Delegating RAM Computations with Adaptive Soundness and Privacy*

Prabhanjan Ananth† Yu-Chi Chen‡ Kai-Min Chung§ Huijia Lin¶
Wei-Kai Lin‖

October 18, 2016

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 computation may read and modify the database and the modifications persist between computations. Delegating RAM computations is important as it has the distinct feature that the run-time of computations maybe *sub-linear* in the size of the database.

We present the first RAM delegation scheme that provide both soundness and privacy guarantees in the *adaptive* setting, where the sequence of delegated RAM programs are chosen adaptively, depending potentially on the encodings of the database and previously chosen programs. Prior works either achieved only adaptive soundness without privacy [Kalai and Paneth, ePrint'15], or only security in the selective setting where all RAM programs are chosen statically [Chen et al. ITCS'16, Canetti and Holmgren ITCS'16].

Our scheme assumes the existence of indistinguishability obfuscation (i$\mathcal{O}$) for circuits and the decisional Diffie-Hellman (DDH) assumption. However, our techniques are quite general and in particular, might be applicable even in settings where iO is not used. We provide a *"security lifting technique"* that "lifts" any proof of selective security satisfying certain special properties into a proof of adaptive security, for arbitrary cryptographic schemes. We then apply this technique to the delegation scheme of Chen et al. and its selective security proof, obtaining that their scheme is essentially already adaptively secure. Because of the general approach, we can also easily extend to delegating parallel RAM (PRAM) computations. We believe that the security lifting technique can potentially find other applications and is of independent interest.

*This is the full version of the extended abstract to appear at Theory of Cryptography Conference (TCC) 2016-B. Information about the grants supporting the authors can be found in "Acknowledgements" section.

†University of California Los Angeles and Center for Encrypted Functionalities. Email: prabhanjan@cs.ucla.edu.

‡Academia Sinica, Taiwan. Email: wycchen@iis.sinica.edu.tw.

§Academia Sinica, Taiwan. Email: kmchung@iis.sinica.edu.tw.

¶University of California Santa Barbara. Email: rachel.lin@cs.ucsb.edu.

‖Cornell University. Email: wklin@cs.cornell.edu.

Contents

1 Introduction **3**
- 1.1 Our Contributions in More Detail . . . 4
- 1.2 Applications . . . 5
- 1.3 On the Existence of IO . . . 6
- 1.4 Concurrent and Related Works . . . 6
- 1.5 Organization . . . 7

2 Overview **7**
- 2.1 Classical Complexity Leveraging . . . 8
- 2.2 Generalized Security Games . . . 8
- 2.3 Small-loss Complexity Leveraging . . . 9
- 2.4 Local Application . . . 11
- 2.5 The CCC+ Scheme and Its Nice Proof . . . 12

3 Preliminaries **13**
- 3.1 Indistinguishability Obfuscation . . . 13
- 3.2 Puncturable Pseudorandom Functions . . . 14
- 3.3 Tools of [KLW15] . . . 14
  - 3.3.1 Iterators . . . 15
  - 3.3.2 Splittable Signatures . . . 15
- 3.4 RAM Computation . . . 17

4 Abstract Proof **18**
- 4.1 Cryptographic Experiments and Games . . . 18
- 4.2 Generalized Cryptographic Games . . . 18
- 4.3 Small-loss Complexity Leveraging . . . 19
  - 4.3.1 Step 1: $\mathcal{G}$-Selective Security . . . 20
  - 4.3.2 Step 2: Fully Adaptive Security . . . 23
- 4.4 Nice Indistinguishability Proof . . . 24

5 Adaptive Delegation for RAM computation **26**
- 5.1 Definition . . . 27

6 History-less Accumulators **29**
- 6.1 Overview . . . 29
- 6.2 Definition . . . 30
  - 6.2.1 Security . . . 32
- 6.3 Extended Two-to-One SPB Hash . . . 34
- 6.4 History-less Accumulators from Extended Two-to-One SPB Hash . . . 36

7 Instantiation: Adaptive Delegation for RAM with Persistent Database **39**
- 7.1 Roadmap . . . 39
- 7.2 Adaptive $\mathsf{Ci}\mathcal{O}$ for RAM with Persistent Database . . . 40
  - 7.2.1 Definition of $\mathsf{Ci}\mathcal{O}$ . . . 40
  - 7.2.2 Construction of $\mathsf{Ci}\mathcal{O}$ . . . 41
  - 7.2.3 Checking Niceness for Security Proof of $\mathsf{Ci}\mathcal{O}$ . . . 44

7.3 Adaptive $\mathcal{GRAM}$ with Persistent Database 45
7.3.1 Definition of $\mathcal{GRAM}$ 46
7.3.2 Construction of $\mathcal{GRAM}$ with Persistent Database 47
7.3.3 Checking Niceness for Security Proof of $\mathcal{GRAM}$ 48
7.4 Garbled RAM to Delegation: Adaptive Setting 52

A Detailed Check for the Proof of Lemma 6 **56**

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: *Soundness* (a.k.a., 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 design *RAM delegation schemes* with both soundness and privacy.

Private RAM Delegation. Consider the following setting. 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.

In order to leverage the efficiency of RAM computations, it is important that RAM delegation schemes are *efficient*: The user runs in time only proportional to the size of the database, or to each program, while the cloud runs in time proportional to the run-time of each computation.

Adaptive v.s. Selective Security. Two "levels" of security exist for delegation schemes: The, *weaker*, selective security provides guarantees only in the restricted setting where all delegated RAM programs and database are chosen statically, whereas, the, *stronger*, adaptive security allows these RAM programs 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.

We present an adaptively secure RAM delegation scheme.

Theorem 1 (Informal Main Theorem)**.** *Assuming DDH and $\mathsf{i}\mathcal{O}$ for circuits, there is an efficient RAM delegation scheme, with adaptive privacy and adaptive soundness.*

Our result closes the gaps left open by previous two lines of research on RAM delegation. In one line, Chen et al. [CCC+16] and Canetti and Holmgren [CH16] constructed the first RAM delegation schemes that achieve *selective privacy* and *selective soundness*, assuming $\mathsf{i}\mathcal{O}$ and one-way functions; their works, however, left open security in the adaptive setting. In another line, Kalai and Paneth [KP15], building upon the seminal result of [KRR14], constructed a RAM delegation scheme with *adaptive soundness*, based on super-polynomial hardness of the LWE assumption, which, however, does not provide privacy at all.[1] Our RAM delegation scheme improves upon previous works — it simultaneously achieves adaptive soundness and privacy. Concurrent to our work, Canetti, Chen, Holmgren, and Raykova [CCHR16] also constructed such a RAM delegation scheme. Our construction and theirs are the first to achieve these properties.

[1] Note that here, privacy cannot be achieved for free using Fully Homomorphic Encryption (FHE), as FHE does not directly support computation with RAM programs, unless they are first transformed into oblivious Turing machines or circuits.

1.1 Our Contributions in More Detail

Our RAM delegation scheme achieves the privacy guarantee that the encodings of a database and many RAM programs, chosen adaptively by a malicious server (i.e., the cloud), reveals nothing more than the outputs of the computations. This is captured via the simulation paradigm, where the encodings can be simulated by a simulator that receives only the outputs. On the other hand, soundness guarantees that no malicious server can convince an honest client (i.e., the user) to accept a wrong output of any delegated computation, even if the database and programs are chosen adaptively by the malicious server.

Efficiency. Our adaptively secure RAM delegation scheme achieves the same level of efficiency as previous selectively secure schemes [CCC+16, CH16]. More specifically,

- CLIENT DELEGATION EFFICIENCY: To outsource a database DB of size n, the client encodes the database in time linear in the database size, $n\,\mathrm{poly}(\lambda)$ (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 client encodes the program in time linear in the output length and polynomial in the program description size $l \times \mathrm{poly}(|M|, \lambda)$, independent of the complexity of the RAM program.
- SERVER EVALUATION EFFICIENCY: The evaluation time and space complexity of the server, scales linearly with the complexity of the RAM programs, that is, $T\,\mathrm{poly}(\lambda)$ and $S\,\mathrm{poly}(\lambda)$ respectively.
- CLIENT VERIFICATION EFFICIENCY: Finally, the user verifies the proof from the server and recovers the output in time $l \times \mathrm{poly}(\lambda)$.

The above level of efficiency is comparable to 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 $\mathrm{poly}(\lambda)$ overhead at the server, and a $\mathrm{poly}(|M|, \lambda)$ overhead at the user.[2] In particular, if the run-time of a delegated RAM program is sub-linear $o(n)$, the server evaluation time is also sub-linear $o(n)\,\mathrm{poly}(\lambda)$, which is crucial for server efficiency.

Technical contributions: Though our RAM delegation scheme relies on the existence of $\mathsf{i}\mathcal{O}$, the techniques that we introduce in this work are quite general and in particular, might be applicable in settings where $\mathsf{i}\mathcal{O}$ is not used at all.

Our main theorem is established by showing that the selectively secure RAM delegation scheme of [CCC+16] (CCC+ scheme henceforth) is, in fact, also adaptively secure (up to some modifications). However, proving its adaptive security is challenging, especially considering the heavy machinery already in the selective security proof (inherited from the line of works on succinct randomized encoding of Turing machines and RAMs [BGL+15, CHJV15]). Ideally, we would like to have a proof of adaptive security that uses the selective security property in a black-box way. A recent elegant example is the work of [ABSV15] that constructed an adaptively secure functional encryption from any selectively secure functional encryption without any additional assumptions.[3] However, such cases are rare: In most cases, adaptive security is treated independently, achieved using completely new constructions and/or new proofs (see examples, the adaptively secure functional encryption scheme by Waters [Wat15], the adaptively secure garbled circuits by [HJO+16],

[2] 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 [AJS15a].

[3] More generally, they use a 1-query adaptively secure functional encryption suffices which can be constructed from one-way functions by [GVW12].

and many others). In the context of RAM delegation, coming up with a proof of adaptive security from scratch requires at least repeating or rephrasing the proof of selective security and adding more details (unless the techniques behind the entire line of research [KLW15, CH16, CCC+16] can be significantly simplified).

Instead of taking this daunting path, we follow a more principled and general approach. We provide an abstract proof that "lifts" any selective security proof satisfying certain properties — called a "nice" proof — into an adaptive security proof, for arbitrary cryptographic schemes. With the abstract proof, the task of showing adaptive security boils down to a mechanic (though possibly tedious) check whether the original selective security proof is nice. We proceed to do so for the CCC+ scheme, and show that when the CCC+ scheme is plugged in with a special kind of positional accummulator [KLW15], called *history-less accummulator*, all niceness properties are satisfied; then its adaptive security follows immediately. At a very high-level, history-less accummulators can statistically bind the value at a particular position q irrespect of the history of read/write accesses, whereas positional accumulators of [KLW15] binds the value at q after a specific sequence of read/write accesses.

Highlights of techniques used in the abstract proof includes a stronger version of complexity leveraging—called small-loss complexity leveraging—that have much smaller security loss than classical complexity leveraging, when the security game and its selective security proof satisfy certain "niceness" properties, as well as a way to apply small-loss complexity leveraging locally inside an involved security proof. We provide an overview of our techniques in more detail in Section 2.

Parallel RAM (PRAM) Delegation As a benefit of our general approach, we can easily handle delegation of PRAM computations as well. Roughly speaking, PRAM programs are RAM programs that additionally support parallel (random) accesses to the database. Chen et al. [CCC+16] presented a delegation scheme for PRAM 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.

Theorem 2 (informal — PRAM Delegation Scheme)**.** *Assuming DDH and the existence of* $i\mathcal{O}$ *for circuits, there exists an efficient PRAM delegation scheme, with adaptive privacy and adaptive soundness.*

1.2 Applications

In the context of cloud computing and big data, designing ways for delegating computation privately and efficiently is important. Different cryptographic tools, such as Fully Homomorphic Encryption (FHE) and Functional Encryption (FE), provide different solutions. However, so far, none supports delegation of *sub-linear* computation (for example, binary search over a large ordered data set, and testing combinatorial properties, like k-connectivity and bipartited-ness, of a large graph in sub-linear time). It is known that FHE does not support RAM computation, for the evaluator cannot decrypt the locations in the memory to be accessed. FE schemes for Turing machines constructed in [AS16] cannot be extended to support RAM, as the evaluation complexity is at least linear in the size of the encrypted database. This is due to a refreshing mechanism crucially employed in their work that "refreshes" the entire encrypted database in each evaluation, in order to ensure privacy. To the best of our knowledge, RAM delegation schemes are the only solution that supports sub-linear computations.

Apart from the relevance of RAM delegation in practice, it has also been quite useful to obtain theoretical applications. Recently, RAM delegation was also used in the context of patchable obfuscation by [AJS15b]. In particular, they crucially required that the RAM delegation satisfies adaptive privacy and only our work (and concurrently [CCHR16]) achieves this property. Our techniques have also found interesting applications. In particular, history-less accumulators, that we introduce, was crucially used in the construction of constrained PRFs for Turing machines [DKW16].

1.3 On the Existence of IO

Our RAM delegation scheme assumes the existence of IO for circuits. So far, in the literature, many candidate IO schemes have been proposed (e.g., [GGH+13b, BR14, BGK+14]) building upon the so called graded encoding schemes [GGH13a, CLT13, CLT15, GGH15]. While the security of these candidates have come under scrutiny in light of two recent attacks [CGH+15, MSZ16] on specific candidates, there are still several IO candidates on which the current cryptanalytic attacks don't apply. Moreover, current multilinear map attacks do not apply to IO schemes obtained after applying bootstrapping techniques to candidate IO schemes for NC^1 [GIS+10, GGH+13b, App14, CLTV15, BGL+15] or special subclass of constant degree comptuations [Lin16], or functional encryption schemes for NC^1 [AJ15, BV15, AJS15a] or NC^0 [LV16]. We refer the reader to [AJN+16] for an extensive discussion of the state-of-affairs of attacks.

1.4 Concurrent and Related Works

Concurrent and independent work: A concurrent and independent work achieving the same result of obtaining adaptively secure RAM delegation scheme is by Canetti et. al. [CCHR16]. Their scheme extends the selectively secure RAM delegation scheme of [CH16], and uses a new primitive called adaptive accumulators, which is interesting and potentially useful for other applications. They give a proof of adaptive security from scratch, extending the selective security proof of [CH16] in a non-black-box way. In contrast, our approach is semi-generic. We isolate our key ideas in an abstract proof framework, and then instantiate the existing selective security proof of [CCC+16] in this framework. The main difference from [CCC+16] is that we use historyless accumulators (instead of using positional accumulators). Our notion of historyless accumulators is seemingly different from adaptive accumulators; its not immediately clear how to get one from the other. One concrete benefit our approach has is that the usage of $\mathsf{i}\mathcal{O}$ is falsifiable, whereas in their construction of adaptive accumulators, $\mathsf{i}\mathcal{O}$ is used in a non-falsifiable way. More specifically, they rely on the $\mathsf{i}\mathcal{O}$-to-differing-input obfuscation transformation of [BCP14], which makes use of $\mathsf{i}\mathcal{O}$ in a non-falsifiable way.

Previous works on non-succinct garbled RAM: 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]. A black-box construction for *parallel* garbled RAM was later proposed by Lu and Ostrovsky [LO15] following the works of [BCP, CLT]. 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, where in the offline phase, the delegator still needs to run in time proportional to the worst case time complexity of the RAM program.

Previous works on succinct garbled RAM: 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] (following [KLW15], which gives a Turing machine delegation scheme) constructed fully succinct garbled RAM, and [CCC+16] additionally gives the first fully succinct garbled PRAM. However, their schemes only achieve selective security. Lifting to adaptive security while keeping succinctness is the contribution of this work.

1.5 Organization

We first give an overview of our approach in Section 2. Preliminaries are shown in Section 3. In Section 4, we present our abstract proof framework. The formal definition of adaptive delegation for RAMs is then presented in Section 5. The new primitive, history-less accumulator, is introduced in Section 6. Instantiation of RAM delegation using our abstract proof framework is presented in Section 7.

2 Overview

We now provide an overview of our abstract proof for lifting "nice" selective security proofs into adaptive security proofs. To the best of our knowledge, so far, the only general method going from selective to adaptive security is *complexity leveraging*, which however has (1) exponential security loss and (2) cannot be applied in RAM delegation setting for two reasons: (i) this will restrict the number of programs an adversary can choose and, (ii) the security parameter has to be scaled proportional to the number of program queries. This means that all the parameters grow proportional to the number of program queries.

Small-loss complexity leveraging: Nevertheless, we overcome the first limitation by showing a stronger version of complexity leveraging that has much smaller security loss, when the original selectively secure scheme (including its security game and security reduction) satisfy certain properties—we refer to the properties as *niceness* properties and the technique as *small-loss complexity leveraging*.

Local application: Still, many selectively secure schemes may not be *nice*, in particular, the CCC+ scheme. We broaden the scope of application of small-loss complexity leveraging using another idea: Instead of applying small-loss complexity leveraging to the scheme directly, we dissect its proof of selective security, and apply it to "smaller units" in the proof. Most commonly, proofs involve hybrid arguments; now, if every pair of neighboring hybrids with indistinguishability is *nice*, small-loss complexity leveraging can be applied *locally* to lift the indistinguishability to be resilient to adaptive adversaries, which then "sum up" to the global adaptive security of the scheme.

We capture the niceness properties abstractly and prove the above two steps abstractly. Interestingly, a challenging point is finding the right "language" (i.e. formalization) for describing selective and adaptive security games in a general way; we solve this by introducing *generalized security games*. With this language, the abstract proof follows with *simplicity* (completely disentangled from the complexity of specific schemes and their proofs, such as, the CCC+ scheme).

2.1 Classical Complexity Leveraging

Complexity leveraging says if a selective security game is $\mathsf{negl}(\lambda)2^{-L}$-secure, where λ is the security parameter and $L = L(\lambda)$ is the length of the information that selective adversaries choose statically (mostly at the beginning of the game), then the corresponding adaptive security game is $\mathsf{negl}(\lambda)$-secure. For example, the selective security of a public key encryption (PKE) scheme considers adversaries that choose two challenge messages v_0, v_1 of length n statically, whereas adaptive adversaries may choose v_0, v_1 adaptively depending on the public key. (See Figure 1.) By complexity leveraging, any PKE that is $\mathsf{negl}(\lambda)2^{-2n}$-selectively secure is also adaptively secure.

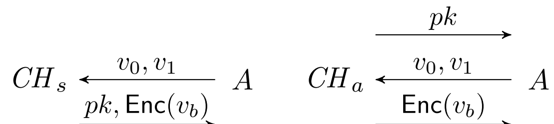

Figure 1 Left: Selective security of PKE. Right: Adaptive security of PKE.

The idea of complexity leveraging is extremely simple. However, to extend it, we need a general way to formalize it. This turns out to be non-trivial, as the selective and adaptive security games are defined separately (e.g., the selective and adaptive security games of PKE have different challengers CH_s and CH_a), and vary case by case for different primitives (e.g., in the security games of RAM delegation, the adversaries choose multiple programs over time, as opposed to in one shot). To overcome this, we introduce generalize security games.

2.2 Generalized Security Games

Generalized security games, like classical games, are between a challenger CH and an adversary A, but are meant to separate the information A chooses statically from its interaction with CH. More specifically, we model A as a non-uniform Turing machine with an additional write-only *special output tape*, which can be written to only at the beginning of the execution (See Figure 1). The special output tape allows us to capture (fully) selective and (fully) adaptive adversaries naturally: The former write all messages to be sent in the interaction with CH on the tape (at the beginning of the execution), whereas the latter write arbitrary information. Now, selective and adaptive security are captured by running the same (generalized) security game, with different types of adversaries (e.g., see Figure 2 for the generalized security games of PKE).

Now, complexity leveraging can be proven abstractly: If there is an adaptive adversary A that wins against CH with advantage $\mathsf{negl}(\lambda)$, there is a selective adversary A' that wins with advantage $\mathsf{negl}(\lambda)/2^L$, as A' simply writes on its tape a random guess ρ of A's messages, which is correct with probability $1/2^L$.

With this formalization, we can further generalize the security games in two aspects. First, we consider the natural class of semi-selective adversaries that choose only partial information statically, as opposed to its entire transcript of messages (e.g., in the selective security game of functional encryption in [GGH+13b] only the challenge messages are chosen selectively, whereas all functions are chosen adaptively). More precisely, an adversary is *F-semi-selective* if the initial choice ρ it writes to the special output tape is always consistent with its messages $m_1, \cdots, m_k$ w.r.t. the output of F, $F(\rho) = F(m_1, \cdots, m_k)$. Clearly, complexity leveraging w.r.t. F-semi-selective adversaries incurs a 2^{L_F}-security loss, where $L_F = |F(\rho)|$.

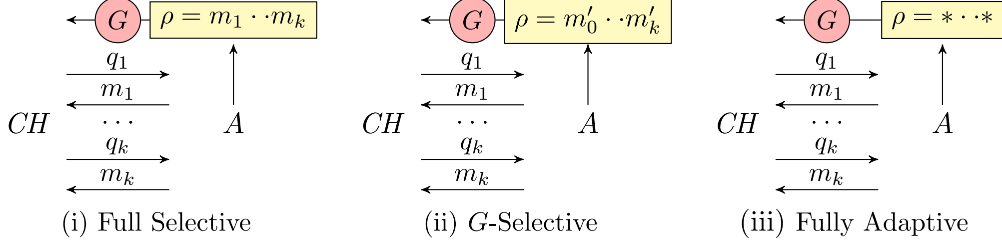

Figure 3 Three levels of adaptivity. In (ii) G-selective means $G(m_1 \cdot\cdot m_k) = G(m'_1 \cdot\cdot m'_k)$.

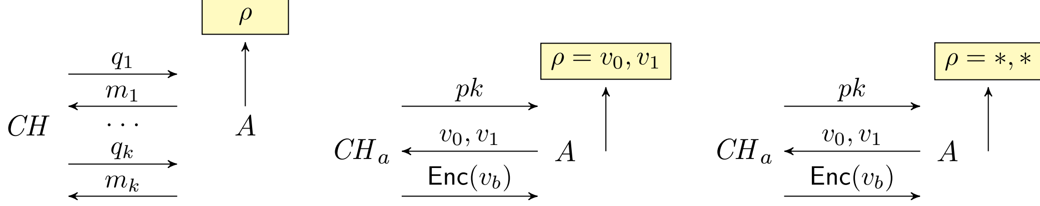

Figure 2 Left: A generalized game. Middle and Right: Selective and adaptive security of PKE described using generalized games.

Second, we allow the challenger to depend on some partial information $G(\rho)$ of the adversary's initial choice ρ, by sending $G(\rho)$ to CH, after A writes to its special output tape (See Figure 3)—we say such a game is G-*dependent*. At a first glance, this extension seems strange; few primitives have security games of this form, and it is unnatural to think of running such a game with a fully adaptive adversary (who does not commit to $G(\rho)$ at all). However, such games are prevalent *inside* selective security proofs, which leverage the fact that adversaries are selective (e.g., the selective security proof of the functional encryption of [GGH+13b] considers an intermediate hybrid where the challenger uses the challenge messages v_0, v_1 from the adversary to program the public key). Hence, this extension is essential to our eventual goal of applying small-loss complexity leveraging to neighboring hybrids, inside selective security proofs.

2.3 Small-loss Complexity Leveraging

In a G-dependent generalized game CH, *ideally*, we want a statement that $\mathsf{negl}(\lambda)2^{-L_G}$-selective security (i.e., against (fully) selective adversaries) implies $\mathsf{negl}(\lambda)$-adaptively security (i.e., against (fully) adaptive adversaries). We stress that the security loss we aim for is 2^{L_G}, related to the length of the information $L_G = G(\rho)$ that the challenger depends on,[4] as opposed to 2^L as in classical complexity leveraging (where L is the total length of messages selective adversaries choose statically). When $L \gg L_G$, the saving in security loss is significant. However, this ideal statement is clearly false in general.

1. For one, consider the special case where G always outputs the empty string, the statement means $\mathsf{negl}(\lambda)$-selective security implies $\mathsf{negl}(\lambda)$-adaptive security. We cannot hope to improve complexity leveraging unconditionally.
2. For two, even if the game is 2^{-L}-selectively secure, complexity leveraging does not apply to *generalized* security games. To see this, recall that complexity leveraging turns an adaptive adversary A with advantage δ, into a selective one B with advantage $\delta/2^L$, who guesses A's

[4] Because the challenger CH depends on L_G-bit of partial information $G(\rho)$ of the adversary's initial choice ρ, we do not expect to go below 2^{-L_G}-security loss unless requiring very strong properties to start with.

messages at the beginning. It relies on the fact that the challenger is oblivious of B's guess ρ to argue that messages to and from A are information theoretically independent of ρ, and hence ρ matches A's messages with probability $1/2^L$ (see Figure 3 again). However, in generalized games, the challenger does depend on some partial information $G(\rho)$ of B's guess ρ, breaking this argument.

To circumvent the above issues, we strengthen the premise with two niceness properties (introduced shortly). Importantly, both niceness properties still only provide $\mathsf{negl}(\lambda)2^{-L_G}$-security guarantees, and hence the security loss remains 2^{L_G}.

Lemma 1 (Informal, Small Loss Complexity Leveraging)**.** *Any G-dependent generalized security games with the following two properties for $\delta = \mathsf{negl}(\lambda)2^{-L_G}$ are adaptively secure.*

- *The game is* δ-G-hiding.
- *The game has a security reduction with* δ-statistical emulation property *to a δ-secure cryptographic assumption.*

We define δ-G-hiding and δ-statistical emulation properties shortly. We prove the above lemma in a modular way, by first showing the following semi-selective security property, and then adaptive security. In each step, we use one niceness property.

> **δ-semi-selective security:** We say that a G-dependent generalized security game CH is δ-semi-selective secure, if the winning advantage of any *G-semi-selective adversary* is bounded by $\delta = \mathsf{negl}(\lambda)2^{-L_G}$. Recall that such an adversary writes ρ to the special output tape at the beginning, and later choose adaptively any messages $m_1, \cdots, m_k$ consistent with $G(\rho)$, that is, $G(m_1, \cdots, m_k) = G(\rho)$ or $\perp$ (i.e., the output of G is undefined for $m_1, \cdots, m_k$).

Step 1 – From selective to G-semi-selective security This step encounters the same problem as in the first issue above: We cannot expect to go from $\mathsf{negl}(\lambda)2^{-L_G}$-selective to $\mathsf{negl}(\lambda)2^{-L_G}$-semi-selective security unconditionally, since the latter is dealing with much more adaptive adversaries. Rather, we consider only cases where the selective security of the game with CH is proven using a *black-box straight-line* security reduction R to a game-based intractability assumption with challenger CH' (c.f. falsifiable assumption [Nao03]). We identify the following sufficient conditions on R and CH' under which semi-selective security follows.

Recall that a reduction R simultaneously interacts with an adversary A (on the right), and leverages A's winning advantage to win against the challenger CH' (on the left). It is convenient to think of R and CH' as a compound machine $CH' \leftrightarrow R$ that interacts with A, and outputs what CH' outputs. Our condition requires that $CH' \leftrightarrow R$ emulates statistically every next message and output of CH. More precisely,

> **δ-statistical emulation property**: For every possible $G(\rho)$ and partial transcript $\tau = (q_1, m_1, \cdots, q_k, m_k)$ consistent with $G(\rho)$ (i.e., $G(m_1, \cdots, m_k) = G(\rho)$ or $\perp$), condition on them $(G(\rho), \tau)$ appearing in interactions with CH or $CH' \leftrightarrow R$, the distributions of the next message or output from CH or $CH' \leftrightarrow R$ are δ-statistically close.

We show that this condition implies that for any G-semi-selective adversary, its interactions with CH and $CH' \leftrightarrow R$ are $\mathrm{poly}(\lambda)\delta$-statistically close (as the total number of messages is $\mathrm{poly}(\lambda)$), as well as the output of CH and CH'. Hence, if the assumption CH' is $\mathsf{negl}(\lambda)2^{-L_G}$-secure against arbitrary adversaries, so is CH against G-semi-selective adversaries.[5]

[5] Technically, we also require that CH and CH' have the same winning threshold, like both $1/2$ or 0.

Further discussion: We remark that the statistical emulation property is a strong condition that is sufficient but not necessary. A weaker requirement would be requiring the game to be G-semi-selective secure directly. However, we choose to formulate the statistical emulation property because it is a typical way how reductions are built, by emulating perfectly the messages and output of the challenger in the honest games. Furthermore, given R and CH', the statistical emulation property is easy to check, as from the description of R and CH', it is usually clear whether they emulate CH statistically close or not.

Step 2 – From G-semi-selective to adaptive security we would like to apply complexity leveraging to go from $\mathsf{negl}(\lambda)2^{-L_G}$-semi-selective security to adaptive security. However, we encounter the same problem as in the second issue above. To overcome it, we require the security game to be G-hiding, that is, the challenger's messages computationally hides $G(\rho)$.

> **δ-G-hiding:** For any ρ and ρ', interactions with CH after receiving $G(\rho)$ or $G(\rho')$ are indistinguishable to any polynomial-time adversaries, except from a δ distinguishing gap.

Let's see how complexity leveraging can be applied now. Consider again using an adaptive adversary A with advantage $1/\operatorname{poly}(\lambda)$ to build a semi-selective adversary B with advantage $1/\operatorname{poly}(\lambda)2^{L_G}$, who guesses A's choice of $G(m_1, \cdots, m_k)$ later. As mentioned before, since the challenger in the generalized game depends on B's guess τ, classical complexity leveraging argument does not apply. However, by the δ-G-hiding property, B's advantage differ by at most δ, when moving to a hybrid game where the challenger generates its messages using $G(\rho)$, where ρ is what A writes to its special output tape at the beginning, instead of τ. In this hybrid, the challenger is oblivious of B's guess τ, and hence the classical complexity leveraging argument applies, giving that B's advantage is at least $1/\operatorname{poly}(\lambda)2^{L_G}$. Thus by G-hiding, B's advantage in the original generalized game is at least $1/\operatorname{poly}(\lambda)2^{L_G} - \delta = 1/\operatorname{poly}(\lambda)2^{L_G}$. This gives a contradiction, and concludes the adaptive security of the game.

Summarizing the above two steps, we obtain our informal lemma on small-loss complexity leveraging.

2.4 Local Application

In many cases, small-loss complexity leveraging may not directly apply, since either the security game is not G-hiding, or the selective security proof does not admit a reduction with the statistical emulation property. We can broaden the application of small-loss complexity leveraging by looking into the selective security proofs and apply small loss complexity leveraging on smaller "steps" inside the proof. For our purpose of getting adaptively secure RAM delegation, we focus on the following common proof paradigm for showing indistinguishability based security. But the same principle of local application could be applied to other types of proofs.

A common proof paradigm for showing the indistinguishability of two games $Real_0$ and $Real_1$ against selective adversaries is the following:

- 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 to selective adversaries.

Then, by standard hybrid arguments, the real games are selectively indistinguishable.

To lift such a selective security proof into an adaptive security proof, we first cast all real and hybrids games into our framework of generalized games, which can be run with both selective and adaptive adversaries. If we can obtain that neighboring hybrids games are also indistinguishable to adaptive adversaries, then the adaptive indistinguishability of the two real games follow simply from hybrid arguments. Towards this, we apply small-loss complexity leveraging on neighboring hybrids. More specifically, H_i and H_{i+1} are adaptively indistinguishable, if they satisfy the following properties:

- H_i and H_{i+1} are respectively G_i and G_{i+1}-dependent, as well as δ-$(G_i||G_{i+1})$-hiding, where $G_i||G_{i+1}$ outputs the concatenation of the outputs of G_i and G_{i+1} and $\delta = \mathsf{negl}(\lambda)2^{-L_{G_i}-L_{G_{i+1}}}$.
- The selective indistinguishability of H_i and H_{i+1} is shown via a reduction R to a δ-secure game-based assumption and the reduction has δ-statistical emulation property.

Thus, applying small-loss complexity leveraging on every neighboring hybrids, the maximum security loss is $2^{2L_{max}}$, where $L_{max} = \max(L_{G_i})$. Crucially, if every hybrid H_i have small L_{G_i}, the maximum security loss is small. In particular, we say that a selective security proof is "*nice*" if it falls into the above framework and all G_i's have only *logarithmic length* outputs — such "nice" proofs can be lifted to proofs of adaptive indistinguishability with only polynomial security loss. This is exactly the case for the CCC+ scheme, which we explain next.

2.5 The CCC+ Scheme and Its Nice Proof

CCC+ proposed a selectively secure RAM delegation scheme in the persistent database setting. We now show how CCC+ scheme can be used to instantiate the abstract framework discussed earlier in this Section. We only provide with relevant details of CCC+ and refer the reader to Section 7 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 highlight below two important building blocks relevant to our discussion. Additional tools such as iterators and splittable signatures are also employed in their construction.

- *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. In the security proof, accumulators 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, such programming requires to know ahead of time all the changes the storage undergoes since its initialization. Henceforth, we refer to the hybrids to be in ENFORCE-MODE when the accumulator parameters are programmed and the setting when it is not programmed to be REAL-MODE.
- *"Puncturable" Oblivious RAM*: Oblivious RAM (ORAM) is a randomized compiler that compiles any RAM program into one with a fixed distribution of random access pattern to hide its actual (logic) access pattern. CCC+ relies on stronger "puncturable" property of specific ORAM construction of [CP13], which roughly says the compiled access pattern of a particular logic memory access can be simulated if certain local ORAM randomness is information theoretically "punctured out," and this local randomness is determined at the time the logic

memory location is last accessed. Henceforth, we refer to the hybrids to be in PUNCTURING-MODE when the ORAM randomness is punctured out.

We show that the security proof of CCC+ has a nice proof. 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 three cases depending on the type of neighboring hybrids H_i and H_{i+1}:

1. ORAM IS IN PUNCTURING-MODE IN ONE OR BOTH OF THE NEIGHBORING HYBRIDS: In this case, the hybrid challenger needs to know which ORAM local randomness to puncture out to hide the logic memory access to location q at a particular time point t. As mentioned, this local randomness appears for the first time at the last time point t' that location q is accessed, possibly by a previous machine. As a result, in the proof, some machine components need to be programmed depending on the memory access of later machines. In this case, G_i or G_{i+1} need to contain information about q, t and t', which can be described in $\text{poly}(\lambda)$ bits.
2. POSITIONAL ACCUMULATOR IS IN ENFORCE-MODE IN ONE OR BOTH OF THE NEIGHBORING HYBRIDS: 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.
3. REMAINING CASES: In remaining cases, the indistinguishability of neighboring hybrids reduces to the security of other cryptographic primitives, such as, iterators, splittable signatures, indistinguishability obfuscation and others. We note that in these cases, we simply have $G_i = G_{i+1} = \mathsf{null}$, which outputs an empty string.

As seen from the above description, only 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 size requirements. That is, the output length of G_i is now short. By plugging this into the CCC+ scheme, we obtain a "nice" security proof.

All that remains is to construct history-less accumulators. The construction of this primitive can be found in Section 6.

3 Preliminaries

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

3.1 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* $C:\{0,1\}^{\mathsf{in}}\rightarrow\{0,1\}$ *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:*

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

3.2 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 $K\xleftarrow{\$}\mathcal{K}$ 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 3 ([GGM86, BW13, BGI14, KPTZ13]). *If* μ*-secure one-way functions*[6] *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.*

3.3 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 3.3.1 and 3.3.2. Then, we propose a variant of positional accumulators, termed as *history-less accumulators* in Section 6.

[6] 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 $\mu(\lambda)$.

3.3.1 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 $v_0 \in \mathsf{St}_\lambda$.

$\mathsf{ItrEnforce}(1^\lambda, T, \vec{m} = (m_1, \ldots, m_k))$ The enforced setup algorithm takes as input the security parameter λ (in unary), an integer bound T (in binary) and k messages $(m_1, \ldots, m_k)$, 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 $k \le T$, we will use the notation $\mathsf{Iterate}^k(\mathsf{PP}_{\mathsf{Itr}}, v_0, (m_1, \ldots, m_k))$ 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 $1 \le j \le k-1$.

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.*

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.

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

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

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

3.4 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 program, and use memory or database to denote storage outside the program.

Let us consider the persistent database setting. A multiple-program RAM computation, $\Pi = (\mathsf{mem}^{0,0}, \{F_{\mathsf{sid}}\}_{\mathsf{sid}=1}^{l})$, is specified by a sequence of programs $\{F_i\}_{i=1}^{l}$ and an initial memory $\mathsf{mem}^{0,0}$, where the session identity and total number of programs are denoted by sid and l. As above, the evaluator prepares initial state and memory, 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}. For simplicity, we adopt conventions w.r.t 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. At $t = t^*_{\mathsf{sid}}$ as termination, F_{sid} does not produce any memory access command.
5. 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 $\mathsf{st}^{\mathsf{sid}+1,0}$.

Computation Trace. For a multiple-program RAM computation Π, computation trace is a tuple of configurations defined as $\mathsf{conf}\langle\Pi\rangle = (\mathsf{mem}^{0,0}, \{\{\mathsf{st}^t_{\mathsf{sid}}\}_{t=0}^{t^*_{\mathsf{sid}}}, \{a^t_{\mathsf{sid}}\}_{t=1}^{t^*_{\mathsf{sid}}-1}\}_{\mathsf{sid}=1}^{l})$ for all session identities sid and time steps t.

4 Abstract Proof

In this section, we present our abstract proof that turns "nice" selective security proofs, to adaptive security proofs. As discussed in the introduction, we use generalized security experiments and games to describe our transformation. We present small-loss complexity leveraging in Section 4.3 and how to locally apply it in Section 4.4. In the latter, we focus our attention on proofs of indistinguishability against selective adversaries, as opposed to proofs of arbitrary security properties.

4.1 Cryptographic Experiments and Games

We recall 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 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 intractability 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.*

4.2 Generalized Cryptographic Games

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 standard cryptographic experiments so that the same experiment can work with both selective and adaptive adversaries. This is achieved 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. (See Figure 3.)

Definition 11 (Generalized Experiments)**.** *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 all the functions 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 $G(x) = y$.

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 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.

We can now quantify the level of selective/adaptive security of a generalized cryptographic game.

Definition 13 ($\mathcal{F}$-Selective Security)**.** *A generalized cryptographic game $(\mathcal{CH}, \mathcal{G}, \tau)$ is $\mathcal{F}$-*selective secure *if the advantage of every non-uniform* PPT *$\mathcal{F}$-selective adversary $\mathcal{A}$ is negligible.*

4.3 Small-loss Complexity Leveraging

In this section, we present our small-loss complexity leveraging technique to lift fully selective security to fully adaptive security for a generalized cryptographic game $\Pi = (\mathcal{CH}, \mathcal{G}, \tau)$, provided

that the game and its (selective) security proof satisfies certain niceness properties. We will focus on the following class of *guessing* games, which captures indistinguishability security. We remark that our technique also applies to generalized cryptographic games with arbitrary threshold (See Remark 1).

Definition 14 (Guessing Games). *A generalized game (CH, G, τ) (for a security parameter λ) is a* guessing game *if it has the following structure.*

- *At beginning of the game, CH samples a uniform bit $b \leftarrow \{0,1\}$.*
- *At the end of the game, the adversary guesses a bit $b' \in \{0,1\}$, and he wins if $b = b'$.*
- *When the adversary aborts, his guess is a uniform bit $b' \leftarrow \{0,1\}$.*
- *The threshold $\tau = 1/2$.*

The definition extends naturally to a sequence of games $\Pi = (\mathcal{CH}, \mathcal{G}, 1/2)$.

Our technique consists of two modular steps: First reach $\mathcal{G}$-selective security, and then adaptive security, where the first step applies to any generalized cryptographic game.

4.3.1 Step 1: $\mathcal{G}$-Selective Security

In general, a fully selectively secure Π may not be $\mathcal{F}$-selective secure for $\mathcal{F} \neq \mathcal{F}_{\text{id}}$, where $\mathcal{F}_{\text{id}}$ denotes the identity function. We restrict our attention to the following case: The security is proved by a straight-line black-box security reduction from Π to an intractability assumption $(\mathcal{CH}', \tau')$, where the reduction is an ensemble of PPT machines $\mathcal{R} = \{R_\lambda\}$ that interacts simultaneously with an adversary for Π and $\mathcal{CH}'$, the reduction is syntactically well-defined with respect to any class of $\mathcal{F}$-selective adversary. This, however, does not imply that R is a correct reduction to prove $\mathcal{F}$-selective security of Π. Here, we identify a sufficient condition on the "niceness" of reduction that implies $\mathcal{G}$-selective security of Π. We start by defining the syntax of a straight-line black-box security reduction.

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. 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 $\mathcal{A}[\mathcal{G}]$.

Definition 15 (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 $\mathsf{Exp}(\lambda, CH' \leftrightarrow R, A[G])$.*

*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).*

At a high-level, we say that a reduction is μ-nice, where μ is a function, if it satisfies the following syntactical property: 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 ($m_t = \bot$ 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 $CH'\leftrightarrow R$ 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 $m_t = \bot$ 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 16 (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:

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

  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.*

Moreover, we say that a reduction $\mathcal{R} = \{R_\lambda\}$ from a generalized cryptographic game $(\mathcal{CH}, \mathcal{G}, \tau)$ to a intractability assumption $(\mathcal{CH}', \tau)$ is nice *if there is a negligible function μ, such that, R_λ is $\mu(\lambda)$-nice for every λ.*

When a reduction is μ-nice with negligible μ, it is sufficient to imply $\mathcal{G}$-selective security of the corresponding generalized cryptographic game.

Lemma 2. *Suppose R is a μ-nice reduction from (CH, G, τ) to (CH', τ) for security parameter λ, and A is a deterministic G-semi-selective adversary that wins (CH, G, τ) with advantage $\gamma(\lambda)$, then $R\leftrightarrow A[G]$ is an adversary for (CH', τ) with advantage $\gamma(\lambda) - t(\lambda) \cdot \mu(\lambda)$, where $t(\lambda)$ is an upper bound on the run-time of R.*

Proof. 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,

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

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

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

The last two lines above correspond to the following:

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

Therefore, $p \le 2\mu + q$ and $2d_{i+1} \le 2\mu + p_0 + p_1 + q$. 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 $d_{i+1} \le d_i + \mu$, which is a contradiction. □

By a standard argument, Lemma 2 implies the following asymptotic version theorem.

Theorem 4. *If there exists a nice reduction $\mathcal{R}$ from a generalized cryptographic game $(\mathcal{CH}, \mathcal{G}, \tau)$ to an intractability assumption $(\mathcal{CH}', \tau)$, then $(\mathcal{CH}, \mathcal{G}, \tau)$ is $\mathcal{G}$-selectively secure.*

4.3.2 Step 2: Fully Adaptive Security

We now show how to move from $\mathcal{G}$-selective security to fully adaptive security for the class of guessing games with security loss $2^{L_G(\lambda)}$, where $L_G(\lambda)$ is the output length of $\mathcal{G}$, provided that the challenger's messages hide the information of $G(\alpha)$ computationally. We start with formalizing this hiding property.

Roughly speaking, the challenger CH of a generalized experiment (CH, G) is G-hiding, if for any α and α', interactions with CH receiving $G(\alpha)$ or $G(\alpha')$ at the beginning are indistinguishable. Denote by $CH(x)$ the challenger with x hardcoded as the first message.

Definition 17 (G-hiding)**.** *We say that a generalized guessing game (CH, G, τ) is $\mu(\lambda)$-G-hiding for security parameter λ, if its challenger CH satisfies that for every α and α', and every non-uniform* PPT *adversary A,*

$$|\Pr[\mathsf{Out}_A(\lambda, CH(G(\alpha)), A) = 1] - \Pr[\mathsf{Out}_A(\lambda, CH(G(\alpha')), A) = 1]| \le \mu(\lambda)$$

Moreover, we say that a generalized cryptographic guessing game $(\mathcal{CH}, \mathcal{G}, \tau)$ is $\mathcal{G}$-hiding, if there is a negligible function μ, such that, $(\mathcal{CH}_\lambda, G_\lambda, \tau(\lambda))$ is $\mu(\lambda)$-G_λ-hiding for every λ.

The following lemma says that if a generalized guessing game $(CH, G, 1/2)$ is G-selectively secure and G-hiding, then it is fully adaptively secure with 2^{L_G} security loss.

Lemma 3. *Let $(CH, G, 1/2)$ be a generalized cryptographic guessing game for security parameter λ. If there exists a fully adaptive adversary A for $(CH, G, 1/2)$ with advantage $\gamma(\lambda)$ and $(CH, G, 1/2)$ is $\mu(\lambda)$-G-hiding with $\mu(\lambda) \leq \gamma/2^{L_G(\lambda)+1}$, then there exists a G-selective adversary A' for $(CH, G, 1/2)$ with advantage $\gamma(\lambda)/2^{L_G(\lambda)+1}$, where L_G is the output length of G.*

Proof. We construct a wrapper adversary A' for $(CH, G, 1/2)$ who runs A internally as follow:

- When A writes initial choice α on its special output tape, A' ignores α. Instead, it samples a random $g \leftarrow \{0,1\}^{L_G}$, and finds an α' such that $G(\alpha') = g$ (recall that we assume G is reversely computable), and writes α' on its own special output tape.
- A' forwards messages to and from A externally, and aborts whenever A aborts (recall that in a guess game, when A or A' aborts, it makes a uniform guess b').
- If A does not abort and outputs bit b', A' checks whether its initial random guess g matches the output of G applied to the transcript of messages ρ that A sends, i.e., $g = G(\rho)$; we denote this event match. If it is the case, then A' outputs b' as well. Otherwise, it aborts.

Since A' aborts whenever its initial guess g does not match the messages A sends w.r.t. G, A' is a G-selective adversary.

We claim that A' achieves at least $\gamma/2^{L_G+1}$ advantage in $(CH, G, 1/2)$. To prove this, we consider another hybrid experiment, which is identical to the experiment $\mathsf{Exp}(1^\lambda, CH, G, A')$, except that CH instead of receiving $G(\alpha')$ at the beginning, receives $G(\alpha)$. Observe that in this hybrid experiment g is information theoretically hidden from both CH and A (emulated by A'), and the views of CH and A are identical to their views in an honest execution between them directly $\mathsf{Exp}(\lambda, CH, G, A)$ (without the wrapper A' in between). Therefore, the guess g by A' matches $G(\rho)$ with probability $1/2^{L_G}$, and conditioned on guessing correctly, it's advantage is γ. Overall, the advantage of A' is $\gamma/2^{L_G}$.

Next, by the G-hiding property of $(CH, G, 1/2)$, the views of A' in experiment $\mathsf{Exp}(1^\lambda, CH, G, A')$ and the hybrid experiment are μ-indistinguishable. By the construction of the guessing game, the advantage of A' in these two experiments differ by at most μ. Hence, the advantage of A' in $\mathsf{Exp}(1^\lambda, CH, G, A')$ is at least $(\gamma/2^{L_G}) - \mu \geq (\gamma/2^{L_G+1})$.

□

Therefore, for a generalized cryptographic guessing game $(\mathcal{CH}, \mathcal{G}, \tau)$, if $\mathcal{G}$ has logarithmic output length $L_G(\lambda) = O(\log \lambda)$ and the game is $\mathcal{G}$-hiding, then its $\mathcal{G}$-selective security implies fully adaptive security.

Theorem 5. *Let $(\mathcal{CH}, \mathcal{G}, \tau)$ be a $\mathcal{G}$-selectively secure generalized cryptographic guessing game. If $(\mathcal{CH}, \mathcal{G}, \tau)$ is $\mathcal{G}$-hiding and $L_G(\lambda) = O(\log \lambda)$, then $(\mathcal{CH}, \mathcal{G}, \tau)$ is fully adaptively secure.*

Remark 1. The above proof of small-loss complexity leveraging can be extended to a more general class of security games, beyond the guessing games. The challenger with an arbitrary threshold τ has the form that if the adversary aborts, the challenger toss a biased coin and outputs 1 with probability τ. The same argument above goes through for games with this class of challengers.

4.4 Nice Indistinguishability 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. We focus on a common type of indistinguishability proof, which consists of a sequence of hybrid experiments and

shows that neighboring hybrids are indistinguishable via a reduction to a intractability assumption. We formalize required nice properties of the hybrids and reductions such that a fully selective security proof can be lifted to prove fully adaptive security by local application of small-loss complexity leveraging technique to neighboring hybrids. We start by describing common indistinguishability proofs using the language of generalized experiments and games.

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 $G = \mathsf{null} : \{0,1\}^* \to \{\varepsilon\}$ 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 a proof of indistinguishability of $(\mathcal{RL}_0, \mathsf{null})$ and $(\mathcal{RL}_1, \mathsf{null})$ against fully selective adversaries via a sequence of hybrid experiments. As discussed in the overview, the challenger of the hybrids often depends non-trivially on partial information of the adversary's initial choice. Namely, the hybrids are generalized cryptographic experiments with non-trivial $\mathcal{G}$ function. Since small-loss complexity leveraging has exponential security loss in the output length of $\mathcal{G}$, we require all hybrid experiments have logarithmic-length $\mathcal{G}$ function. Below, for convenience, we use the notation $\mathcal{X}_i$ to denote an ensemble of the form $\{X_{i,\lambda}\}$, and the notation $\mathcal{X}_I$ with a function I, as the ensemble $\{X_{I(\lambda),\lambda}\}$.

1. **Security via hybrids with logarithmic-length $\mathcal{G}$ function:** 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,

$$(\mathcal{H}_0, \mathcal{G}_0) = (\{H_{0,\lambda}\}, \{G_{0,\lambda}\}) = (\mathcal{RL}_0, \mathsf{null})$$
$$(\mathcal{H}_\ell, \mathcal{G}_\ell) = (\{H_{\ell(\lambda),\lambda}\}, \{G_{\ell(\lambda),\lambda}\}) = (\mathcal{RL}_1, \mathsf{null}),$$

Furthermore, there exists a function $L_G(\lambda) = O(\log \lambda)$ such that for every λ and i, the output length of $G_{i,\lambda}$ is at most $L_G(\lambda)$.

We next formalize required properties to lift security proof of neighboring hybrids. Towards this, we formulate indistinguishability of two generalized cryptographic experiments as a generalized cryptographic guessing game. The following is a known fact.

Fact. Let $(\mathcal{CH}_0, \mathcal{G}_0)$ and $(\mathcal{CH}_1, \mathcal{G}_1)$ be two ensembles of generalized cryptographic experiments, $\mathcal{F}$ be an ensemble of efficiently computable functions, and $\mathcal{C}_\mathcal{F}$ 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 equivalent to $\mathcal{F}$-selective security of a generalized cryptographic guessing game $(\mathcal{D}, \mathcal{G}_0||\mathcal{G}_1, 1/2)$, 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 $b = b'$.

By the above fact, indistinguishability of neighboring hybrids $(\mathcal{H}_i, \mathcal{G}_i)$ and $(\mathcal{H}_{i+1}, \mathcal{G}_{i+1})$ against $\mathcal{F}$-selective adversary is equivalent to $\mathcal{F}$-selective security of the generalized cryptographic guessing game $(\mathcal{D}_i, \mathcal{G}_i||\mathcal{G}_{i+1}, 1/2)$, where $\mathcal{D}_i = \{D_{i,\lambda[H_{i,\lambda},H_{i+1,\lambda}]}\}$. We can now state the required properties for every pair of neighboring hybrids:

2. **Indistinguishability of neighboring hybrids via nice reduction** For every neighboring hybrids $(\mathcal{H}_i, \mathcal{G}_i)$ and $(\mathcal{H}_{i+1}, \mathcal{G}_{i+1})$, their indistinguishability proof against fully selective adversary is established by a nice reduction $\mathcal{R}_i$ from the corresponding guessing game $(\mathcal{D}_i, \mathcal{G}_i||\mathcal{G}_{i+1}, 1/2)$ to some intractability assumption.
3. **$\mathcal{G}_i||\mathcal{G}_{i+1}$-hiding** For every neighboring hybrids $(\mathcal{H}_i, \mathcal{G}_i)$ and $(\mathcal{H}_{i+1}, \mathcal{G}_{i+1})$, their corresponding guessing game $(\mathcal{D}_i, \mathcal{G}_i||\mathcal{G}_{i+1}, 1/2)$ is $\mathcal{G}_i||\mathcal{G}_{i+1}$-hiding.

In summary,

Definition 18 (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, and 3 described above.*

It is now straightforward to lift security of nice indistinguishability proof by local application of small-loss complexity leveraging for neighboring hybrids.

Theorem 6. *A "nice" proof for the indistinguishability of two real experiments* $(\mathcal{RL}_0, \mathsf{null})$ *and* $(\mathcal{RL}_1, \mathsf{null})$ *implies that these experiments are indistinguishable against fully adaptive adversaries.*

Proof. For every neighboring hybrids $(\mathcal{H}_i, \mathcal{G}_i)$ and $(\mathcal{H}_{i+1}, \mathcal{G}_{i+1})$, the corresponding generalized cryptographic guessing game $(\mathcal{D}_i, \mathcal{G}_i||\mathcal{G}_{i+1}, 1/2)$ is fully selectively secure. By property 2 and Theorem 4, $(\mathcal{D}_i, \mathcal{G}_i||\mathcal{G}_{i+1}, 1/2)$ is $\mathcal{G}_i||\mathcal{G}_{i+1}$-selectively secure. By the fact that the output length of $\mathcal{G}_i||\mathcal{G}_{i+1}$ is at most $2L_G(\lambda) = O(\log \lambda)$, the $\mathcal{G}_i||\mathcal{G}_{i+1}$-hiding property, and Theorem 5, $(\mathcal{D}_i, \mathcal{G}_i||\mathcal{G}_{i+1}, 1/2)$ is fully adaptively secure. This is equivalent to say that $(\mathcal{H}_i, \mathcal{G}_i)$ and $(\mathcal{H}_{i+1}, \mathcal{G}_{i+1})$ are indistinguishable against fully adaptive adversaries. Finally, since all neighboring hybrids are indistinguishable against fully adaptive adversary, a standard hybrid argument implies that $(\mathcal{RL}_0, \mathsf{null})$ and $(\mathcal{RL}_1, \mathsf{null})$ are indistinguishable against fully adaptive adversaries. □

5 Adaptive Delegation for RAM computation

In this section, we introduce the notion of adaptive delegation for RAM computation ($\mathcal{DEL}$) and state our formal theorem. 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.

We first give a brief overview of the structure of the delegation scheme. First, the setup algorithm DBDel, which takes as input the database, is executed. The result is the database encoding and the secret key. PDel is the program encoding procedure. It takes as input the secret key, session ID and the program to be encoded. Eval takes as input the program encoding of session ID sid along with a memory encoding associated with sid. The result is an encoding which is output along with a proof. Along with this the updated memory state is also output. We employ

a verification algorithm Ver to verify the correctness of computation using the proof output by Eval. Finally, Dec is used to decode the output encoding.

We present the formal definition below.

5.1 Definition

Definition 19 ($\mathcal{DEL}$ with Persistent Database)**.** *A $\mathcal{DEL}$ scheme with persistent database, consists of PPT 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 $\mathcal{P}$.*

- $\mathcal{DEL}.\mathsf{DBDel}(1^\lambda, \mathsf{mem}^0, S) \to (\widetilde{\mathsf{mem}}^1, \mathsf{sk})$*: 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*.*

- $\mathcal{DEL}.\mathsf{PDel}(1^\lambda, \mathsf{sk}, \mathsf{sid}, P_{\mathsf{sid}}) \to \widetilde{P}_{\mathsf{sid}}$*: 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* $\widetilde{P}_{\mathsf{sid}}$*.*

- $\mathcal{DEL}.\mathsf{Eval}\left(1^\lambda, T, S, \widetilde{P}_{\mathsf{sid}}, \widetilde{\mathsf{mem}}^{\mathsf{sid}}\right) \to \left(c_{\mathsf{sid}}, \sigma_{\mathsf{sid}}, \widetilde{\mathsf{mem}}^{\mathsf{sid}+1}\right)$*: The evaluating algorithm* Eval *is a deterministic algorithm which takes as input the security parameter* 1^λ*, time bound* T*, 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* $(y_{\mathsf{sid}}, \mathsf{mem}^{\mathsf{sid}+1}) = P_{\mathsf{sid}}(\mathsf{mem}^{\mathsf{sid}})$*.*

- $\mathcal{DEL}.\mathsf{Ver}(1^\lambda, \mathsf{sk}, c_{\mathsf{sid}}, \sigma_{\mathsf{sid}}) \to b_{\mathsf{sid}} \in \{0,1\}$*: The verification algorithm takes as input the security parameter* 1^λ*, secret key* sk*, encoding* c_{sid}*, proof* σ_{sid} *and returns* $b_{\mathsf{sid}} = 1$ *if* σ_{sid} *is a valid proof for* c_{sid}*, or returns* $b_{\mathsf{sid}} = 0$ *if not.*

- $\mathcal{DEL}.\mathsf{Dec}(1^\lambda, \mathsf{sk}, c_{\mathsf{sid}}) \to y_{\mathsf{sid}}$*: 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*.*

Associated to the above scheme are correctness, (adaptive) security, (adaptive) soundness and efficiency properties.

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

- $(\widetilde{\mathsf{mem}}^1, \mathsf{sk}) \leftarrow \mathcal{DEL}.\mathsf{DBDel}(1^\lambda, \mathsf{mem}^0, S)$;
- $\widetilde{P}_{\mathsf{sid}} \leftarrow \mathcal{DEL}.\mathsf{PDel}(1^\lambda, \mathsf{sk}, \mathsf{sid}, P_{\mathsf{sid}})$;
- $(c_{\mathsf{sid}}, \sigma_{\mathsf{sid}}, \widetilde{\mathsf{mem}}^{\mathsf{sid}+1}) \leftarrow \mathcal{DEL}.\mathsf{Eval}(1^\lambda, T, S, \widetilde{P}_{\mathsf{sid}}, \widetilde{\mathsf{mem}}^{\mathsf{sid}})$;
- $b_{\mathsf{sid}} = \mathcal{DEL}.\mathsf{Ver}(1^\lambda, \mathsf{sk}, c_{\mathsf{sid}}, \sigma_{\mathsf{sid}})$;
- $y_{\mathsf{sid}} = \mathcal{DEL}.\mathsf{Dec}(1^\lambda, \mathsf{sk}, c_{\mathsf{sid}})$;
- $(y'_{\mathsf{sid}}, \mathsf{mem}^{\mathsf{sid}+1}) \leftarrow P_{\mathsf{sid}}(\mathsf{mem}^{\mathsf{sid}})$;

The following holds:

$$\Pr\left[(y_{\mathsf{sid}} = y'_{\mathsf{sid}} \wedge b_{\mathsf{sid}} = 1)\ \forall \mathsf{sid}, 1 \leq \mathsf{sid} \leq l\right] = 1.$$

Adaptive Security (full privacy). This property is designed to protect the privacy of the database and the programs from the adversarial server. We formalize this using a simulation based definition. In the real world, the adversary is supposed to declare the database at the beginning of the game. The challenger computes the database encoding and sends it across to the adversary. After this, the adversary can submit programs to the challenger and in return it receives the corresponding program encodings. We emphasize the program queries can be made adaptively. On the other hand, in the simulated world, the simulator does not get to see either the database or the programs submitted by the adversary. But instead it receives as input the length of the database, the lengths of the individual programs and runtimes of all the corresponding computations.[7] It then generates the simulated database and program encodings. The job of the adversary in the end is to guess whether he is interacting with the challenger (real world) or whether he is interacting with the simulator (ideal world).

Definition 20. *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* $\mathsf{Exp\text{-}Del\text{-}Privacy}(1^\lambda, \mathcal{DEL}, \mathcal{A}, \mathcal{S})$ *is at most negligible in* λ.

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

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. This property is designed to protect the clients against adversarial servers producing invalid output encodings. This is formalized in the form of a security experiment: the adversary submits the database to the challenger. The challenger responds with the database encoding. The adversary then chooses programs to be encoded adaptively. In response, the challenger sends the corresponding program encodings. In the end, the adversary is required to submit the output encoding and the corresponding proof. The soundness property requires that the adversary can only submit a convincing "false" proof only with negligible probability.

Definition 21. *A delegation scheme* $\mathcal{DEL}$ *is said to be* adaptively sound *if for all sufficiently large* $\lambda \in \mathbb{N}$*, for all total round* $l \in \mathrm{poly}(\lambda)$*, time bound* T*, space bound* S*, there exists an interactive*

[7] Note that unlike the standard simulation based setting, the simulator does not receive the output of the programs. This is because the output of the computation is never revealed to the adversary.

PPT adversary $\mathcal{A}$*, such that the probability of* $\mathcal{A}$ *win in the following security game* $\textsf{\textit{Exp-Del-Soundness}}(1^\lambda, \mathcal{DEL}, \mathcal{A})$ *is at most negligible in* λ*.*

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

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, where* c_k *is generated as follows: for* $\mathsf{sid} = 1, \ldots, k$*,* $(c_{\mathsf{sid}}, \sigma_{\mathsf{sid}}, \widetilde{\mathsf{mem}}^{\mathsf{sid}+1}) \leftarrow \mathcal{DEL}.\mathsf{Eval}(1^\lambda, T, S, \widetilde{P}_{\mathsf{sid}}, \widetilde{\mathsf{mem}}^{\mathsf{sid}})$*.*

Efficiency. For every session with session ID sid, we require that DBDel and PDel execute in time $\text{poly}(\lambda, |\mathsf{mem}^0|)$ and $\text{poly}(\lambda, |P_{\mathsf{sid}}|)$ respectively. Furthermore we require that Eval run in time $\text{poly}(\lambda, t^*_{\mathsf{sid}})$, where t^*_{sid} denotes the running time of P_{sid} on $\mathsf{mem}^{\mathsf{sid}}$. We require that both Ver and Dec run in time $\text{poly}(\lambda, |y_{\mathsf{sid}}|)$. Finally, the length of $c_{\mathsf{sid}}, \sigma_{\mathsf{sid}}$ should depend only on $|y_{\mathsf{sid}}|$.

A construction of adaptive delegation is provided in Section 7 with its security proof.

Theorem 7. *Assuming the existence of* $\mathsf{i}\mathcal{O}$ *for circuits and DDH, there exists an efficient RAM delegation scheme* $\mathcal{DEL}$ *with persistent database with adaptive security and soundness.*

6 History-less Accumulators

6.1 Overview

We introduce the notion of history-less accumulators, which is a variant of positional accumulators introduced by [KLW15]. History-less accumulator is a cryptographic data structure associated with a large storage component and a "short" accumulator value. The storage component, computed using public parameters, encapsulates a database. There are two functionality aspects associated to this data structure: (a) updatability and (b) verifiability. Updatability ensures that the database, that is part of the storage component, can be updated. Suppose we have s to be a storage component of database M and let w be its associated accumulator value. Then the verifiability property ensures that given w, symbol M_i^*, location i and proof (whose size is independent of $|M|$), we can verify whether $M_i^* = M_i$ where M_i is the ith symbol of M. Associated to the above data structure is an information-theoretic property that is stated as follows: suppose we are given fake public parameters "programmed w.r.t position i". Consider the storage tree associated with accumulator value w computed using these fake parameters such that the ith symbol is M_i. If it is possible to produce a proof that correctly verifies a symbol M_i^* at location i then it is the case that $M_i^* = M_i$. This is referred to as the *Read-Enforcing* property. There is another (computational) property, termed as *Indistinguishability of Read-Setup*, that ensures that the programmed parameters in the Read-Enforcing property and the (real) public parameters are computationally indistinguishable. We additionally associate another information-theoretic property termed as *Write-Enforcing* that is stated as follows: Let w_{k-1} be the accumulator value at the $(k-1)^{th}$ step generated using fake public parameters "programmed w.r.t position i" – this will be generated differently from the parameters in the read-enforcing property. This property states that any two auxiliary informations

aux, aux' used to update the value w_{k-1} *should* always result in the same value. Again, we have *Indistinguishability of Write-Setup* that ensures that the programmed parameters in the Write-Enforcing property and the (real) public parameters are computationally indistinguishable.
First, we formally define the notion of history-less accumulators in Section 6.2. We present a construction of history-less accumulators based on decisional Diffie-Hellman (DDH) assumption. This construction follows along the footsteps, and at the same time making some crucial modifications, of the DDH-based construction of positional accumulators of Okamoto et al. [OPWW15]. It 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 extended two-to-one SPB hash, where SPB stands for somewhere perfectly binding hash. This notion is defined in Section 6.3. 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 in such a way 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. We then present a DDH-based construction of extended two-to-one SPB hashin the same section Sec. 6.3.
2. In the next step, we show how to go from extended two-to-one SPB hash to history-less accumulators. This is presented in Section 6.4. While the construction is identical to the OPWW transformation from two-to-one hash to positional accumulators (of KLW), we need to do more work in the analysis since we achieve the stronger primitive of history-less accumulators (as against positional accumulators). At the heart of our analysis is showing that the uniqueness of root property of extended two-to-one SPB hash implies the write-enforcing property of history-less accumulators.

Remark 2. The read-enforcing property and the indistinguishability of read-setup imply that a history-less accumulator is also a family of collision-resistant hash function (CRH). Assuming for contradiction that there exist an algorithm finds the collision of two databases (M, M') yielding the same accumulator value w in polynomial time. Then, we can break the indistinguishability of read-setup: If the public parameters of history-less accumulator was programmed (enforced) at the given location i, then $M_i = M'_i$ is guaranteed by the read-enforcing property. Thus, the programmed public parameters and the real public parameters are distinguishable. Similarly, a two-to-one SPB hash is also a CRH. In contrast, the (weaker) read-enforcing of positional accumulator does not imply CRH as it requires not only location i but also all the history of database M, including locations and values. Given the negative result on constructing CRH from indistinguishability obfuscation [AS15], we cannot hope for constructing history-less accumulator or two-to-one SPB hash from $\mathsf{i}\mathcal{O}$ and one-way function.

6.2 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 first informally describe the algorithms used in the scheme and later we provide the formal definitions.

A history-less accumulator is described by the PPT algorithms $\mathsf{hAcc} = \mathsf{hAcc}.\{\mathsf{Setup}, \mathsf{EnforceRead}, \mathsf{EnforceWrite}, \mathsf{PrepRead}, \mathsf{PrepWrite}, \mathsf{VerifyRead}, \mathsf{WriteStore}, \mathsf{Update}\}$ be an accumulator with message space $\mathcal{M}_\lambda$. The algorithm Setup generates the accumulator public parameters along with the initial storage value and the initial root value. It helps to think of the storage as being a hash tree and its associated accumulator value being the root and they both are initialized to $\perp$. There are two algorithms that generate "fake" public parameters, namely, EnforceRead and EnforceWrite. The algorithm EnforceRead takes as input a special index INDEX^* and produces fake public parameters along with initialized storage and root values. Later, we will put forth a requirement that any PPT adversary cannot distinguish "real" public parameters (generated by Setup) and "fake" public parameters (generated by EnforceRead). Also we put forth an information theoretic requirement that any storage generated by the "fake" public parameters is such that the accumulator value associated with the storage *determines a unique value at the location* INDEX^*.

Once the setup algorithm is executed, there are two algorithms that deal with arguing about the correctness of the storage. The first one, PrepRead takes as input a storage, an index and produces the symbol at the location at that index and an accompanying proof – we later require this proof to be "short" (in particular, independent of the size of storage). PrepWrite essentially does the same task except that it does not output the symbol at that location – that it only produces the proof (in the formal definition, we call this aux). Another procedure, VerifyRead then verifies whether the proof produced by PrepRead is valid or not. The above algorithms help to verify the correctness of storage. But how do we compute the storage? WriteStore takes as input an old storage along with a new symbol and the location where the new symbol needs to be assigned. It updates the storage appropriately and outputs the new storage. The algorithm Update describes how to "succinctly" update the accumulator by just knowing the public parameters, accumulator value, message symbol, index and auxiliary information aux (produced by WriteStore). Here, "succinctness" refers to the fact that the update time of Update is independent of the size of the storage.
We now present the formal definition of history-less accumulators below.

Setup, $\mathsf{SetupAcc}(1^\lambda, T) \to \mathsf{PP}_{\mathsf{Acc}}, w_0, store_0$: 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 $store_0$.

Read-Enforcing Setup, $\mathsf{EnforceRead}(1^\lambda, T, \text{INDEX}^*) \to \mathsf{PP}_{\mathsf{Acc}}, w_0, store_0$: 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 $store_0$.

Write-Enforcing Setup, $\mathsf{EnforceWrite}(1^\lambda, T, \text{INDEX}^*) \to \mathsf{PP}_{\mathsf{Acc}}, w_0, store_0$: 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 $store_0$.

Read-and-Prove, $\mathsf{PrepRead}(\mathsf{PP}_{\mathsf{Acc}}, store_{in}, \text{INDEX}) \to m, \pi$: 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.

Proof Generation, $\mathsf{PrepWrite}(\mathsf{PP}_{\mathsf{Acc}}, store_{in}, \mathrm{INDEX}) \to aux$: 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.

Verify Proof, $\mathsf{VerifyRead}(\mathsf{PP}_{\mathsf{Acc}}, w_{in}, m_{read}, \mathrm{INDEX}, \pi) \to \{True, False\}$: 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 $False$.

Write to Storage, $\mathsf{WriteStore}(\mathsf{PP}_{\mathsf{Acc}}, store_{in}, \mathrm{INDEX}, m) \to store_{out}$: 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 $store_{out}$.

Update Acc., $\mathsf{Update}(\mathsf{PP}_{\mathsf{Acc}}, w_{in}, m_{write}, \mathrm{INDEX}, aux) \to w_{out}$ **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 $Reject$.

Remark 3. A reader familiar with the work of Koppula et al. [KLW15] will notice that there is a crucial difference between the above notion and the notion of positional accumulators introduced by KLW: in KLW, EnforceRead *takes as input the entire history of computation $m_1, m_2, \ldots$ and the information-theoretic security guarantee is that if the "fake" public parameters is executed on this particular history $m_1, m_2, \ldots$ then the resulting root uniquely determines the value at a particular special index. But in our setting,* EnforceRead *only takes as input the special index and nothing more. The same difference between the two notions is also present for the case of* EnforceWrite.

Correctness. Consider the following process: Let $(m_1, \mathrm{INDEX}_1), \ldots, (m_k, \mathrm{INDEX}_k)$ be a sequence of symbols $m_1, \ldots, m_k$ and indices $\mathrm{INDEX}_1, \ldots, \mathrm{INDEX}_k$ each between 0 and $T-1$.

- Execute $\mathsf{PP}_{\mathsf{Acc}}, w_0, store_0 \leftarrow \mathsf{SetupAcc}(1^\lambda, T)$.
- For j from 1 to k, define $store_j$ iteratively as $store_j := \mathsf{WriteStore}(\mathsf{PP}_{\mathsf{Acc}}, store_{j-1}, \mathrm{INDEX}_j, m_j)$.
- Define aux_j and w_j iteratively as $aux_j := \mathsf{PrepWrite}(\mathsf{PP}_{\mathsf{Acc}}, store_{j-1}, \mathrm{INDEX}_j)$ and $w_j := \mathsf{Update}(\mathsf{PP}_{\mathsf{Acc}}, w_{j-1}, m_j, \mathrm{INDEX}_j, aux_j)$.

We require that the following correctness property holds:

1. For every INDEX between 0 and $T-1$, $\mathsf{PrepRead}(\mathsf{PP}_{\mathsf{Acc}}, store_k, \mathrm{INDEX})$ returns m_i, π, where i is the largest value in $[k]$ such that $\mathrm{INDEX}_i = \mathrm{INDEX}$. If no such value exists, then $m_i = \epsilon$.
2. For any INDEX, let $(m, \pi) \leftarrow \mathsf{PrepRead}(\mathsf{PP}_{\mathsf{Acc}}, store_k, \mathrm{INDEX})$. Then $\mathsf{VerifyRead}(\mathsf{PP}_{\mathsf{Acc}}, w_k, m, \mathrm{INDEX}, \pi) = True$.

6.2.1 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 of the positional accumulators scheme to the history-less setting.

Indistinguishability of Read-Setup. The following security property ensures that the public parameters produced by $\mathsf{SetupAcc}$ is computationally indistinguishable from the public parameters produced by $\mathsf{EnforceRead}$.

Definition 22 (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* $\mathrm{INDEX}^* \in \{0, \dots, S-1\}$.
- *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* $(\mathsf{PP}_{\mathsf{hAcc}}, w_0, store_0) \leftarrow \mathsf{hAcc.EnforceRead}(1^\lambda, S, \mathrm{INDEX}^*)$.
- *$\mathcal{A}$ sends a bit b'.*

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

Indistinguishability of Write-Setup. On the same lines as the above definition, we present the following security notion which ensures that any PPT adversary cannot distinguish the parameters output by $\mathsf{SetupAcc}$ and the parameters output by $\mathsf{EnforceWrite}$.

Definition 23 (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* ***Exp-Setup-Write***$(1^\lambda, \mathsf{hAcc}, \mathcal{A})$ *at most is negligible in λ, where*
Exp-Setup-Write *is defined as follows.*

Exp-Setup-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* $\mathrm{INDEX}^* \in \{0, \dots, S-1\}$.
- *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 $b = b'$.

Read-Enforcing. Suppose the public parameters $\mathsf{PP}_{\mathsf{hAcc}}$ is produced by $\mathsf{EnforceRead}(1^\lambda, S, \mathrm{INDEX}^*)$. Compute a storage tree w.r.t $\mathsf{PP}_{\mathsf{hAcc}}$ on a sequence of symbol-location pairs $(m_1, \mathrm{INDEX}_1), (m_2, \mathrm{INDEX}_2), \dots$. In the end, let w_k be the root. Suppose (m_i, INDEX_i) be the last pair in the sequence such that $\mathrm{INDEX}_i = \mathrm{INDEX}^*$. Then, any "valid" proof authenticating that a symbol m residing at INDEX in the storage tree with root w_k implies that $m = m_i$. If no such last pair exists then the symbol m should be $\emptyset$.

Definition 24 (Read-Enforcing)**.** *Consider any $\lambda \in \mathbb{N}, S \in \Theta(2^\lambda)$, $m_1, \dots, m_k \in \mathcal{M}_\lambda$, any* $\mathrm{INDEX}_1, \dots, \mathrm{INDEX}_k \in \{0, \dots, S-1\}$, *and any* $\mathrm{INDEX}^* \in \{0, \dots, S-1\}$.

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* $\mathsf{VerifyRead}(\mathsf{PP_{hAcc}}, w_k, m, \mathrm{INDEX}^*, \pi) = 1$*, 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.*

Write Enforcing. Suppose the public parameters $\mathsf{PP_{hAcc}}$ is produced by $\mathsf{EnforceWrite}(1^\lambda, S, \mathrm{INDEX}^*)$. Compute a storage tree w.r.t $\mathsf{PP_{hAcc}}$ on a sequence of symbol-location pairs $(m_1, \mathrm{INDEX}_1), \ldots, (m_k, \mathrm{INDEX}_k)$. In the end, let w_k be the root of the final storage tree. Then the output of $\mathsf{Update}(\mathsf{PP_{hAcc}}, w_{k-1}, m_k, \mathrm{INDEX}_k, aux)$, for any auxiliary information "aux", is either w_k or $\perp$, where (m_k, INDEX_k) is the final pair in the sequence.

Definition 25 (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 $(\mathsf{PP_{hAcc}}, w_0, store_0) \leftarrow \mathsf{hAcc.EnforceWrite}(1^\lambda, S, \mathrm{INDEX}_k)$*.*

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

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

Then, hAcc *is said to be* write-enforcing *if* $\mathsf{Update}(\mathsf{PP_{hAcc}}, w_{k-1}, m_k, \mathrm{INDEX}_k, aux) = w_{\mathsf{out}} \neq$ `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* `reject` *does not exist.*

6.3 Extended Two-to-One SPB Hash

We define the notion of extended two-to-one SPB hash below. The hash function takes as input two blocks and outputs a value whose length is only slightly larger than the length of one block. We associate this hash function with some properties as explained later.

Definition 26 (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* $\ell(\lambda, s) = s \cdot (1 + 1/\Omega(\lambda)) + \mathrm{poly}(\lambda)$ *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,** $\left(H_{\mathsf{hk}} : \Sigma^2 \rightarrow \{0,1\}^\ell\right)$*: A deterministic poly-time algorithm that takes as input* $x = (x_0, x_1) \in \Sigma^2$ *and outputs the hash value* $H_{\mathsf{hk}}(x)$*.*

We require that the 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 27 (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 ith block $\mathbf{x}_i$, where $\mathsf{hk} \leftarrow \mathsf{Gen}(1^\lambda, 1^s, i)$.

Definition 28 (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, u', π, π' such that $u \neq u'$ and $\mathsf{Verify}(\mathsf{hk}, y, i, u, \pi) = \mathsf{Verify}(\mathsf{hk}, y, i, u', \pi') = 1$. We require that for any parameter s and any index $i \in \{0, 1\}$:*

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

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 29. *Suppose $\mathsf{hk} \leftarrow \mathsf{Gen}(1^\lambda, 1^s, i \in \{0, 1\})$. 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 $H_{\mathsf{hk}}(x_0, x^*) = H_{\mathsf{hk}}(x_0', x^*)$*

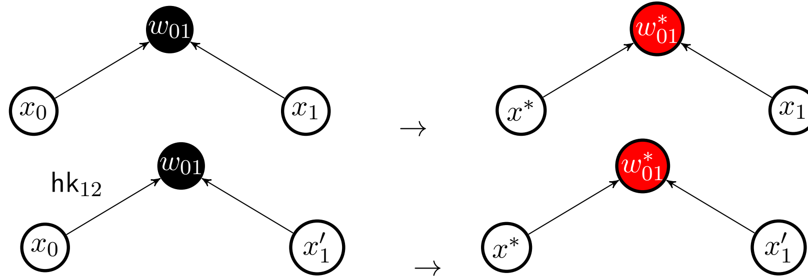

Figure 4 Let $\mathsf{hk} \leftarrow \mathsf{Gen}(1^\lambda, 1^s, i = 0)$. Let $H_{\mathsf{hk}}(x_0, x_1) = H_{hkey}(x_0, x_1') = w_{01}$. Then the uniqueness of root property states that $H_{\mathsf{hk}}(x^*, x_1) = H_{\mathsf{hk}}(x^*, x_1') = w_{01}^*$.

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 with $t = \text{poly}(\lambda)$ 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 $\mathsf{hk} = (g^{\mathbf{a}}, g^{\mathbf{b}}, g^{\mathbf{A}}, g^{\mathbf{B}})$.

$\underline{H_{\mathsf{hk}}(x_A, x_B)}$: We view $x_A \in \Sigma = \{0,1\}^s$ and $x_B \in \Sigma = \{0,1\}^s$ 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 $\mathbf{x}_A$ (resp., $\mathbf{x}_B$) 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 8. *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,

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

□

6.4 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.[8]

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

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

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 $w_{out} = \widetilde{\eta_0}^0$. See Figure 5 for an illustration.

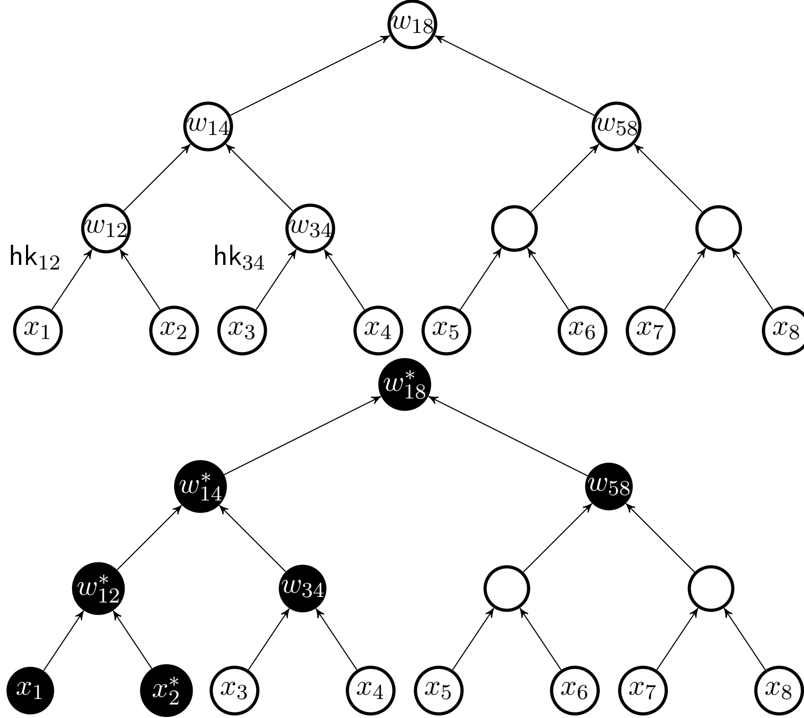

Figure 5 Consider the accumulator tree (top) initialized with $x_1, \ldots, x_8$. To generate this tree, a hash key hk_{12} is generated and then $\mathsf{hk}_{12}(x_1, x_2)$ is computed. This is denoted by w_{12}. Similarly, a fresh instantiation of hk_{34} is sampled and then we compute $\mathsf{hk}_{34}(x_3, x_4)$ and so on. Lets say we want to change x_2 to x_2^*. We update as shown in the bottom tree. We update the root w_{12}^* and then we update the root w_{14}^* and so on. Note that the updating process takes time only logarithmic in the size of the tree (or proportional to the depth of the tree).

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

Proof. We first describe the proof idea and then we provide the formal details.

Proof Idea. Consider a sequence of symbol-index pairs $(m_1, \text{INDEX}_1), \ldots, (m_k, \text{INDEX}_k)$. Let $\mathsf{PP}_{\mathsf{hAcc}}$ be an enforcing public parameter that is "programmed with respect to INDEX_k". Initialize the storage tree on the above sequence of symbol-index pairs. Let w_{k-1} be the accumulator value resulting by initializing the storage tree on the sequence $(m_1, \text{INDEX}_1), \ldots, (m_{k-1}, \text{INDEX}_{k-1})$. Let aux be the "valid" auxiliary information such that $\mathsf{Update}(\mathsf{PP}_{\mathsf{hAcc}}, w_{k-1}, m_k, \text{INDEX}_k, aux) = w_k$. Suppose let aux' be such that $\mathsf{Update}(\mathsf{PP}_{\mathsf{hAcc}}, w_{k-1}, m_k, \text{INDEX}_k, aux') = w'$. If $w' \neq \perp$ then it has to be the case that $w = w'$.

We first observe that the root value in both aux and aux' should be w_{k-1}. From here, on we argue that all the nodes along the path (top-down) from w_{k-1} to the leaf corresponding to INDEX_k should be the same in both aux and aux'. Here, we crucially use the somewhere perfect binding property of the SPB hash. The second observation is that once we update m_k at INDEX_k then we argue that every node along with path (bottom-up) from INDEX_k to the root has to be same in both aux and aux'. Here, we crucially use the uniqueness of root property of the SPB hash. Once we show this, we have shown that the root of aux and aux' is the same which in turn means that $w = w'$.

Formal Details. 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 $(\mathsf{PP}_{\mathsf{hAcc}}, w_0, store_0) \leftarrow$ $\mathsf{hAcc.EnforceWrite}(1^\lambda, S, \text{INDEX}_k)$. For all $j \in [k]$, we define $store_j$ iteratively as $store_j := \mathsf{WriteStore}(\mathsf{PP}_{\mathsf{hAcc}}, store_{j-1}, \text{INDEX}_j, m_j)$. We similarly define aux_j and w_j iteratively as $aux_j := \mathsf{PrepWrite}(\mathsf{PP}_{\mathsf{hAcc}}, store_{j-1}, \text{INDEX}_j)$ and $w_j := \mathsf{Update}(\mathsf{PP}_{\mathsf{hAcc}}, w_{j-1}, m_j, \text{INDEX}_j, aux_j)$. Denote the value $w_{out} = w_k$. Denote aux_k by aux. And let aux' be such that $\mathsf{Update}(\mathsf{PP}_{\mathsf{hAcc}}, w_{k-1}, m_k, \text{INDEX}_k, aux') = w'$. 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:

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

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

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 4. *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 5. *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 4, 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.

□

7 Instantiation: Adaptive Delegation for RAM with Persistent Database

Given the adaptive RAM delegation scheme defined in Section 5, we then present a construction of an adaptive RAM delegation scheme. Formally, we show:

Theorem 10. *Assuming the existence of iO for circuits and the existence of historyless accumulators, we show that there exists adaptive delegation scheme for RAMs with persistent database.*

If we instantiate history-less accumulators using the DDH-based construction in Section 6, we get the following corollary.

Corollary 1. *Assuming the existence of iO for circuits and DDH, we show that there exists an adaptive delegation scheme for RAMs with persistent database.*

We begin by presenting the roadmap of the construction used to prove Theorem 10.

7.1 Roadmap

Building towards this construction, we first show a primitive, called computation-trace indistinguishability obfuscation ($\mathsf{Ci}\mathcal{O}$) with persistent database [CCC+16]. Then we show, in Section 7.2, how to adapt the construction of selective $\mathsf{Ci}\mathcal{O}$ of CCC+ to our setting. A crucial change we make to the construction of selective $\mathsf{Ci}\mathcal{O}$ is that we replace the tool of positional accumulators with *history-less accumulators*; a notion we introduced in Section 6. We then argue that the construction of selective $\mathsf{Ci}\mathcal{O}$ has niceness property. To do this, we first dive into the details for 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 4. 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+ construction is adaptively secure.

In the next step, we consider the concept of selective garbled RAM in the persistent database setting. We then show, in Section 7.3.2, how to build selective garbled RAM from adaptive $\mathsf{Ci}\mathcal{O}$, and check that its security proof satisfies niceness property. In the final step, we use the known generic transformations to obtain selective RAM delegation from adaptive garbled RAM in the persistent setting, and then apply the same procedure to obtain adaptive RAM delegation. This is demonstrated in Section 7.4.

Extension to adaptive PRAM delegation. With the strategy of CCC+ [CCC+16], we can realize the adaptive PRAM delegation with persistent database. We provide a brief overview below as to how to make this extension work. At first, we convert our selective $\mathsf{Ci}\mathcal{O}$ for RAM and branch-and-combine technique of [CCC+16] into the selective $\mathsf{Ci}\mathcal{O}$ for PRAM with niceness property. Then, we build the selective PRAM garbling from selective $\mathsf{Ci}\mathcal{O}$ for PRAM. Finally, the selective PRAM delegation is achieved with full privacy and soundness by applying the generic transformation, which also satisfies niceness property. Plugging the security-lifting, we obtain the adaptive PRAM delegation, where 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.

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

Computation-trace indistinguishable obfuscation $\mathsf{Ci}\mathcal{O}$ [CCC+16] is a weaker security notion of delegation without database and output privacy and soundness (see Section 7.2.1 for its formal definition). Similar to delegation, we consider multiple-program RAM computation with an initial database, where a sequence of programs work on the database content processed and left over by the previous program. However, the security of $\mathsf{Ci}\mathcal{O}$ is to require indistinguishability between two sequences of obfuscated programs with the same computation trace (which is defined in Section 3.4). The intuition of using $\mathsf{Ci}\mathcal{O}$ is to force the evaluator to evaluate obfuscated programs as intended to produce the intended computation trace. Also, the sequence of programs is required to be executed in the intended order.

In this section, we present the construction of selective $\mathsf{Ci}\mathcal{O}$ [CCC+16] which is based on indistinguishable obfuscation scheme $\mathsf{i}\mathcal{O}$, puncturable pseudorandom function scheme PRF, iterator scheme Itr, splittable signature scheme Spl, and *stronger* accumulator scheme hAcc. Then, we will show that the selective proof of the constructed $\mathsf{Ci}\mathcal{O}$ satisfies the "niceness" property. With niceness, we can apply the abstract framework to obtain adaptive security. Here we remark that positional accumulator applied by [KLW15, CCC+16, CH16] cannot preserve niceness in the selective proof, so we choose history-less accumulator instead.

7.2.1 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. $\mathsf{Ci}\mathcal{O}$ [CCC+16] forces the evaluator to evaluate obfuscated programs as intended to produce the intended computation trace (see Section 3.4). The sequence of programs is required to be executed in the intended order.

Definition 30 ($\mathsf{Ci}\mathcal{O}$ 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 projection function.*

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 $|\Pr[b = b'] - \frac{1}{2}| \leq \mathsf{negl}(\lambda)$ 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 initial $\widetilde{\mathsf{mem}}^{1,0}$ and previous $\widetilde{F}_1, ..., \widetilde{F}_{i-1}$, $\mathcal{A}$ adaptively chooses a new pair of (F^0_i, F^1_i) to $\mathcal{C}$. If F^0_i, F^1_i are computation-trace identical, $\mathcal{C}$ returns $\widetilde{F}_i = \mathsf{Obf}(\mathsf{sk}, i, F^b_i)$ to $\mathcal{A}$. If not, $\mathcal{C}$ aborts.*
- *$\mathcal{A}$ outputs b'. $\mathcal{A}$ is said to win if $b' = b$.*

We remark that an unrestricted adaptive adversary *can adaptively choose RAM programs P^i depending on the program compilations 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. $\mathsf{DBCompile}$ *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 $\tilde{O}(t^*_{\mathsf{sid}})$.*

7.2.2 Construction of $\mathsf{Ci}\mathcal{O}$

At a very high level, CCC+ construction of $\mathsf{Ci}\mathcal{O}$ [CCC+16] transforms a RAM program into an encapsulated next step function which authenticates the output of a time step and verifies the integrity of the input in every time step. This encapsulated next step function is then obfuscated. To successfully carry out the authentication phase, the output state (which is small in size) will be signed by using a signature scheme. In order to control the size of the state, the construction

additionally relies on an accumulator scheme [KLW15]. Accumulator is a data structure that allows for producing "short" state that binds a large memory.

Let a stateful algorithm F_{sid} denote a single program with session identity sid in a multiple-program RAM computation, 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 $\mathsf{DBCompile}(1^\lambda, \mathsf{mem}^{0,0}) \to (\widetilde{\mathsf{mem}}^{1,0}, \widetilde{\mathsf{st}}^{1,0}, \mathsf{sk})$**.** It computes the following parameters:

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

where K_T is the termination key. Based on $\mathsf{mem}^{0,0}$, this algorithm computes the initial configuration for computation 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 $\mathsf{Obf}(1^\lambda, \mathsf{sk}, \mathsf{sid}, F_{\mathsf{sid}}) \to \widetilde{F}_{\mathsf{sid}}$**.** First, it generates following session parameters for the given 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 $\mathsf{sid} \in [l]$, $1 \le \mathsf{sid} \le l$. This algorithm Obf then computes an obfuscation of the program $\widehat{F}_{\mathsf{sid}}$. That is, $\widetilde{F}_{\mathsf{sid}} \leftarrow \mathsf{i}\mathcal{O}.\mathsf{Gen}(\widehat{F}_{\mathsf{sid}})$, and it outputs $\widetilde{F}_{\mathsf{sid}}$.

Algorithm 1: $\widehat{F}_{\mathsf{sid}}$ in 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 $a^{\mathsf{in}}_{\mathtt{A}\leftarrow\mathtt{M}} = (\bot, \bot)$;
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 $r_A = \mathsf{PRF}(K_A, (\mathsf{sid}, t-1))$;
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 $\mathsf{Eval}(\widetilde{\mathsf{mem}}^{\mathsf{sid},0}, \widetilde{\mathsf{st}}^{\mathsf{sid},0}, \widetilde{F}_{\mathsf{sid}}) \rightarrow (\widetilde{\mathsf{mem}}^{\mathsf{sid}+1,0}, \widetilde{\mathsf{st}}^{\mathsf{sid}+1,0})$**.** 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 $\widetilde{a}^{\mathsf{sid},0}_{\mathtt{A}\leftarrow\mathtt{M}} = \bot$. 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-lifting.

7.2.3 Checking Niceness for Security Proof of $\mathsf{Ci}\mathcal{O}$

Lemma 6. *The security proof of the above construction of selective* $\mathsf{Ci}\mathcal{O}$ *satisfies the niceness property.*

Proof. We follow the abstract proof (see Section 4) and Theorem 6. Note that we already have a *selective proof* to show our $\mathsf{Ci}\mathcal{O}$ construction is selectively secure since we can directly use that proof in [CCC+16, KLW15] with the stronger history-less accumulator. To apply Theorem 6, we firstly model the selective proof as generalized cryptographic games, reductions, and specific functions G. Secondly, given the above, it suffices for checking the selective proof is a "nice" proof which satisfies properties 1, 2 and 3 listed in Definition 18.

Even though there is a long sequence of hybrids, cryptographic games (to distinguish adjacent hybrids), and reductions, we use a systematic way to checking that the selective proof indeed satisfies niceness. In general, the ith 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 an intractability assumption (CH'_i, τ'_i).

To complete the above well-defined games and reductions with specific function G_i, we observe that in general there are two cases of hybrid H_i in this selective 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 $\mathsf{EnforceWrite}$. On the one hand, $\mathsf{PP}_{\mathsf{hAcc}}$ is necessary while compiling database $\widetilde{\mathsf{mem}}^{0,0}$. On the other hand, however, computing the enforced $\mathsf{PP}_{\mathsf{hAcc}}$ depends on global information. It is because $\mathsf{EnforceRead}$ (or $\mathsf{EnforceWrite}$) needs INDEX^* which is specified by $(F_1, \dots F_{\mathsf{sid}})$ on mem at time t, where sid and t are further specified by H_i. In other words, INDEX^* is a global information that depends on $(\mathsf{mem}, F_1, \dots F_{\mathsf{sid}})$. Thus, we need a proper function G_i to provide INDEX^*.

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

Remark 4. Let one of neighboring hybrids (H_i, H_{i+1}) be in the enforcing mode (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. Specifically, in original selective $\mathsf{Ci}\mathcal{O}$, the enforcing accumulator can be either positional (that needs complete memory-accessing history which may be long) or history-less (that needs only one index). To complete our proof, we require only a short guess, so the history-less accumulator is necessary. We can further apply the abstract framework to achieve adaptive security.

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 Definition 18 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 via hybrids with logarithmic-length $\mathcal{G}$ function.** This property holds, since all hybrids H_i have the same interface as the real experiments when interacting with the adversary. Also note that function G_i has only two cases for all i, either null or $\mathrm{INDEX}^*_i(\alpha)$, with

logarithmic-length output. Specifically, the output length of G_i is exactly $\log S$ which is in $O(\log \lambda)$ as long as the space bound $S = O(\text{poly } \lambda)$.

2. **Indistinguishability of neighboring hybrids via nice reduction.** For every neighboring hybrids (H_i, G_i) and (H_{i+1}, G_{i+1}), we consider 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. 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 16 from the corresponding guessing game to some intractability assumptions, such as $\mathsf{i}\mathcal{O}$ and puncturable PRF.

3. **$\mathcal{G}_i || \mathcal{G}_{i+1}$-hiding.** We claim the guessing game $(CH_i, \bar{G}_i, 1/2)$ is $\bar{G}_i$ hiding by considering H_i and H_{i+1} separately. If G_i and G_{i+1} are both hidden from hybrid experiments H_i and H_{i+1} respectively, then $\bar{G}_i$ is also hidden. Thus, this property requires that $(H_i, G_i(\alpha))$ and $(H_i, G_i(\alpha'))$ are indistinguishable for every α and α', for every adversary, where $G_i(\alpha)$ or $G_i(\alpha')$ is hard-coded by the challenger (Definition 17). We claim (H_i, G_i) is G_i-hiding for all i.

   Proof. For any G_i, there are two cases, either null or $\text{INDEX}^*_i(\alpha)$. In Case 1, $(H_i, G_i(\alpha) = \mathsf{null})$ and $(H_i, G_i(\alpha') = \mathsf{null})$ are identical for any adversary since H_i never uses null. 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 $(H_i, G_i(\alpha))$ and $(H_i, G_i(\alpha'))$ are indistinguishable to any PPT G_i-selective adversary by the read/write setup indistinguishability of history-less accumulator (Definitions 22 and 23) in which we first switch the enforce-mode $(H_i, G_i(\alpha))$ to normal (H_i, null) and then do again back to the enforce-mode $(H_i, G_i(\alpha'))$.

   □

More detailed checks are described in Appendix A. In particular we will show that some experiments have guessing (i.e., $\mathbf{Hyb}_{0,2,j,i,0}, ..., \mathbf{Hyb}_{0,2,j,i,13}$ in [CCC+16]).

□

Finally, it follows by Theorem 6 that real experiments $\mathsf{Exp\text{-}IND\text{-}Ci}\mathcal{O}\{b = 0\}$ and $\mathsf{Exp\text{-}IND\text{-}Ci}\mathcal{O}\{b = 1\}$ of the $\mathsf{Ci}\mathcal{O}$ construction are indistinguishable against adaptive adversaries. We state the following corollary.

Corollary 2. *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 the construction of* $\mathsf{Ci}\mathcal{O}$ *is adaptively secure.*

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

A garbling scheme for RAM computation with persistent database ($\mathcal{GRAM}$) is conceptually a type of a delegation scheme (Definition 19) and is equipped with program encoding and a database encoding algorithms. The security guarantee is however different from delegation – it has no output privacy and no soundness. More specifically, the adversary learns the output in clear (see Section 7.3.1 for its formal definition).

To obtain adaptive $\mathcal{GRAM}$, we recall the selective construction of $\mathcal{GRAM}$ from [CCC+16], and then we then verify whether the security proof fits with our abstraction framework proposed earlier (Section 4). This selective $\mathcal{GRAM}$ is constructed with selective $\mathsf{Ci}\mathcal{O}$. To adopt our framework of generalizing cryptographic games, we substitute selective $\mathsf{Ci}\mathcal{O}$ with the stronger adaptive $\mathsf{Ci}\mathcal{O}$

presented in the previous section. We then follow along the selective security proof of [CCC+16] we syntactically check whether it satisfies "niceness" in our framework of generalized cryptographic games. As the proof has "nice", it follows by our abstraction that this construction of $\mathcal{GRAM}$ is secure against adaptive adversary, and then this adaptive $\mathcal{GRAM}$ scheme can further be applied to construct adaptive delegation.

The checking of niceness property is involved. Our adaptive construction is almost identical to the selective construction in [CCC+16], where the only difference is that adaptive $\mathcal{GRAM}$ is built on adaptive $\mathsf{Ci}\mathcal{O}$. For simplicity, we usually omit adaptive when denoting $\mathcal{GRAM}$ or $\mathsf{Ci}\mathcal{O}$ in this section.

7.3.1 Definition of $\mathcal{GRAM}$

For any RAM program P that computes on database mem and outputs a 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 the program P. We stress that database is *persistent* if mem' can be taken as input to the succeeding program P'. For simplicity, 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.

In general, a garbling scheme to garble RAM programs and persistent database consists of four algorithms: the first one is to generate a secret key and garble initial database, the second one is to garble a RAM program that could read and write that database and then return an output, and the last one is to evaluate those garbled database and programs.

Definition 31 ($\mathcal{GRAM}$ 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.*

- $\mathcal{GRAM}.\mathsf{DBGarble}(1^\lambda, \mathsf{mem}^1, S) \to (\mathsf{sk}, \widetilde{\mathsf{mem}}^1)$*: The database garbling algorithm* $\mathsf{DBGarble}$ *is a randomized algorithm which takes as input the security parameter 1^λ, the secret key sk, database mem^1, and a space bound S. It outputs a secret key sk and a garbled database $\widetilde{\mathsf{mem}}^1$.*
- $\mathcal{GRAM}.\mathsf{PGarble}(1^\lambda, \mathsf{sk}, \mathsf{sid}, P^{\mathsf{sid}}) \to \widetilde{P}^{\mathsf{sid}}$*: 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 current database $\widetilde{\mathsf{mem}}^{\mathsf{sid}}$. It outputs $(y^{\mathsf{sid}}, \widetilde{\mathsf{mem}}^{\mathsf{sid}+1})$ or $\perp$.*

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

$$\begin{aligned}\Pr[(\widetilde{\mathsf{mem}}^1, \mathsf{sk}) \leftarrow \mathcal{GRAM}.\mathsf{DBGarble}(1^\lambda, \mathsf{mem}^1, S); \widetilde{P}^{\mathsf{sid}} \leftarrow \mathcal{GRAM}.\mathsf{PGarble}(1^\lambda,\\ \mathsf{sk}, \mathsf{sid}, P^{\mathsf{sid}}); (y^{\mathsf{sid}}, \widetilde{\mathsf{mem}}^{\mathsf{sid}+1}) \leftarrow \mathcal{GRAM}.\mathsf{Eval}(1^\lambda, T, S, \widetilde{P}^{\mathsf{sid}}, \widetilde{\mathsf{mem}}^{\mathsf{sid}});\\ (z^{\mathsf{sid}}, \mathsf{mem}^{\mathsf{sid}+1}) \leftarrow P^{\mathsf{sid}}(\mathsf{mem}^{\mathsf{sid}}) \ : \ y^{\mathsf{sid}} = z^{\mathsf{sid}} \ \forall \mathsf{sid}, 1 \le \mathsf{sid} \le l] = 1.\end{aligned}$$

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^1 *to challenger $\mathcal{C}$.*
3. *If $b = 0$, challenger $\mathcal{C}$ computes* $(\widetilde{\mathsf{mem}}^1, \mathsf{sk}) \leftarrow \mathsf{DBGarble}(1^\lambda, \mathsf{mem}^1, S)$*. Otherwise, $\mathcal{C}$ simulates* $(\widetilde{\mathsf{mem}}^1, \mathsf{sk}) \leftarrow \mathcal{S}(1^\lambda, |\mathsf{mem}^1|)$*, where* $|\mathsf{mem}^1|$ *is the length of* mem^1*. $\mathcal{C}$ 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 remark that an unrestricted adaptive adversary *can adaptively choose RAM programs P^i depending on the garbled programs and garbled database 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* $\tilde{O}(|\mathsf{mem}^1|)$ *and* $\tilde{O}(\mathrm{poly}(|P^{\mathsf{sid}}|))$*, and efficient* Eval *runs in time* $\tilde{O}(t^*)$.

7.3.2 Construction of $\mathcal{GRAM}$ with Persistent Database

The construction of $\mathcal{GRAM}$ with persistent database is based on puncturable PRF, $\mathcal{PKE}$ and $\mathsf{Ci}\mathcal{O}$ with persistent database. To garble both database and programs, the idea of [CCC+16] is natural.

At a high level, the goal of the garbling procedure is to generate obfuscated program $\tilde{P}$ and the encoded input $\tilde{x}$ such that the output of $P(x)$ can be recovered from these encodings and at the same time both P and x should be hidden. To protect the privacy of P, we must restrict $\tilde{P}$ to evaluate *only* on input x (whose encoding is provided), and not for instance, $P(x')$ where the encoding of x' is not provided. We need some *authentication mechanism* to *authenticate* the whole evaluation of P on x so that the adversary follows along this path of computation. Moreover, the evaluation of P on x produces a long computation trace in addition to y. We need some *hiding mechanism* to hide the evaluation process.

To authenticate the whole evaluation, $\mathsf{Ci}\mathcal{O}$ will do. To hide information through the evaluation process, a natural approach is to use encryption schemes to hide the CPU states and the memory content. Namely, $\tilde{P}$ always outputs encrypted CPU states and memory, and on (authenticated) input of ciphtertexts, performs decryption before the actual computation. Note, however, that the memory access pattern cannot be encrypted (otherwise the server cannot evaluate), which may also leak information. A natural approach is to use oblivious RAM (ORAM) to hide the access pattern. Namely, we use ORAM compiler to compile the program (and add an "encryption layer"). Thus, the garbling algorithm of $\mathcal{GRAM}$ is to compile programs and database with hiding and authenticating property as follows.

1. (Hiding.) First, compile RAM programs and database with oblivious RAM (ORAM) to hide the access pattern. Second, compile RAM programs and database with public-key encryption ($\mathcal{PKE}$) to hide its CPU states and memory contents. Note that both ORAM and $\mathcal{PKE}$ need randomness, which is generated with puncturable PRF carefully.
2. (Authenticating.) Use $\mathsf{Ci}\mathcal{O}$ to obfuscate the compiled programs and database generated by the hiding step.

To evaluate garbled programs, the evaluator performs $\mathsf{Ci}\mathcal{O}$ evaluation directly on encoded programs and database. The algorithms of database and program garbling ($\mathcal{GRAM}.\mathsf{DBGarble}$ and $\mathcal{GRAM}.\mathsf{PGarble}$) are designed as follows.

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

1. It randomly chooses puncturable PRF keys K_E, K_N and $\mathsf{Ci}\mathcal{O}$ key $\mathsf{sk}_{\mathsf{Ci}\mathcal{O}}$. It outputs secret key $\mathsf{sk} = (K_E, K_N, \mathsf{sk}_{\mathsf{Ci}\mathcal{O}})$.
2. (ORAM) Compile database mem^1 with ORAM algorithm into oblivious database mem^1_o, where the randomness used by ORAM is sampled uniformly.
3. ($\mathcal{PKE}$) Encrypt oblivious database mem^1_o into mem^1_e using $\mathcal{PKE}$ scheme and pk created from K_E.
4. ($\mathsf{Ci}\mathcal{O}$) Obfuscate mem^1_e using $\mathsf{Ci}\mathcal{O}.\mathsf{DBCompile}(1^\lambda, \mathsf{mem}^1_e) \to (\widetilde{\mathsf{mem}}^{1,0}, \widetilde{\mathsf{st}}^{1,0}, \mathsf{sk}_{\mathsf{Ci}\mathcal{O}})$.
5. Output 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 $P_{\mathsf{sid},o}[K_N]$, which computes pseudo-randomnesses from K_N and then passes them to ORAM.
3. ($\mathcal{PKE}$) Compile $P_{\mathsf{sid},o}[K_N]$ into $P_{\mathsf{sid},e}[K_N, K_E]$ that decrypts input and encrypts output at each step with $\mathcal{PKE}$ and keys generated from K_E (Algorithm 2), 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 $P_{\mathsf{sid},e}[K_N, K_E]$ into $\widetilde{P}_{\mathsf{sid}}$ using $\mathsf{Ci}\mathcal{O}.\mathsf{Obf}(1^\lambda, \mathsf{sk}_{\mathsf{Ci}\mathcal{O}}, P_{\mathsf{sid},e}[K_N, K_E]) \to \widetilde{P}_{\mathsf{sid}}$.
5. Output garbled program $\widetilde{P}_{\mathsf{sid}}$.

7.3.3 Checking Niceness for Security Proof of $\mathcal{GRAM}$

As the above construction is almost identical to [CCC+16] with stronger $\mathsf{Ci}\mathcal{O}$ substituted, its correctness and efficiency is straightforward. Next, we recall its security proof at a high level, and then argue this proof is nice and can be lifted by our framework.

Algorithm 2: $P_{\mathsf{sid},e}[K_N, K_E]$, 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 $\mathsf{st}^{\mathsf{in}} = \mathtt{init}, I^{\mathsf{in}} = \bot, b^{\mathsf{in}} = \bot$;
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 $P_{\mathsf{sid},o}[K_N]$ with state $\mathsf{st}^{\mathsf{in}}$, accessing index I^{in}, reading bit b^{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 $\widetilde{\mathsf{st}}^{\mathsf{out}} = (\boxed{\mathsf{st}}^{\mathsf{out}}, t+1)$, $\widetilde{a}^{\mathsf{out}} = (I^{\mathsf{out}}, (\boxed{b}^{\mathsf{out}}, (\mathsf{sid}, t)))$;

The Intuition of Selective Security Proof. In the security proof, the approach taken by [CCC+16] is to establish indistinguishability of hybrids *backwards in time.* Namely, they consider intermediate hybrids $\mathbf{Hyb}_i$ where the computations of the first i time steps are real, and those of the remaining time steps are simulated (appropriately). A key idea here (following [KLW15]) is to encrypt each message (a CPU state or a memory cell) using a different key, and generate these keys (as well as encryption randomness) using puncturable PRF (PPRF), which allows us to use a standard puncturing argument in the security proof (extended to work with $\mathsf{Ci}\mathcal{O}$ instead of $\mathsf{i}\mathcal{O}$) to move to a hybrid where semantic security holds for a specific message so that we can "erase" it.

Yet, since the computation trace of the first $(i-1)$ time steps is real, it contains enough information to carry out the rest of the (deterministic) computation. In particular, the access pattern at time step i is determined by the first $(i-1)$ time steps, that means we cannot replace it by a simulated access pattern. However, it is unlikely that we can use the security of ORAM in a black-box way, since ORAM security only holds when the adversary does not learn any content of the computation. Indeed, we can only use puncturing argument to argue that semantic security holds *locally* for some encryption at a specific computation step of P.

To solve this problem, [CCC+16] developed a puncturing ORAM technique to reason about the simulation specifically for CP-ORAM [CP13]. At a very high level, to move from $\mathbf{Hyb}_i$ to $\mathbf{Hyb}_{i-1}$ (*i.e.*, erase the computation at the i-th time step), we "puncture" ORAM at time step i (*i.e.*, the i-th memory access), which enables us to replace the access pattern by a simulated one at this time step. We can then move (from $\mathbf{Hyb}_i$) to $\mathbf{Hyb}_{i-1}$ by erasing the memory content and computation, and undoing the "puncturing." Roughly speaking, puncturing CP-ORAM at i-th time step can be viewed as injecting a piece of "*puncturing code*" in the ORAM access procedure to erase the information rand_i about access pattern at time step i information-theoretically. For instance, rand_i is generated at the latest time step t' that accesses the same memory location as time step i. The puncturing code simply removes the generation of rand_i at time step t'. However, the last access time t' can be much smaller than i, so the puncturing may cause global changes in the computation. Thus, moving to the punctured mode requires carefully defining a sequence of hybrids that modifies the computation step by step. They do so by further introducing an auxiliary "*partially puncturing*" code that punctures rand_i from certain threshold time step $j \geq t'$. The sequence of hybrids which move to the punctured code corresponds to moving the threshold $j \leq i$ backwards from i to t'.

Specifically, the puncturing code is informally defined as follows. Let $P_{\mathbf{Hyb}_i}$ be the program encoded in $\mathbf{Hyb}_i$. Let ℓ be the memory block accessed at the i-th time step of $P(x)$, $p = pos[\ell]$ be the position map value at time step i, and t' be the last access time of block ℓ before time step i. Note that p is exactly rand_i that is generated at time step t'. The goal is to information-theoretically erase the value p from time t'. The punctured hybrid $\mathbf{Hyb}_i^{\text{punct}}$ is defined with a hybrid encoding $\mathsf{Ci}\mathcal{O}(P_{\mathbf{Hyb}_i^{\text{punct}}}, \mathsf{mem}_{\mathsf{hide}})$ where $P_{\mathbf{Hyb}_i^{\text{punct}}}$ is $P_{\mathbf{Hyb}_i}$ plus the following puncturing code:

> **Puncturing Code**: At time step $t = t'$, do not generate the value p, and instead of putting back the (encrypted) fetched block ℓ to the root of the ORAM tree, an encryption of a dummy block is put back. Moreover, the position map value p is not updated. Additionally, the value p is hardwired, and is used to emulate the memory access at the i-th time step.

In other words, block ℓ is deleted at time step t' and p remains to store the old value (used to fetch the block at time step t'). So, in $\mathbf{Hyb}_i^{\text{punct}}$, the value p is information-theoretically hidden in the computation trace of the first $(i-1)$ time steps and is only used to determine the access pattern at time step i. We can then use puncturing arguments for PPRF to replace p by one generated by the simulation (as opposed to real) PPRF key. Similarly, the partially puncturing code is defined

as follows to move from $P_{\mathbf{Hyb}_i}$ to $P_{\mathbf{Hyb}_i^{\text{punct}}}$.

> **Partially Puncturing Code**$[j]$: At any time step $t > j$, if the input CPU state or memory contains the block ℓ, then replace it by a dummy block before performing the computation.

We observe that in both puncturing and partially puncturing code, only the index ℓ of the block (to be accessed at time step i) is needed to perform such removal.

Checking Niceness of the Selective Proof We now show that the above proof of selective security can be generalized by our framework of generalized cryptographic game, and consists of "nice" neighboring hybrids with proper global function G.

In the proof, recall that intermediate hybrid $\mathbf{Hyb}_i$ has its computation which is real in the first i time steps and simulated (appropriately) in the remaining time steps. To switch time step i from real to simulated, there are two major steps: the first is to use puncturable PRF and encryption scheme to simulate the ciphertexts of messages (including CPU state and a memory cell value); the second is to use the puncturing ORAM technique to simulate its access pattern. In the first step, all the ciphertexts of messages depend only on known input and programs in generalized hybrid experiment. According to this step, no global information is needed, and thus arguing its niceness is straightforward. In the second step, intermediate puncturing and partially puncturing hybrids $\mathbf{Hyb}_{i,j}$ are needed, where $j < i$ is the time step to insert (partially) puncturing code. However, these hybrids do not directly work under generalized hybrid experiment. Let α be a sequence of input database and program queries with persistent database, $\alpha = (\mathsf{mem}, P_1, \ldots, P_l)$. Let time step i be the u-th step in the r-th program P_r, and let time step j be the v-th step in the q-th program P_q, where $1 \leq q \leq r \leq l$. The last access time t' can be much smaller than i, and thus access time t' and i can be in different programs such that $q < r$. Therefore, garbled encoding of P_q depends on the query of P_r which is not known while generating P_q in the hybrid experiment. As a result, we need proper global information and function G_i to formulate our generalized cryptographic experiments or games.

To tackle this, we observe that, for all hybrid experiments $\mathbf{Hyb}_{i,j}$, the challenger needs only a small amount of global information: in both puncturing and partially puncturing codes, the only global information needed is the index ℓ of the block to be accessed in time step u of program P_r. In other words, to add the (partially) puncturing code, it must know (a) the value of ℓ and (b) the value of r. Thus, we can define

$$G_{i,j}(\alpha) := (r, \ell),$$

where $i = (r, u)$ is the u-th time step of the r-th program, $j = (q, v)$ is the v-th time step of the q-th program, and ℓ is the index of block to be accessed at time i. The output of $G_{i,j}$ is logarithmic-length, and $G_{i,j}$ is hiding because $\mathbf{Hyb}_{i,j}$ is indistinguishable from $\mathbf{Hyb}_i$, which has no puncturing and no G.

Lemma 7. *The security proof of the above construction of selective $\mathcal{GRAM}$ satisfies the niceness property.*

Proof. In this proof, we use a systematic way to check the selective proof which satisfies all niceness properties even though there is a long sequence of hybrids and reductions. Let the sequence of hybrids in the selective proof be $\mathsf{Exp\text{-}Real} = H_0 \approx H_1 \cdots \approx H_{\ell(\lambda)+1} = \mathsf{Exp\text{-}Sim}$. We firstly model the selective proof as generalized cryptographic games and reductions with specific functions G, which are efficiently and reversely computable functions. Secondly, we check the selective proof is a "nice" proof which satisfies all the properties listed in Definition 18.

To work with our framework of generalized cryptographic games, we need to define the function G for each hybrid game corresponding to reduction. In general, we slightly abuse the notation i as a *global timestamp* including session identity and program time step. The ith 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 tries to distinguish 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 the selective proof.

To complete the above well-defined games and reductions with specific functions G_i, we check all hybrids H_i and observe that (a) non-puncturing H_i takes as input only the prefix message to compute their every output, and (b) (partially) puncturing H_i takes additional values (r, ℓ) as input. Thus, we define the function G_i which outputs null or a pair of global program index r and location ℓ for any input $\alpha = (\mathsf{mem}, \{P_{\mathsf{sid}}\}_{\mathsf{sid}=1}^{l}, \dots)$ for all hybrid experiments H_i.

Let us briefly discuss some steps of using *puncturing ORAM* in the proof (i.e., $\mathbf{Hyb}_{2,i,0,j} \approx \mathbf{Hyb}_{2,i,0,j+1}$ in [CCC+16]). We have to carefully apply the abstraction. In the such case, with well-defined generalized games $(CH_i, G_i||G_{i+1}, 1/2)$ and reductions R_i, we check that they satisfy the properties in Definition 18, which then states they constitute a "nice" proof.

1. **Security via hybrids with logarithmic-length $\mathcal{G}$ function.** This holds as all hybrids H_i have the same interface as the real experiments when interacting with the adversary. Additionally, $G_{i,j}$ (for any (partially) puncturing hybrid $H_{i,j}$) outputs a program index r and a location ℓ with logarithmic-length.
2. **Indistinguishability of neighboring hybrids via nice reduction.** To meet the definition of the nice reduction (see Definition 16), we need to syntactically check the whole procedures in $M_1 = CH_{i,j}$ and $M_2 = (CH'_{i,j} \leftrightarrow R_{i,j})$. M_1 and M_2 perform almost the same procedures (except a few interactions between $CH'_{i,j} \leftrightarrow R_{i,j}$), so $R_{i,j}$ is a nice reduction.
3. **$\mathcal{G}_i||\mathcal{G}_{i+1}$-hiding.** 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. By the assumption applied in Property 1, any (partially) puncturing hybrid experiment $H_{i,j}$ using global information from $G_{i,j}$ is indistinguishable from its non-puncturing precedent hybrid H_i (through a sequence of hybrids) where no G is used. Thus, $G_{i,j}$ is hiding with any large enough λ.

Finally, we conclude that the selective proof is a nice proof that satisfies the three properties.

□

It follows by Theorem 6 and Lemma 7 that experiments Exp-Real and Exp-Sim of our $\mathcal{GRAM}$ construction are indistinguishable against adaptive adversaries.

Corollary 3. *Let $\mathcal{PKE}$ be an IND-CPA secure public key encryption scheme,* CiO *be an adaptive computation-trace indistinguishability obfuscation scheme in RAM model,* PRF *be a secure puncturable PRF scheme. Then $\mathcal{GRAM}$ is an adaptive secure garbled RAM scheme with persistent database.*

7.4 Garbled RAM to Delegation: Adaptive Setting

To construct the adaptive RAM delegation, we have to consider full privacy and soundness (Definition 19). There are known generic transformations [AIK10, GHRW14] from $\mathcal{GRAM}$ to delegation ($\mathcal{GRAM}$ 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 then, this RAM delegation achieves 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 is supported in part by grant #360584 from the Simons Foundation and supported in part from a DARPA/ARL SAFEWARE award, NSF Frontier Award 1413955, NSF grants 1228984, 1136174, 1118096, and 1065276. This material is based upon work supported by the Defense Advanced Research Projects Agency through the ARL under Contract W911NF-15-C-0205. The views expressed are those of the author and do not reflect the official policy or position of the Department of Defense, the National Science Foundation, or the U.S. Government.

Kai-Min Chung was partially supported by Ministry of Science and Technology, Taiwan, under Grant no. MOST 103-2221-E-001-022-MY3.

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.

[AJ15] Prabhanjan Ananth and Abhishek Jain. Indistinguishability obfuscation from compact functional encryption. pages 308–326, 2015.

[AJN+16] Prabhanjan Ananth, Aayush Jain, Moni Naor, Amit Sahai, and Eylon Yogev. Universal obfuscation and witness encryption: Boosting correctness and combining security. In *CRYPTO*, 2016.

[AJS15a] Prabhanjan Ananth, Abhishek Jain, and Amit Sahai. Achieving compactness generically: Indistinguishability obfuscation from non-compact functional encryption. *IACR Cryptology ePrint Archive*, 2015:730, 2015.

[AJS15b] Prabhanjan Ananth, Abhishek Jain, and Amit Sahai. Patchable obfuscation. *IACR Cryptology ePrint Archive*, 2015:1084, 2015.

[App14] Applebaum, Benny. Bootstrapping obfuscators via fast pseudorandom functions. In *ASIACRYPT*, 2014.

[AS15] G. Asharov and G. Segev. Limits on the power of indistinguishability obfuscation and functional encryption. In *Foundations of Computer Science (FOCS), 2015 IEEE 56th Annual Symposium on*, pages 191–209, Oct 2015.

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

[BCP] Elette Boyle, Kai-Min Chung, and Rafael Pass. Oblivious parallel RAM and applications. In *TCC 2016-A*.

[BCP14] Elette Boyle, Kai-Min Chung, and Rafael Pass. On extractability obfuscation. In *TCC*, 2014.

[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.

[BGK+14] Boaz Barak, Sanjam Garg, Yael Tauman Kalai, Omer Paneth, and Amit Sahai. Protecting obfuscation against algebraic attacks. In *EUROCRYPT*, 2014.

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

[BR14] Zvika Brakerski and Guy N. Rothblum. Virtual black-box obfuscation for all circuits via generic graded encoding. In *TCC*, 2014.

[BV15] Nir Bitansky and Vinod Vaikuntanathan. Indistinguishability obfuscation from functional encryption. In *IEEE 56th Annual Symposium on Foundations of Computer Science, FOCS 2015, Berkeley, CA, USA, 17-20 October, 2015*, pages 171–190, 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.

[CCHR16] Ran Canetti, Yilei Chen, Justin Holmgren, and Mariana Raykova. Succinct adaptive garbled RAM. In *TCC 2016-B*, 2016.

[CGH+15] Jean-Sébastien Coron, Craig Gentry, Shai Halevi, Tancrede Lepoint, Hemanta K Maji, Eric Miles, Mariana Raykova, Amit Sahai, and Mehdi Tibouchi. Zeroizing without low-level zeroes: New MMAP attacks and their limitations. In *CRYPTO*. 2015.

[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.

[CLT] Binyi Chen, Huijia Lin, and Stefano Tessaro. Oblivious parallel ram: Improved efficiency and generic constructions. In *TCC 2016-A*.

[CLT13] Jean-Sébastien Coron, Tancrède Lepoint, and Mehdi Tibouchi. Practical multilinear maps over the integers. In *CRYPTO*, 2013.

[CLT15] Jean-Sébastien Coron, Tancrede Lepoint, and Mehdi Tibouchi. New multilinear maps over the integers. In *CRYPTO*. 2015.

[CLTV15] Ran Canetti, Huijia Lin, Stefano Tessaro, and Vinod Vaikuntanathan. Obfuscation of probabilistic circuits and applications. In *Theory of Cryptography - 12th Theory of Cryptography Conference, TCC 2015, Warsaw, Poland, March 23-25, 2015, Proceedings, Part II*, pages 468–497, 2015.

[CP13] Kai-Min Chung and Rafael Pass. A simple ORAM. *IACR Cryptology ePrint Archive*, 2013:243, 2013.

[DKW16] Apoorvaa Deshpande, Venkata Koppula, and Brent Waters. Constrained pseudorandom functions for unconstrained inputs. Cryptology ePrint Archive, Report 2016/301, 2016. http://eprint.iacr.org/2016/301.

[GGH13a] Sanjam Garg, Craig Gentry, and Shai Halevi. Candidate multilinear maps from ideal lattices. In *EUROCRYPT*, 2013.

[GGH+13b] Sanjam Garg, Craig Gentry, Shai Halevi, Mariana Raykova, Amit Sahai, and Brent Waters. Candidate indistinguishability obfuscation and functional encryption for all circuits. In *FOCS*, 2013.

[GGH15] Craig Gentry, Sergey Gorbunov, and Shai Halevi. Graph-induced multilinear maps from lattices. In *TCC*. 2015.

[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 *EUROCRYPT*, 2014.

[GHRW14] Craig Gentry, Shai Halevi, Mariana Raykova, and Daniel Wichs. Outsourcing private RAM computation. In *FOCS*, 2014.

[GIS+10] Vipul Goyal, Yuval Ishai, Amit Sahai, Ramarathnam Venkatesan, and Akshay Wadia. Founding cryptography on tamper-proof hardware tokens. In *TCC*. 2010.

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

[GLOS15] Sanjam Garg, Steve Lu, Rafail Ostrovsky, and Alessandra Scafuro. Garbled RAM from one-way functions. In *STOC*, 2015.

[GVW12] Sergey Gorbunov, Vinod Vaikuntanathan, and Hoeteck Wee. Functional encryption with bounded collusions via multi-party computation. In *CRYPTO*, 2012.

[HJO+16] Brett Hemenway, Zahra Jafargholi, Rafail Ostrovsky, Alessandra Scafuro, and Daniel Wichs. Adaptively secure garbled circuits from one-way functions. In *CRYPTO*, 2016.

[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 *STOC*, 2014.

[Lin16] Huijia Lin. Indistinguishability obfuscation from constant-degree graded encoding schemes. In Marc Fischlin and Jean-Sébastien Coron, editors, *Advances in Cryptology - EUROCRYPT 2016 - 35th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Vienna, Austria, May 8-12, 2016, Proceedings, Part I*, volume 9665 of *Lecture Notes in Computer Science*, pages 28–57. Springer, 2016.

[LO13] Steve Lu and Rafail Ostrovsky. How to garble RAM programs. In *EUROCRYPT*, 2013.

[LO15] Steve Lu and Rafail Ostrovsky. Black-box parallel garbled RAM. Cryptology ePrint Archive, Report 2015/1068, 2015. http://eprint.iacr.org/2015/1068.

[LV16] Huijia Lin and Vinod Vaikuntanathan. Indistinguishability obfuscation from ddh-like assumptions on constant-degree graded encodings. Cryptology ePrint Archive, Report 2016/795, 2016. http://eprint.iacr.org/2016/795.

[MSZ16] Eric Miles, Amit Sahai, and Mark Zhandry. Annihilation attacks for multilinear maps: Cryptanalysis of indistinguishability obfuscation over GGH13. In *CRYPTO*, 2016.

[Nao03] Moni Naor. On cryptographic assumptions and challenges. In *CRYPTO*, 2003.

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

[Wat15] Brent Waters. A punctured programming approach to adaptively secure functional encryption. In *CRYPTO*, 2015.

A Detailed Check for the Proof of Lemma 6

We define the first layer hybrids $\mathbf{Hyb}_i$ for $i \in \{0,1\}$, which are exactly two experiments $\mathsf{Exp}\text{-}\mathsf{IND}\text{-}\mathsf{Ci}\mathcal{O}\{b = 0\}$ and $\mathsf{Exp}\text{-}\mathsf{IND}\text{-}\mathsf{Ci}\mathcal{O}\{b = 1\}$ defined in the selective 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 obfuscated computation $\widetilde{\mathsf{mem}}^{1,0}, \{\widetilde{F}^i_{\mathsf{sid}}\}^l_{\mathsf{sid}=1}$ as Algorithm 3.

Let $\mathsf{Adv}^k_{\mathcal{A}}$ be the advantage of $\mathbf{Hyb}_k$ against an adversary $\mathcal{A}$. We argue that $|\mathsf{Adv}^0_{\mathcal{A}} - \mathsf{Adv}^1_{\mathcal{A}}| \le \mathsf{negl}(\lambda)$. To show this, we define the second-layer hybrids $\mathbf{Hyb}_{0,0}, \mathbf{Hyb}_{0,1}$, $\{\mathbf{Hyb}_{0,2,j}, \mathbf{Hyb}_{0,3,j}, \mathbf{Hyb}_{0,4,j}\}^l_{j=1}$. 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^*_{sid} be the terminating time of both programs F^0_{sid} and F^1_{sid} where $t^*_{\mathsf{sid}} < T$. 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, $0 \le i < t^*_j$.

Algorithm 3: $\widehat{F}^i_{\mathsf{sid}}$ for $i \in \{0,1\}$

Input : $\widetilde{\mathsf{st}}^{\mathsf{in}} = ((\mathsf{sid}, t), \mathsf{st}^{\mathsf{in}}, v^{\mathsf{in}}, w^{\mathsf{in}}, \sigma^{\mathsf{in}})$, $\widetilde{a}^{\mathsf{in}}_{\mathtt{A}\leftarrow\mathtt{M}} = (a^{\mathsf{in}}_{\mathtt{A}\leftarrow\mathtt{M}}, \pi^{\mathsf{in}})$ 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 $\mathtt{reject}$;

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 $\mathtt{reject}$;

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 $\mathtt{reject}$;

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

12 If $\mathsf{st}^{\mathsf{out}} = \mathtt{reject}$, output $\mathtt{reject}$;

13 Compute $w^{\mathsf{out}} = \mathsf{hAcc.Update}(\mathsf{PP}_{\mathsf{hAcc}}, w^{\mathsf{in}}, b^{\mathsf{out}}, \pi^{\mathsf{in}})$, output $\mathtt{reject}$ 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 $\mathtt{reject}$ 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* $\mathtt{halt}$ *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 $\widetilde{\mathsf{st}}^{\mathsf{out}} = ((\mathsf{sid}, t+1), \mathsf{st}^{\mathsf{out}}, v^{\mathsf{out}}, w^{\mathsf{out}}, \sigma^{\mathsf{out}})$, $\widetilde{a}^{\mathsf{out}}_{\mathtt{M}\leftarrow\mathtt{A}} = a^{\mathsf{out}}_{\mathtt{M}\leftarrow\mathtt{A}}$;

$\mathbf{Hyb}_{0,0}$. 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 are similar to $\{\widehat{F}^0_{\mathsf{sid}}\}^l_{\mathsf{sid}=1}$ except that they have PRF key K_B hardwired and accept 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 4. 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 5. 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 all t, $i+2 \leq t \leq t^*_{\mathsf{sid}} - 1$, 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.

$\mathbf{Hyb}_{0,4,j}$. 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 formally show Lemma 8 (from $\mathbf{Hyb}_{0,2,j,i}$ to $\mathbf{Hyb}_{0,2',j,i}$) where the proof presented by [CCC+16] is a nice proof.

Summarizing the above result, all hybrids from $\mathbf{Hyb}_0$ to $\mathbf{Hyb}_{0,4,l}$, which gradually substitute F^0s with F^1s, satisfy the three properties in Definition 18. Symmetrically, all hybrids from $\mathbf{Hyb}_1$ to $\mathbf{Hyb}_{1,4,l}$, which gradually substitute F^1 with F^0, also satisfy these properties in Definition 18. 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 8. *Let $1 \leq j \leq l$, $1 \leq i < t^*_j$, 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.*

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: $T, \mathsf{PP}_{\mathsf{hAcc}}, \mathsf{PP}_{\mathsf{Itr}}, v^0, K_A, K_T, K_B, m^i$

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 $(t > t^*$ 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'* *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** $t = i$ *and* $m^{\mathsf{out}} = m^i$ **then**
25 Compute $\sigma^{\mathsf{out}} = \mathsf{Spl.Sign}(\mathsf{sk}'_A, m^{\mathsf{out}})$;
26 **else if** $t = i$ *and* $m^{\mathsf{out}} \neq m^i$ **then**
27 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 $\widetilde{\mathsf{st}}^{\mathsf{out}} = ((\mathsf{sid}, t + 1), \mathsf{st}^{\mathsf{out}}, v^{\mathsf{out}}, w^{\mathsf{out}}, \sigma^{\mathsf{out}})$, $\widetilde{a}^{\mathsf{out}}_{\mathtt{M}\leftarrow\mathtt{A}} = a^{\mathsf{out}}_{\mathtt{M}\leftarrow\mathtt{A}}$;

Algorithm 5: $\widehat{F}^{0,2',j,i}_{\mathsf{sid}=j}$

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: $T, \mathsf{PP}_{\mathsf{hAcc}}, \mathsf{PP}_{\mathsf{Itr}}, v^0, K_A, K_T, K_B, m^i$

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 ($t > t^*$ 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'* *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** $t = i+1$ *and* $m^{\mathsf{in}} = m^i$ **then**
25   Compute $\sigma^{\mathsf{out}} = \mathsf{Spl.Sign}(\mathsf{sk}'_A, m^{\mathsf{out}})$;
26 **else if** $t = i+1$ *and* $m^{\mathsf{in}} \neq m^i$ **then**
27   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 $\widetilde{\mathsf{st}}^{\mathsf{out}} = ((\mathsf{sid}, t+1), \mathsf{st}^{\mathsf{out}}, v^{\mathsf{out}}, w^{\mathsf{out}}, \sigma^{\mathsf{out}})$, $\widetilde{a}^{\mathsf{out}}_{\mathtt{M}\leftarrow\mathtt{A}} = a^{\mathsf{out}}_{\mathtt{M}\leftarrow\mathtt{A}}$;

Proof. Here we only focus on the program $\mathsf{sid} = j$ for simplicity. Define next (deepest) layer hybrids $\mathbf{Hyb}_{0,2,j,i,0}$, $\mathbf{Hyb}_{0,2,j,i,1}$, $\ldots$, $\mathbf{Hyb}_{0,2,j,i,13}$. 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 $0 \leq k < 13$, 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 $\mathbf{Hyb}_{0,2,j,i,k}$, $0 \leq k \leq 13$. To see how to verify the niceness (i.e., 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), and then 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 $K_A\{(j, i)\} \leftarrow \mathsf{PRF.Puncture}(K_A, (j, i))$, $r_C = \mathsf{PRF}(K_A, (j, i))$, $(\mathsf{sk}_C, \mathsf{vk}_C, \mathsf{vk}_{C,\text{rej}}) = \mathsf{Spl.Setup}(1^\lambda; r_C)$ and $K_B\{(j, i)\} \leftarrow \mathsf{PRF.Puncture}(K_B, (j, i))$, $r_D = \mathsf{PRF}(K_B, (j, i))$, $(\mathsf{sk}_D, \mathsf{vk}_D, \mathsf{vk}_{D,\text{rej}}) = \mathsf{Spl.Setup}(1^\lambda; r_D)$. The challenger finally outputs an obfuscation of $\widehat{F}^{0,2,j,i,1}$ which is identical to $\widehat{F}^{0,2,j,i,0}$ 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 $t \neq i+1$, compute $r_{\mathsf{type}} = \mathsf{PRF}(K_{\mathsf{type}}\{(j, i)\}, (\mathsf{sid}, t-1))$, and $(\mathsf{sk}_{\mathsf{type}}, \mathsf{vk}_{\mathsf{type}}, \mathsf{vk}_{\mathsf{type},\text{rej}}) = \mathsf{Spl.Setup}(\lambda; r_{\mathsf{type}})$ 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 $t \neq i$, compute $r'_{\mathsf{type}} = \mathsf{PRF}(K_{\mathsf{type}}\{(j, i)\}, (\mathsf{sid}, t))$, and $(\mathsf{sk}'_{\mathsf{type}}, \mathsf{vk}'_{\mathsf{type}}, \mathsf{vk}'_{\mathsf{type},\text{rej}}) = \mathsf{Spl.Setup}(\lambda; r'_{\mathsf{type}})$ 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 $\mathsf{i}\mathcal{O}$ security, $(CH'_{0,2,j,i,0}, 1/2) = (CH_{\mathsf{i}\mathcal{O}}, 1/2)$.

$\mathbf{Hyb}_{0,2,j,i,2}$. 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 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, $(CH'_{0,2,j,i,1}, 1/2) = (CH_{\mathsf{PRF}}, 1/2)$.

$\mathbf{Hyb}_{0,2,j,i,3}$. 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, $(CH'_{0,2,j,i,2}, 1/2) = (CH_{\mathsf{i}\mathcal{O}}, 1/2)$.

$\mathbf{Hyb}_{0,2,j,i,4}$. 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 $\mathsf{vk}_{D,\mathrm{abo}}$ instead of vk_D. As in the previous hybrid, the challenger uses $\mathsf{Spl.Split}$ to compute $(\sigma_{C,\mathrm{one}}, \mathsf{vk}_{C,\mathrm{one}}, \mathsf{sk}_{C,\mathrm{abo}}, \mathsf{vk}_{C,\mathrm{abo}})$ and $(\sigma_{D,\mathrm{one}}, \mathsf{vk}_{D,\mathrm{one}}, \mathsf{sk}_{D,\mathrm{abo}}, \mathsf{vk}_{D,\mathrm{abo}})$ 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 $\mathsf{vk}_{\mathrm{abo}}$ indistinguishability, $(CH'_{0,2,j,i,4}, 1/2) = (CH_{\mathsf{vk}_{\mathrm{abo}}}, 1/2)$.

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 $G_{0,2,j,i,7} = \mathrm{INDEX}^*_{0,2,j,i,7}(\cdot)$, 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 1. *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 three properties of the nice proof hold.*

Proof. We check each property as follows.

- Property 1: It holds, since function $G_{0,2,j,i,7}$ outputs $\mathrm{INDEX}^*_{0,2,j,i,7}(\alpha)$ with logarithmic-length.
- Property 2: To meet the definition of the nice reduction (See Definition 16), 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 $CH_{\mathsf{hAcc},r} \leftrightarrow R_{0,2,j,i,6}$), so $R_{0,2,j,i,6}$ is a nice reduction.
- Property 3: It suffices for showing that $(\mathbf{Hyb}_{0,2,j,i,7}, G_{0,2,j,i,7})$ 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 imply the first and the third are indistinguishable, and thus this property holds.

□

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 $G_{0,2,j,i,8} = \mathrm{INDEX}^*_{0,2,j,i,7}(\cdot)$ (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 1.

$\mathbf{Hyb}_{0,2,j,i,9}$. 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 1, we claim this is a nice proof.

$\mathbf{Hyb}_{0,2,j,i,10}$. 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, $(CH'_{0,2,j,i,9}, 1/2) = (CH_{\mathsf{Spl}}, 1/2)$.

$\mathbf{Hyb}_{0,2,j,i,11}$. 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, $(CH'_{0,2,j,i,10}, 1/2) = (CH_{\mathsf{i}\mathcal{O}}, 1/2)$.

$\mathbf{Hyb}_{0,2,j,i,12}$. 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, $(CH'_{0,2,j,i,11}, 1/2) = (CH_{\mathsf{PRF}}, 1/2)$.

$\mathbf{Hyb}_{0,2,j,i,13}$. This hybrid corresponds to $\mathbf{Hyb}_{0,2',j,i}$. 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, $(CH'_{0,2,j,i,12}, 1/2) = (CH_{\mathsf{i}\mathcal{O}}, 1/2)$.

□