{PP}_{\mathsf{hAcc}}, w_{k-1}, m_k, \mathrm{INDEX}_k, aux) = w_{\mathsf{out}} \neq \mathtt{reject}$ *for any* aux*, then* $w_{\mathsf{out}} = w_k$.

Note that this is an information-theoretic property: we are requiring that an aux *value producing an accumulated value other than* w_k *or* $\mathtt{reject}$ *does not exist.*

5.2 Extended Two-to-One SPB Hash

We define the notion of extended two-to-one SPB hash below.

Definition 23 (Extended Two-to-One SPB Hash)**.** *A extended two-to-one SPB hash is a hash function with input-length = 2, block length s and output-length is $\ell(\lambda, s) = s \cdot (1 + 1/\Sigma(\lambda)) + \mathrm{poly}(\lambda)$ with the associated algorithms described below.*

- **Hash key generation,** $\mathsf{hk} \leftarrow \mathsf{Gen}(1^\lambda, 1^s, i)$*: It takes as input a security parameter λ, block length s, an index $i \in \{0,1\}$ and outputs the hash key* hk*. We let $\Sigma = \{0,1\}^s$ denote the block alphabet.*
- **Hashing algorithm,** $\Big(H_{\mathsf{hk}} : \{0,1\}^* \to \{0,1\}^\ell\Big)$*: A deterministic poly-time algorithm that takes as input $x = (x[0], \ldots, x[L-1]) \in \Sigma^2$ and outputs the hash value $H_{\mathsf{hk}}(x)$.*

We require that extended two-to-one SPB hash satisfies the following properties.

I. **Index Hiding**: This says that a PPT adversary should not be able to determine which index was used in the generation of the hash key. Formally,

Definition 24 (Index Hiding)**.** *We consider the following game between an attacker $\mathcal{A}$ and a challenger:*

- *The attacker $\mathcal{A}(1^\lambda)$ sends two indices $i_0, i_1 \in \{0,1\}$.*
- *The challenger chooses a bit $b \leftarrow \{0,1\}$ and sets $\mathsf{hk} \leftarrow \mathsf{Gen}(1^\lambda, 1^s, i_b)$.*
- *The attacker $\mathcal{A}$ gets* hk *and outputs a bit b'*

We require that for any PPT attacker $\mathcal{A}$ we have $\left|\Pr[b' = b] - \frac{1}{2}\right| \leq \mathsf{negl}(\lambda)$ in the above game.

II. **Somewhere Perfectly Binding**: This property states that the output of $H_{\mathsf{hk}}(\mathbf{x}_0, \mathbf{x}_1)$ uniquely determines the b^{th} block of input $\mathbf{x}_b$, where $\mathsf{hk} \leftarrow \mathsf{Gen}(1^\lambda, 1^s, b)$.

Definition 25 (Somewhere Perfectly Binding)**.** *We say that the hash key* hk *is somewhere perfectly binding (SPB) for an index i if there does not exist any values $y, u \neq u', \pi, \pi'$ such that $\mathsf{Verify}(\mathsf{hk}, y, i, u, \pi) = \mathsf{Verify}(\mathsf{hk}, y, i, u', \pi') = 1$. We require that for any parameter s and any index $i \in \{0,1\}$:*

$$\Pr[\mathsf{hk} \textit{ is SPB for index } i \ : \ \mathsf{hk} \leftarrow \mathsf{Gen}(1^\lambda, 1^s, i)] = 1$$

III. **Uniqueness of root**: This property states that if $H_{\mathsf{hk}}(x_0, x_1) = H_{\mathsf{hk}}(x_0, x_1')$ then for every x_0', we have that $H_{\mathsf{hk}}(x_0', x_1) = H_{\mathsf{hk}}(x_0', x_1')$, where $\mathsf{hk} \leftarrow \mathsf{Gen}(1^\lambda, 1^s, 0)$. The case when the first index is information-theoretically bound, that is $\mathsf{hk} \leftarrow \mathsf{Gen}(1^\lambda, 1^s, 1)$, can similarly be defined.

Definition 26. *Suppose* $\mathsf{hk} \leftarrow \mathsf{Gen}(1^\lambda, 1^s, b \in \{0,1\})$*. We say that the extended two-to-one SPB hash satisfies uniqueness of root property if for all $x_0, x_0', x_1, x_1' \in \Sigma$, we have the following:*

- *Case $b = 0$: Let $H_{\mathsf{hk}}(x_0, x_1) = H_{\mathsf{hk}}(x_0, x_1')$. Then for all $x^* \in \Sigma$, we have $H_{\mathsf{hk}}(x^*, x_1) = H_{\mathsf{hk}}(x^*, x_1')$.*
- *Case $b = 1$: Let $H_{\mathsf{hk}}(x_0, x_1) = H_{\mathsf{hk}}(x_0', x_1)$. Then for all $x^* \in \Sigma$, we have $H_{\mathsf{hk}}(x_0, x^*) = H_{\mathsf{hk}}(x_0', x^*)$*

Construction of Extended Two-to-One SPB Hash. We adapt the decisional Diffie-Hellman (DDH)-based construction of Okamoto et al. [OPWW15] to achieve a construction of Extended Two-to-One SPB Hash. In order to do that, we first present their construction verbatim below and then we show that it satisfies uniqueness of root property. The properties of index hiding and SPB will be imported from their result.

We consider a PPT group generator $\mathcal{G}$, that takes as input 1^λ, parameter $1^{t=t(\lambda)}$ and outputs a description of group $\mathbb{G}$ and the order of the group $p \in \theta(2^{t+1})$. We assume that the decisional Diffie-Hellman assumption holds on $\mathcal{G}$.

$\underline{\mathsf{Gen}(1^\lambda, 1^s, b \in \{0,1\})}$: Let $t = \max(\lambda, |\sqrt{s \cdot c}|)$. Generate $(\mathbb{G}, p) \leftarrow \mathcal{G}(1^\lambda, 1^t)$. Choose a random generator $g \in \mathbb{G}$.

Set $d = \lceil \frac{s}{t} \rceil$. Choose at random vectors $\mathbf{w} = (w_1, \ldots, w_d) \in \mathbb{Z}_p$, $\mathbf{a} = (a_1, \ldots, a_d) \in \mathbb{Z}_p^d$ and $\mathbf{b} = (b_1, \ldots, b_d) \in \mathbb{Z}_p^d$. We let $\widetilde{A} \in \mathbb{Z}_p^{d \times d} = \mathbf{a} \otimes \mathbf{w}$ be the tensor product of vectors $\mathbf{a}$ and $\mathbf{w}$, where $(\mathbf{a} \otimes \mathbf{w})_{ij} = a_i \cdot w_j$. Similarly, let $\widetilde{B} = \mathbf{b} \otimes \mathbf{w}$. Finally, let $A = \widetilde{A} + (1 - b) \cdot I$ and $B = \widetilde{B} + b \cdot I$, where $I \in \mathbb{Z}_p^{d \times d}$ is an identity matrix.

The hash key $\mathsf{hk} = (g^{\mathbf{a}}, g^{\mathbf{b}}, g^{\mathbf{A}}, g^{\mathbf{B}})$.

$\underline{H_{\mathsf{hk}}(x_A \in \Sigma = \{0,1\}^s, x_B \in \Sigma = \{0,1\}^s)}$: We view x_A and x_B each consisting of d blocks each of t bits (if this is not the case then we will suitably pad with 0s). That is, $\mathbf{x}_A = (x_{A,1}, \ldots, x_{A,d})$ and $\mathbf{x}_B = (x_{B,1}, \ldots, x_{B,d})$, where $x_{A,j}$ (or $x_{B,j}$) are represented as integers and by our setting of parameters these values are upper bounded by p. It then outputs the value,

$$\left(V = g^{\mathbf{x}_A \mathbf{a} + \mathbf{x}_B \mathbf{b}}, Y = g^{\mathbf{x}_A \mathbf{A} + \mathbf{x}_B \mathbf{B}}\right),$$

where $\mathbf{x}_A \mathbf{a}$ (resp., $\mathbf{x}_B \mathbf{b}$) is an inner product of $\mathbf{x}_A$ and $\mathbf{a}$ (resp., $\mathbf{x}_B$ and $\mathbf{b}$). Similarly, $\mathbf{x}_A \mathbf{A}$ (resp., $\mathbf{x}_B \mathbf{B}$) is a row vector obtained as a result of matrix multiplication of row vector $\mathbf{x}_A$ and matrix $\mathbf{A}$ (resp., $\mathbf{B}$).

The analysis of size overhead (output length), index hiding and the binding properties of the above scheme can be found in Okamoto et al. [OPWW15]. We prove the uniqueness of root property below.

Theorem 4. *The above scheme satisfies uniqueness of root property.*

Proof. Suppose $\mathsf{hk} \leftarrow \mathsf{Gen}(1^\lambda, 1^s, b \in \{0,1\})$. We consider the case when $b = 0$. The same argument symmetrically holds when $b = 1$. Let $x_A, x_B, x_B' \in \{0,1\}^s$ be such that $H_{\mathsf{hk}}(x_A, x_B) = H_{\mathsf{hk}}(x_A, x_B')$. We denote $H_{\mathsf{hk}}(x_A, x_B) = (g^{\mathbf{x}_A \mathbf{a} + \mathbf{x}_B \mathbf{b}}, g^{\mathbf{x}_A \mathbf{A} + \mathbf{x}_B \mathbf{B}})$ and $H_{\mathsf{hk}}(x_A, x_B') = (g^{\mathbf{x}_A \mathbf{a} + \mathbf{x}_B \mathbf{b}}, g^{\mathbf{x}_A \mathbf{A} + \mathbf{x}_B' \mathbf{B}})$, where $\mathbf{x}_A, \mathbf{x}_B, \mathbf{x}_B'$ are generated as in the description of the scheme.

The fact that $H_{\mathsf{hk}}(x_A, x_B) = H_{\mathsf{hk}}(x_A, x'_B)$ implies $g^{\mathbf{x}_A\mathbf{a}+\mathbf{x}_B\mathbf{b}} = g^{\mathbf{x}_A\mathbf{a}+\mathbf{x}'_B\mathbf{b}}$ and also, $g^{\mathbf{x}_A\mathbf{A}+\mathbf{x}_B\mathbf{B}} = g^{\mathbf{x}_A\mathbf{A}+\mathbf{x}'_B\mathbf{B}}$. From these two equalities, we have $\mathbf{x}_B\mathbf{b} = \mathbf{x}'_B\mathbf{b}$ and $\mathbf{x}_B\mathbf{B} = \mathbf{x}'_B\mathbf{B}$.

Now, let $x^* \in \{0,1\}^s$. We have,

$$\begin{aligned} H_{\mathsf{hk}}(x^*, x_B) &= (g^{\mathbf{x}^*\mathbf{a}+\mathbf{x}_B\mathbf{b}}, g^{\mathbf{x}^*\mathbf{A}+\mathbf{x}_B\mathbf{B}}) \\ &= (g^{\mathbf{x}^*\mathbf{a}+\mathbf{x}'_B\mathbf{b}}, g^{\mathbf{x}^*\mathbf{A}+\mathbf{x}'_B\mathbf{B}}) \\ &= H_{\mathsf{hk}}(x^*, x'_B), \text{ as desired.} \end{aligned}$$

□

5.3 History-less Accumulators from Extended Two-to-One SPB Hash

We show how to achieve history-less accumulators from extended two-to-one SPB hash. Our construction will be identical to Okamoto et al. [OPWW15] transformation of positional accumulators from two-to-one hash. We sketch the construction at a high level and a formal description of the construction can be found in their paper[3].

We adopt a Merkle-tree based approach of constructing a history-less accumulator. Suppose we want to initialize the accumulator storage tree with the initial memory $x \in \{0,1\}^{\mathrm{poly}(\lambda)}$. This tree is defined as follows. We divide x into equal halves, namely, x_A and x_B. We recursively, build an accumulator storage tree on x_A and x_B. We denote by w_0 and w_1 to be the corresponding root nodes. We now pick a fresh instantiation of the extended two-to-one SPB hash scheme. Denote the hash key generated from this instantiation to be hk. We define the hash of (w_0, w_1), computed using the key hk, to be the root w. Our initial accumulator value will now be w. Lets say we update a memory element at the location index. Once this is updated, we re-compute the root of the storage tree. This is done by recursively updating the root of the left sub-tree (or the right sub-tree) depending on where index lies. Note that the sub-tree that does not contain memory location at index will not be touched.

In more detail, the update algorithm (Update) takes as input ($\mathsf{PP}_{\mathsf{Acc}}, w_{in}, m_{write}, \mathsf{index}, aux$) and does the following: It parses aux as $\left(m, \pi = (\eta_0^L, \eta_1^L, \ldots, \eta_0^1, \eta_1^1, w_{in} = \eta_0^0)\right)$. It then checks whether (i) For $i \in \{0, \ldots, L-1\}$, η_0^i is the root of $(\eta_0^{i+1}, \eta_1^{i+1})$ in the storage tree, (ii) $\eta_0^L = m$ at the location INDEX. If either one of the checks do not pass then output $\perp$, else continue. In the next step, update the leaf node η_0^L to be $\widetilde{\eta_0}^L$. Then, recursively compute $\widetilde{\eta_0}^i = H_{\mathsf{hk}}(\widetilde{\eta_0}^{i+1}, \eta_1^{i+1})$, for $i \in \{0, \ldots, L-1\}$. Finally assign $w_{out} = \widetilde{\eta_0}^0$.

We argue that the above construction satisfies the definition of history-less accumulators. The only property we need to argue is write-enforcing property. The rest of the properties, namely, indistinguishability of read and write setup, (history-less) read-enforcing have proofs identical to their counterparts in the proof of security of Okamoto et al. [OPWW15].

Theorem 5. *The above construction satisfies history-less write-enforcing property.*

[3]Refer Appendix B.1, dated September 7, 2015 of the ePrint version of [OPWW15].

Proof. Consider any $\lambda \in \mathbb{N}, S \in \Theta(2^\lambda), m_1, \ldots, m_k \in \mathcal{M}_\lambda$, $\text{INDEX}_1, \ldots, \text{INDEX}_k \in \{0, \ldots, S-1\}$. Let $(\mathsf{PP}_{\mathsf{hAcc}}, w_0, \mathit{store}_0) \leftarrow \mathsf{hAcc.EnforceWrite}(1^\lambda, S, \text{INDEX}_k)$. For all $j \in [k]$, we define store_j iteratively as $\mathit{store}_j := \mathsf{WriteStore}(\mathsf{PP}_{\mathsf{hAcc}}, \mathit{store}_{j-1}, \text{INDEX}_j, m_j)$. We similarly define aux_j and w_j iteratively as $\mathit{aux}_j := \mathsf{PrepWrite}(\mathsf{PP}_{\mathsf{hAcc}}, \mathit{store}_{j-1}, \text{INDEX}_j)$ and $w_j := \mathsf{Update}(\mathsf{PP}_{\mathsf{hAcc}}, w_{j-1}, m_j, \text{INDEX}_j, \mathit{aux}_j)$. Denote the value $w_{out} = w_k$. Denote aux_k by aux. And let aux' be such that $\mathsf{Update}(\mathsf{PP}_{\mathsf{hAcc}}, w_{k-1}, m_k, \text{INDEX}_k, \mathit{aux}') = w'$. We claim that if $w' \neq \perp$ then $w' = w_{out}$.

Before we prove this claim, we introduce some notation. We denote aux and aux' as follows:

$$\mathit{aux} = \left(\eta_0^L, \eta_1^L, \ldots, \eta_0^1, \eta_1^1, w_{k-1} = \eta_0^0\right), \mathit{aux}' = \left(\mu_0^L, \mu_1^L, \ldots, \mu_0^1, \mu_1^1, w_{k-1} = \mu_0^0\right)$$

Since Update does not output $\perp$ when both aux and aux' are input into it, we can argue the following: (i) $w_{k-1} = \eta_0^0 = \mu_0^0$, (ii) For $i \in \{0, \ldots, L-1\}$, η_0^i (resp., μ_0^i) is the root of $(\eta_0^{i+1}, \eta_1^{i+1})$ (resp., $(\mu_0^{i+1}, \mu_1^{i+1})$) in the storage tree, (iii) $\eta_0^L = \mu_0^L = m$.

Consider the following lemma. The lemma states that every node along the path of aux, corresponding to a prefix of INDEX_k, has the same value as its corresponding node in aux'. We now state the following lemma.

Lemma 2. *Assuming somewhere perfectly binding property of the underlying extended two-to-one SPB hash scheme, we have* $\eta_0^i = \mu_0^i$ *for all* $i \in \{1, \ldots, L\}$.

Proof. We prove this recursively, top-down, starting from the root of the storage tree. Recall that the root of the tree is $\eta_0^0 = \mu_0^0 = w_{k-1}$. From the perfect binding property of the underlying extended two-to-one SPB hash scheme, we have that w_{k-1} uniquely determines η_0^1 and similarly w_{k-1} uniquely determines μ_0^1. This is only possible if $\eta_0^1 = \mu_0^1$. Similarly, we can show that η_0^1 uniquely determines η_0^2 and μ_0^2, which implies that $\eta_0^2 = \mu_0^2$. Proceeding this way, we get $\eta_0^i = \mu_0^i$, for all $i \in \{1, \ldots, L\}$. □

Now, we recursively define $\widetilde{\eta_0}^i$ to be root of children $\widetilde{\eta_0}^{i+1}$ and η_1^{i+1}, for all $i \in \{0, \ldots, L-1\}$, where (i) $\widetilde{\eta_0}^L = m_k$, (ii) $\widetilde{\eta_0}^0 = w_{out}$. Similarly, we can define $\widetilde{\mu_0}^i$ to be root of children $\widetilde{\mu_0}^{i+1}$ and μ_1^{i+1}, for all $i \in \{0, \ldots, L-1\}$, where (i) $\widetilde{\mu_0}^L = m_k$, (ii) $\widetilde{\mu_0}^0 = w'$.

Lemma 3. *Assuming uniqueness of root property of the underlying extended two-to-one SPB hash scheme, we have* $\widetilde{\eta_0^i} = \widetilde{\mu_0^i}$.

Proof. From Lemma 2, we have that $\eta_0^{L-1} = \mu_0^{L-1}$ and also, $\eta_0^L = \mu_0^L$. This means that $H_{\mathsf{hk}}(\eta_0^L, \eta_1^L) = H_{\mathsf{hk}}(\mu_0^L, \mu_1^L)$, where hk is the hash key used for that particular node. Now, we have $\widetilde{\eta_0}^L = \widetilde{\mu_0}^L = m_k$, which is the value being updated at location INDEX_k. From the uniqueness of root property, we have that $H_{\mathsf{hk}}(\widetilde{\eta_0}^L, \eta_1^L) = H_{\mathsf{hk}}(\widetilde{\mu_0}^L, \mu_1^L)$. From our previous notation, this means that $\widetilde{\eta_0}^{L-1} = \widetilde{\mu_0}^{L-1}$. Proceeding this way up the tree, we get $\widetilde{\eta_i}^0 = \widetilde{\mu_i}^0$, for all $i \in \{0, \ldots, L\}$. □

A consequence of the above lemma is that the root nodes $\widetilde{\eta_0}^0$ and $\widetilde{\mu_0}^0$ are the same. In our terminology, this means that $w_{out} = w'$. This completes the proof of the theorem.

□

6 Adaptive $\mathsf{Ci}\mathcal{O}$ for RAM with Persistent Database

In this section, we introduce the primitive, computation-trace indistinguishable obfuscation $\mathsf{Ci}\mathcal{O}$ with persistent database against *adaptive* adversaries. We will formally define the adaptive security, give a construction based on the new notion history-less accumulator, and then prove its adaptive security using our abstract framework.

6.1 Definition of $\mathsf{Ci}\mathcal{O}$

Consider an initial memory, and then execute a sequence of programs which work on the memory content processed and left over by the previous program. Our adaptive $\mathsf{Ci}\mathcal{O}$ is similar to that *selective* $\mathsf{Ci}\mathcal{O}$ described in [CCC+16], and it forces the evaluator to evaluate obfuscated programs as intended to produce the intended computation trace. The sequence of programs is required to be executed in the intended order. In addition, an adaptive $\mathsf{Ci}\mathcal{O}$ allows its adversary to choose each program *after* receiving previous programs or memory content (compared to a selective adversary must choose its programs and memory content at the beginning).

Let a multiple-program RAM computation be written as $\Pi = (\mathsf{mem}^{0,0}, \{F_{\mathsf{sid}}\}_{\mathsf{sid}=1}^{l})$, where the session identity and total number of programs are denoted by sid and l. For simplicity, we adopt conventions regarding the construction and timestamp as follows.

1. Denote by $(\mathsf{sid}, 0)$ the beginning of session sid.
2. Denote by (sid, i) the time step i of session sid for $i \neq 0$.
3. Each stateful function F_{sid} hardwires the program and its short input x_{sid}.
4. Denote by $(\mathsf{mem}^{\mathsf{sid}+1,0}, \mathsf{st}^{\mathsf{sid}+1,0}) \leftarrow F^*_{\mathsf{sid}}(\mathsf{mem}^{\mathsf{sid},0}, \mathsf{st}^{\mathsf{sid},0})$ the iterative evaluation of F_{sid} on memory database $\mathsf{mem}^{\mathsf{sid},0}$ and CPU state $\mathsf{st}^{\mathsf{sid},0}$ until termination with leftover database $\mathsf{mem}^{\mathsf{sid}+1,0}$ and output state $\mathsf{st}^{\mathsf{sid}+1,0}$.

Definition 27 (Adaptive $\mathsf{Ci}\mathcal{O}$ with Persistent Database)**.** *A computation-trace indistinguishability obfuscation scheme with persistent database denoted by* $\mathsf{Ci}\mathcal{O} = \mathsf{Ci}\mathcal{O}.\{\mathsf{DBCompile}, \mathsf{Obf}, \mathsf{Eval}\}$*, is defined as follows:*

Database compilation algorithm $(\widetilde{\mathsf{mem}}^{1,0}, \widetilde{\mathsf{st}}^{1,0}, \mathsf{sk}) := \mathsf{DBCompile}(1^\lambda, \mathsf{mem}^{0,0}; \rho)$**:** $\mathsf{DBCompile}()$ *is a probabilistic algorithm which takes as input the security parameter λ, the database* $\mathsf{mem}^{0,0}$ *and some randomness ρ, and returns the complied database, state, and secret key* $(\widetilde{\mathsf{mem}}^{1,0}, \widetilde{\mathsf{st}}^{1,0}, \mathsf{sk})$ *as output.*

Program compilation algorithm $\widetilde{F}_{\mathsf{sid}} := \mathsf{Obf}(1^\lambda, \mathsf{sk}, \mathsf{sid}, F_{\mathsf{sid}}; \rho')$**:** $\mathsf{Obf}()$ *is a probabilistic algorithm which takes as input the security parameter λ, the secret key* sk*, the session ID* sid*, the stateful function F_{sid} and some randomness ρ', and returns a complied function $\widetilde{F}_{\mathsf{sid}}$ as output.*

Evaluation algorithm $\mathsf{conf} := \mathsf{Eval}(\widetilde{\mathsf{mem}}^{\mathsf{sid},0}, \widetilde{\mathsf{st}}^{\mathsf{sid},0}, \widetilde{F}_{\mathsf{sid}})$**:** $\mathsf{Eval}()$ *is a deterministic algorithm which takes as input* $(\widetilde{\mathsf{mem}}^{\mathsf{sid},0}, \widetilde{\mathsf{st}}^{\mathsf{sid},0}, \widetilde{F}_{\mathsf{sid}})$*, and returns a configuration* $\mathsf{conf} = (\widetilde{\mathsf{mem}}^{\mathsf{sid}+1,0}, \widetilde{\mathsf{st}}^{\mathsf{sid}+1,0})$ *as output.*

Correctness. *For all* $\mathsf{sid} \in [l]$*, database* $\mathsf{mem}^{0,0}$*,* F_{sid} *with termination time* t^*_{sid} *and randomness* ρ'*, let* $(\widetilde{\mathsf{mem}}^{1,0}, \widetilde{\mathsf{st}}^{1,0}, \mathsf{sk}) := \mathsf{DBCompile}(1^\lambda, \mathsf{mem}^{0,0})$*,* $\widetilde{F}_{\mathsf{sid}} := \mathsf{Obf}(1^\lambda, \mathsf{sk}, \mathsf{sid}, F_{\mathsf{sid}}; \rho')$*, and* $(\widetilde{\mathsf{mem}}^{\mathsf{sid}+1,0}, \widetilde{\mathsf{st}}^{\mathsf{sid}+1,0}) := \mathsf{Eval}(\widetilde{\mathsf{mem}}^{\mathsf{sid},0}, \widetilde{\mathsf{st}}^{\mathsf{sid},0}, \widetilde{F}_{\mathsf{sid}})$*, it holds that*

$$\mathsf{Project}(\widetilde{\mathsf{st}}^{\mathsf{sid}+1,0}) = \mathsf{st}^{\mathsf{sid}+1,0},$$

where $\mathsf{Project}$ *is a simple projecting function.*

Selective Security. *A* $\mathsf{Ci}\mathcal{O}$ *construction is said to be selectively computation-trace indistinguishable if for all PPT adversary* $\mathcal{A}$*, we have* $|\Pr[b = b'] - \frac{1}{2}| \leq \mathsf{negl}$ *in the following game.*

$\mathsf{Exp\text{-}IND\text{-}Ci}\mathcal{O}$

- $\mathcal{C}$ *chooses a bit* $b \in \{0, 1\}$.
- $\mathcal{A}$ *gives* $\mathcal{C}$ *an initial memory* $\mathsf{mem}^{0,0}$ *and program pairs* $\{(F_i^0, F_i^1)\}_{i=1}^l$*. If* F_i^0, F_i^1 *are computation-trace identical,* $\mathcal{C}$ *computes* $(\widetilde{\mathsf{mem}}^{1,0}, \widetilde{\mathsf{st}}^{1,0}, \mathsf{sk}) := \mathsf{DBCompile}(1^\lambda, \mathsf{mem}^{0,0})$*, and returns* $(\widetilde{\mathsf{mem}}^{1,0}, \widetilde{\mathsf{st}}^{1,0})$ *and* $\widetilde{F}_1, ..., \widetilde{F}_l$ *to* $\mathcal{A}$ *where* $\widetilde{F}_i = \mathsf{Obf}(\mathsf{sk}, i, F_i^b)$*. If not,* $\mathcal{C}$ *aborts.*
- $\mathcal{A}$ *outputs* b'*.* $\mathcal{A}$ *is said to win if* $b' = b$*.*

Adaptive Security. *A* $\mathsf{Ci}\mathcal{O}$ *construction is said to be adaptively computation-trace indistinguishable if for all PPT adversary* $\mathcal{A}$*, we have* $|\Pr[b = b'] - \frac{1}{2}| \leq \mathsf{negl}$ *in the following game.*

$\mathsf{Exp\text{-}IND\text{-}Ci}\mathcal{O}$

- $\mathcal{C}$ *chooses a bit* $b \in \{0, 1\}$.
- $\mathcal{A}$ *gives* $\mathcal{C}$ *an initial memory* $\mathsf{mem}^{0,0}$*.* $\mathcal{C}$ *computes* $(\widetilde{\mathsf{mem}}^{1,0}, \widetilde{\mathsf{st}}^{1,0}, \mathsf{sk}) := \mathsf{DBCompile}(1^\lambda, \mathsf{mem}^{0,0})$ *and returns* $(\widetilde{\mathsf{mem}}^{1,0}, \widetilde{\mathsf{st}}^{1,0})$ *to* $\mathcal{A}$*.*
- *At each round* i*, based on the current* $\widetilde{\mathsf{mem}}^{i,0}$ *and previous* $\widetilde{F}_1, ..., \widetilde{F}_{i-1}$*,* $\mathcal{A}$ *adaptively chooses a new pair of* (F_i^0, F_i^1) *to* $\mathcal{C}$*. If* F_i^0, F_i^1 *are computation-trace identical,* $\mathcal{C}$ *returns* $\widetilde{F}_i = \mathsf{Obf}(\mathsf{sk}, i, F_i^b)$ *to* $\mathcal{A}$*. If not,* $\mathcal{C}$ *aborts.*
- $\mathcal{A}$ *outputs* b'*.* $\mathcal{A}$ *is said to win if* $b' = b$*.*

Efficiency. $\mathsf{DBCompile}$ *and* Obf *runs in time* $\tilde{O}(|\mathsf{mem}^{0,0}|)$ *and* $\tilde{O}(\mathrm{poly}(|F_{\mathsf{sid}}|))$*, and efficient* Eval *runs in time* $\tilde{O}(t^*_{\mathsf{sid}})$*.*

6.2 Constructing Adaptive $\mathsf{Ci}\mathcal{O}$ for RAM with Persistent Database

In this section, we construct an adaptive $\mathsf{Ci}\mathcal{O}$, which is based on indistinguishable obfuscation scheme $\mathsf{i}\mathcal{O}$, puncturable pseudorandom function scheme PRF, iterator scheme Itr, splittable signature scheme Spl, and history-less accumulator scheme hAcc. Let a stateful algorithm F denotes a RAM program. Let T be the time bound and S be the space bound of all programs. The construction of $\mathsf{Ci}\mathcal{O}$ consists of the following three algorithms.

Database Compilation Algorithm $\mathsf{DBCompile}(1^\lambda, \mathsf{mem}^{0,0}) \rightarrow (\widetilde{\mathsf{mem}}^{1,0}, \widetilde{\mathsf{st}}^{1,0}, \mathsf{sk})$**.** It computes the following parameters for the obfuscated computation system:

$$K_T \leftarrow \mathsf{PPRF.Setup}(1^\lambda)$$
$$(\mathsf{PP}_{\mathsf{hAcc}}, \hat{w}_0, \hat{store}_0) \leftarrow \mathsf{hAcc.Setup}(S),$$

where K_T is the termination key. Based on $\mathsf{mem}^{0,0}$, this algorithm computes the initial configuration for the complied computation system as follows.

- (Compile storage.) For each $j \in \{1, \ldots, |\mathsf{mem}^{0,0}|\}$ and $x_j = \mathsf{mem}^{0,0}[j]$, it computes iteratively:

$$\pi_j \leftarrow \mathsf{hAcc.PrepWrite}(\mathsf{PP}_{\mathsf{hAcc}}, \hat{store}_{j-1}, j)$$
$$\hat{w}_j \leftarrow \mathsf{hAcc.Update}(\mathsf{PP}_{\mathsf{hAcc}}, \hat{w}_{j-1}, j, x_j, \pi_j)$$
$$\hat{store}_j \leftarrow \mathsf{hAcc.WriteStore}(\mathsf{PP}_{\mathsf{hAcc}}, \hat{store}_{j-1,}, j, x_j)$$

  Set $w^0 := \hat{w}_{|\mathsf{mem}^{0,0}|}$, and $store^0 := \hat{store}_{|\mathsf{mem}^{0,0}|}$.
- (Sign initial state.) Set $\mathsf{sid} = 1, \mathsf{st}^{0,0} = \mathtt{init}, v^0 = \bot$. Compute signature σ^0 as follows:

$$r_T \leftarrow \mathsf{PRF}(K_T, \mathsf{sid} - 1)$$
$$(\mathsf{sk}^0, \mathsf{vk}^0, \mathsf{vk}^0_{\mathrm{rej}}) \leftarrow \mathsf{Spl.Setup}(1^\lambda; r_T)$$
$$\sigma^0 \leftarrow \mathsf{Spl.Sign}(\mathsf{sk}^0, (\mathsf{sid}, \mathsf{st}^{0,0}, v^0, w^0))$$

- (Output) Now we can output the initial configuration as

$$\widetilde{\mathsf{mem}}^{1,0} = store^0$$
$$\widetilde{\mathsf{st}}^{1,0} = ((\mathsf{sid}, 0), \mathsf{st}^{0,0}, v^0, w^0, \sigma^0)$$
$$\mathsf{sk} = (\mathsf{PP}_{\mathsf{hAcc}}, K_T)$$

Program Compilation Algorithm $\mathsf{Obf}(1^\lambda, \mathsf{sk}, \mathsf{sid}, F_{\mathsf{sid}}) \rightarrow \widetilde{F}_{\mathsf{sid}}$**.** First, it generates following session parameters for program F_{sid}:

$$K_A \leftarrow \mathsf{PPRF.Setup}(1^\lambda)$$
$$(\mathsf{PP}_{\mathsf{Itr}}, v^0) \leftarrow \mathsf{Itr.Setup}(T)$$

Second, it parses $\mathsf{sk} = (\mathsf{PP}_{\mathsf{hAcc}}, K_T)$. With parameters $T, \mathsf{PP}_{\mathsf{hAcc}}, \mathsf{PP}_{\mathsf{Itr}}, v^0, K_A$ and termination key K_T, as well as program F_{sid}, we define the program $\widehat{F}_{\mathsf{sid}}$ (Algorithm 1) for given $\mathsf{sid} \in [l]$, $1 \leq \mathsf{sid} \leq l$. This algorithm Obf then computes an obfuscation of the program $\widehat{F}_{\mathsf{sid}}$. That is, $\widetilde{F}_{\mathsf{sid}} \leftarrow i\mathcal{O}.\mathsf{Gen}(\widehat{F}_{\mathsf{sid}})$, and it outputs $\widetilde{F}_{\mathsf{sid}}$.

Algorithm 1: $\widehat{F}_{\mathsf{sid}}$ in $\mathsf{Ci}\mathcal{O}$ for RAM

```
    Input : $\widetilde{\mathsf{st}}^{\mathsf{in}} = ((s,t), \mathsf{st}^{\mathsf{in}}, v^{\mathsf{in}}, w^{\mathsf{in}}, \sigma^{\mathsf{in}})$, $\widetilde{a}^{\mathsf{in}}_{\mathtt{A}\leftarrow\mathtt{M}} = (a^{\mathsf{in}}_{\mathtt{A}\leftarrow\mathtt{M}}, \pi^{\mathsf{in}})$ where $a^{\mathsf{in}}_{\mathtt{A}\leftarrow\mathtt{M}} = (I^{\mathsf{in}}, b^{\mathsf{in}})$
    Data: $T, \mathsf{PP}_{\mathsf{hAcc}}, \mathsf{PP}_{\mathsf{Itr}}, v^0, K_A, K_T$
 1  if $s = \mathsf{sid}$ and $t = 0$ then
 2      Compute $r_{\mathsf{sid}-1} = \mathsf{PRF}(K_T, \mathsf{sid}-1)$ and
        $(\mathsf{sk}_{\mathsf{sid}-1}, \mathsf{vk}_{\mathsf{sid}-1}, \mathsf{vk}_{\mathsf{sid}-1,\mathrm{rej}}) = \mathsf{Spl.Setup}(1^\lambda; r_{\mathsf{sid}-1})$;
 3      If $\mathsf{Spl.Verify}(\mathsf{vk}_{\mathsf{sid}-1}, (\mathsf{sid}, \mathsf{st}^{\mathsf{in}}, v^{\mathsf{in}}, w^{\mathsf{in}}), \sigma^{\mathsf{in}}) = 0$, output reject;
 4      Initialize $t = 1, \mathsf{st}^{\mathsf{in}} = \mathtt{init}, v^{\mathsf{in}} = v^0$, and $a^{\mathsf{in}}_{\mathtt{A}\leftarrow\mathtt{M}} = (\perp, \perp)$;
 5  else
 6      If $\mathsf{hAcc.VerifyRead}(\mathsf{PP}_{\mathsf{hAcc}}, w^{\mathsf{in}}, I^{\mathsf{in}}, b^{\mathsf{in}}, \pi^{\mathsf{in}}) = 0$ output reject;
 7      Compute $r_A = \mathsf{PRF}(K_A, (\mathsf{sid}, t-1))$;
 8      Compute $(\mathsf{sk}_A, \mathsf{vk}_A, \mathsf{vk}_{A,\mathrm{rej}}) = \mathsf{Spl.Setup}(1^\lambda; r_A)$;
 9      Set $m^{\mathsf{in}} = (v^{\mathsf{in}}, \mathsf{st}^{\mathsf{in}}, w^{\mathsf{in}}, I^{\mathsf{in}})$;
10      If $\mathsf{Spl.Verify}(\mathsf{vk}_A, m^{\mathsf{in}}, \sigma^{\mathsf{in}}) = 0$ output reject;

11  Compute $(\mathsf{st}^{\mathsf{out}}, a^{\mathsf{out}}_{\mathtt{M}\leftarrow\mathtt{A}}) \leftarrow F(\mathsf{st}^{\mathsf{in}}, a^{\mathsf{in}}_{\mathtt{A}\leftarrow\mathtt{M}})$ where $a^{\mathsf{out}}_{\mathtt{M}\leftarrow\mathtt{A}} = (I^{\mathsf{out}}, b^{\mathsf{out}})$;
12  If $\mathsf{st}^{\mathsf{out}} = \mathtt{reject}$, output reject;

13  Compute $w^{\mathsf{out}} = \mathsf{hAcc.Update}(\mathsf{PP}_{\mathsf{hAcc}}, w^{\mathsf{in}}, b^{\mathsf{out}}, \pi^{\mathsf{in}})$, output reject if $w^{\mathsf{out}} = \mathtt{reject}$;
14  Compute $v^{\mathsf{out}} = \mathsf{Itr.Iterate}(\mathsf{PP}_{\mathsf{Itr}}, v^{\mathsf{in}}, (\mathsf{st}^{\mathsf{in}}, w^{\mathsf{in}}, I^{\mathsf{in}}))$, output reject if $v^{\mathsf{out}} = \mathtt{reject}$;
15  Compute $r'_A = \mathsf{PRF}(K_A, (\mathsf{sid}, t))$;
16  Compute $(\mathsf{sk}'_A, \mathsf{vk}'_A, \mathsf{vk}'_{A,\mathrm{rej}}) = \mathsf{Spl.Setup}(1^\lambda; r'_A)$;
17  Set $m^{\mathsf{out}} = (v^{\mathsf{out}}, \mathsf{st}^{\mathsf{out}}, w^{\mathsf{out}}, I^{\mathsf{out}})$;
18  Compute $\sigma^{\mathsf{out}} = \mathsf{Spl.Sign}(\mathsf{sk}'_A, m^{\mathsf{out}})$;

19  if $\mathsf{st}^{\mathsf{out}}$ returns halt for termination then
20      Compute $r_{\mathsf{sid}} = \mathsf{PRF}(K_T, \mathsf{sid})$ and $(\mathsf{sk}_{\mathsf{sid}}, \mathsf{vk}_{\mathsf{sid}}, \mathsf{vk}_{\mathsf{sid},\mathrm{rej}}) = \mathsf{Spl.Setup}(1^\lambda; r_{\mathsf{sid}})$;
21      Compute $\sigma^{\mathsf{out}} = \mathsf{Spl.Sign}(\mathsf{sk}_{\mathsf{sid}}, (\mathsf{sid}, \mathsf{st}^{\mathsf{out}}, v^{\mathsf{out}}, w^{\mathsf{out}}))$;
22      Output $\widetilde{\mathsf{st}}^{\mathsf{out}} = ((\mathsf{sid}+1, 0), \mathsf{st}^{\mathsf{out}}, v^{\mathsf{out}}, w^{\mathsf{out}}, \sigma^{\mathsf{out}})$ ;
23  else
24      Output $\widetilde{\mathsf{st}}^{\mathsf{out}} = ((\mathsf{sid}, t+1), \mathsf{st}^{\mathsf{out}}, v^{\mathsf{out}}, w^{\mathsf{out}}, \sigma^{\mathsf{out}})$, $\widetilde{a}^{\mathsf{out}}_{\mathtt{M}\leftarrow\mathtt{A}} = a^{\mathsf{out}}_{\mathtt{M}\leftarrow\mathtt{A}}$;
```

Evaluation algorithm $\mathsf{Eval}(\widetilde{\mathsf{mem}}^{\mathsf{sid},0}, \widetilde{\mathsf{st}}^{\mathsf{sid},0}, \widetilde{F}_{\mathsf{sid}}) \to (\widetilde{\mathsf{mem}}^{\mathsf{sid}+1,0}, \widetilde{\mathsf{st}}^{\mathsf{sid}+1,0})$**.** Upon receiving a compiled database $(\widetilde{\mathsf{mem}}^{\mathsf{sid},0}, \widetilde{\mathsf{st}}^{\mathsf{sid},0})$ and a sequence of obfuscated programs $(\widetilde{F}_1, \ldots \widetilde{F}_l)$, the evaluation algorithm carries out the following for each session sid:

1. Set $\widetilde{a}^{\mathsf{sid},0}_{\mathtt{A}\leftarrow\mathtt{M}} = \perp$. For $t = 1$ to T, perform following procedures until $\widetilde{F}_{\mathsf{sid}}$ outputs a halting state $\widetilde{\mathsf{st}}^{\mathsf{sid},t^*}$ at that halting time t^*:
   - Compute $(\widetilde{\mathsf{st}}^{\mathsf{sid},t}, \widetilde{a}^{\mathsf{sid},t}_{\mathtt{M}\leftarrow\mathtt{A}}) \leftarrow \widetilde{F}_{\mathsf{sid}}(\widetilde{\mathsf{st}}^{\mathsf{sid},t-1}, \widetilde{a}^{\mathsf{sid},t-1}_{\mathtt{A}\leftarrow\mathtt{M}})$;
   - Run $(\widetilde{\mathsf{mem}}^{\mathsf{sid},t}, \widetilde{a}^{\mathsf{sid},t}_{\mathtt{A}\leftarrow\mathtt{M}}) \leftarrow \widetilde{\mathsf{access}}(\widetilde{\mathsf{mem}}^{\mathsf{sid},t-1}, \widetilde{a}^{\mathsf{sid},t}_{\mathtt{M}\leftarrow\mathtt{A}})$, where $\widetilde{\mathsf{access}}$ is the function for

memory access command.

2. At time t^*, output $(\widetilde{\mathsf{mem}}^{\mathsf{sid}+1,0}, \widetilde{\mathsf{st}}^{\mathsf{sid}+1,0}) = (\widetilde{\mathsf{mem}}^{\mathsf{sid},t^*}, \widetilde{\mathsf{st}}^{\mathsf{sid},t^*})$.

To fulfill correctness, we simply define the function $\mathsf{Project}(\widetilde{\mathsf{st}}^{\mathsf{sid}+1,0})$ as follows: first, parse $\widetilde{\mathsf{st}}^{\mathsf{sid}+1,0}$ as $((\mathsf{sid}+1,0), \mathsf{st}^{\mathsf{out}}, v^{\mathsf{out}}, w^{\mathsf{out}}, \sigma^{\mathsf{out}})$; second, output $\mathsf{st}^{\mathsf{out}}$ only.

With the same argument from the selective $\mathsf{Ci}\mathcal{O}$, it is straightforward to verify the correctness and efficiency of the above construction. Next, we present a theorem for its security.

Theorem 6. *Let* $\mathsf{i}\mathcal{O}$ *be a secure indistinguishability obfuscator,* PRF *be a selectively secure puncturable PRF,* Spl *be a secure splittable signature scheme,* Itr *be a secure iterator scheme, and* hAcc *be a secure history-less accumulator scheme. Then* $\mathsf{Ci}\mathcal{O}$ *is an adaptive computation-trace indistinguishability obfuscation scheme in RAM model with persistent database.*

Proof. In this proof, we follow the abstract proof (Section 3) and Theorem 3 to argue our $\mathsf{Ci}\mathcal{O}$ construction is adaptively secure. Note that we already have a *selective proof* to show our $\mathsf{Ci}\mathcal{O}$ construction is selectively secure since we can use that proof of (selective) security in [CCC+16, KLW15] with stronger history-less accumulator. To apply Theorem 3, we firstly model the selective proof as generalized cryptographic games, reductions, and specific functions G. Secondly, given the above, we check the selective proof is a "nice" proof which satisfies all fiveproperties 1, 2, 3", 4, 5 listed in Claim 2. Finally, it follows by Theorem 3 that experiments $\mathsf{Exp\text{-}IND\text{-}Ci}\mathcal{O}\{b=0\}$ and $\mathsf{Exp\text{-}IND\text{-}Ci}\mathcal{O}\{b=1\}$ of our $\mathsf{Ci}\mathcal{O}$ construction are indistinguishable against adaptive adversaries.

Even though there is a long sequence of hybrids, cryptographic games (to distinguish adjacent hybrids), and reductions, we use a systematic way to check the selective proof indeed satisfies all five properties. In general, the i^{th} hybrid can be modeled as an interactive compiler/obfuscator H_i that receives $G_i(\alpha)$ as its global information. Let $(CH_i, G_i||G_{i+1}, 1/2)$ be the generalized cryptographic game to challenge an adversary to distinguish between neighboring hybrids (H_i, H_{i+1}). Let the interactive machine R_i be the reduction from a game $(CH_i, G_i||G_{i+1}, 1/2)$ to a falsifiable assumption (CH'_i, τ'_i) which is one of the assumptions stated in Theorem 6.

To complete the above well-defined games and reductions with specific functions G_i, we observe that there are two cases of hybrid H_i in this proof:

- Case 1: H_i takes as input only the prefix message to compute its output for each step.
- Case 2: H_i *enforces* its accumulator $\mathsf{PP}_{\mathsf{hAcc}}$ which uses either $\mathsf{EnforceRead}$ or $\mathsf{EnforceWrite}$. $\mathsf{PP}_{\mathsf{hAcc}}$ is used to produce the compiled database $\widetilde{\mathsf{mem}}^{0,0}$. It is necessary to enforce on an INDEX* specified by $(F_1, \ldots F_{\mathsf{sid}})$ on mem at time t, where sid and t are further specified by H_i. Therefore, INDEX* is a global information that depends on $(\mathsf{mem}, F_1, \ldots F_{\mathsf{sid}})$.

We define function G_i be null in Case 1. However, in Case 2, define $G_i(\alpha) := \text{INDEX}^*_i(\alpha)$ where $\alpha = (\mathsf{mem}, F_1, F_2, \ldots F_l)$. Note that $\text{INDEX}^*_i(\alpha)$ is efficiently computable by its definition. Also, it is reversibly computable, since for any given $\text{INDEX}^*_i(\alpha)$ in space bound S we can simply set a program F at session sid and time t to access $\text{INDEX}^*_i(\alpha)$.

Remark 1. Let one of neighboring hybrids (H_i, H_{i+1}) be in the enforcing mode (e.g. Case 2). Comparing to game (H_i, H_{i+1}), reduction R_i, and assumption CH'_i in the security proof of the *selective* $\mathsf{Ci}\mathcal{O}$ construction, we stress those are slightly modified in our proof of the *adaptive* $\mathsf{Ci}\mathcal{O}$ construction. In particular, the enforcing accumulator can be complete memory-accessing history or history-less in selective $\mathsf{Ci}\mathcal{O}$. However, the history-less accumulator is necessary in our adaptive $\mathsf{Ci}\mathcal{O}$.

With well-defined generalized games $(CH_i, G_i||G_{i+1}, 1/2)$ and reductions R_i, we check that they satisfy these properties in Claim 2 step by step, which then states they constitute a "nice" proof. For simplicity, $G_i||G_{i+1}$ is denoted by $\bar{G}_i$.

1. **Security based on falsifiable assumptions.** The selective proof is based on: indistinguishability of $\mathsf{i}\mathcal{O}$, selective security of puncturable PRF, four key indistinguishabilities of splittable signature, and indistinguishability of iterator. In addition, the new stronger notion, history-less accumulator, is utilized with its read/write-setup indistinguishabilities.

2. **Security via hybrid argument.** This property holds, since all hybrids H_i have the same interface as the real experiments when interacting with the adversary.

3". **Nice reductions.** For each pair of CH_i and $CH'_i \leftrightarrow R_i$ that both receive $\bar{G}_i(\alpha)$, we need to check if their output distributions are μ-close for every prefix $\rho = (m_1, a_1, m_2, a_2, \cdots, m_{\ell-1}, a_{\ell-1}, m_\ell)$ of a $\bar{G}_i$-consistent transcript of messages. Fortunately, by looking into CH_i and $CH'_i \leftrightarrow R_i$ and comparing their procedures syntactically, we observe those procedures are almost identical even though R_i passes its partial procedures to CH'_i. As a result, $\Delta(\mathsf{D}_{CH_i}(\lambda, \rho), \mathsf{D}_{CH'_i \leftrightarrow R_i}(\lambda, \rho))$ is 0, and R_i is 0-nice by Definition 15.

4. **G_i with polynomial-sized ranges.** Note that function G_i has only 2 cases for all i, either null or $\mathrm{INDEX}^*_i(\alpha)$, which have polynomial-sized range S.

5. **Hiding to adaptive adversaries G_i.** This property requires $(\mathcal{H}_I, \mathcal{G}_I)$ and $(\mathcal{H}_I, \mathbf{0})$ are indistinguishable to $\mathcal{G}_I$-selective adversaries for any function $I(\lambda)$, where $\mathbf{0}$ is the constant zero function. Let $\mathcal{H}_i = \{H_{i,\lambda}\}$, $\mathcal{G}_i = \{\bar{G}_{i,\lambda}\}$ for large enough λ. We claim G_i is hiding for all i with any large enough λ.

   Proof. For any G_i, there are 2 cases, either null or $\mathrm{INDEX}^*_i(\alpha)$. In Case 1, (H_i, null) and $(H_i, \mathbf{0})$ are identical for any adversary since H_i never uses null nor $\mathbf{0}$. In Case 2, the output $G_i(\alpha)$ is always an index that passed as an input to either $\mathsf{EnforceRead}$ or $\mathsf{EnforceWrite}$, and then (H_i, G_i) and $(H_i, \mathbf{0})$ are indistinguishable to any PPT G_i-selective adversary by the read/write setup indistinguishability of history-less accumulator (Definition 19, 20). □

We define the first layer hybrids $\mathbf{Hyb}_i$ for $i \in \{0, 1\}$, which are exactly two experiments $\mathsf{Exp\text{-}IND\text{-}Ci}\mathcal{O}\{b = 0\}$ and $\mathsf{Exp\text{-}IND\text{-}Ci}\mathcal{O}\{b = 1\}$ defined in the adaptive security of $\mathsf{Ci}\mathcal{O}$.

$\mathbf{Hyb}_i$ for $i \in \{0, 1\}$. In this hybrid, the challenger defined by the generalized game outputs an obfuscation computation $\widetilde{\mathsf{mem}}^{1,0}, \{\widetilde{F}^i_{\mathsf{sid}}\}^l_{\mathsf{sid}=1}$ as Algorithm 2.

Algorithm 2: $\widehat{F}^i_{\mathsf{sid}}$ for $i \in \{0,1\}$

```
Input  : $\widetilde{\mathsf{st}}^{\mathsf{in}} = ((\mathsf{sid}, t), \mathsf{st}^{\mathsf{in}}, v^{\mathsf{in}}, w^{\mathsf{in}}, \sigma^{\mathsf{in}})$,   $\widetilde{a}^{\mathsf{in}}_{\mathtt{A \leftarrow M}} = (a^{\mathsf{in}}_{\mathtt{A \leftarrow M}}, \pi^{\mathsf{in}})$ where $a^{\mathsf{in}}_{\mathtt{A \leftarrow M}} = (I^{\mathsf{in}}, b^{\mathsf{in}})$
Data: $T, \mathsf{PP}_{\mathsf{hAcc}}, \mathsf{PP}_{\mathsf{Itr}}, v^0, K_A, K_T$
1  if $s = \mathsf{sid}$ and $t = 0$ then
2      Compute $r_{\mathsf{sid}-1} = \mathsf{PRF}(K_T, \mathsf{sid} - 1)$ and
       $(\mathsf{sk}_{\mathsf{sid}-1}, \mathsf{vk}_{\mathsf{sid}-1}, \mathsf{vk}_{\mathsf{sid}-1,\mathrm{rej}}) = \mathsf{Spl.Setup}(1^\lambda; r_{\mathsf{sid}-1})$;
3      If $\mathsf{Spl.Verify}(\mathsf{vk}_{\mathsf{sid}-1}, (\mathsf{sid}, \mathsf{st}^{\mathsf{in}}, v^{\mathsf{in}}, w^{\mathsf{in}}), \sigma^{\mathsf{in}}) = 0$, output reject;
4      Initialize $t = 1, \mathsf{st}^{\mathsf{in}} = \mathtt{init}, v^{\mathsf{in}} = v^0$, and $a^{\mathsf{in}}_{\mathtt{A \leftarrow M}} = (\perp, \perp)$;
5  else
6      If $\mathsf{hAcc.VerifyRead}(\mathsf{PP}_{\mathsf{hAcc}}, w^{\mathsf{in}}, I^{\mathsf{in}}, b^{\mathsf{in}}, \pi^{\mathsf{in}}) = 0$ output reject;
7      Compute $r_A = \mathsf{PRF}(K_A, (\mathsf{sid}, t - 1))$;
8      Compute $(\mathsf{sk}_A, \mathsf{vk}_A, \mathsf{vk}_{A,\mathrm{rej}}) = \mathsf{Spl.Setup}(1^\lambda; r_A)$;
9      Set $m^{\mathsf{in}} = (v^{\mathsf{in}}, \mathsf{st}^{\mathsf{in}}, w^{\mathsf{in}}, I^{\mathsf{in}})$;
10     If $\mathsf{Spl.Verify}(\mathsf{vk}_A, m^{\mathsf{in}}, \sigma^{\mathsf{in}}) = 0$ output reject;

11 Compute $(\mathsf{st}^{\mathsf{out}}, a^{\mathsf{out}}_{\mathtt{M \leftarrow A}}) \leftarrow F^i_{\mathsf{sid}}(\mathsf{st}^{\mathsf{in}}, a^{\mathsf{in}}_{\mathtt{A \leftarrow M}})$;
12 If $\mathsf{st}^{\mathsf{out}} = \mathtt{reject}$, output reject;

13 Compute $w^{\mathsf{out}} = \mathsf{hAcc.Update}(\mathsf{PP}_{\mathsf{hAcc}}, w^{\mathsf{in}}, b^{\mathsf{out}}, \pi^{\mathsf{in}})$, output reject if $w^{\mathsf{out}} = \mathtt{reject}$;
14 Compute $v^{\mathsf{out}} = \mathsf{Itr.Iterate}(\mathsf{PP}_{\mathsf{Itr}}, v^{\mathsf{in}}, (\mathsf{st}^{\mathsf{in}}, w^{\mathsf{in}}, I^{\mathsf{in}}))$, output reject if $v^{\mathsf{out}} = \mathtt{reject}$;
15 Compute $r'_A = \mathsf{PRF}(K_A, (\mathsf{sid}, t))$;
16 Compute $(\mathsf{sk}'_A, \mathsf{vk}'_A, \mathsf{vk}'_{A,\mathrm{rej}}) = \mathsf{Spl.Setup}(1^\lambda; r'_A)$;
17 Set $m^{\mathsf{out}} = (v^{\mathsf{out}}, \mathsf{st}^{\mathsf{out}}, w^{\mathsf{out}}, I^{\mathsf{out}})$;
18 Compute $\sigma^{\mathsf{out}} = \mathsf{Spl.Sign}(\mathsf{sk}'_A, m^{\mathsf{out}})$;

19 if $\mathsf{st}^{\mathsf{out}}$ returns halt for termination then
20     Compute $r_{\mathsf{sid}} = \mathsf{PRF}(K_T, \mathsf{sid})$ and $(\mathsf{sk}_{\mathsf{sid}}, \mathsf{vk}_{\mathsf{sid}}, \mathsf{vk}_{\mathsf{sid},\mathrm{rej}}) = \mathsf{Spl.Setup}(1^\lambda; r_{\mathsf{sid}})$;
21     Compute $\sigma^{\mathsf{out}} = \mathsf{Spl.Sign}(\mathsf{sk}_{\mathsf{sid}}, (\mathsf{sid}, \mathsf{st}^{\mathsf{out}}, v^{\mathsf{out}}, w^{\mathsf{out}}))$;
22     Output $\widetilde{\mathsf{st}}^{\mathsf{out}} = ((\mathsf{sid} + 1, 0), \mathsf{st}^{\mathsf{out}}, v^{\mathsf{out}}, w^{\mathsf{out}}, \sigma^{\mathsf{out}})$ ;
23 else
24     Output $\widetilde{\mathsf{st}}^{\mathsf{out}} = ((\mathsf{sid}, t + 1), \mathsf{st}^{\mathsf{out}}, v^{\mathsf{out}}, w^{\mathsf{out}}, \sigma^{\mathsf{out}})$,   $\widetilde{a}^{\mathsf{out}}_{\mathtt{M \leftarrow A}} = a^{\mathsf{out}}_{\mathtt{M \leftarrow A}}$;
```

Let $\mathsf{Adv}^k_{\mathcal{A}}$ be the advantage of $\mathbf{Hyb}_k$ against adversary $\mathcal{A}$. We argue that $|\mathsf{Adv}^0_{\mathcal{A}} - \mathsf{Adv}^1_{\mathcal{A}}| \leq \mathsf{negl}(\lambda)$. To show this, we define the second-layer hybrids $\mathbf{Hyb}_{0,0}, \mathbf{Hyb}_{0,1}$, $\{\mathbf{Hyb}_{0,2,j}, \mathbf{Hyb}_{0,3,j}, \mathbf{Hyb}_{0,4,j}\}_{j=1}^{l}$. The order from j to $j+1$ is $\mathbf{Hyb}_{0,2,j}, \mathbf{Hyb}_{0,3,j}, \mathbf{Hyb}_{0,4,j}$, $\mathbf{Hyb}_{0,2,j+1}, \mathbf{Hyb}_{0,3,j+1}, \mathbf{Hyb}_{0,4,j+1}$. Let $t^*_{\mathsf{sid}} < T$ be the terminating time of both programs F^0_{sid} and F^1_{sid}. For the j-th session, we also define third-layer hybrids $\mathbf{Hyb}_{0,2,j,i}$ and $\mathbf{Hyb}_{0,2',j,i}$ for time i, $0 \leq i < t^*_j$.

$\mathbf{Hyb}_{0,0}$. This hybrid is identical to $\mathbf{Hyb}_0$ in the first layer.

$\mathbf{Hyb}_{0,1}$. In this hybrid, the challenger outputs obfuscations of $\{\widehat{F}^{0,1}_{\mathsf{sid}}\}^{l}_{\mathsf{sid}=1}$ which is similar to $\{\widehat{F}^{0}_{\mathsf{sid}}\}^{l}_{\mathsf{sid}=1}$ except that it has PRF key K_B hardwired, accepts both 'A' and 'B' type signatures for $t < t^*_{\mathsf{sid}}$, for all $\mathsf{sid} \in [l]$. The type of the outgoing signature follows the type of the incoming signature. Also, if the incoming signature is 'B' type and $t < t^*_{\mathsf{sid}}$, then the program uses F^1_{sid} to compute the output.

$\mathbf{Hyb}_{0,2,j}$. In this hybrid, the challenger sets $\widehat{F}_{\mathsf{sid}} = \widehat{F}^{0,4,j-1}_{\mathsf{sid}}$ if $\mathsf{sid} < j$; otherwise, it sets $\widehat{F}_{\mathsf{sid}} = \widehat{F}^{0,1}_{\mathsf{sid}}$ if $\mathsf{sid} \geq j$. This hybrid is identical to $\mathbf{Hyb}_{0,2,j,0}$ defined below.

$\mathbf{Hyb}_{0,2,j,i}$. In this hybrid, the challenger sets $\widehat{F}_{\mathsf{sid}} = \widehat{F}^{0,4,j-1}_{\mathsf{sid}}$ if $\mathsf{sid} < j$; otherwise, it sets $\widehat{F}_{\mathsf{sid}} = \widehat{F}^{0,1}_{\mathsf{sid}}$ if $\mathsf{sid} > j$. If $\mathsf{sid} = j$, the challenger outputs an obfuscation of $\widehat{F}^{0,2,j,i}_{\mathsf{sid}}$ defined in Algorithm 3. This program is similar to $\widehat{F}^{0,1}_{\mathsf{sid}}$ except that it accepts 'B' type signatures only for inputs corresponding to $i+1 \leq t \leq t^*_{\mathsf{sid}} - 1$. It also has the correct output message m^i for step i hardwired. For $i+1 \leq t \leq t^*_{\mathsf{sid}} - 1$, the type of the outgoing signature follows the type of the incoming signature. At $t = i$, it outputs an 'A' type signature if $m^{\mathsf{out}} = m^i$, and outputs 'B' type signature otherwise.

$\mathbf{Hyb}_{0,2',j,i}$. In this hybrid, the challenger sets $\widehat{F}_{\mathsf{sid}} = \widehat{F}^{0,4,j-1}_{\mathsf{sid}}$ if $\mathsf{sid} < j$; otherwise, it sets $\widehat{F}_{\mathsf{sid}} = \widehat{F}^{0,1}_{\mathsf{sid}}$ if $\mathsf{sid} > j$. If $\mathsf{sid} = j$, the challenger outputs an obfuscation of $\widehat{F}^{0,2',j,i}_{\mathsf{sid}}$ defined in Algorithm 4. This program is similar to $\widehat{F}^{0,2,j,i}$ except that it accepts 'B' type signatures only for inputs corresponding to $i+2 \leq t \leq t^*_{\mathsf{sid}} - 1$. It also has the correct input message m^i for step $i+1$ hardwired. For $i+2 \leq t \leq t^*_{\mathsf{sid}} - 1$, the type of the outgoing signature follows the type of the incoming signature. At $t = i+1$, it outputs an 'A' type signature if $m^{\mathsf{in}} = m^i$, and outputs 'B' type signature otherwise.

$\mathbf{Hyb}_{0,3,j}$. In this hybrid, the challenger sets $\widehat{F}_{\mathsf{sid}} = \widehat{F}^{0,4,j-1}_{\mathsf{sid}}$ if $\mathsf{sid} < j$; otherwise, it sets $\widehat{F}_{\mathsf{sid}} = \widehat{F}^{0,1}_{\mathsf{sid}}$ if $\mathsf{sid} > j$. If $\mathsf{sid} = j$, the challenger outputs an obfuscation of $\widehat{F}^{0,3,j}_{\mathsf{sid}}$. This program is similar to $\widehat{F}^{0,2',j,t^*_j-1}$, except that it does not output 'B' type signatures.

Algorithm 3: $\widehat{F}^{0,2,j,i}_{\mathsf{sid}=j}$

```
Input : $\widetilde{\mathsf{st}}^{\mathsf{in}} = ((\mathsf{sid}, t), \mathsf{st}^{\mathsf{in}}, v^{\mathsf{in}}, w^{\mathsf{in}}, \sigma^{\mathsf{in}})$, $\widetilde{a}^{\mathsf{in}}_{\mathtt{A}\leftarrow\mathtt{M}} = (a^{\mathsf{in}}_{\mathtt{A}\leftarrow\mathtt{M}}, \pi^{\mathsf{in}})$ where $a^{\mathsf{in}}_{\mathtt{A}\leftarrow\mathtt{M}} = (I^{\mathsf{in}}, b^{\mathsf{in}})$
Data: $T, \mathsf{PP}_{\mathsf{hAcc}}, \mathsf{PP}_{\mathsf{Itr}}, v^0, K_A, K_T, K_B, \underline{m^i}$
1  if $s = \mathsf{sid}$ and $t = 0$ then
2      Compute $r_{\mathsf{sid}-1} = \mathsf{PRF}(K_T, \mathsf{sid}-1)$ and
       $(\mathsf{sk}_{\mathsf{sid}-1}, \mathsf{vk}_{\mathsf{sid}-1}, \mathsf{vk}_{\mathsf{sid}-1,\mathrm{rej}}) = \mathsf{Spl.Setup}(1^\lambda; r_{\mathsf{sid}-1})$;
3      If $\mathsf{Spl.Verify}(\mathsf{vk}_{\mathsf{sid}-1}, (\mathsf{sid}, \mathsf{st}^{\mathsf{in}}, v^{\mathsf{in}}, w^{\mathsf{in}}), \sigma^{\mathsf{in}}) = 0$, output reject;
4      Initialize $t = 1, \mathsf{st}^{\mathsf{in}} = \mathtt{init}, v^{\mathsf{in}} = v^0$, and $a^{\mathsf{in}}_{\mathtt{A}\leftarrow\mathtt{M}} = (\bot, \bot)$;
5  else
6      If $\mathsf{hAcc.VerifyRead}(\mathsf{PP}_{\mathsf{hAcc}}, w^{\mathsf{in}}, I^{\mathsf{in}}, b^{\mathsf{in}}, \pi^{\mathsf{in}}) = 0$ output reject;
7      Compute $r_A = \mathsf{PRF}(K_A, (\mathsf{sid}, t-1))$, $r_B = \mathsf{PRF}(K_B, (\mathsf{sid}, t-1))$;
8      Compute $(\mathsf{sk}_A, \mathsf{vk}_A, \mathsf{vk}_{A,\mathrm{rej}}) = \mathsf{Spl.Setup}(1^\lambda; r_A)$, $(\mathsf{sk}_B, \mathsf{vk}_B, \mathsf{vk}_{B,\mathrm{rej}}) = \mathsf{Spl.Setup}(1^\lambda; r_B)$;
9      Set $m^{\mathsf{in}} = (v^{\mathsf{in}}, \mathsf{st}^{\mathsf{in}}, w^{\mathsf{in}}, I^{\mathsf{in}})$ and $\alpha =$ '-' ;
10     If $\mathsf{Spl.Verify}(\mathsf{vk}_A, m^{\mathsf{in}}, \sigma^{\mathsf{in}}) = 1$ set $\alpha =$ 'A' ;
11     If $\alpha =$ '-' and ($t > t^*$ $\underline{\text{or } t \leq i}$ ) output reject;
12     If $\alpha \neq$ 'A' and $\mathsf{Spl.Verify}(\mathsf{vk}_B, m^{\mathsf{in}}, \sigma^{\mathsf{in}}) = 1$ set $\alpha =$ 'B' ;
13     If $\alpha =$ '-' output reject;

14 if $\alpha =$ 'B' $\underline{\text{or } t \leq i}$ then
15     Compute $(\mathsf{st}^{\mathsf{out}}, a^{\mathsf{out}}_{\mathtt{M}\leftarrow\mathtt{A}}) \leftarrow F^1(\mathsf{st}^{\mathsf{in}}, a^{\mathsf{in}}_{\mathtt{A}\leftarrow\mathtt{M}})$
16 else
17     Compute $(\mathsf{st}^{\mathsf{out}}, a^{\mathsf{out}}_{\mathtt{M}\leftarrow\mathtt{A}}) \leftarrow F^0(\mathsf{st}^{\mathsf{in}}, a^{\mathsf{in}}_{\mathtt{A}\leftarrow\mathtt{M}})$
18 If $\mathsf{st}^{\mathsf{out}} = \mathtt{reject}$, output reject;

19 Compute $w^{\mathsf{out}} = \mathsf{hAcc.Update}(\mathsf{PP}_{\mathsf{hAcc}}, w^{\mathsf{in}}, b^{\mathsf{out}}, \pi^{\mathsf{in}})$, output reject if $w^{\mathsf{out}} = \mathtt{reject}$;
20 Compute $v^{\mathsf{out}} = \mathsf{Itr.Iterate}(\mathsf{PP}_{\mathsf{Itr}}, v^{\mathsf{in}}, (\mathsf{st}^{\mathsf{in}}, w^{\mathsf{in}}, I^{\mathsf{in}}))$, output reject if $v^{\mathsf{out}} = \mathtt{reject}$;
21 Compute $r'_A = \mathsf{PRF}(K_A, (\mathsf{sid}, t))$, $r'_B = \mathsf{PRF}(K_B, (\mathsf{sid}, t))$;
22 Compute $(\mathsf{sk}'_A, \mathsf{vk}'_A, \mathsf{vk}'_{A,\mathrm{rej}}) = \mathsf{Spl.Setup}(1^\lambda; r'_A)$, $(\mathsf{sk}'_B, \mathsf{vk}'_B, \mathsf{vk}'_{B,\mathrm{rej}}) = \mathsf{Spl.Setup}(1^\lambda; r'_B)$;
23 Set $m^{\mathsf{out}} = (v^{\mathsf{out}}, \mathsf{st}^{\mathsf{out}}, w^{\mathsf{out}}, I^{\mathsf{out}})$;
24 if $\underline{t = i \text{ and } m^{\mathsf{out}} = m^i}$ then
25     $\underline{\text{Compute } \sigma^{\mathsf{out}} = \mathsf{Spl.Sign}(\mathsf{sk}'_A, m^{\mathsf{out}})}$;
26 else if $\underline{t = i \text{ and } m^{\mathsf{out}} \neq m^i}$ then
27     $\underline{\text{Compute } \sigma^{\mathsf{out}} = \mathsf{Spl.Sign}(\mathsf{sk}'_B, m^{\mathsf{out}})}$;
28 else
29     $\underline{\text{Compute } \sigma^{\mathsf{out}} = \mathsf{Spl.Sign}(\mathsf{sk}'_\alpha, m^{\mathsf{out}})}$;

30 if $\mathsf{st}^{\mathsf{out}}$ returns halt for termination then
31     Compute $r_{\mathsf{sid}} = \mathsf{PRF}(K_T, \mathsf{sid})$ and $(\mathsf{sk}_{\mathsf{sid}}, \mathsf{vk}_{\mathsf{sid}}, \mathsf{vk}_{\mathsf{sid},\mathrm{rej}}) = \mathsf{Spl.Setup}(1^\lambda; r_{\mathsf{sid}})$;
32     Compute $\sigma^{\mathsf{out}} = \mathsf{Spl.Sign}(\mathsf{sk}_{\mathsf{sid}}, (\mathsf{sid}, \mathsf{st}^{\mathsf{out}}, v^{\mathsf{out}}, w^{\mathsf{out}}))$;
33     Output $\widetilde{\mathsf{st}}^{\mathsf{out}} = ((\mathsf{sid}+1, 0), \mathsf{st}^{\mathsf{out}}, v^{\mathsf{out}}, w^{\mathsf{out}}, \sigma^{\mathsf{out}})$ ;
34 else
35     Output $\widetilde{\mathsf{st}}^{\mathsf{out}} = ((\mathsf{sid}, t+1), \mathsf{st}^{\mathsf{out}}, v^{\mathsf{out}}, w^{\mathsf{out}}, \sigma^{\mathsf{out}})$, $\widetilde{a}^{\mathsf{out}}_{\mathtt{M}\leftarrow\mathtt{A}} = a^{\mathsf{out}}_{\mathtt{M}\leftarrow\mathtt{A}}$;
```

$\mathbf{Hyb}_{0,4,j}$. In this hybrid, the challenger sets $\widehat{F}_{\mathsf{sid}} = \widehat{F}^{0,4,j-1}_{\mathsf{sid}}$ if $\mathsf{sid} < j$; otherwise, it sets $\widehat{F}_{\mathsf{sid}} = \widehat{F}^{0,1}_{\mathsf{sid}}$ if $\mathsf{sid} > j$. If $\mathsf{sid} = j$, the challenger outputs an obfuscation of $\widehat{F}^{0,4,j}_{\mathsf{sid}}$. This program is similar to $\widehat{F}^{0,3,j}$, except that it outputs `reject` for all $t > t^*_j$ including the case when the signature is a valid 'A' type signature.

In the remaining of this subsection, we only prove Lemma 4 (from $\mathbf{Hyb}_{0,2,j,i}$ to $\mathbf{Hyb}_{0,2',j,i}$). Summarizing the above result, we have shown that all hybrids from $\mathbf{Hyb}_0$ to $\mathbf{Hyb}_{0,4,l}$, which gradually substitutes F^0s with F^1s, satisfy the five properties in Claim 2. Symmetrically, all hybrids from $\mathbf{Hyb}_1$ to $\mathbf{Hyb}_{1,4,l}$, which gradually substitutes F^1 with F^0, also satisfy these properties in Claim 2. Finally, we conclude that the proof is a nice proof from $\mathbf{Hyb}_0$ to $\mathbf{Hyb}_{0,4,l} = \mathbf{Hyb}_{1,4,l}$ and to $\mathbf{Hyb}_1$, which completes this proof.

Lemma 4. *Let $1 \le j \le l$, $1 \le i < t^*_j$, and $\mathbf{Hyb}_{0,2,j,i,k}$ for $k \in [0,13]$ defined as follows. We claim the proof from $\mathbf{Hyb}_{0,2,j,i}$ to $\mathbf{Hyb}_{0,2',j,i,k}$ is a nice proof.*

Proof. Here we only focus on the program $\mathsf{sid} = j$ for simplicity. Define next (deepest) layer hybrids $\mathbf{Hyb}_{0,2,j,i,0}$, $\mathbf{Hyb}_{0,2,j,i,1}$, $\dots$, $\mathbf{Hyb}_{0,2,j,i,13}$. The first hybrid corresponds to $\mathbf{Hyb}_{0,2,j,i}$, and the last one corresponds to $\mathbf{Hyb}_{0,2',j,i}$. For all $0 \le k < 13$, the generalized cryptographic game $(CH_{0,2,j,i,k}, \bar{G}_{0,2,j,i,k}, 1/2)$ is to distinguish between $\mathbf{Hyb}_{0,2,j,i,k}$ and $\mathbf{Hyb}_{0,2,j,i,k+1}$, and reduction $R_{0,2,j,i,k}$ is the straight-line black-box reduction from $(CH_{0,2,j,i,k}, \bar{G}_{0,2,j,i,k}, 1/2)$ to assumption $(CH'_{0,2,j,i,k}, 1/2)$. In addition, we specify $G_{0,2,j,i,k}$ for each $\mathbf{Hyb}_{0,2,j,i,k}$, $0 \le k \le 13$. To see how to verify a "nice" proof if G is not null, go to $\mathbf{Hyb}_{0,2,j,i,7}$ directly.

$\mathbf{Hyb}_{0,2,j,i,0}$. This hybrid corresponds to $\mathbf{Hyb}_{0,2,j,i}$. To generalize it, simply let $G_{0,2,j,i,0} = \mathsf{null}$.

$\mathbf{Hyb}_{0,2,j,i,1}$. In this hybrid, the challenger punctures key K_A, K_B at input (j,i), uses $\mathsf{PRF}(K_A,(j,i))$ and $\mathsf{PRF}(K_B,(j,i))$ to compute $(\mathsf{sk}_C, \mathsf{vk}_C)$ and $(\mathsf{sk}_D, \mathsf{vk}_D)$ respectively. More formally, it computes $K_A\{(j,i)\} \leftarrow \mathsf{PRF.Puncture}(K_A,(j,i))$, $r_C = \mathsf{PRF}(K_A,(j,i))$, $(\mathsf{sk}_C, \mathsf{vk}_C, \mathsf{vk}_{C,\mathrm{rej}}) = \mathsf{Spl.Setup}(1^\lambda; r_C)$ and $K_B\{(j,i)\} \leftarrow \mathsf{PRF.Puncture}(K_B,(j,i))$, $r_D = \mathsf{PRF}(K_B,(j,i))$, $(\mathsf{sk}_D, \mathsf{vk}_D, \mathsf{vk}_{D,\mathrm{rej}}) = \mathsf{Spl.Setup}(1^\lambda; r_D)$. The challenger finally outputs an obfuscation of $\widehat{F}^{0,2,j,i,1}$ which is identical to $\widehat{F}^{0,2,j,i,0}$ defined except that it uses punctured PRF keys $K_A\{(j,i)\}, K_B\{(j,i)\}$ and 'C' and 'D' type keys, and modifies the following codes.

- Lines 6 and 7: If $t \ne i+1$, compute $r_{\mathsf{type}} = \mathsf{PRF}(K_{\mathsf{type}}\{(j,i)\}, (\mathsf{sid}, t-1))$, and $(\mathsf{sk}_{\mathsf{type}}, \mathsf{vk}_{\mathsf{type}}, \mathsf{vk}_{\mathsf{type},\mathrm{rej}}) = \mathsf{Spl.Setup}(\lambda; r_{\mathsf{type}})$ for all $\mathsf{type} \in \{A, B\}$. Else, set $\mathsf{vk}_A = \mathsf{vk}_C$ and $\mathsf{vk}_B = \mathsf{vk}_D$.
- Lines 22 and 23: If $t \ne i$, compute $r'_{\mathsf{type}} = \mathsf{PRF}(K_{\mathsf{type}}\{(j,i)\}, (\mathsf{sid}, t))$, and $(\mathsf{sk}'_{\mathsf{type}}, \mathsf{vk}'_{\mathsf{type}}, \mathsf{vk}'_{\mathsf{type},\mathrm{rej}}) = \mathsf{Spl.Setup}(\lambda; r'_{\mathsf{type}})$ for all $\mathsf{type} \in \{A, B\}$. Else, set $\mathsf{sk}'_A = \mathsf{sk}_C$ and $\mathsf{sk}'_B = \mathsf{sk}_D$.

To generalize it, simply let $G_{0,2,j,i,1} = \mathsf{null}$. The indistinguishability between this and the previous one is based on $i\mathcal{O}$ security, $(CH'_{0,2,j,i,0}, 1/2) = (CH_{i\mathcal{O}}, 1/2)$.

$\mathbf{Hyb}_{0,2,j,i,2}$. In this hybrid, in $\widehat{F}^{0,2,j,i,2}$ the challenger chooses r_C, r_D uniformly at random instead of computing them using $\mathsf{PRF}(K_A,(j,i))$ and $\mathsf{PRF}(K_B,(j,i))$. In other words, the

Algorithm 4: $\widehat{F}^{0,2',j,i}_{\mathsf{sid}=j}$

```
   Input : $\widetilde{\mathsf{st}}^{\mathsf{in}} = ((\mathsf{sid}, t), \mathsf{st}^{\mathsf{in}}, v^{\mathsf{in}}, w^{\mathsf{in}}, \sigma^{\mathsf{in}})$,  $\widetilde{a}^{\mathsf{in}}_{\mathtt{A}\leftarrow\mathtt{M}} = (a^{\mathsf{in}}_{\mathtt{A}\leftarrow\mathtt{M}}, \pi^{\mathsf{in}})$ where $a^{\mathsf{in}}_{\mathtt{A}\leftarrow\mathtt{M}} = (I^{\mathsf{in}}, b^{\mathsf{in}})$
   Data: $T, \mathsf{PP}_{\mathsf{hAcc}}, \mathsf{PP}_{\mathsf{Itr}}, v^0, K_A, K_T, K_B, \underline{m^i}$
1  if $s = \mathsf{sid}$ and $t = 0$ then
2      Compute $r_{\mathsf{sid}-1} = \mathsf{PRF}(K_T, \mathsf{sid}-1)$ and
       $(\mathsf{sk}_{\mathsf{sid}-1}, \mathsf{vk}_{\mathsf{sid}-1}, \mathsf{vk}_{\mathsf{sid}-1,\mathrm{rej}}) = \mathsf{Spl.Setup}(1^\lambda; r_{\mathsf{sid}-1})$;
3      If $\mathsf{Spl.Verify}(\mathsf{vk}_{\mathsf{sid}-1}, (\mathsf{sid}, \mathsf{st}^{\mathsf{in}}, v^{\mathsf{in}}, w^{\mathsf{in}}), \sigma^{\mathsf{in}}) = 0$, output reject;
4      Initialize $t = 1, \mathsf{st}^{\mathsf{in}} = \mathtt{init}, v^{\mathsf{in}} = v^0$, and $a^{\mathsf{in}}_{\mathtt{A}\leftarrow\mathtt{M}} = (\perp, \perp)$;
5  else
6      If $\mathsf{hAcc.VerifyRead}(\mathsf{PP}_{\mathsf{hAcc}}, w^{\mathsf{in}}, I^{\mathsf{in}}, b^{\mathsf{in}}, \pi^{\mathsf{in}}) = 0$ output reject;
7      Compute $r_A = \mathsf{PRF}(K_A, (\mathsf{sid}, t-1))$, $r_B = \mathsf{PRF}(K_B, (\mathsf{sid}, t-1))$;
8      Compute $(\mathsf{sk}_A, \mathsf{vk}_A, \mathsf{vk}_{A,\mathrm{rej}}) = \mathsf{Spl.Setup}(1^\lambda; r_A)$, $(\mathsf{sk}_B, \mathsf{vk}_B, \mathsf{vk}_{B,\mathrm{rej}}) = \mathsf{Spl.Setup}(1^\lambda; r_B)$;
9      Set $m^{\mathsf{in}} = (v^{\mathsf{in}}, \mathsf{st}^{\mathsf{in}}, w^{\mathsf{in}}, I^{\mathsf{in}})$ and $\alpha =$ ‘-’ ;
10     If $\mathsf{Spl.Verify}(\mathsf{vk}_A, m^{\mathsf{in}}, \sigma^{\mathsf{in}}) = 1$ set $\alpha =$ ‘A’ ;
11     If $\alpha =$ ‘-’ and $(t > t^*$ $\underline{\text{or } t \leq i+1}$ $)$ output reject;
12     If $\alpha \neq$ ‘A’ and $\mathsf{Spl.Verify}(\mathsf{vk}_B, m^{\mathsf{in}}, \sigma^{\mathsf{in}}) = 1$ set $\alpha =$ ‘B’ ;
13     If $\alpha =$ ‘-’ output reject;

14 if $\alpha =$ ‘B’ $\underline{\text{or } t \leq i+1}$ then
15     Compute $(\mathsf{st}^{\mathsf{out}}, a^{\mathsf{out}}_{\mathtt{M}\leftarrow\mathtt{A}}) \leftarrow F^1(\mathsf{st}^{\mathsf{in}}, a^{\mathsf{in}}_{\mathtt{A}\leftarrow\mathtt{M}})$
16 else
17     Compute $(\mathsf{st}^{\mathsf{out}}, a^{\mathsf{out}}_{\mathtt{M}\leftarrow\mathtt{A}}) \leftarrow F^0(\mathsf{st}^{\mathsf{in}}, a^{\mathsf{in}}_{\mathtt{A}\leftarrow\mathtt{M}})$
18 If $\mathsf{st}^{\mathsf{out}} = \mathtt{reject}$, output reject;

19 Compute $w^{\mathsf{out}} = \mathsf{hAcc.Update}(\mathsf{PP}_{\mathsf{hAcc}}, w^{\mathsf{in}}, b^{\mathsf{out}}, \pi^{\mathsf{in}})$, output reject if $w^{\mathsf{out}} = \mathtt{reject}$;
20 Compute $v^{\mathsf{out}} = \mathsf{Itr.Iterate}(\mathsf{PP}_{\mathsf{Itr}}, v^{\mathsf{in}}, (\mathsf{st}^{\mathsf{in}}, w^{\mathsf{in}}, I^{\mathsf{in}}))$, output reject if $v^{\mathsf{out}} = \mathtt{reject}$;
21 Compute $r'_A = \mathsf{PRF}(K_A, (\mathsf{sid}, t))$, $r'_B = \mathsf{PRF}(K_B, (\mathsf{sid}, t))$;
22 Compute $(\mathsf{sk}'_A, \mathsf{vk}'_A, \mathsf{vk}'_{A,\mathrm{rej}}) = \mathsf{Spl.Setup}(1^\lambda; r'_A)$, $(\mathsf{sk}'_B, \mathsf{vk}'_B, \mathsf{vk}'_{B,\mathrm{rej}}) = \mathsf{Spl.Setup}(1^\lambda; r'_B)$;
23 Set $m^{\mathsf{out}} = (v^{\mathsf{out}}, \mathsf{st}^{\mathsf{out}}, w^{\mathsf{out}}, I^{\mathsf{out}})$;
24 if $\underline{t = i+1 \text{ and } m^{\mathsf{in}} = m^i}$ then
25     $\underline{\text{Compute } \sigma^{\mathsf{out}} = \mathsf{Spl.Sign}(\mathsf{sk}'_A, m^{\mathsf{out}})}$;
26 else if $\underline{t = i+1 \text{ and } m^{\mathsf{in}} \neq m^i}$ then
27     $\underline{\text{Compute } \sigma^{\mathsf{out}} = \mathsf{Spl.Sign}(\mathsf{sk}'_B, m^{\mathsf{out}})}$;
28 else
29     Compute $\sigma^{\mathsf{out}} = \mathsf{Spl.Sign}(\mathsf{sk}'_\alpha, m^{\mathsf{out}})$;

30 if $\mathsf{st}^{\mathsf{out}}$ returns halt for termination then
31     Compute $r_{\mathsf{sid}} = \mathsf{PRF}(K_T, \mathsf{sid})$ and $(\mathsf{sk}_{\mathsf{sid}}, \mathsf{vk}_{\mathsf{sid}}, \mathsf{vk}_{\mathsf{sid},\mathrm{rej}}) = \mathsf{Spl.Setup}(1^\lambda; r_{\mathsf{sid}})$;
32     Compute $\sigma^{\mathsf{out}} = \mathsf{Spl.Sign}(\mathsf{sk}_{\mathsf{sid}}, (\mathsf{sid}, \mathsf{st}^{\mathsf{out}}, v^{\mathsf{out}}, w^{\mathsf{out}}))$;
33     Output $\widetilde{\mathsf{st}}^{\mathsf{out}} = ((\mathsf{sid}+1, 0), \mathsf{st}^{\mathsf{out}}, v^{\mathsf{out}}, w^{\mathsf{out}}, \sigma^{\mathsf{out}})$ ;
34 else
35     Output $\widetilde{\mathsf{st}}^{\mathsf{out}} = ((\mathsf{sid}, t+1), \mathsf{st}^{\mathsf{out}}, v^{\mathsf{out}}, w^{\mathsf{out}}, \sigma^{\mathsf{out}})$,   $\widetilde{a}^{\mathsf{out}}_{\mathtt{M}\leftarrow\mathtt{A}} = a^{\mathsf{out}}_{\mathtt{M}\leftarrow\mathtt{A}}$;
```

secret key/verification key pairs are sampled as $(\mathsf{sk}_C, \mathsf{vk}_C) \leftarrow \mathsf{Spl.Setup}(1^\lambda)$ and $(\mathsf{sk}_D, \mathsf{vk}_D) \leftarrow \mathsf{Spl.Setup}(1^\lambda)$. To generalize it, simply let $G_{0,2,j,i,2} = \mathsf{null}$. The indistinguishability between this and the previous one is based on selectively secure puncturable PRF, $(CH'_{0,2,j,i,1}, 1/2) = (CH_{\mathsf{PRF}}, 1/2)$.

Hyb$_{0,2,j,i,3}$. In this hybrid, the challenger computes constrained signing keys using the $\mathsf{Spl.Split}$ algorithm. As in the previous hybrids, the challenger first computes the i-th message m^i. Then, it computes the following:
$(\sigma_{C,\text{one}}, \mathsf{vk}_{C,\text{one}}, \mathsf{sk}_{C,\text{abo}}, \mathsf{vk}_{C,\text{abo}}) = \mathsf{Spl.Split}(\mathsf{sk}_C, m^i)$ and $(\sigma_{D,\text{one}}, \mathsf{vk}_{D,\text{one}}, \mathsf{sk}_{D,\text{abo}}, \mathsf{vk}_{D,\text{abo}}) = \mathsf{Spl.Split}(\mathsf{sk}_D, m^i)$.
The challenger finally outputs an obfuscation of $\widehat{F}^{0,2,j,i,3}$ which is similar to $\widehat{F}^{0,2,j,i,1}$ except that the following codes.

- Data: Hardwire $\sigma_{C,\text{one}}, \mathsf{sk}_{D,\text{abo}}$ instead of $\mathsf{sk}_C, \mathsf{sk}_D$.
- Line 26: Compute $\sigma^{\mathsf{out}} = \sigma_{C,\text{one}}$.
- Line 28: Compute $\sigma^{\mathsf{out}} = \mathsf{Spl.AboSign}(\mathsf{sk}'_{D,\text{abo}}, m^{\mathsf{out}})$.

To generalize it, simply let $G_{0,2,j,i,3} = \mathsf{null}$. The indistinguishability between this and the previous one is based on $\mathsf{i}\mathcal{O}$ security, $(CH'_{0,2,j,i,2}, 1/2) = (CH_{\mathsf{i}\mathcal{O}}, 1/2)$.

Hyb$_{0,2,j,i,4}$. This hybrid is similar to the previous one, except that in $\widehat{F}^{0,2,j,i,4}$ the challenger hardwires $\mathsf{vk}_{C,\text{one}}$ instead of vk_C. To generalize it, simply let $G_{0,2,j,i,4} = \mathsf{null}$. The indistinguishability between this and the previous one is based on vk_{one} indistinguishability, $(CH'_{0,2,j,i,3}, 1/2) = (CH_{\mathsf{vk}_{\text{one}}}, 1/2)$.

Hyb$_{0,2,j,i,5}$. This hybrid is similar to the previous one, except that in $\widehat{F}^{0,2,j,i,5}$ the challenger hardwires $\mathsf{vk}_{D,\text{abo}}$ instead of vk_D. As in the previous hybrid, the challenger uses $\mathsf{Spl.Split}$ to compute $(\sigma_{C,\text{one}}, \mathsf{vk}_{C,\text{one}}, \mathsf{sk}_{C,\text{abo}}, \mathsf{vk}_{C,\text{abo}})$ and $(\sigma_{D,\text{one}}, \mathsf{vk}_{D,\text{one}}, \mathsf{sk}_{D,\text{abo}}, \mathsf{vk}_{D,\text{abo}})$ from sk_C and sk_D respectively. To generalize it, simply let $G_{0,2,j,i,5} = \mathsf{null}$. The indistinguishability between this and the previous one is based on vk_{abo} indistinguishability, $(CH'_{0,2,j,i,4}, 1/2) = (CH_{\mathsf{vk}_{\text{abo}}}, 1/2)$.

Hyb$_{0,2,j,i,6}$. In this hybrid, the challenger outputs an obfuscation of $\widehat{F}^{0,2,j,i,6}$ which performs extra checks before computing the signature. In particular, the program additionally checks if the input corresponds to step $i+1$. If so, it checks whether $m^{\mathsf{in}} = m^i$ or not, and accordingly outputs either 'A' or 'B' type signature. Formally, $\widehat{F}^{0,2,j,i,6}$ is similar to $\widehat{F}^{0,2,j,i,5}$ except adding the code.

- Between Lines 28 and 29: Else if $t = i{+}1$ and $m^{\mathsf{in}} = m^i$, compute $\sigma^{\mathsf{out}} = \mathsf{Spl.Sign}(\mathsf{sk}'_A, m^{\mathsf{out}})$. Else if $t = i+1$ and $m^{\mathsf{in}} \neq m^i$, compute $\sigma^{\mathsf{out}} = \mathsf{Spl.Sign}(\mathsf{sk}'_B, m^{\mathsf{out}})$.

To generalize it, simply let $G_{0,2,j,i,6} = \mathsf{null}$. The indistinguishability between this and the previous one is based on $\mathsf{i}\mathcal{O}$ security, $(CH'_{0,2,j,i,5}, 1/2) = (CH_{\mathsf{i}\mathcal{O}}, 1/2)$.

Hyb$_{0,2,j,i,7}$. In this hybrid, the challenger makes the accumulator *read enforcing* to prepare the initial configuration from $(\mathsf{PP}_{\mathsf{hAcc}}, \hat{w}_0, \hat{store}_0) \leftarrow \mathsf{hAcc.SetupEnforceRead}(1^\lambda; T, I^i)$. Then, the challenger outputs an obfuscation of $\widehat{F}^{0,2,j,i,7}$ which is similar to $\widehat{F}^{0,2,j,i,6}$ but uses $\mathsf{PP}_{\mathsf{hAcc}}$ of the enforcing mode instead.

To generalize it, let $G_{0,2,j,i,7} = \text{INDEX}^*_{0,2,j,i,7}(\cdot)$, which outputs reading location I^i at session program F_j and time i for any input $\alpha = (\mathsf{mem}, F_1, F_2, \dots)$. The indistinguishability between this and the previous hybrid is based on indistinguishability of read-setup hAcc, assumption $(CH'_{0,2,j,i,6}, 1/2) = (CH_{\mathsf{hAcc},r}, 1/2)$.

Claim 4. *With the generalized game $(CH_{0,2,j,i,6}, \bar{G}_{0,2,j,i,6}, 1/2)$ and reduction $R_{0,2,j,i,6}$, we claim that all of the five properties of the nice proof hold.*

Proof. We check each property as follows.

- Property 1 holds because reduction $R_{0,2,j,i,6}$ is based on assumption $(CH_{\mathsf{hAcc},r}, 1/2)$.
- Property 2 holds because $\mathbf{Hyb}_{0,2,j,i,6}$ and $\mathbf{Hyb}_{0,2,j,i,7}$ are neighboring hybrids, which have the same interface.
- Property 3": To meet the definition of the nice reduction (See Definition 15), we need to syntactically check the whole procedures in $M_1 = CH_{0,2,j,i,6}$ and $M_2 = (CH_{\mathsf{hAcc},r} \leftrightarrow R_{0,2,j,i,6})$. M_1 and M_2 perform almost the same procedures (except there are interactions between $CH_{\mathsf{hAcc},r} \leftrightarrow R_{0,2,j,i,6}$), $R_{0,2,j,i,6}$ is a nice reduction.
- Property 4 holds because $G_{0,2,j,i,7}$ outputs an INDEX in a polynomial range.
- Property 5: To show this, it suffice to show $(\mathbf{Hyb}_{0,2,j,i,7}, G_{0,2,j,i,7})$ and $(\mathbf{Hyb}_{0,2,j,i,7}, \mathbf{0})$ indistinguishable to $G_{0,2,j,i,7}$-selective adversaries. That is, $G(\alpha)$ is either I^i or 0, and then $\mathbf{Hyb}_{0,2,j,i,7}$ passes I^i or 0 to $\mathsf{hAcc.SetupEnforceRead}$ to prepare its initial configuration. By indistinguishability of read-setup hAcc, $\mathsf{hAcc.SetupEnforceRead}(\cdot, I^i)$ and $\mathsf{hAcc.Setup}(\cdot)$ are indistinguishable, and $\mathsf{hAcc.Setup}(\cdot)$ and $\mathsf{hAcc.SetupEnforceRead}(\cdot, 0)$ are indistinguishable. Those implies the first and the third are indistinguishable, and thus Property 5 holds.

□

$\mathbf{Hyb}_{0,2,j,i,8}$. In this hybrid, the challenger outputs an obfuscation of $\widehat{F}^{0,2,j,i,8}$ which runs F^1 instead of F^0 , if on $(i+1)$-st step, the input signature 'A' verifies. Note that the accumulator is 'read enforced' in this hybrid. The modification is shown below.

– Line 13: If $\alpha =$ 'B' or $t \leq i+1$ then

To generalize it, let $G_{0,2,j,i,8} = \text{INDEX}^*_{0,2,j,i,7}(\cdot)$ (because this hybrid still perfroms $\mathsf{SetupEnforceRead}$). The indistinguishability between this and the previous one is based on $\mathsf{i}\mathcal{O}$ security, $(CH'_{0,2,j,i,7}, 1/2) = (CH_{\mathsf{i}\mathcal{O}}, 1/2)$. With $\mathsf{SetupEnforceRead}$, we claim this is a nice proof similar to Claim 4.

$\mathbf{Hyb}_{0,2,j,i,9}$. In this hybrid, the challenger uses setup of the normal mode for the accumulator related parameters, so it computes $(\mathsf{PP}_{\mathsf{hAcc}}, \hat{w}_0, \hat{store}_0) \leftarrow \mathsf{hAcc.Setup}(1^\lambda; T)$. The remaining steps are exactly identical to the corresponding ones in the previous hybrid. Finally, the challenger outputs an obfuscation of $\widehat{F}^{0,2,j,i,9}$ which is similar to $\widehat{F}^{0,2,j,i,8}$ except

$\mathsf{PP}_{\mathsf{hAcc}}$ of the normal mode. To generalize it, simply let $G_{0,2,j,i,9} = \mathsf{null}$. The indistinguishability between this and the previous one is based on indistinguishability of read-setup hAcc, $(CH'_{0,2,j,i,8}, 1/2) = (CH_{\mathsf{hAcc},r}, 1/2)$. Again, similar to Claim 4, we claim this is a nice proof.

Hyb$_{0,2,j,i,10}$. In this hybrid, the challenger computes $(\sigma_{C,\mathrm{one}}, \mathsf{vk}_{C,\mathrm{one}}, \mathsf{sk}_{C,\mathrm{abo}}, \mathsf{vk}_{C,\mathrm{abo}}) = \mathsf{Spl.Split}(\mathsf{sk}_C, m^i)$, but does not compute $(\mathsf{sk}_D, \mathsf{vk}_D)$. It outputs an obfuscation of $\widehat{F}^{0,2,j,i,10}$ which is similar to $\widehat{F}^{0,2,j,i,9}$ except that it hardwires
$(K_A\{(j,i)\}, K_B\{(j,i)\}, \sigma_{C,\mathrm{one}}, \mathsf{vk}_{C,\mathrm{one}}, \mathsf{sk}_{C,\mathrm{abo}}, \mathsf{vk}_{C,\mathrm{abo}}, m_i)$. Note that the hardwired keys for verification/signing $(\sigma_{C,\mathrm{one}}, \mathsf{vk}_{C,\mathrm{one}}, \mathsf{sk}_{C,\mathrm{abo}}, \mathsf{vk}_{C,\mathrm{abo}})$ are all derived from the same signing key sk_C, whereas the first two from sk_C and the next two from sk_D in the previous hybrid. To generalize it, simply let $G_{0,2,j,i,10} = \mathsf{null}$. The indistinguishability between this and the previous one is based on splitting indistinguishability of splittable signature, $(CH'_{0,2,j,i,9}, 1/2) = (CH_{\mathsf{Spl}}, 1/2)$.

Hyb$_{0,2,j,i,11}$. In this hybrid, the challenger chooses $(\mathsf{sk}_C, \mathsf{vk}_C) \leftarrow \mathsf{Spl.Setup}(\lambda)$ and then outputs an obfuscation of $\widehat{F}^{0,2,j,i,11}$ which only hardwires $(K_A\{(j,i)\}, K_B\{(j,i)\}, \mathsf{sk}_C, \mathsf{vk}_C, m_i)$. Comparing to $\widehat{F}^{0,2,j,i,1}$, it does the following modifications in $\widehat{F}^{0,2,j,i,11}$.

- Lines 6 and 7: If $t \neq i+1$, compute $r_{\mathsf{type}} = \mathsf{PRF}(K_{\mathsf{type}}\{(j,i)\}, (\mathsf{sid}, t-1))$, and $(\mathsf{sk}_{\mathsf{type}}, \mathsf{vk}_{\mathsf{type}}, \mathsf{vk}_{\mathsf{type},\mathrm{rej}}) = \mathsf{Spl.Setup}(\lambda; r_{\mathsf{type}})$ for all $\mathsf{type} \in \{A, B\}$. Else, set $\mathsf{vk}_A = \mathsf{vk}_C$.
- Line 10: If $\alpha =$ '-' and $(t > t^*_j$ or $t \leq i+1)$ output `reject`.
- Lines 22 and 23: If $t \neq i$, compute $r'_{\mathsf{type}} = \mathsf{PRF}(K_{\mathsf{type}}\{(j,i)\}, (\mathsf{sid}, t))$, and $(\mathsf{sk}'_{\mathsf{type}}, \mathsf{vk}'_{\mathsf{type}}, \mathsf{vk}'_{\mathsf{type},\mathrm{rej}}) = \mathsf{Spl.Setup}(\lambda; r'_{\mathsf{type}})$ for all $\mathsf{type} \in \{A, B\}$. Else, set $\mathsf{sk}'_A = \mathsf{sk}_C$.
- Lines 25 to 30: If $t = i$, compute $\sigma_{\mathsf{out}} = \mathsf{Spl.Sign}(sk'_A, m_{\mathsf{out}})$.
  Else if $t = i+1$ and $m_{\mathsf{in}} = m^i$, compute $\sigma_{\mathsf{out}} = \mathsf{Spl.Sign}(sk'_A, m_{\mathsf{out}})$.
  Else if $t = i+1$ and $m_{\mathsf{in}} \neq m^i$, compute $\sigma_{\mathsf{out}} = \mathsf{Spl.Sign}(sk'_B, m_{\mathsf{out}})$.
  Else, compute $\sigma_{\mathsf{out}} = \mathsf{Spl.Sign}(sk'_\alpha, m_{\mathsf{out}})$

To generalize it, simply let $G_{0,2,j,i,11} = \mathsf{null}$. The indistinguishability between this and the previous one is based on $\mathsf{i}\mathcal{O}$ security, $(CH'_{0,2,j,i,10}, 1/2) = (CH_{\mathsf{i}\mathcal{O}}, 1/2)$.

Hyb$_{0,2,j,i,12}$. In this hybrid, the challenger chooses the randomness r_C used to compute $(\mathsf{sk}_C, \mathsf{vk}_C)$ pseudorandomly; that is, it sets $r_C = \mathsf{PRF}(K_A, (j,i))$. To generalize it, simply let $G_{0,2,j,i,12} = \mathsf{null}$. The indistinguishability between this and the previous one is based on selectively secure puncturable PRF, $(CH'_{0,2,j,i,11}, 1/2) = (CH_{\mathsf{PRF}}, 1/2)$.

Hyb$_{0,2,j,i,13}$. This hybrid corresponds to **Hyb**$_{0,2',j,i}$. To generalize it, simply let $G_{0,2,j,i,13} = \mathsf{null}$. The indistinguishability between this and the previous one is based on $\mathsf{i}\mathcal{O}$ security, $(CH'_{0,2,j,i,12}, 1/2) = (CH_{\mathsf{i}\mathcal{O}}, 1/2)$.

□

□

7 Adaptive $\mathcal{GRAM}$ with Persistent Database

After obtaining adaptive $\mathsf{Ci}\mathcal{O}$ which forces the obfuscated program to be executed as intended, we can extend it to an adaptive garbling scheme for RAM computation ($\mathcal{GRAM}$) which provides input and program privacy. In the persistent database setting, the privacy of the entire sequence of inputs and programs is preserved, but the output of each program in the sequence will be learnt by the decoder in the clear. At a high level of our construction, for database garbling, the database will be compiled into the oblivious version by ORAM algorithm, and then all database cells will be encrypted. For program garbling, we compile each program P into an ORAM program P_o, produce P_e by adding decryption and encryption in the beginning and end of P_o, and finally obtain the garbled program $\widetilde{P} \leftarrow \mathsf{Ci}\mathcal{O}(P_e)$.

7.1 Definition

For any RAM program P that computes on database mem and outputs bit b at halting time t^*, we denote it by

$$(y = (t^*, b), \mathsf{mem}') \leftarrow P(\mathsf{mem}),$$

where mem' is the updated database modified by program P. We stress that database is *persistent* if mem' can be taken as input to the succeeding program P'. We assume that w.l.o.g. any short input to P can be hard-coded in P directly, and halting time t^* is given in output y.

A garbling scheme to garble RAM programs and persistent database consists of three algorithms: the first is to garble initial database, the second is to garble a RAM program that could read and write that database and then return an output, and the third is to evaluate those garbled database and programs.

Definition 28 ($\mathcal{GRAM}$ with Persistent Database)**.** *A $\mathcal{GRAM}$ scheme with persistent database consists of algorithms $\mathcal{GRAM} = \mathcal{GRAM}.\{\mathsf{DBGarble}, \mathsf{PGarble}, \mathsf{Eval}\}$ described below.*

- $\mathcal{GRAM}.\mathsf{DBGarble}(1^\lambda, \mathsf{mem}^0, S) \to (\widetilde{\mathsf{mem}}^1, \mathsf{sk})$*: The database garbling algorithm* $\mathsf{DBGarble}$ *is a randomized algorithm which takes as input the security parameter 1^λ, database* mem^0*, and a space bound S. It outputs a garbled database $\widetilde{\mathsf{mem}}^1$ and a secret key* sk*.*
- $\mathcal{GRAM}.\mathsf{PGarble}(1^\lambda, \mathsf{sk}, \mathsf{sid}, P_{\mathsf{sid}}) \to \widetilde{P}_{\mathsf{sid}}$*: The algorithm* $\mathsf{PGarble}$ *is a randomized algorithm which takes as input the security parameter 1^λ, the secret key* sk*, the session ID* sid*, the description of a* RAM *program P_{sid} with time bound T and space bound S. It outputs a garbled program $\widetilde{P}_{\mathsf{sid}}$.*
- $\mathcal{GRAM}.\mathsf{Eval}(1^\lambda, T, S, \widetilde{P}_{\mathsf{sid}}, \widetilde{\mathsf{mem}}^{\mathsf{sid}}) \to (y_{\mathsf{sid}}, \widetilde{\mathsf{mem}}^{\mathsf{sid}+1})$*: The evaluating algorithm* Eval *is a deterministic algorithm which takes as input the security parameter 1^λ, time bound T and space bound S, a garbled program $\widetilde{P}_{\mathsf{sid}}$, and the previous database $\widetilde{\mathsf{mem}}^{\mathsf{sid}}$. It outputs $(y_{\mathsf{sid}}, \widetilde{\mathsf{mem}}^{\mathsf{sid}+1})$ or $\perp$, where $(y_{\mathsf{sid}}, \mathsf{mem}^{\mathsf{sid}+1}) = P_{\mathsf{sid}}(\mathsf{mem}^{\mathsf{sid}})$.*

Correctness. *A garbling scheme $\mathcal{GRAM}$ is said to be* correct *if*

$$\begin{aligned}\Pr[&(\widetilde{\mathsf{mem}}^1, \mathsf{sk}) \leftarrow \mathcal{GRAM}.\mathsf{DBGarble}(1^\lambda, \mathsf{mem}^0, S); \widetilde{P}_{\mathsf{sid}} \leftarrow \mathcal{GRAM}.\mathsf{PGarble}(1^\lambda, \mathsf{sk}, \mathsf{sid}, P_{\mathsf{sid}});\\ &(y_{\mathsf{sid}}, \widetilde{\mathsf{mem}}^{\mathsf{sid}+1}) \leftarrow \mathcal{GRAM}.\mathsf{Eval}(1^\lambda, T, S, \widetilde{P}_{\mathsf{sid}}, \widetilde{\mathsf{mem}}^{\mathsf{sid}});\\ &(y'_{\mathsf{sid}}, \mathsf{mem}^{\mathsf{sid}+1}) \leftarrow P_{\mathsf{sid}}(\mathsf{mem}^{\mathsf{sid}}) \ : \ y_{\mathsf{sid}} = y'_{\mathsf{sid}} \ \forall \mathsf{sid}, 1 \leq \mathsf{sid} \leq l] = 1.\end{aligned}$$

Adaptive Security. *A garbling scheme $\mathcal{GRAM} = \mathcal{GRAM}.\{\mathsf{DBGarble}, \mathsf{PGarble}, \mathsf{Eval}\}$ with persistent database is said to be* adaptively secure *if for all sufficiently large $\lambda \in \mathbb{N}$, for all total round $l \in \mathrm{poly}(\lambda)$, time bound T, space bound S, for every interactive PPT adversary $\mathcal{A}$, there exists an interactive PPT simulator $\mathcal{S}$ such that $\mathcal{A}$'s advantage in the following security game* $\mathsf{Exp\text{-}GRAM}(1^\lambda, \mathcal{GRAM}, \mathcal{A}, \mathcal{S})$ *is at most negligible in λ.*

$\mathsf{Exp\text{-}GRAM}(1^\lambda, \mathcal{GRAM}, \mathcal{A}, \mathcal{S})$

1. *The challenger $\mathcal{C}$ chooses a bit $b \in \{0, 1\}$.*
2. *$\mathcal{A}$ chooses and sends database* mem^0 *to challenger $\mathcal{C}$.*
3. *If $b = 0$, challenger $\mathcal{C}$ computes* $(\widetilde{\mathsf{mem}}^1, \mathsf{sk}) \leftarrow \mathsf{DBGarble}(1^\lambda, \mathsf{mem}^0, S)$. *Otherwise, $\mathcal{C}$ simulates* $(\widetilde{\mathsf{mem}}^1, \mathsf{sk}) \leftarrow \mathcal{S}(1^\lambda, |\mathsf{mem}^0|)$, *where* $|\mathsf{mem}^0|$ *is the length of* mem^0. *$\mathcal{C}$ sends* $\widetilde{\mathsf{mem}}^1$ *back to $\mathcal{A}$.*
4. *For each round* sid *from 1 to l,*
   (a) *$\mathcal{A}$ chooses and sends program P_{sid} to $\mathcal{C}$.*
   (b) *If $b = 0$, challenger $\mathcal{C}$ sends $\widetilde{P}_{\mathsf{sid}} \leftarrow \mathcal{GRAM}.\mathsf{PGarble}(1^\lambda, \mathsf{sk}, \mathsf{sid}, P_{\mathsf{sid}})$ to $\mathcal{A}$. Otherwise, $\mathcal{C}$ simulates and sends $\widetilde{P}_{\mathsf{sid}} \leftarrow \mathcal{S}(1^\lambda, \mathsf{sk}, \mathsf{sid}, 1^{|P_{\mathsf{sid}}|}, y_{\mathsf{sid}}, T, S)$ to $\mathcal{A}$, where y_{sid} is defined by the honest computation of program P_{sid} on database* $\mathsf{mem}^{\mathsf{sid}}$: $P_{\mathsf{sid}}(\mathsf{mem}^{\mathsf{sid}}) \rightarrow (\mathsf{mem}^{\mathsf{sid}+1}, y_{\mathsf{sid}})$.
5. *$\mathcal{A}$ outputs a bit b'. $\mathcal{A}$ wins the security game if $b = b'$.*

We notice that an unrestricted adaptive adversary *can adaptively choose RAM programs P_i depending on the program encodings it receives in the above game, whereas a restricted* selective adversary *can only make the choice of programs statically at the beginning of the execution.*

Efficiency. *For all session* sid, *we require* $\mathsf{DBGarble}$ *and* $\mathsf{PGarble}$ *runs in time* $\tilde{O}(|\mathsf{mem}^0|)$ *and* $\tilde{O}(\mathrm{poly}(|P_{\mathsf{sid}}|))$, *and efficient* Eval *runs in time* $\tilde{O}(t^*_{\mathsf{sid}})$.

7.2 Constructing Adaptive $\mathcal{GRAM}$ from $\mathsf{Ci}\mathcal{O}$

Our construction of adaptive $\mathcal{GRAM}$ with persistent database is very similar to the selective version. It uses puncturable PRF, ORAM, $\mathcal{PKE}$ and the new primitive, adaptively secure $\mathsf{Ci}\mathcal{O}$ with persistent database (presented in Section 6). To garble both database and programs, we follow the same natural idea: we use oblivious RAM to hide the access pattern and public-key encryptions to hide the content (including the input), and then we use $\mathsf{Ci}\mathcal{O}$ to obfuscate the compiled computation instance. To evaluate garbled programs, we directly evaluate them using $\mathsf{Ci}\mathcal{O}$ evaluation.

$\mathcal{GRAM}.\mathsf{DBGarble}(1^\lambda, \mathsf{mem}, S) \to (\mathsf{sk}, \widetilde{\mathsf{mem}}^1)$**:** The database garbling algorithm $\mathsf{DBGarble}$ takes following steps to generate secret key sk and the encoding $\widetilde{\mathsf{mem}}^1$.

1. Randomly chooses puncturable PRF keys K_E and K_N.
2. (ORAM) Compile database mem^0 with ORAM algorithm into oblivious database mem^0_o, where the randomness used by ORAM is sampled uniformly.

   The description length of mem^0 could be independently less than space bound S, and thus $|\mathsf{mem}^0_o|$ is proportional to $|\mathsf{mem}^0|$.
3. ($\mathcal{PKE}$) Encrypt oblivious database mem^0_o into mem^0_e using $\mathcal{PKE}$ scheme and pk created from K_E.
4. ($\mathsf{Ci}\mathcal{O}$) Obfuscate mem^0_e using $\mathsf{Ci}\mathcal{O}.\mathsf{DBCompile}(1^\lambda, \mathsf{mem}^0_e) \to (\widetilde{\mathsf{mem}}^{1,0}, \widetilde{\mathsf{st}}^{1,0}, \mathsf{sk}_{\mathsf{Ci}\mathcal{O}})$.
5. Output secret key $\mathsf{sk} = (K_E, K_N, \mathsf{sk}_{\mathsf{Ci}\mathcal{O}})$ and garbled database $\widetilde{\mathsf{mem}}^1 = (\widetilde{\mathsf{mem}}^{1,0}, \widetilde{\mathsf{st}}^{1,0})$.

$\mathcal{GRAM}.\mathsf{PGarble}(1^\lambda, \mathsf{sk}, \mathsf{sid}, P_{\mathsf{sid}}) \to \widetilde{P}_{\mathsf{sid}}$**:** The program garbling algorithm $\mathsf{PGarble}$ takes following steps to generate garbled program $\widetilde{P}_{\mathsf{sid}}$.

1. Parse secret key $\mathsf{sk} = (K_E, K_N, \mathsf{sk}_{\mathsf{Ci}\mathcal{O}})$.
2. (ORAM) Compile program P_{sid} with ORAM algorithm into oblivious program $P[K_N]_{\mathsf{sid},o}$, which computes pseudo-randomnesses from K_N and then passes them to ORAM.
3. ($\mathcal{PKE}$) Compile $P[K_N]_{\mathsf{sid},o}$ into $P[K_N, K_E]_{\mathsf{sid},e}$ that decrypts input and encrypts output at each step with $\mathcal{PKE}$ and keys generated from K_E (Algorithm 5), where the only exception is not to encrypt the halting state which contains the output of P_{sid}. Here the encrypted state (resp., encrypted memory contain) is denoted by $\boxed{\mathsf{st}}$ (resp., $\boxed{b}$).
4. ($\mathsf{Ci}\mathcal{O}$) Obfuscate $P[K_N, K_E]_{\mathsf{sid},e}$ into $\widetilde{P}_{\mathsf{sid}}$ using $\mathsf{Ci}\mathcal{O}.\mathsf{Obf}(1^\lambda, \mathsf{sk}_{\mathsf{Ci}\mathcal{O}}, P[K_N, K_E]_{\mathsf{sid},e}) \to \widetilde{P}_{\mathsf{sid}}$.
5. Output garbled program $\widetilde{P}_{\mathsf{sid}}$.

$\mathcal{GRAM}.\mathsf{Eval}(1^\lambda, T, S, \widetilde{P}_{\mathsf{sid}}, \widetilde{\mathsf{mem}}^{\mathsf{sid}}) \to (y_{\mathsf{sid}}, \widetilde{\mathsf{mem}}^{\mathsf{sid}+1})$**:**

1. Parse database $\widetilde{\mathsf{mem}}^{\mathsf{sid}} = (\widetilde{\mathsf{mem}}^{\mathsf{sid},0}, \widetilde{\mathsf{st}}^{\mathsf{sid},0})$.
2. ($\mathsf{Ci}\mathcal{O}$) Evaluate $\mathsf{Ci}\mathcal{O}.\mathsf{Eval}(\widetilde{\mathsf{mem}}^{\mathsf{sid},0}, \widetilde{\mathsf{st}}^{\mathsf{sid},0}, \widetilde{P}_{\mathsf{sid}}) \to (\widetilde{\mathsf{mem}}^{\mathsf{sid}+1,0}, \widetilde{\mathsf{st}}^{\mathsf{sid}+1,0})$.
3. Output y_{sid} by parsing it from $\widetilde{\mathsf{st}}^{\mathsf{sid}+1,0}$, and output new garbled database $\widetilde{\mathsf{mem}}^{\mathsf{sid}+1} = (\widetilde{\mathsf{mem}}^{\mathsf{sid}+1,0}, \widetilde{\mathsf{st}}^{\mathsf{sid}+1,0})$. Note that parsing y_{sid} is trivial because it is not encrypted in $P[K_N, K_E]_{\mathsf{sid},e}$.

Algorithm 5: $P[K_N, K_E]_{\mathsf{sid},e}$, the program working with encrypted data (formalized as a step circuit).

```
   Input : $\widetilde{\mathsf{st}}^{\mathsf{in}} = (\boxed{\mathsf{st}}^{\mathsf{in}}, t)$,   $\widetilde{a}^{\mathsf{in}} = (I^{\mathsf{in}}, (\boxed{b}^{\mathsf{in}}, \mathsf{lw}^{\mathsf{in}}))$
   Data: $T, K_E, K_N, \mathsf{sid}$
1  if $t = 0$ then
2      Set $\mathsf{st}^{\mathsf{in}} = \mathtt{init}, I^{\mathsf{in}} = \perp, b^{\mathsf{in}} = \perp$;
3  else
4      // Decrypt input state $\boxed{\mathsf{st}}^{\mathsf{in}}$ and reading bit $\boxed{b}^{\mathsf{in}}$
5      Compute $(r_1^{\mathsf{lw}}, r_2^{\mathsf{lw}}, r_3^{\mathsf{lw}}, r_4^{\mathsf{lw}}) = \mathsf{PRF}(K_E, \mathsf{lw}^{\mathsf{in}})$, $(\mathsf{pk}^{\mathsf{lw}}, \mathsf{sk}^{\mathsf{lw}}) = \mathcal{PKE}.\mathsf{Gen}(1^\lambda; r_1^{\mathsf{lw}})$, and
       decrypt $b^{\mathsf{in}} = \mathcal{PKE}.\mathsf{Decrypt}(\mathsf{sk}^{\mathsf{lw}}, \boxed{b}^{\mathsf{in}})$;
6      Compute $(r_1^{t-1}, r_2^{t-1}, r_3^{t-1}, r_4^{t-1}) = \mathsf{PRF}(K_E, (\mathsf{sid}, t-1))$,
       $(\mathsf{pk}^{t-1}, \mathsf{sk}^{t-1}) = \mathcal{PKE}.\mathsf{Gen}(1^\lambda; r_3^{t-1})$, and decrypt $\mathsf{st}^{\mathsf{in}} = \mathcal{PKE}.\mathsf{Decrypt}(\mathsf{sk}^{t-1}, \boxed{\mathsf{st}}^{\mathsf{in}})$;

7  Run one step of the ORAM compiled program $P[K_N]_{\mathsf{sid},o}$ with state $(\mathsf{st}^{\mathsf{in}}$, accessing
   index $I^{\mathsf{in}}$, reading bit $b^{\mathsf{in}}$, and ORAM randomness $\rho = \mathsf{PRF}(K_N, t)$. Let the output be
   $(\mathsf{st}^{\mathsf{out}}, I^{\mathsf{out}}, b^{\mathsf{out}})$;

8  if $\mathsf{st}^{\mathsf{out}} = (\mathtt{halt}, \cdot)$ then
9      Output $\widetilde{\mathsf{st}}^{\mathsf{out}} = \mathsf{st}^{\mathsf{out}}$
10 else
11     // Encrypt output $\boxed{\mathsf{st}}^{\mathsf{out}}$ and writing bit $\boxed{b}^{\mathsf{out}}$
12     Compute $(r_1^t, r_2^t, r_3^t, r_4^t) = \mathsf{PRF}(K_E, (\mathsf{sid}, t))$;
13     Compute $(\mathsf{pk}^{\mathsf{lw}}, \mathsf{sk}^{\mathsf{lw}}) = \mathcal{PKE}.\mathsf{Gen}(1^\lambda; r_1^t)$, and encrypt
       $\boxed{b}^{\mathsf{out}} = \mathcal{PKE}.\mathsf{Encrypt}(\mathsf{pk}^{\mathsf{lw}}, b^{\mathsf{out}}; r_2^t)$;
14     Compute $(\mathsf{pk}^t, \mathsf{sk}^t) = \mathcal{PKE}.\mathsf{Gen}(1^\lambda; r_3^t)$, and encrypt
       $\boxed{\mathsf{st}}^{\mathsf{out}} = \mathcal{PKE}.\mathsf{Encrypt}(\mathsf{pk}^t, \mathsf{st}^{\mathsf{out}}; r_4^t)$;
15     Output $\widetilde{\mathsf{st}}^{\mathsf{out}} = (\boxed{\mathsf{st}}^{\mathsf{out}}, t+1)$,   $\widetilde{a}^{\mathsf{out}} = (I^{\mathsf{out}}, (\boxed{b}^{\mathsf{out}}, (\mathsf{sid}, t)))$ ;
```

With the same argument from the selective $\mathcal{GRAM}$, it is straightforward to verify the correctness and efficiency of the above construction. Next, we present a theorem for its security.

Theorem 7. *Let $\mathcal{PKE}$ be an IND-CPA secure public key encryption scheme,* $\mathsf{Ci}\mathcal{O}$ *be an adaptive computation-trace indistinguishability obfuscation scheme in RAM model,* PRF *be a secure puncturable PRF scheme. Then $\mathcal{GRAM}$ is a secure garble RAM scheme with persistent database.*

Proof. In this proof, we follow the abstraction (Section 3) and Theorem 3 again to argue our $\mathcal{GRAM}$ construction is adaptively secure. Note that we already have a *selective proof* to show our $\mathcal{GRAM}$ construction is selectively secure since we can use that proof of (selective) security in [CCC+16] with the stronger notion of adaptively secure $\mathsf{Ci}\mathcal{O}$. Let the sequence of hybrids in the selective proof be $\mathsf{Exp\text{-}Real} = H_0 \approx H_1 \cdots \approx H_{\ell(\lambda)+1} = \mathsf{Exp\text{-}Sim}$. To apply Theorem 3, we firstly model the selective proof as generalized cryptographic games and reductions with specific functions G, which are efficiently and reversibly computable functions. Secondly, we check the selective proof is a "nice" proof which satisfies all properties 1, 2, 3", 4, 5 listed in Claim 2. Finally, it follows by Theorem 3 that experiments $\mathsf{Exp\text{-}Real}$ and $\mathsf{Exp\text{-}Sim}$ of our $\mathcal{GRAM}$ construction are indistinguishable against adaptive adversaries.

We use a systematic way to check the selective proof which satisfies all these properties even though there is a long sequence of hybrids, cryptographic games to distinguish neighboring hybrids, and reductions. In particular, to work with generalized cryptographic games, we need to define the function G for each hybrid game corresponding to reduction. In general, the i^{th} hybrid can be modeled as an interactive garbler H_i that receives $G_i(\alpha)$ as its global information. Define generalized cryptographic game $(CH_i, G_i||G_{i+1}, 1/2)$ where an adversary distinguishes between (H_i, H_{i+1}) with given $G_i(\alpha)||G_{i+1}(\alpha)$. Let the interactive machine R_i be the reduction from a game $(CH_i, G_i||G_{i+1}, 1/2)$ to a falsifiable assumption (CH'_i, τ'_i), which is one of the assumptions stated in Theorem 7.

To complete the above well-defined games and reductions with specific functions G_i, we check all hybrids H_i and observe that H_i takes as input only the prefix message to compute their every output. Thus, we simply define function G_i be null for all H_i. Note that $\mathsf{Ci}\mathcal{O}$ (used in $\mathsf{DBGarble}$ and $\mathsf{PGarble}$ as a black-box way) does not need any global information $G(\alpha)$.

With well-defined generalized games $(CH_i, G_i||G_{i+1}, 1/2)$ and reductions R_i, we check they satisfy all five properties in Claim 2, which then states they constitute a "nice" proof.

1. **Security based on falsifiable assumptions.** The selective proof is based on: indistinguishability of *adaptive* $\mathsf{Ci}\mathcal{O}$, selective security of puncturable PRF, and IND-CPA security of $\mathcal{PKE}$ scheme.

2. **Security via hybrid argument.** This holds as all hybrids H_i have the same interface as the real experiments when interacting with the adversary.

3". **Nice reductions.** For each pair of CH_i and $CH'_i \leftrightarrow R_i$ that both receive $\bar{G}_i(\alpha)$, we need to check if their output distributions are μ-close for every prefix $\rho = (m_1, a_1,$

$m_2, a_2, \cdots, m_{\ell-1}, a_{\ell-1}, m_\ell)$ of a $\bar{G}_i$-consistent transcript of messages. Fortunately, by looking into CH_i and $CH'_i \leftrightarrow R_i$ and comparing their procedures syntactically, we observed their procedures are almost identical even though R_i passes its partial procedures to CH'_i. Therefore, $\Delta(\mathsf{D}_{CH_i}(\lambda, \rho), \mathsf{D}_{CH'_i \leftrightarrow R_i}(\lambda, \rho))$ is 0, and R_i is 0-nice by Definition 15.

Remark 2. To show this property holds for each reduction R_i such that reduces game $(CH_i, \bar{G}_i, 1/2)$ to $\mathsf{Ci}\mathcal{O}$ assumption $(CH'_{\mathsf{Ci}\mathcal{O}}, 1/2)$, we require that $\mathsf{Ci}\mathcal{O}$ is secure against *adaptive* adversaries. If $\mathsf{Ci}\mathcal{O}$ was only *selective*, then its assumption $(CH'_{\mathsf{Ci}\mathcal{O},\mathrm{sel}}, 1/2)$ would be a cryptographic game that takes the whole global information $(\mathsf{mem}, P_1, P_2, \dots)$ as the first message; thus, it would be quite difficult (if any) to find a nice reduction R such that $\Delta(\mathsf{D}_{CH_i}(\lambda, \rho), \mathsf{D}_{CH'_{\mathsf{Ci}\mathcal{O},\mathrm{sel}} \leftrightarrow R}(\lambda, \rho))$ is negligible. With adaptive $\mathsf{Ci}\mathcal{O}$, it is trivial to show R_i is syntactically nice, and that is the reason why we need *adaptive* $\mathsf{Ci}\mathcal{O}$ in above Property 1.

4. **G_i with polynomial-sized ranges.** Note that, for all i, function G_i is always null, which has constant-sized range.
5. **Hiding to adaptive adversaries G_i.** This property requires $(\mathcal{H}_I, \mathcal{G}_I)$ and $(\mathcal{H}_I, \mathbf{0})$ are indistinguishable to $\mathcal{G}_I$-selective adversaries for any function $I(\lambda)$, where $\mathbf{0}$ is the constant zero function. Let $\mathcal{H}_i = \{H_{i,\lambda}\}$, $\mathcal{G}_i = \{G_{i,\lambda}\}$ for large enough λ. Because H_i never uses $G_i(\alpha)$ no matter G_i is null or $\mathbf{0}$, G_i is hiding for all i with any large enough λ.

Finally, we conclude that the selective proof is a nice proof that satisfy these five properties, and therefore by Theorem 3, experiments $\mathsf{Exp\text{-}Real}$ and $\mathsf{Exp\text{-}Sim}$ are indistinguishable against adaptive adversaries. □

8 Garbled RAM to Delegation: Adaptive Setting

To construct the adaptive RAM delegation, we have to consider full privacy and soundness (Definition 18). There are known generic transformations [AIK10,GHRW14] from $\mathcal{GRAM}$ to that with full privacy and soundness. We follow that transformation with slight modifications to build the RAM delegation.

- Firstly, *output privacy* is achieved by compiling P into P_{op} which hardwires one-time key key, takes memory and state as input, and finally computes y as output and returns $c = \mathsf{Enc}_{\mathsf{key}}(y)$. The client can recover y by decrypting c with key. The entire view of the evaluator can be simulated given the values c (or the length of c by some security of encryption scheme Enc), and thus the evaluator learns nothing about the outputs y.
- Secondly, we can also add *soundness* by using verifiable computation in which the received output c is indeed the correct output encoding of the computation. To do this, we compile P_{op} into $P_{\mathsf{op,ver}}$ which additionally hardwires a one-time MAC key sk, returns c as above, and moreover generates a one-time MAC σ of c. The client can use the MAC key sk and encryption c to verify whether σ is valid. The entire view can be simulated given the values (c, σ), and thus the evaluator cannot come up with a valid MAC σ' for any $y' \neq y$.
- Finally, use our construction of $\mathcal{GRAM}$ to compile $P_{\mathsf{op,ver}}$ to the program encoding $\widetilde{P}$.

Our underlying $\mathcal{GRAM}$ is adaptively secure, and thus this RAM delegation also has adaptively full privacy and soundness.

Acknowledgements

We thank Yael Kalai for insightful discussions in the early stages of this project.

This work was done in part while the authors were visiting the Simons Institute for the Theory of Computing, supported by the Simons Foundation and by the DIMACS/Simons Collaboration in Cryptography through NSF grant CNS-1523467. Prabhanjan Ananth was partially supported by grant #360584 from the Simons Foundation. Huijia Lin was partially supported by NSF grants CNS-1528178 and CNS-1514526.

References

[ABSV15] Prabhanjan Ananth, Zvika Brakerski, Gil Segev, and Vinod Vaikuntanathan. From selective to adaptive security in functional encryption. In *CRYPTO*, 2015.

[AIK10] Benny Applebaum, Yuval Ishai, and Eyal Kushilevitz. From secrecy to soundness: Efficient verification via secure computation. In *Automata, languages and Programming*, pages 152–163. Springer, 2010.

[AJS15] Prabhanjan Ananth, Abhishek Jain, and Amit Sahai. Indistinguishability obfuscation with constant size overhead. Cryptology ePrint Archive, Report 2015/1023, 2015. http://eprint.iacr.org/.

[AS16] Prabhanjan Ananth and Amit Sahai. Functional encryption for turing machines. In *TCC*, 2016.

[BGI$^+$01] Boaz Barak, Oded Goldreich, Russell Impagliazzo, Steven Rudich, Amit Sahai, Salil P. Vadhan, and Ke Yang. On the (im)possibility of obfuscating programs. In Joe Kilian, editor, *Advances in Cryptology - CRYPTO 2001, 21st Annual International Cryptology Conference, Santa Barbara, California, USA, August 19-23, 2001, Proceedings*, volume 2139 of *Lecture Notes in Computer Science*, pages 1–18. Springer, 2001.

[BGI14] Elette Boyle, Shafi Goldwasser, and Ioana Ivan. Functional signatures and pseudorandom functions. In *Public-Key Cryptography–PKC 2014*, pages 501–519. Springer, 2014.

[BGL$^+$15] Nir Bitansky, Sanjam Garg, Huijia Lin, Rafael Pass, and Siddartha Telang. Succinct randomized encodings and their applications. In *STOC*, 2015.

[BW13] Dan Boneh and Brent Waters. Constrained pseudorandom functions and their applications. In *Advances in Cryptology-ASIACRYPT 2013*, pages 280–300. Springer, 2013.

[CCC$^+$16] Yu-Chi Chen, Sherman S. M. Chow, Kai-Min Chung, Russell W. F. Lai, Wei-Kai Lin, and Hong-Sheng Zhou. Cryptography for parallel ram from indistinguishability obfuscation. ITCS, 2016.

[CH16] Ran Canetti and Justin Holmgren. Fully succinct garbled RAM. ITCS, 2016.

[CHJV15] Ran Canetti, Justin Holmgren, Abhishek Jain, and Vinod Vaikuntanathan. Indistinguishability obfuscation of iterated circuits and RAM programs. In *STOC*, 2015.

[GGH$^+$13] Sanjam Garg, Craig Gentry, Shai Halevi, Mariana Raykova, Amit Sahai, and Brent Waters. Candidate indistinguishability obfuscation and functional encryption for all circuits. In *54th Annual IEEE Symposium on Foundations of Computer Science, FOCS 2013, 26-29 October, 2013, Berkeley, CA, USA*, pages 40–49. IEEE Computer Society, 2013.

[GGM86] Oded Goldreich, Shafi Goldwasser, and Silvio Micali. How to construct random functions. *Journal of the ACM (JACM)*, 33(4):792–807, 1986.

[GHL$^+$14] Craig Gentry, Shai Halevi, Steve Lu, Rafail Ostrovsky, Mariana Raykova, and Daniel Wichs. Garbled RAM revisited. In *Advances in Cryptology - EUROCRYPT 2014 - 33rd Annual International Conference on the Theory and Applications of Cryptographic Techniques, Copenhagen, Denmark, May 11-15, 2014. Proceedings*, pages 405–422, 2014.

[GHRW14] Craig Gentry, Shai Halevi, Mariana Raykova, and Daniel Wichs. Outsourcing private RAM computation. In *55th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2014, Philadelphia, PA, USA, October 18-21, 2014*, pages 404–413, 2014.

[GLO15] Sanjam Garg, Steve Lu, and Rafail Ostrovsky. Black-box garbled RAM. FOCS, 2015.

[GLOS15] Sanjam Garg, Steve Lu, Rafail Ostrovsky, and Alessandra Scafuro. Garbled RAM from one-way functions. In *Proceedings of the Forty-Seventh Annual ACM on Symposium on Theory of Computing, STOC 2015, Portland, OR, USA, June 14-17, 2015*, pages 449–458, 2015.

[KLW15] Venkata Koppula, Allison Bishop Lewko, and Brent Waters. Indistinguishability obfuscation for turing machines with unbounded memory. In *STOC*, 2015.

[KP15] Yael Tauman Kalai and Omer Paneth. Delegating ram computations. *IACR Cryptology ePrint Archive*, 2015:957, 2015.

[KPTZ13] Aggelos Kiayias, Stavros Papadopoulos, Nikos Triandopoulos, and Thomas Zacharias. Delegatable pseudorandom functions and applications. In *Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security*, pages 669–684. ACM, 2013.

[KRR14] Yael Tauman Kalai, Ran Raz, and Ron D. Rothblum. How to delegate computations: The power of no-signaling proofs. In *Proceedings of the 46th Annual*

ACM Symposium on Theory of Computing, STOC '14, pages 485–494, New York, NY, USA, 2014. ACM.

[LO13] Steve Lu and Rafail Ostrovsky. How to garble RAM programs. In *Advances in Cryptology - EUROCRYPT 2013, 32nd Annual International Conference on the Theory and Applications of Cryptographic Techniques, Athens, Greece, May 26-30, 2013. Proceedings*, pages 719–734, 2013.

[OPWW15] Tatsuaki Okamoto, Krzysztof Pietrzak, Brent Waters, and Daniel Wichs. New realizations of somewhere statistically binding hashing and positional accumulators. ASIACRYPT, 2015.