Forward Secret Encrypted RAM: Lower Bounds and Applications*

Alexander Bienstock[1], Yevgeniy Dodis[1], and Kevin Yeo[2]

[1] New York University
{abienstock,dodis}@cs.nyu.edu
[2] Google and Columbia University
kwlyeo@google.com

Abstract. In this paper, we study *forward secret encrypted RAMs* (FS eRAMs) which enable clients to outsource the storage of an n-entry array to a server. In the case of a catastrophic attack where both client and server storage are compromised, FS eRAMs guarantee that the adversary may not recover any array entries that were deleted or overwritten prior to the attack. A simple folklore FS eRAM construction with $O(\log n)$ overhead has been known for at least two decades. Unfortunately, no progress has been made since then. We show the lack of progress is fundamental by presenting an $\Omega(\log n)$ lower bound for FS eRAMs proving that the folklore solution is optimal. To do this, we introduce the *symbolic model* for proving cryptographic data structures lower bounds that may be of independent interest.

Given this limitation, we investigate applications where forward secrecy may be obtained without the additional $O(\log n)$ overhead. We show this is possible for oblivious RAMs, memory checkers, and multicast encryption by incorporating the ideas of the folklore FS eRAM solution into carefully chosen constructions of the corresponding primitives.

1 Introduction

In recent years, there is an increasing desire to outsource the storage of data to remote servers (such as cloud service providers). By outsourcing, organizations can avoid dealing with problems arising from storing data such as global availability, replication, handling outages, etc. On the other hand, outsourcing incurs new problems with respect to privacy. In many settings, the outsourced data is stored by third-party entities that may not be completely trustworthy. As a result, there is a need for cryptographic protocols that guarantee the outsourced data remains private even when stored by the potentially untrusted storage servers.

A straightforward attempt to obtain privacy is to encrypt all data before being sent to the servers. In more detail, the data owner (also referred to as the client) will store a private key locally and encrypt all data that will be outsourced to the servers. The storage servers will never see the outsourced data in plaintext. Unfortunately, this protocol critically assumes that the client's storage always

* The full version [5] is available as entry 2021/244 in the IACR eprint archive.

remains secure. In the case of a catastrophic attack where the client storage is compromised, the adversary may be able to decrypt all prior ciphertexts observed by the server to obtain the outsourced data in plaintext. Catastrophic attacks will inevitably leak the current state of outsourced data as the client should be able to retrieve the current outsourced data for use. However, we can still aim to provide strong privacy guarantees for prior iterations of outsourced data that may have been overwritten and/or deleted in the past. This is the core problem that we will study in our work.

In more precise terminology, we denote this primitive as *forward secret encrypted RAMs* or *FS eRAMs*. The notion of forward secret encrypted RAMs is not new and has been studied several times in the past two decades under different names such as "secure deletion" [35,37,38,3,39], "how to forget a secret" [14], "self-destruction" [16] and "revocability" [7] to list some examples. FS eRAMs consider the setting with a client and server where the client outsources the storage of an array with n entries to the potentially untrusted server that enables the client to perform read and write operations to any of the n array entries. Note that deletion is supported by simply writing $\perp$ to any array entry. For security, FS eRAMs guarantee that even after a catastrophic corruption of both the client and server storage (and with knowledge of the access pattern to server storage cells from reads and writes), the adversary may only decrypt the array's contents at the time of the compromise. Any array entries that have been overwritten prior to the attack may not be recovered.

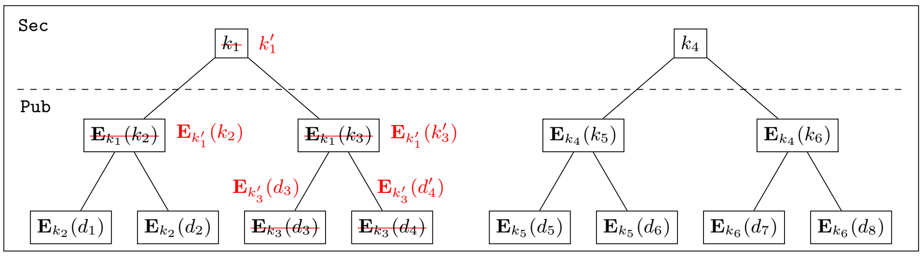

Fig. 1. Folklore Forward Secret Encrypted RAM construction from [14]. In this case, we have $s = 2, n = 8$, and so we have two trees rooted at the secret cells, each with four leaves corresponding to data cells. The two roots store encryption keys, while every other interior node stores an encryption of a key which was used to encrypt the contents of the node's children. All cells, except for the two roots, reside in public storage. We depict with red font the execution of the operation write($\texttt{Sec}, \texttt{Pub}, d_4', 4$) by showing that the keys at the interior nodes on the path of the leaf holding $\mathbf{E}_{k_3}(d_4)$ are replaced by new keys which in turn are used to re-encrypt their children (including new data d_4'). This operation reads and overwrites $O(\log(n/s))$ cells asymptotically and deletes from `Sec` any keys that could be used to recover the old data d_4.

All prior FS eRAM constructions may be unified with a single folklore solution with logarithmic overhead that was, to our knowledge, first presented in [14].

The folklore construction utilizes a binary tree with n leaf nodes that will be used to store array entries. Each internal node stores a symmetric encryption key that is used to encrypt the contents of its two children. The leaf nodes store array entries encrypted by their parent node's key. Finally, the root's encryption key is stored in client memory. To read an array entry, the corresponding root-to-leaf path is downloaded and decrypted sequentially starting from the root node. For writing to an array entry, the root-to-leaf path is downloaded along with the children of all nodes in the path and the leaf node's contents are replaced with the new entry. All encryption keys of internal nodes in the root-to-leaf path are re-generated randomly and all ciphertexts are re-encrypted using the new keys before being uploaded back to the server. Figure 1 presents a diagram of the folklore construction. For both operations, the communication and computation costs are $O(\log n)$. If the client has the capability to store $O(s)$ keys, the efficiency may be reduced to $O(\log(n/s))$ by storing the top $O(\log s)$ levels of the tree in client storage. This folklore construction has been extended in many interesting ways such as handling dynamic array sizes [38] and more complex tree structures like B-trees [39].

Even though this folklore solution has been known for more than two decades, there have been no improvements on the asymptotic efficiency. This leads to the very important question of "Is the folklore forward secret encrypted RAM construction optimal?". Another important question is "If the folklore solution is optimal, are there important applications or settings where one can incorporate forward secret encrypted RAMs without incurring the additional logarithmic overhead?". In this work, we study both questions and answer affirmatively.

1.1 Our Main Result: Lower Bound

As the main result of our work, we present a lower bound for forward secret encrypted RAMs showing the folklore construction is optimal. In the past, lower bounds for cryptographic data structures were mainly proved in either the balls-and-bins [17,8] or cell probe model [47,26,24,34,23,33,27] that lie on opposite ends of the spectrum in terms of flexibility. The balls-and-bins models insists that each server memory location (bins) may store at most one opaque encrypted array entry (balls). Nothing else may be stored in server memory. Therefore, the balls-and-bins model does not encompass the folklore FS eRAM construction that utilizes server memory to store encrypted keys. On the other hand, the cell probe model allows arbitrary encodings of array entries to be stored in server memory. Due to this flexibility, proving cell probe lower bounds is viewed as the holy grail. However, cell probe lower bounds are very difficult to prove and there are long-standing gaps between lower bounds in the cell probe model and more restrictive models.

At a high level, the cell probe model only charges data structures for probing (either reading/writing) a cell. Probing a single cell accounts for one unit cost of computation. All other resources of the data structure may be used without cost, including computation *besides* cell probes, randomness generation, etc. However, this does not apply to adversaries who are typically expected to be PPT. We refer readers to [47] for a formal definition with respect to data structures and [26] with respect to cryptographic data structures.

It turns out that when computation is free, we can construct a simple FS eRAM with only $O(1)$ cell probes. Each of the n entries are encrypted using authenticated encryption. To retrieve an entry, the data structure retrieves the ciphertext and tries to decrypt with all keys until succeeding. This is technically a valid, but practically infeasible, construction in the cell probe model.

To circumvent this problem, one could also require that the data structure is a PPT algorithm. However, this only rules out natural usages of encryption and other cryptographic primitives. Data structures may use cryptographic primitives in unnatural manners (like with weaker security parameters) trying to find an encryption scheme that is breakable by the data structure yet is intractable for the adversary. Unfortunately, this phenomenon occurs due to the assumptions of the adversary's and data structure's computational powers (along with the data structure's private state) as opposed to studying the hardness of the FS eRAM problem.

To address this gap, we introduce the *symbolic model* for proving lower bounds for cryptographic data structures inspired by prior works [29,4] for other primitives. The symbolic model enables server cells to only store strings that are derived from a structured grammar that incorporates *natural usage* of important cryptographic primitives (encryption, (dual)PRFs, etc.). The symbolic model strikes a balance between the balls-and-bins and cell probe model by enabling more flexible server storage beyond just encrypted array entries (like the balls-and-bins model) but not arbitrary encodings (like the cell probe model). Importantly, our symbolic model encompasses the folklore solution. In the symbolic model, we prove our main result showing that the folklore solution is optimal.

Theorem 1 (Informal). *For any client storage s, any forward secret encrypted RAM in the symbolic model must use $\Omega(\log(n/s))$ overhead.*

Our Lower Bound Techniques. We provide a simple, non-adaptive adversary that at each time t selects a random virtual cell to overwrite with new data. We show that after each of these operations, in expectation, the contents of logarithmically (in n/s) many public and secret cells that were used by the protocol become unusable by the protocol. I.e., because the protocol must force itself to delete some of the secrets which it used to recover the old data, for any encoding of some other secret in a public or secret cell, if decoding requires a deleted secret, the encoding is useless. Since we show that we can identify $\Omega(\log(n/s))$ different such cells at each instant, we reach our lower bound.

To show that the contents of logarithmically-many cells become useless at each t, we abstract the relationship between the contents of the secret cells, the keys that the protocol uses, and the virtual cells at each instant into a directed graph $\mathcal{G}$ in Definition 9, which we call the *key-data* graph. More specifically, the contents in each secret cell, the keys that the protocol uses (and can thus recover using the contents of the secret and public cells), and the virtual cells at some time t are the vertices of $\mathcal{G}$. Each key vertex has an edge to another vertex if it was used in a (d)PRF computation to generate the target vertex, or it was used in the generation of a string (e.g., as an encryption key) in one of the public or secret cells that encodes the target vertex. Additionally, each vertex corresponding to the contents of a secret cell that is not a key has an edge to another vertex if it encodes the target vertex.

By correctness, we show that at each instant of the protocol, for every data item stored in the virtual cells, there exists a collection of paths starting from vertices corresponding to secret cells and ending at the vertex corresponding to the data. These paths abstract the notion that together with the contents of the public cells at that instant, those secret cells (and not any subset of them) can recover the data. Moreover, these paths satisfy the special property that if the corresponding data cell is overwritten, all of the vertices along at least one of these paths must be made indefinitely inaccessible by the protocol for it to no longer be able to recover the old data. This choice of paths is completely determined by the protocol and indeed the protocol may make this choice in an effort to minimize the amount of computation it has to do. By proving a graph-theoretic lemma about the out-degrees of the nodes on any such chosen path in the key-data graph (i.e. that they sum to $\log n/s$), we show that in expectation over the virtual cell which the adversary chooses to overwrite, the number of cells which can no longer be used to access the secrets they encode as a consequence of all of the vertices on the path becoming inaccessible is $\Omega(\log(n/s))$ in expectation.

1.2 "Bypassing" the Lower Bound

Equipped with the knowledge that the folklore solution is optimal, we investigate applications where FS eRAMs may be incorporated without the additional logarithmic overhead. At a high level, we will show that the folklore FS eRAM construction may be overlaid into constructions that already utilize tree-like structures. We prove this is true for three such primitives.

Oblivious RAMs. Oblivious RAMs (ORAMs) [17,36,18,42,32,1] are cryptographic primitives in the client-server setting that obfuscate the client's access pattern to the underlying array entry even when the server observes physical accesses to server storage. Note ORAMs do not protect against client corruption. The best ORAM constructions require $O(\log n)$ overhead. A naive composition with FS eRAMs incurs $O(\log^2 n)$ overhead. In Section 5, we provide a construction that essentially avoids additional overhead over ORAM:

Theorem 2 (Informal). *There exists a construction that is both a forward secret encrypted RAM and an oblivious RAM with $O(\log n \cdot f(n))$ overhead and $O(1)$ client storage for any function $f(n) = \omega(1)$.*

As an additional contribution, we also show that stronger notions of forward secret obliviousness are expensive in the cell probe model. One natural notion might be to provide forward secrecy for access patterns. After client compromise, the server may not learn information about the prior accesses to data. We denote this as strong oblivious forward secret encrypted RAMs. We note a similar lower bound was presented in [39], but only in the balls-and-bins model. Our result is slightly stronger since it is proved in the cell probe model.

Theorem 3 (Informal). *For any client storage s, any strong oblivious forward secret encrypted RAM in the cell probe model must use $\Omega(n - s)$ overhead.*

Memory Checkers. Memory checkers (MCs) [6,31,11], are cryptographic primitives in the client-server setting which provide authenticity of an outsourced data array for the client. MCs require $\Omega(\log n / \log \log n)$ overhead [15], and the best known constructions require $O(\log n)$ overhead.[1] Again, a naive composition with FS eRAM also provides forward secrecy of the data, but requires $O(\log^2 n)$ overhead. In Section 6, we provide a construction that avoids the extra overhead:

Theorem 4 (Informal). *There exists a forward secret memory checker with $O(\log n)$ overhead and $O(1)$ client storage.*

Multicast Encryption. Multicast encryption (ME) is a primitive that allows a group manager to securely and efficiently distribute secrets to an evolving group of users. After each group membership change, a new epoch is initiated, and the group manager sends ciphertexts over a broadcast channel which allow only the *current* group members to derive the next group secret.[2] ME has been widely studied in the literature [45,46,21,22,9,30,41]. Indeed, these works can be unified into a folklore construction based on binary trees which tightly achieves optimal $O(\log n)$ communication and computational complexity per epoch with respect to the lower bound of [29]. However, this folklore construction has large group manager secret state and does not protect against corruption of this state. In the full version [5], we provide a construction that achieves group manager FS, while reducing its secret state to $O(1)$ size and retaining the optimal efficiency of the folklore solution, without an extra $O(\log n)$ factor of computational overhead:

Theorem 5 (Informal). *There exists an ME construction that is forward secret with respect to group manager corruptions, has $O(1)$ group manager secret storage, and $O(\log n)$ communication and computation per epoch.*

2 Lower Bound Model

In this section, we present a general framework for proving computational lower bounds on cryptographic or privacy-preserving data structures using a symbolic

[1] In both cases, for *online* MCs that access the remote storage in a deterministic and non-adaptive manner. Online MCs report any inauthentic retrieval from the server immediately, as opposed to after a long sequence of retrievals. Recall that the folklore FS eRAM construction also makes deterministic, non-adaptive accesses.

[2] In the full version [5], we briefly compare ME with a harder setting called Continuous Group Key Agreement.

model, then formalize it for the case of FS eRAM. The symbolic model we present is inspired by the one used for communication complexity lower bounds originally by Micciancio and Panjwani in [29] for multicast encryption and also recently in [4] for concurrent group ratcheting.[3]

2.1 Framework for Symbolic Private Data Structure Lower Bounds

The first step in proving a lower bound on some private data structure in our symbolic framework is to decide which primitives are allowed in the constructions. Based on these primitives, one has to define a *grammar* within the model which specifies exactly the types of strings that can be created and stored in the structure to keep the data private. For example, if one of the allowed primitives is encryption, then the grammar must specify strings that correspond to encryption keys and encryptions of certain other strings derived from the grammar. If no other primitives are allowed, for example PRFs, then the constructions can only use such a key to generate more ciphertexts according to the grammar, and not other keys through PRF computations, for example. This grammar only defines the exact form of strings that can be generated by the allowed primitives, but does not on its own define how these strings can be used to recover the data.

The manner in which constructions can use sets of strings derived from the grammar to recover other strings, including data, is specified by an *entailment relation*. We emphasize that this relation takes as input strings derived from the grammar, generated by the functionality of the allowed primitives, and outputs other strings within the grammar, also based on the functionality of the allowed primitives. For example, if a construction stores a set of strings which include a ciphertext within the grammar that is generated by an encryption algorithm, along with the encryption key, the entailment relation specifies that the plaintext that also falls within the grammar can be recovered via the decryption algorithm. We note that unlike in traditional models, we only define the syntax and security of the allowed primitives implicitly within the grammar and entailment relation. Further utilizing the encryption example, if the key for a ciphertext is not available, then the entailment relation prohibits derivation of the underlying plaintext (and thus it is secure).

Grammars and corresponding entailment relations can generally be used for lower bounds on any private data structures. Indeed, we will provide our own in Section 2.2. However, we again stress that for some arbitrary private data structure, one might use a completely different grammar and entailment relation to show a lower bound in the symbolic framework. To apply the grammar and entailment relation to a lower bound for a specific private data structure, one has to provide syntax, correctness, and security definitions in the symbolic model. For correctness, in general, constructions must derive strings from the grammar to store in the data structure and later use these strings, along with the entailment relation, to derive the plaintext data that they store. For security, we

[3] The model is also related to that of automated protocol verification (see e.g., [25]).

use the implicit security definitions for the allowed primitives within the grammar and entailment relation to show that an adversary cannot derive certain data using certain other strings (e.g., encrypted storage). In general, such security definitions may not be as strong as the standard (e.g., indistinguishability based) definitions for private data structures. However, proving a lower bound based on a weaker adversarial model only strengthens the result.

2.2 Symbolic Definitions for Allowed Primitives

Our symbolic model allows the FS eRAM cells (secret and public) to store only certain types of strings generated by the functionality of our allowed primitives and specified by our grammar, which can only be interpreted and utilized via our entailment relation. We introduce the allowed primitives, the grammar and entailment relation in this section. Note again: our grammar and entailment relation for these primitives can be used for other private data structure lower bounds in the symbolic framework, but some such lower bounds may also allow for other primitives (e.g., public key techniques), and use some other grammar, and/or entailment relation.

Cryptographic primitives. The primitives which we choose to consider in our lower bound encompass all of the reasonable primitives which one would use for FS eRAM, while excluding those that are too powerful and inefficient in practice. Indeed, a trivial construction exists from Puncturable Encryption (PE) [10,12,19,20,13,44,43], by simply revoking all overwritten ciphertexts. However, PE is a strong primitive and is seen as too computationally inefficient for most practical applications (see e.g., [2]). Therefore, we do not consider PE in our model and indeed view a corollary of our lower bound as a lower bound on the overhead of PE schemes from practical primitives.

Thus, we choose to include only those primitives that are natural and useful for FS eRAM, as well as not too powerful as to make the problem trivial. Since FS eRAM protocols are carried out by a single entity, we only consider symmetric techniques. Namely, we include symmetric encryption, (dual) pseudorandom functions, and secret sharing. These are all of the primitives (and more) which the folklore construction and all other constructions in the literature use [38,7,14]. In fact, only symmetric encryption is used in these prior constructions. We include (dual) pseudorandom functions, as they have been shown to achieve speedups for many other primitives. We note that FS eRAMs are easy to achieve with two non-colluding servers as the array may be secret shared between the two servers. Therefore, we add secret sharing in the case that it may be useful even in the single-server setting. We introduce the primitives here, before defining them formally in our grammar and entailment relation.

An encryption algorithm $\mathbf{E}$ takes as input a key K and string C, and outputs a ciphertext $C' = \mathbf{E}_K(C)$. Informally, $\mathbf{E}_K(C)$ hides the string C. Symbolically, the string C can be recovered from the ciphertext, only by using the key K and corresponding decryption algorithm: $\mathbf{D}_K(\mathbf{E}_K(C)) = C$.

A PRF is an efficiently computable function $\mathbf{F}$ that takes as input a single key K_1 and some string C and outputs a pseudorandom and independent key K_2: $\mathbf{F}(K_1, C) = K_2$. Symbolically, K_2 can only be computed with $\mathbf{F}$ on input K_1 and not some other input key $K_1' \neq K_1$.

We also consider dPRFs which are efficiently computable functions $\mathbf{dF}$ that take as input two keys K_1 and K_2 and output a pseudorandom and independent key K_3: $\mathbf{dF}(K_1, K_2) = K_3$. Symbolically, K_3 can only be computed with $\mathbf{dF}$ on input K_1 and K_2 and not some other pair of keys (K_1', K_2') such that *either* $K_1' \neq K_1$ *or* $K_2' \neq K_2$. The output key of both PRFs and dPRFs can be used as encryption keys.

A secret sharing scheme allows one to *split* a string C into shares $\mathbf{S}_1(C), \ldots,$ $\mathbf{S}_{\mathtt{n}}(C)$, for some fixed integer $\mathtt{n}$, such that C can only be recovered from certain subsets of the shares. These subsets are defined using an *access structure* $\Gamma \subseteq 2^{\{1,\ldots,\mathtt{n}\}}$. For each subset $I \in \Gamma$, if we are given the set of shares $\mathbf{S}_I(C) = \{\mathbf{S}_i(c)\}_{i \in I}$ of some string C, then we can efficiently recover C using the reconstruction function $\mathbf{R}$: $\mathbf{R}(I, S_I(C)) = C$. Symbolically, if we only have a set of shares $S_{I'}(C)$ for some $I' \notin \Gamma$, then we cannot recover C.

Grammar and entailment relation definitions. We will allow cells of the FS eRAM to contain strings which are arbitrary nested combinations of encryption, PRFs, dPRFs, and secret sharing. We formally allow cell contents to be described by strings derived from the grammar in the left of Figure 2.

$C \to K \vert \mathbf{E}_K(C) \vert \mathbf{S}_1(C) \vert \ldots \vert \mathbf{S}_{\mathtt{n}}(C) \vert D$ $K \to R \vert \mathbf{F}(K, C) \vert \mathbf{dF}(K, K)$	$c \in \mathsf{C} \implies \mathsf{C} \vdash c$ $\mathsf{C} \vdash k \implies \mathsf{C} \vdash \mathbf{F}(k, c), \forall c$ $\mathsf{C} \vdash k_1, k_2 \implies \mathsf{C} \vdash \mathbf{dF}(k_1, k_2)$ $\mathsf{C} \vdash \mathbf{E}_k(c), k \implies \mathsf{C} \vdash c$ $\exists I \in \Gamma : \forall i \in I, \mathsf{C} \vdash \mathbf{S}_i(c) \implies \mathsf{C} \vdash c$

Fig. 2. Definitions of: 1. (left) the grammar used in our symbolic model to describe the strings which any FS eRAM protocol can create and store in secret and public cells, and 2. (right) the entailment relation $\vdash$, where in the second through fourth rules, k, k_1, k_2 are of type K in our grammar.

The variables C and K in the grammar represent strings and keys, D is a variable that ranges over the arbitrarily large set of plaintext strings which we allow the user to write into the virtual cells of the FS eRAM, and R ranges over an arbitrarily large set of truly random keys. We will call strings created from variable K as keys and those from variable D as data throughout the next two sections. Observe that the input to PRFs that is not the key can be any string C derived from the grammar. Some remarks:

- We emphasize that the data cells can only hold strings of type D. We mainly make this restriction for simplicity so that data cannot be used as keys, secret

shares, or ciphertexts in any protocol. Note: the adversary in the security definition, as will be later seen, chooses the contents of the data cells.
– For convenience, we do not allow ciphertexts or secret shares to be encryption keys or (d)PRF keys. Our proof may be modified to enable ciphertexts or secret shares to be keys if desired.
– We assume that all cells are of approximately the same length, and that each of them can hold any one string derived from the grammar. This means that the functions $\mathbf{E}, \mathbf{F}, \mathbf{dF}, \mathbf{S}$ have outputs of about the same length.

Again, it is important to observe that the strings specified by the above grammar are *purely syntactic* and without any meaning. Their only intended meaning is described by the entailment relation $\mathsf{C} \vdash c$, specifying which strings c can be recovered given a set of strings C. This entailment relation follows naturally from the standard cryptographic definitions for the primitives we consider (and inductively from the base case): 1. For PRFs, one can only compute the function if the key is revealed; 2. For dPRFs, one can only derive an output for a given input if they have *both* keys (security holds if only one key is revealed); 3. For encryption, one can only derive the underlying plaintext of a ciphertext if they have the key; and 4. For secret sharing, one can reconstruct a secret that has been shared only if they have all shares for a given subset of the access structure. We formally define the relation in the right part of Figure 2 and add the following useful definition.

Definition 1. *For any* C*, we denote the set of strings that can be recovered from* C *using* $\vdash$ *as* $Rec(\mathsf{C})$*.*

2.3 FS eRAM Symbolic Definition

Now we provide the formal definition of FS eRAM in our symbolic model. Throughout the execution of an FS eRAM protocol Π, it forward secretly stores a user-chosen ordered n-element array $\mathtt{D}$ of virtual cells, which hold strings of type D in our grammar. We refer to these cells as the *data cells* and their initial contents as array $\mathtt{D}_0$. We again emphasize that data cells can only hold strings of type D in our grammar, and thus cannot hold strings arbitrarily derived from C in our grammar.

At each instant t, Π is given one data cell $i \in [n]$ to overwrite with new data d so that the contents of the data cells at time t, $\mathtt{D}_t$, satisfy: $\mathtt{D}_t[i] = d$ and $\mathtt{D}_t[j] = \mathtt{D}_{t-1}[j], \forall j \in [n] \setminus \{i\}$. Π proceeds using an array of s ($\ll n$) secret cells, $\mathtt{Sec}$, and a (possibly arbitrarily large) array of public cells $\mathtt{Pub}$, which can hold strings derived from our above grammar.[4] We emphasize that in normal protocol execution, the cells in $\mathtt{Pub}$ can always be viewed by an adversary $\mathcal{A}$, while the cells in $\mathtt{Sec}$ cannot be viewed by $\mathcal{A}$ until corruption. Π has the following syntax:

Definition 2 (Syntax). *A* Forward Secret Encrypted RAM *protocol* $\Pi =$ (init, write) *consists of the following algorithms:*

[4] Every cell of $\mathtt{Sec}$ and $\mathtt{Pub}$ initially contains the special *empty* symbol $\perp$.

– init($\mathtt{D}_0 = (d_1, \ldots, d_n)$)*, which takes in the n initial data cell contents $\mathtt{D}_0$ and computes the initial state of the cells* `Sec` *and* `Pub` *using strings derived from C in our grammar.*
– write(i, d)*, which takes in a cell index i and new data d and overwrites the contents of some cells of* `Sec` *and* `Pub` *using strings derived from C in our grammar.*
– read(i)*, which takes in a cell index i and using the contents of the cells of* `Sec` *and* `Pub` *(and the entailment relation) returns the data d_i stored there.*

We note that Π need not be deterministic either in choosing contents of cells or in choosing which cells to write to, i.e., Π could have access to an arbitrarily long, finite random string $\mathcal{R}$ at each instant. However, as we will later describe, the adversary that we consider to reach our lower bound is agnostic to any randomness that Π uses.

An adversary $\mathcal{A}$ specifies the data $\mathtt{D}_0$ input to the init algorithm, as well as for each $t > 0$, the cell i and data d input to the write algorithm. At each instant t, we refer to the contents of `Sec` and `Pub` as arrays $\mathtt{Sec}_t$ and $\mathtt{Pub}_t$, respectively. For any sequence of data cells chosen by $\mathcal{A}$, $\widetilde{\mathtt{D}}_t = (\mathtt{D}_0, \mathtt{D}_1, \ldots, \mathtt{D}_t)$, let $\mathtt{Pub}(\widetilde{\mathtt{D}}_t)$ denote the union of all strings written to public cells by Π when $\mathtt{D}_0, \mathtt{D}_1, \ldots, \mathtt{D}_t$ are specified by calls to init and write, i.e. $\mathtt{Pub}(\widetilde{\mathtt{D}}_t) = \cup_{i=0}^{t} \mathtt{Pub}_i$. Similarly, let $\mathtt{Sec}(\widetilde{\mathtt{D}}_t)$ denote the union of all strings written to secret cells by Π as a result of $\widetilde{\mathtt{D}}_t$, i.e., $\mathtt{Sec}(\widetilde{\mathtt{D}}_t) = \cup_{i=0}^{t} \mathtt{Sec}_i$. Additionally, at any time t, we refer to all of the previous strings of each cell of $\mathtt{D}_t$ as $Prev(\mathtt{D}_t) = (\bigcup_{j=0}^{t-1} \mathtt{D}_j) \setminus \mathtt{D}_t$.[5]

For correctness, we intuitively require that with $\mathtt{Sec}_t$ and $\mathtt{Pub}_t$, Π should be able to recover and successfully return $\mathtt{D}_t[i] \leftarrow \text{read}(i)$ for all $i \in [n]$. For forward secrecy, at each instant t, we also want to protect all data that used to be in `D`, but was since overwritten, in case of a corruption of $\mathtt{Sec}_t$ by $\mathcal{A}$. We abstract the above conditions (without explicitly providing read or corruption oracles to $\mathcal{A}$) using the following definition:

Definition 3 (Correctness and Security). *Π is correct and secure if for all t, and all sequences $\widetilde{\mathtt{D}}_t$ determined by an adversary $\mathcal{A}$:*

– (Correctness): *All of the data cells $\mathtt{D}_t$ can be recovered by the contents of the secret and public cells and successfully returned at time t: For every $i \in [n], d_i \leftarrow \text{read}(i)$, where $d_i = \mathtt{D}_t[i]$ and $\mathtt{D}_t[i] \in Rec(\mathtt{Sec}_t \cup \mathtt{Pub}_t)$.*[6]
– (Security): *The previous contents of all cells in* `D` *with respect to time t cannot be recovered by the contents of the secret cells at time t and all public cells written to up to time t: $\forall d \in Prev(\mathtt{D}_t), d \notin Rec(\mathtt{Sec}_t \cup \mathtt{Pub}(\widetilde{\mathtt{D}}_t))$.*

[5] All set operations specified in the definitions of $\mathtt{Pub}(\widetilde{D}_t)$, $\mathtt{Sec}(\widetilde{D}_t)$, $Prev(\mathtt{D}_t)$, and the remainder of the proof are taken with respect to the sets containing the unique elements of the corresponding operands, ignoring cells with $\perp$. We may in fact directly refer to these defined arrays as sets in the remainder. While such definitions will not consider duplicate cells, our lower bound proof will not have to take into account the number of duplicates.

[6] Condition $\mathtt{D}_t[i] \in Rec(\mathtt{Sec}_t \cup \mathtt{Pub}_t)$ forces Π to be a proper symbolic construction.

Again, we note that the security of this definition is not as strong as an indistinguishability-based definition, as we require explicit recovery of previous data to break security. However, this only strengthens our lower bound.

We will measure the computational complexity of the protocol Π by amortizing over the number of unique strings derived from our grammar that are written to the public and secret cells throughout the execution of the protocol. This measure of course does not track *all* of the computation of some protocol Π, which only strengthens our lower bound. For example, we do not count the computational cost of any of our primitives. Formally, the computational cost $c(\widetilde{\mathtt{D}}_t)$, incurred by an execution of the protocol when run on input $\widetilde{\mathtt{D}}_t$, is defined as

$$c(\widetilde{\mathtt{D}}_t) \coloneqq \frac{|\mathtt{Pub}(\widetilde{D}_t) \cup \mathtt{Sec}(\widetilde{D}_t)|}{t+1}.$$

3 Forward Secret Encrypted RAM Lower Bound

In this section, we prove the following Theorem in several steps.

Theorem 6. *There exists a non-adaptive, randomized adversarial sequence of* init *and* write *operations such that for any FS eRAM protocol Π that is correct and secure with respect to Definition 3, the amortized computational complexity cost incurred by the protocol when executed against that strategy is in expectation $\Omega(\log(n/s))$.*

Before we formalize our lower bound, we start with a simple example which demonstrates an important observation needed in our proof, and which we will refer to throughout the proof. Suppose that at time t, we have $s = 1$, $n = 4$ and keys k_1, k_2, k_3 such that $k_1 \in \mathtt{Sec}_t$. Further, we have $\{\mathbf{E}_{k_1}(k_2), \mathbf{E}_{k_1}(k_3), \mathbf{E}_{k_2}(d_1), \mathbf{E}_{k_2}(d_2), \mathbf{E}_{k_2}(\mathbf{E}_{k_3}(d_3)), \mathbf{E}_{k_3}(d_4)\} \subseteq \mathtt{Pub}(\widetilde{\mathtt{D}}_t)$, for $\mathtt{D}_t = \{d_1, d_2, d_3, d_4\}$. We can informally abstract this into a graph depicting the relations between keys and data shown in Figure 3.

The graph demonstrates that k_1 encodes information about k_2 and k_3, while k_2 and k_3 both encode some information about d_3. Additionally, k_2 encodes information about d_1 and d_2, while k_3 encodes information about d_4. Now suppose that virtual cell 3 (corresponding to d_3) is overwritten by some adversary. The goal of the protocol is effectively to create a new graph that retains as much reachability, in the graph theoretic sense, from the secret cells as possible (since we want to minimize the amount of computation needed to achieve correctness), while still disabling any ability to recover d_3 if any of the keys in the new graph are obtained by the adversary.

In our example, it is sufficient to remove k_1, as then k_2 and k_3, and thus d_3 cannot be recovered. However, this may be a rather inefficient method, as k_2 and k_3 also encode information about data d_1, d_2, d_4, and thus the protocol would have to generate fresh encodings of them for correctness. Unfortunately, we cannot retain reachability to both k_2 and k_3: it is necessary to remove *either* one of the edges $k_2 \to d_3$ or $k_3 \to d_3$ in the graph (which is done by removing k_2 or k_3,

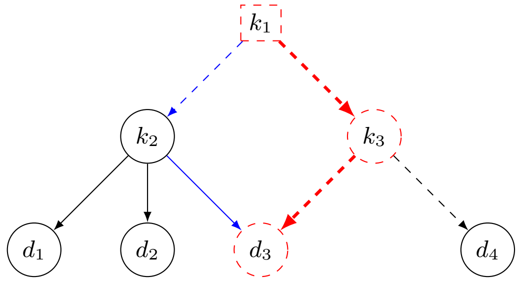

Fig. 3. Key-data graph $\mathcal{G}_{\Pi}(\widetilde{\mathtt{D}}_{t-1})$ at time $t-1$ as defined by Definition 9 for an execution of an FS eRAM protocol Π in which $s=1$, $n=4$, $k_1 \in \mathtt{Sec}_{t-1}$ (denoted by its square border), $\{\mathbf{E}_{k_1}(k_2), \mathbf{E}_{k_1}(k_3), \mathbf{E}_{k_2}(d_1), \mathbf{E}_{k_2}(d_2), \mathbf{E}_{k_2}(\mathbf{E}_{k_3}(d_3)), \mathbf{E}_{k_3}(d_4)\} \subseteq \mathtt{Pub}(\widetilde{\mathtt{D}}_{t-1})$, and $\mathtt{D}_{t-1} = \{d_1, d_2, d_3, d_4\}$. Since k_1 *minimally recovers* d_3 according to Definition 4, the collection of paths $\mathcal{P}_3^{t-1}$ that exist according to Lemma 1 are $P_{3,1} \coloneqq k_1 \to k_2 \to d_3$, represented by blue edges, and $P_{3,2} \coloneqq k_1 \to k_3 \to d_3$, represented by thick red edges. By Lemma 2, if at time t, d_3 is overwritten, all of the vertices on one of these paths must become indefinitely useless (see Definitions 5, 6). Since k_2 can be used to recover more data (besides d_3) than k_3 can, the protocol may choose to make the vertices of $P_{3,2}$ useless at time t (represented by dashed borders). In this case, all of the strings in $\mathtt{Pub}(\widetilde{\mathtt{D}}_t)$ corresponding to the dashed edges $k_1 \to k_2$, $k_1 \to k_3$, $k_3 \to d_3$, and $k_3 \to d_4$ become indefinitely useless at time t (see Definition 8), as the protocol can no longer recover the keys and data that they encapsulate. However, we stress that the choice of the path $P_{3,1}$ or $P_{3,2}$ on which all nodes must become useless at time t is determined by the protocol.

respectively), since together, they can be used to recover d_3 from $\mathbf{E}_{k_2}(\mathbf{E}_{k_3}(d_3))$, but the absence of one of them enables secrecy of d_3. Furthermore, it is still necessary to remove k_1, as it alone can recover d_3 by recovering k_2 and k_3. In this case, the protocol may want to remove just k_1 and k_3, as retaining reachability to k_2 retains reachability to two data cells, without any extra computation (as opposed to just one if reachability to k_3 is retained). However, we emphasize that the choice of which key to retain reachability to is indeed completely decided by the protocol, and the adversary does not have any control over this. The key to our proof is to lower bound the minimal amount of reachability, corresponding to edges in the graph and ciphertexts stored in public and secret cells in the protocol, that must be lost after each operation. Note that this is not the exact formalization, as the proof needs to handle several subtleties and complex cases, as later detailed in this section.

To do this, we will first abstract a meaningful graph-theoretic notion to determine the (amortized) minimum number of unique strings that any FS eRAM protocol, captured by our symbolic model, must write to its public and secret cells during each operation to preserve forward secrecy and correctness.

3.1 Minimality and Usefulness

We first introduce the notion of sets of strings defined by our grammar *minimally* recovering other sets of strings. Intuitively, a set C of strings minimally recovers another set C' of strings if the strings of C can together recover the strings of C' (using the contents of the public cells up to time t, too), but the removal of any string $c \in \mathsf{C}$ prevents the recovery of at least one string $c' \in \mathsf{C}'$.

Definition 4. *A set* C *of strings* minimally recovers set C' *of strings if* $\mathsf{C}' \subseteq \mathit{Rec}(\mathsf{C} \cup \mathtt{Pub}(\widetilde{\mathtt{D}}_t))$ *and for any* $c \in \mathsf{C}$, $\mathsf{C}' \nsubseteq \mathit{Rec}((\mathsf{C} \setminus \{c\}) \cup \mathtt{Pub}(\widetilde{\mathtt{D}}_t))$*.*

If C' contains a single element c', we may only write "C recovers c'" at certain points throughout our exposition (instead of C' or $\{c'\}$).

Now, for any execution of an FS eRAM protocol Π, at certain instants some of the keys that have been used by it may be accessible by the secret cells, while others may not. We call such keys *useful* and *useless*, respectively.

Definition 5. *A key k is* useful *at time t if* $\exists S \subseteq \mathtt{Sec}_t$ *such that S minimally recovers k. It is* useless *otherwise.*

The set of useful keys at time t is denoted $\mathtt{UsefulKeys}_{\Pi}(\widetilde{\mathtt{D}}_t)$. The intuition behind this definition is that as data cells are overwritten, at least some of the keys that the protocol used to recover the previous data (if there are any) for correctness cannot be used anymore, for otherwise, previous data would be accessible, which would violate security. For example, in Figure 3, keys k_1 and k_3 become useless at time t, because they can be used to recover d_3. Similarly, we can define data in terms of *usefulness*.

Definition 6. *Data d is* useful *at time t if* $d \in \mathtt{D}_t$*. It is* useless *otherwise.*

We also define *usefulness* for strings derived by our grammar (that are not keys), based on the keys that are used to create them, and if they are accessible by the adversary at a given time. We first recall that there are four types of cryptographic operations we allow to derive strings from our grammar: encryptions, secret sharing, and (d)PRF computations. Based on this, we define how a string from our grammar can *encapsulate* a key or data:

Definition 7. *We say that a string c* encapsulates *key or data c', if c is the result of arbitrarily nested encryption and secret sharing operations on c'; i.e.,* $c = e_1(e_2(\ldots(e_l(c')\ldots))$ *for some* $l \geq 1$*, where each e_i is either* $\mathbf{E}_{k_i}$ *for some key k_i or* $\mathbf{S}_j$ *for some* $j \in [\mathtt{n}]$*.*

Definition 8. *A string c that encapsulates c′, as in Definition 7 above, is* useless at time t *if*

- *at least one e_i corresponds to* $\mathbf{E}_{k_i}$ *for* $k_i \notin \mathtt{UsefulKeys}_{\Pi}(\widetilde{\mathtt{D}}_t)$*, or*
- $c \notin \mathtt{Sec}_t \cup \mathtt{Pub}(\widetilde{\mathtt{D}}_t)$*.*[7]

[7] Note that we do not say $c \notin \mathit{Rec}(\mathtt{Sec}_t \cup \mathtt{Pub}(\widetilde{\mathtt{D}}_t))$, since for our lower bound we want to only count those strings that were actually stored in some secret cell and are erased at some time. Also, as will be seen later, it must be that for any such string that we count in our lower bound, it must further be that $c \notin \mathit{Rec}(\mathtt{Sec}_t \cup \mathtt{Pub}(\widetilde{\mathtt{D}}_t))$.

Otherwise, it is useful*.*

It is important to note that usefulness is dynamic: keys and strings that are useful at one instant may be useless at another instant. As noted above, every time a data cell is overwritten, some of the keys that can be used to recover its old contents must become useless for security and thus all strings that were in part generated by such keys (via the encryption algorithm, **E**), as well as some strings which encapsulated these keys or the data, must become useless too. For example, in Figure 3, since keys k_1 and k_3 become useless at time t, strings $\{\mathbf{E}_{k_1}(k_2), \mathbf{E}_{k_1}(k_3), \mathbf{E}_{k_2}(\mathbf{E}_{k_3}(d_3)), \mathbf{E}_{k_3}(d_4)\} \subseteq \mathtt{Pub}(\widetilde{\mathtt{D}}_t)$ must too.

Our goal is to show that after every write operation, logarithmically many (in n/s) strings stored in public and secret cells must become useless. Such strings may have been generated and stored in their respective cells at any time in the past, but the protocol incurs a computational cost of at least one per string.

3.2 Key-Data Graph

We will interpret secret cells, keys, and data using graph-theoretic terminology. For any execution of the protocol Π on a sequence of data cells $\widetilde{\mathtt{D}}_t$, we associate a directed graph $\mathcal{G}_\Pi(\widetilde{\mathtt{D}}_t)$ called the *key-data graph* at that time. Each vertex in the graph is either a string in a secret cell at time t, a useful key at time t, or a data cell at time t, and edges between the vertices abstract the process of key or data recovery (by use of the entailment relation $\vdash$).

Definition 9. *Let Π be a Forward Secret Encrypted RAM protocol executed with a sequence of data cells $\widetilde{\mathtt{D}}_t$. The* Key-Data Graph *for the protocol at time t is a directed graph $\mathcal{G}_\Pi(\widetilde{\mathtt{D}}_t) = (\mathcal{V}, \mathcal{E})$, where $\mathcal{V} = \mathtt{UsefulKeys}_\Pi(\widetilde{\mathtt{D}}_t) \cup \mathtt{Sec}_t \cup \mathtt{D}_t$ and $\mathcal{E}$ is the set of all ordered pairs $(v_1, v_2) \in \mathcal{V} \times \mathcal{V}$ such that at least one of the following is true:*

1. *$\exists c$ s.t. $v_2 = \mathbf{F}(v_1, c)$.*
2. *$\exists k \in \mathtt{UsefulKeys}_\Pi(\widetilde{\mathtt{D}}_t)$ s.t. $v_2 = \mathbf{dF}(v_1, k)$ or $v_2 = \mathbf{dF}(k, v_1)$.*
3. *$\exists c \in \mathtt{Pub}(\widetilde{\mathtt{D}}_t) \cup \mathtt{Sec}_t$ s.t. $c = e_1(\mathbf{E}_{v_1}(e_2(v_2)))$ (for arbitrary sequences of encryption and sharing operations e_1, e_2).*
4. *$v_1 \in \mathtt{Sec}_t, v_2 \in \mathtt{UsefulKeys}_\Pi(\widetilde{\mathtt{D}}_t) \cup \mathtt{D}_t$ and v_1 encapsulates v_2.*

We note that in Figure 3, we depict exactly $\mathcal{G}_\Pi(\widetilde{\mathtt{D}}_{t-1})$ for the simple example, which demonstrates that this definition can flexibly handle general schemes

In the following lemma, we show that if a set of vertices C in $\mathcal{G}_\Pi(\widetilde{\mathtt{D}}_t)$ minimally recovers some other vertex v_j, then there is a collection of paths from the vertices of C to v_j in $\mathcal{G}_\Pi(\widetilde{\mathtt{D}}_t)$ such that every vertex on the path (except for the vertices of C) is minimally recovered by its predecessors on these paths.

Lemma 1. *For any FS eRAM protocol Π, for every $t \geq 0$, for every sequence of data cell updates $\widetilde{\mathtt{D}}_t$ performed by $\mathcal{A}$ and for every useful key or data $v_j \in \mathcal{V}$ and (non-empty) set $\mathsf{C} \subseteq \mathcal{V}$, if C minimally recovers v_j, then there exists a collection of paths $\mathcal{P}$ from all $v_i \in \mathsf{C}$ to v_j (at least one per individual vertex v_i) in $\mathcal{G}_\Pi(\widetilde{\mathtt{D}}_t)$*

such that for every v along these paths, except for the $v_i \in \mathsf{C}$, the set of strings $\{v_1, \ldots, v_\ell\} \subseteq \mathtt{UsefulKeys}_{\Pi}(\widetilde{\mathtt{D}}_t) \cup \mathtt{Sec}_t$ that have an edge to v on one of the paths minimally recover v.

Proof. Consider some C and v_j satisfying the conditions in the lemma statement. Let $q \geq 1$ be the smallest number of applications of the entailment relation $\vdash$ required to recover v_j from C and $\mathtt{Pub}(\widetilde{\mathtt{D}}_t)$. We prove the lemma using induction over q. I.e., for all $q \geq 1$ and every C that can minimally recover v_j in q steps, there is a collection of paths $\mathcal{P}$ originating from the v_i in C and ending at v_j in $\mathcal{G}_{\Pi}(\widetilde{\mathtt{D}}_t)$ satisfying the lemma statement.

The statement is true for $q = 1$ since in this case $\mathsf{C} = \{v_j\}$ and so there is a trivial path from v_j to itself. Now, suppose that the statement is true for all values of q smaller than an integer $Q > 1$. So, for all $q < Q$, every set $\mathsf{C} \subseteq \mathcal{V}$ that can minimally recover v_j in q steps has a collection of paths in $\mathcal{G}_{\Pi}(\widetilde{\mathtt{D}}_t)$ leading from every $v_i \in \mathsf{C}$ to v_j and for every vertex v along the paths, the vertices from which it has incoming edges along the paths form a set which minimally recovers v.

Consider any set $\mathsf{C} \subseteq \mathcal{V}$ that can minimally recover v_j in Q steps. It must be that there exists some set C' such that 1. C' can minimally recover v_j in less than Q steps, and 2. for each $c' \in \mathsf{C}'$, either

(i) there exists some $k \in \mathsf{C}$ and some string c such that $\mathbf{F}(k, c) = c'$ and k minimally recovers c', or
(ii) there exists keys $k_1, k_2 \in \mathtt{UsefulKeys}_{\Pi}(\widetilde{\mathtt{D}}_t)$ such that (a) $k_1 \in \mathsf{C}$, (b) $k_2 \in \mathsf{C}$, or (c) both $k_1, k_2 \in \mathsf{C}$ and $\mathbf{dF}(k_1, k_2) = c'$, where in cases (a), (b), (c), just k_1, just k_2, or $\{k_1, k_2\}$ minimally recover c', respectively, or
(iii) c' is a key or data and there exists sets $\mathsf{C}'' \subseteq \mathtt{Pub}(\widetilde{\mathtt{D}}_t)$ and $\mathsf{C}''' \subseteq \mathtt{Sec}_t \cap \mathsf{C}$ (each possibly empty) whose strings encapsulate c' and $\mathsf{C}_e \subseteq \mathsf{C}$, a set of useful encryption keys used in the generation of these strings, such that $\mathsf{C}''' \cup \mathsf{C}_e$ minimally recovers c',[8] or
(iv) $c' \in \mathsf{C}$.

Moreover, for any string $c' \in \mathsf{C}'$ satisfying case (i), it must be $c' \in \mathtt{UsefulKeys}_{\Pi}(\widetilde{\mathtt{D}}_t)$ because $c' \in Rec(\{k\}) \subseteq Rec(\mathtt{Sec}_t \cup \mathtt{Pub}(\widetilde{\mathtt{D}}_t))$, since $k \in \mathtt{UsefulKeys}_{\Pi}(\widetilde{\mathtt{D}}_t)$. For any string satisfying case (ii), it must be that $c' \in \mathtt{UsefulKeys}_{\Pi}(\widetilde{\mathtt{D}}_t)$ because $c' \in Rec(\{k_1, k_2\}) \subseteq Rec(\mathtt{Sec}_t \cup \mathtt{Pub}(\widetilde{\mathtt{D}}_t))$, since $k_1, k_2 \in \mathtt{UsefulKeys}_{\Pi}(\widetilde{\mathtt{D}}_t)$. For any key or data $c' \in \mathsf{C}'$ satisfying case (iii), it must be that $c' \in \mathtt{UsefulKeys}_{\Pi}(\widetilde{\mathtt{D}}_t) \cup \mathtt{D}_t$ because: If c' is a key, we have that $c' \in Rec(\mathsf{C}_e \cup \mathsf{C}''' \cup \mathtt{Pub}(\widetilde{\mathtt{D}}_t)) \subseteq Rec(\mathtt{Sec}_t \cup \mathtt{Pub}(\widetilde{\mathtt{D}}_t))$, since $\mathsf{C}_e \subseteq \mathtt{UsefulKeys}_{\Pi}(\widetilde{\mathtt{D}}_t)$ and $\mathsf{C}''' \subseteq \mathtt{Sec}_t$. If c' is data, we must have that $v_j \in \mathcal{V}$ so $c' = v_j \in \mathtt{D}_t$, for otherwise, since data cells have no outgoing edges, C' does not minimally recover v_j. Thus, we have shown that $\mathsf{C}' \subseteq \mathcal{V}$.

We also note that we can choose to consider only C' that contain c' corresponding to case (iii) that are the keys or data which are encapsulated by C''

[8] This is the case in which c' is recovered through secret share reconstruction (using $\mathsf{C}'' \cup \mathsf{C}'''$), decryption (using keys of C_e and possibly some that can be recovered using only $\mathtt{Pub}(\widetilde{\mathtt{D}}_t)$, along with $\mathsf{C}'' \cup \mathsf{C}'''$), or possibly a combination of both.

and C''' (and not some intermediary strings $c_{\texttt{bad}}$ encapsulating c' in C', which could be in $\texttt{Sec}_t$, for example) because v_j in the lemma statement is a useful key or data. Therefore the encapsulated c' must either be v_j itself or some key that must be used to minimally recover v_j. Thus, if C contains enough keys which were used in the generation of the strings of $\mathsf{C}'' \cup \mathsf{C}'''$ that (in addition to the cells of C''') can minimally recover c' anyway, then we can just directly include c' in C'. If C does not contain enough of those keys, then we can just choose C' that lessens the number of applications of the entailment relation needed to recover a sufficient set of those keys, and defer inclusion of c' until some later set.

Continuing, by the inductive hypothesis, there are a collection of paths from the $c' \in \mathsf{C}'$ satisfying the lemma statement, and prepending the paths with:

- in case (i), the edge $k \to c'$,
- in case (ii,a), the edge $k_1 \to c'$, in case (ii,b) the edge $k_2 \to c'$, in case (ii,c) the edges $k_1 \to c'$ and $k_2 \to c'$,
- in case (iii), the edges $k \to c'$, for each $k \in \mathsf{C}_e$, and the edges $c_{\texttt{Sec}} \to c'$, for each $c_{\texttt{Sec}} \in \mathsf{C}'''$, and
- in case (iv), nothing,

extends these paths appropriately. Indeed, by the case analysis above, for each $c' \in \mathsf{C}'$ the nodes in C corresponding to its incoming edges on the paths minimally recover c'. Furthermore, if any nodes $c \in \mathsf{C}$ do not have any edges to nodes in C', then since C' minimally recovers v_j and all of the nodes in C' can be recovered by the other nodes of C, C does not minimally recover v_j, a contradiction. Thus we have shown the inductive step holds for all $Q > 1$. □

Now we show a quick proposition which establishes that at each time t, any data $d_i \in \texttt{D}_t$ cannot be solely recovered by the contents of the public cells up to time t and thus there must exist some non-empty set of strings in the secret cells that minimally recovers d_i:

Proposition 1. *For any $d_i \in \texttt{D}_t$, it must be that $d_i \notin Rec(\texttt{Pub}(\widetilde{\texttt{D}}_t))$ and thus $\exists S_i \subseteq \texttt{Sec}_t$ such that $S_i \neq \emptyset$ and S_i minimally recovers d_i.*

Proof. The proof for this is quite basic: if $d_i \in Rec(\texttt{Pub}(\widetilde{\texttt{D}}_t))$, then if the adverary's next operation is write(i, d) for some data d, then at time $t+1$, for example, it is clear that $d_i \in Prev(\texttt{D}_{t+1})$ and also that $d_i \in Rec(\texttt{Pub}(\widetilde{\texttt{D}}_{t+1})) \subseteq Rec(\texttt{Sec}_{t+1} \cup \texttt{Pub}(\widetilde{\texttt{D}}_{t+1}))$, which violates security. Thus, a contradiction is reached.

Now, by correctness, it must be that $d_i \in Rec(\texttt{Sec}_t \cup \texttt{Pub}_t)$, so with the above, it must be that there exists some non-empty $S_i \subseteq \texttt{Sec}_t$ such that S_i minimally recovers d_i. □

As a result of this proposition, we have that for each $d_i \in \texttt{D}_t$ at time t, there exists *at least one* collection of paths $\mathcal{P}_i^t$ from the $s_j \in S_i$ to d_i satisfying the conditions of Lemma 1. In the example of Figure 3, the collection of paths $\mathcal{P}_3^{t-1} = \{P_{3,1}, P_{2,1}\}$, where $P_{3,1} \coloneqq k_1 \to k_2 \to d_3$ and $P_{3,2} \coloneqq k_1 \to k_3 \to d_3$, represents one such collection of paths that must exist as a result of Lemma 1, since $\{k_1\}$ minimally recovers d_3. Observe, however, that in general, there could be multiple such path collections (even from the same $S_i \subsetneq \texttt{Sec}_t$) for each d_i (for example, if in Figure 3 we also have $\mathbf{E}_{k_1}(d_3) \in \texttt{Pub}(\widetilde{\texttt{D}}_{t+1})$, then we also have the collection consisting of just $k_1 \to d_3$); we cannot restrict protocols to just one collection, even though having more does not seem fruitful. So, for each $i \in [n]$, there may exist r_i non-empty secret cell subsets $S_{i,j} \subseteq \texttt{Sec}_t, j \in [r_i]$ that minimally recover d_i, and $r'_i \geq r_i$ corresponding path collections $\mathcal{P}^t_{i,l}, l \in [r'_i]$, which therefore must exist by Lemma 1.

We also note that by the proof of Lemma 1, every string stored in a secret cell $c_{\texttt{Sec}} \in \texttt{Sec}_t$ that exists on these paths and is not a key or data has no incoming edges on the paths from other path nodes.

We now show how any FS eRAM protocol Π must handle such collections of paths at every time t in order to preserve security.

Lemma 2. *If at time t, an adversary $\mathcal{A}$ executes* write(i, d'_i) *for $d'_i \neq \texttt{D}_{t-1}[i]$, then, for every collection of paths $\mathcal{P}^{t-1}_{i,l}$, $l \in [r'_i]$, from some non-empty $S_{i,j} \subseteq \texttt{Sec}_{t-1}$ that minimally recovers $\texttt{D}_{t-1}[i]$, all of the vertices (data, keys, and strings in secret cells) on at least one of the paths $P^*_i \in \mathcal{P}^{t-1}_{i,l}$, must become useless for all times $t' \geq t$.*

Proof. Since $\mathcal{A}$ queries write(i, d'_i), Π must alter the contents of the secret cells and the partition of useful and useless keys so that $d_i \notin Rec(\texttt{Sec}_{t'} \cup \texttt{Pub}(\widetilde{\texttt{D}}_{t'}))$ for all $t' \geq t$, where d_i was the string stored in data cell i at time $t-1$ (i.e., what d'_i replaces at time t).

Now, consider one such collection of paths $\mathcal{P}^{t-1}_{i,l}$. If $d_i \in \texttt{Sec}_{t-1}$, i.e., $\mathcal{P}^{t-1}_{i,l}$ only consists of the trivial path from d_i to itself, then, looking ahead to Section 3.3, the adversary will never again write d_i to $\texttt{D}$, and thus d_i becomes useless for all $t' \geq t$. Otherwise, we will proceed inductively starting from the sink of all of these paths, d_i (and ending at a node right after the source of one of them). Let V_i and E_i be the set of vertices and edges in all of the paths of $\mathcal{P}^{t-1}_{i,l}$, respectively. We know from Lemma 1 that the set $R := \{v : v \in V_i, (v, d_i) \in E_i\}$, indeed minimally recovers d_i. Thus, for security, since we must have that $d_i \notin Rec(\texttt{Sec}_{t'} \cup \texttt{Pub}(\widetilde{\texttt{D}}_{t'}))$, it must be that $\exists v^* \in R$ such that $v^* \notin Rec(\texttt{Sec}_{t'} \cup \texttt{Pub}(\widetilde{\texttt{D}}_{t'}))$ for all $t' \geq t$, i.e., $v^* \notin \texttt{UsefulKeys}_{\Pi}(\widetilde{\texttt{D}}_{t'}) \cup \texttt{Sec}_{t'}$. So, if v^* was in $\texttt{UsefulKeys}_{\Pi}(\widetilde{\texttt{D}}_{t-1})$ or $\texttt{Sec}_{t-1}$, it can no longer be useful at each time t' (it is deleted at time t and can never be generated again, by definition of the symbolic model). We must then recurse on v^*, continuing until we reach some source in $\mathcal{P}^{t-1}_{i,l}$, i.e., some key or string $c^* \in \texttt{Sec}_{t-1}$, that must become useless.

Therefore, we conclude that all of the vertices on at least one of the paths $P^*_i \in \mathcal{P}^{t-1}_{i,l}$ must become indefinitely useless at time t. □

It is important to note that the protocol Π has control over which path P^*_i in each $\mathcal{P}^{t-1}_{i,l}$ must have all of its nodes become useless at time t (and there could in fact be overlap between collections, even for the same i). For example in Figure 3, it may be that either of $P_{3,1}$ or $P_{3,2}$ have all of their nodes become useless at time t. We depict in that figure that the path which Π chooses is $P_{3,2}$.

The rest of the lower bound proof will proceed by considering for each t and $i \in [n]$, an arbitrary collection of paths $\mathcal{P}_i^t$ and for each such collection, the path P_i^* that minimizes the sum of the number of outgoing edges of type 3 or 4 in Definition 9 from its vertices. We do so to focus on the minimum number of strings that are actually stored in public and secret cells, corresponding to tangible actions that Π took in the past to put them there, that Π must make useless at each time t. In particular, we remove all edges (such as those for (d)PRFs) that may not require such storage. We do this by, in Definition 11, first choosing an arbitrary collection $\mathcal{P}_i^t$ for each $i \in [n]$ and considering only the edges on those paths, then restricting these paths to only edges of type 3 or 4, and finally further restricting to only those edges on path P_i^* for each $i \in [n]$. First, a helpful definition:

Definition 10. *Let $G = (\mathcal{V}, \mathcal{E})$ be a directed graph and let P be some path in G. The* out-degree *of P in G, denoted $out_G(P)$, is the number of edges in $\mathcal{E}$ that start from a node in P. That is,*

$$out_G(P) = \sum_{v \in P} out_G(v).$$

Definition 11. *For any FS eRAM protocol Π, executed with a sequence of data cell updates $\widetilde{\mathtt{D}}_t$, the* succinct key-data graph *at time t, denoted $\hat{\mathcal{G}}_\Pi(\widetilde{\mathtt{D}}_t)$, is a modification of $\mathcal{G}_\Pi(\widetilde{\mathtt{D}}_t)$ formed by*

- *first, choosing for each $i \in [n]$ an arbitrary collection of paths $\mathcal{P}_i^t$ in $\mathcal{G}_\Pi(\widetilde{\mathtt{D}}_t)$ specified by Lemma 1 and which must exist by Proposition 1, starting from the vertices of some arbitrary non-empty $S_i \in \mathtt{Sec}_t$ that minimally recovers d_i and ending at d_i,*
- *second, including an edge from $\mathcal{G}_\Pi(\widetilde{\mathtt{D}}_t)$ if and only if it is contained in the collection of paths $\mathcal{P}_i^t$ for some $i \in [n]$,*
- *third, for each $i \in [n], l \in [\ell_i]$, where ℓ_i the number of paths in $\mathcal{P}_i^t$, for any non-empty sequence of edges $(e_1, \ldots, e_m)$ on path $P_{i,l} \in \mathcal{P}_i^t$ such that each $e_j = (v_{j-1}, v_j)$ for $j \in [m-1]$ is a (d)PRF edge of type 1 or 2, $e_m = (v_{m-1}, v_m)$ is not a (d)PRF edge, and v_0 does not have any incoming (d)PRF edges on $P_{i,l}$, removing all of the edges in the sequence and replacing them with the single edge $e = (v_0, v_m)$ for only $P_{i,l}$,*
- *fourth, taking the union of these edges for all $i \in [n]$, and finally*
- *including an edge if and only if it is contained in the path P_i^* for some $i \in [n]$ such that $out(P_i^*) = \min_{P_{i,l} \in \mathcal{P}_i^t} out(P_{i,l})$, where the outdegree of the nodes is over only the edges from the last step.*

One can observe that vertices representing data cells are of course sinks, i.e., they have no outgoing edges, in $\mathcal{G}_\Pi(\widetilde{\mathtt{D}}_t)$ and $\hat{\mathcal{G}}_\Pi(\widetilde{\mathtt{D}}_t)$. In addition, at most s secret cells can be represented in $\mathcal{G}_\Pi(\widetilde{\mathtt{D}}_t)$ and $\hat{\mathcal{G}}_\Pi(\widetilde{\mathtt{D}}_t)$, by definition. Furthermore, if a data cell d_i is not itself in $\mathtt{Sec}_t$ at time t, by Proposition 1, it must be that some non-empty subset $S_i \subseteq \mathtt{Sec}_t$ minimally recovers d_i, and so there must be some collection of paths $\mathcal{P}_i^t$ in $\mathcal{G}_\Pi(\widetilde{\mathtt{D}}_t)$ starting at the secret cells of S_i and ending in edges of type 3 or 4 at d_i. Thus, there must still be one path in $\hat{\mathcal{G}}_\Pi(\widetilde{\mathtt{D}}_t)$ starting at some secret cell $c_{\mathtt{Sec}} \in \mathtt{Sec}_t$ and ending at d_i. We will call such paths $\widehat{P}_i^*$.

3.3 Adversarial Strategy

We now describe the strategy used by the adversary to prove the lower bound. The adversary will use a simple non-adaptive randomized strategy. Initially, the adversary inputs n distinct strings of type D to be stored in the data cells: $\mathtt{D}_0 = (d_1, \ldots, d_n)$ to the init algorithm. At each instant t, the adversary will pick a cell i uniformly at random to write data d that has never before been in $\mathtt{D}_{t'}$ for $0 \le t' \le t$, i.e. write(i, d). We will show that in expectation, the number of strings stored in secret and public cells combined that become *useless* after time t is logarithmically-many (in terms of n/s). Before proving Theorem 6, we provide an instrumental graph-theoretic lemma in proving our lower bound. We prove this lemma in the full version [5].

Lemma 3. *Let $G = (\mathcal{V}, \mathcal{E})$ be an arbitrary graph and let $S = \{u_1, \ldots, u_s\}$, $D = \{v_1, \ldots, v_n\}$ be any sets of nodes in the graph such that for each $v_i \in D$, $\exists u_j \in S$ such that there is a path P_i from u_j to v_i and there is no $k \neq i$ such that v_k occurs in P_i. Then if $i \in [n]$ is chosen uniformly at random:*

$$\mathbb{E}[out_G(P_i)] \ge \log_2(n/s).$$

Proof (Theorem 6). In the setting of succinct key-data graphs, sets S and D in Lemma 3 correspond to the secret cells and data cells, respectively, at time t, i.e., $\mathtt{Sec}_t$ and $\mathtt{Pub}_t$. Each path $P_i = \widehat{P}_i^*$ corresponds to the path starting from some secret cell u_j to the data cell v_i that has the minimum number of outgoing edges over all $P_{i,l} \in \mathcal{P}_i^t$, for some arbitrary $\mathcal{P}_i^t$, which correspond to strings that must be stored in the public or secret cells of Π. Since no data cells have outgoing edges, for every $i \in [n]$, no data cell v_k, $k \neq i$ lies on the path $\widehat{P}_i^*$. Thus, Lemma 3 implies that for every $t \ge 1$, the data cell v_i replaced by $\mathcal{A}$ at time t is such that in expectation, $\widehat{P}_i^*$ will always have out-degree at least $\log_2(n/s)$ in $\hat{\mathcal{G}}_\Pi(\widetilde{\mathtt{D}}_t)$. Since $\widehat{P}_i^*$ corresponds to the path from a secret cell to d_i which minimizes the number of outgoing edges corresponding to strings that must be stored in the public or secret cells of Π, any other path that Π chooses whose nodes must become useless after time t (by Lemma 2) must have at least $\log_2(n/s)$ such edges in expectation.

Once all of the vertices on some such path become useless, all strings in public or secret cells that were in part generated by them become useless by definition. For example, in Figure 3, since the vertices on $P_{3,2}$ become useless, each string $\mathbf{E}_{k_1}(k_2), \mathbf{E}_{k_1}(k_3), \mathbf{E}_{k_2}(\mathbf{E}_{k_3}(d_3)), \mathbf{E}_{k_3}(d_4)$ becomes useless. So, under the above adversarial strategy, at least $\log_2(n/s)$ strings stored in public or secret cells become useless after time t, in expectation. We only mark each string as useless once because:

- if a key in the set, which minimally recovers the key or data that the string encapsulates, using that string, is indefinitely inaccessible by the protocol, the protocol is indefinitely unable to recover the key or data using the string (and also the string can never be reproduced, by definition of the symbolic model),

– otherwise, if the string was stored in a secret cell on the path P_i^*, then from Lemma 2, we know that it must become useless indefinitely and therefore never be stored in a secret cell again.

Therefore, in both cases, there will never be any edge in any future succinct key-data graph that corresponds to the string. Moreover, since v_i is never added back to a data cell by the adversary, the strings remain useless for all $t' \geq t$.

Therefore, we have shown that throughout the execution of any protocol Π, at least $\frac{t}{t+1} \cdot \log_2(n/s) \geq (1 - \frac{1}{t}) \cdot \log_2(n/s) = (1 - o(1)) \cdot \log_2(n/s)$ strings which were stored in public or secret cells become useless in expectation through time t, proving Theorem 6. ■

4 Stronger Forward Secret Encrypted RAM Definitions

We note that our FS eRAM computational lower bound in Section 3 uses a different, weaker definition in our symbolic model based on recoverability (which only makes our lower bound stronger). For our upper bounds, we will use the typical indistinguishability-based security that we define here in the standard model of computation.

Definition 12 (Forward Secret Encrypted RAM). *A* Forward Secret Encrypted RAM (FS eRAM) *protocol* $\Pi = (\text{init}, \text{read}, \text{write})$ *consists of the following algorithms:*

– $(\texttt{Sec}, \texttt{Pub}) \leftarrow \text{init}(1^\lambda, s, n)$*, which initializes public cells* `Pub` *and* s *secret cells* `Sec`*, where* n *is the number of virtual cells and* λ *is the security parameter.*
– $(\texttt{Sec}', \texttt{Pub}', d) \leftarrow \text{read}(\texttt{Sec}, \texttt{Pub}, i)$*, which returns data* d *of virtual cell* $i \in [n]$*.*
– $(\texttt{Sec}', \texttt{Pub}') \leftarrow \text{write}(\texttt{Sec}, \texttt{Pub}, d, i)$*, which replaces the contents of virtual cell* $i \in [n]$ *with data* d*.*

Correctness. An FS eRAM scheme is *correct* if for any sequence of operations: $\Pr[(\cdot, \cdot, d^*) \leftarrow \text{read}(\texttt{Sec}, \texttt{Pub}, i) : d^* = d] = 1$, for any execution of $\text{read}(\texttt{Sec}, \texttt{Pub}, i)$ in the sequence after an execution of $\text{write}(\texttt{Sec}, \texttt{Pub}, d, i)$ and before a subsequent execution of $\text{write}(\texttt{Sec}, \texttt{Pub}, d', i)$, for some data d', in the sequence where the probability is over the random coin tosses of the protocol.

Security. We define security with respect to the following game between a challenger and an adversary. We emphasize that the adversary has read-access to *all* of `Pub` (which is usually encrypted) on which the FS eRAM operates, as well as the access pattern to cells of `Pub` during operations.

The challenger initially chooses $b \in \{0, 1\}$ uniformly at random and runs $(\texttt{Sec}, \texttt{Pub}) \leftarrow \text{init}(1^\lambda, s, n)$ (where $s \ll n = \texttt{poly}(\lambda)$). Then, the adversary has access to (polynomial-many queries of) the following oracles:

– **write**(d, i), which computes $\text{write}(\texttt{Sec}, \texttt{Pub}, d, i)$.
– **corrupt**$()$, which simply returns `Sec`.
– **chall**(d_0, d_1, i), which computes $\text{write}(\texttt{Sec}, \texttt{Pub}, d_b, i)$.

An adversary is not allowed to call **corrupt**() after a call to **chall**(d_0, d_1, i), without first using **write**(d, i) to overwrite the i-th virtual cell with some other data d, since otherwise they would trivially win. Observe that w.l.o.g. there is no oracle for read() since the adversary already knows the data in cells which they filled using **write**(), and should not know the data in cells filled via **chall**(). Further observe that the following definition provides forward secrecy, since upon some legal **corrupt**() call after some **chall**(d_0, d_1, i) call, the adversary $\mathcal{A}$ should not be able to guess the bit b, i.e. whether d_0 or d_1 was stored in virtual cell i.

Definition 13 (Forward Secret Encrypted RAM Security). *A Forward Secret Encrypted RAM protocol Π is* secure *if for every adversary $\mathcal{A}$ that runs in time* $\texttt{poly}(\lambda)$*:* $|\Pr[\mathcal{A} \to 1 | b = 1] - \Pr[\mathcal{A} \to 1 | b = 0]| \leq \texttt{negl}(\lambda)$.

Recall the folklore construction of Figure 1. It is clear that this construction is secure with respect to Definition 13 due to standard IND-CPA security of symmetric encryption. Once the adversary queries **write**(d, i) after a **chall**(d_0, d_1, i) query, all of the keys on the path of cell i, including those in `Sec`, are refreshed, and encryptions of the children of the path nodes are recomputed. Thus no information about the challenge bit b can be garnered from a **corrupt**() query.

5 Oblivious Forward Secret Encrypted RAM

In this section, we consider combining the notion of forward secret encrypted RAMs with *oblivious RAMs* (or ORAMs). ORAMs are a well-studied cryptographic primitive (see [17,36,18,42,32,1] and references therein) that provides security for the patterns of data access. At a high level, ORAMs guarantee that adversaries may not distinguish between two equal-length operational sequences even when viewing accesses to encrypted data. We note that ORAMs do not consider the setting where adversaries corrupt client storage.

Looking forward, we first formally define oblivious forward secret encrypted RAMs. We present both a strong and a weak notion combining obliviousness and forward secrecy. First, we present a linear cell probe lower bound for the strong variant. To obtain sub-linear overhead, we consider the weaker notion and present an optimal construction with logarithmic overhead. As a result, we show that one can add a weaker notion of obliviousness to forward secret encrypted RAMs without asymptotic overhead.

5.1 Definitions

The syntax of ORAMs are identical to the syntax of FS eRAMs presented in Definition 12 where secret cells `Sec` are client storage and public cells `Pub` are server storage. Therefore, we omit a formal notion of ORAM syntax and refer readers back to FS eRAM syntax if needed.

We start by defining the most natural notion of oblivious forward secret encrypted RAMs. When the adversary corrupts client storage and views the current memory contents, the adversary should not be able to distinguish between any two equal-length operational sequences that result in the current memory contents. At a high level, this notion provides forward secrecy for both the memory contents as well as the access patterns performed by the client prior to corruption. We denote this notion as *strong* oblivious forward secret encrypted RAMs.

Definition 14 (Strong Oblivious Forward Secret Encrypted RAM). *Consider any two equal-length sequences of read and write operations O_1 and O_2 such that the contents of the array after both sequences are executed are identical. Let $\mathcal{V}(O)$ be the adversary's view when executing sequence O that includes contents of server (public) cells, all accesses to server cells and corrupted secret cells after O is executed. For any pair of such sequences O_1 and O_2 and any PPT adversary $\mathcal{A}$, a protocol Π is a strong oblivious forward secret encrypted RAM if* $|\Pr[\mathcal{A}(\mathcal{V}(O_1)) = 1] - \Pr[\mathcal{A}(\mathcal{V}(O_2)) = 1]| \leq \mathsf{negl}(\lambda)$.

We show in the full version [5] that this strong notion requires linear overhead in the cell probe model. As a result, we need to consider a weaker security notion to obtain reasonable (or even sub-linear) overhead. Another natural composition of forward secret encrypted RAMs and oblivious RAMs is to trivially combine the two security notions together. If we consider an adversary that never corrupts client storage, then the access pattern to data remains secure (identical to ORAM guarantees). When the adversary corrupts client storage, all deleted memory contents are not recoverable by the adversary (identical to forward secret guarantees). In the case of client corruption, no security guarantees are offered about the client's access patterns prior to corruption. We denote this security as *weak* oblivious forward secret encrypted RAMs. As this is the notion that enables interesting sub-linear constructions, we will also refer to this notion as simply oblivious forward secret encrypted RAMs. We start by defining oblivious RAM:

Definition 15 (Oblivious RAM). *Consider any two equal-length sequences of read and write operations O_1 and O_2. Let $\mathcal{V}(O)$ be the adversary's view when executing sequence O that includes contents of server (public) cells and all accesses to server cells. For any pair of such sequences O_1 and O_2 and any PPT adversary $\mathcal{A}$, a protocol Π is an oblivious RAM if* $|\Pr[\mathcal{A}(\mathcal{V}(O_1)) = 1] - \Pr[\mathcal{A}(\mathcal{V}(O_2)) = 1]| \leq \mathsf{negl}(\lambda)$.

Definition 16 ((Weak) Oblivious Forward Secret Encrypted RAM). *A protocol Π is a (weak) oblivious forward secret encrypted RAM if Π is both a forward secret encrypted RAM and an oblivious RAM.*

5.2 Oblivious Forward Secret Encrypted RAM Construction

We start by presenting a naive composition of oblivious RAM and forward secret encrypted RAM constructions. Throughout this section, we will measure overhead with respect to encrypted array entries. For example, $O(\log n)$ communication means $O(\log n)$ encrypted array entries. We make the natural assumption that cell size is $\Omega(\log n)$ bits, and also array entries are $O(\log n)$ bits. The idea is to take any forward secret encrypted RAM and replace each memory access using an oblivious RAM. While this guarantees both obliviousness and forward secrecy, the efficiency is not optimal. Note that forward secret encrypted RAMs use $O(\log n)$ memory accesses. Each memory access in an ORAM costs $O(\log n)$ overhead incurring a total $O(\log^2 n)$ overhead. We note that prior works have studied this primitive such as [39]. To our knowledge, the best current construction requires $O(\log^2 n)$ overhead.[9]

Our construction utilizes two observations. First, tree-based ORAMs are quite conducive to incorporate the folklore FS eRAM solution. However, all tree-based ORAMs [42] require $O(\log^2 n)$ overhead. On the other hand, hierarchical ORAMs [17,36,32,1] obtain $O(\log n)$ overhead but there is no straightforward way to incorporate the folklore FS eRAM solution. To obtain our result, we compose tree-based and hierarchical ORAMs to obtain a faster solution. At a high level, we use tree-based ORAMs and replace the recursive position map with a hierarchical ORAMs. We describe our new constructions below.

Overview. Our construction will avoid this additional logarithmic overhead incurred by ORAM over forward secret encrypted RAMs. Without loss of generality, suppose we are storing n array entries $\mathtt{D}[0], \ldots, \mathtt{D}[n-1]$ where n is a power of two. Our construction uses three components: a complete binary tree, a stash and an oblivious RAM. The binary tree is inspired by prior works for tree-based ORAMs [42]. The tree will have n leaf nodes and $\log n$ levels used to store the n array entries. Every node in the binary tree has capacity to store up to a constant number of array entries. Each of the n array entries, $\mathtt{D}[i]$, will be uniquely assigned a uniformly random leaf node of the tree denoted by $\mathsf{Leaf}(i)$. The tree maintains the invariant that if any array entry $\mathtt{D}[i]$ is stored in the binary tree, then $\mathtt{D}[i]$ will be stored in a node that appears on the unique root-to-leaf path for leaf $\mathsf{Leaf}(i)$. If $\mathtt{D}[i]$ is not stored in the tree, then it will be stored (along with $\mathsf{Leaf}(i)$) in the *stash* denoted by Stash. Additionally, we need to maintain a *position map*, PMAP, that stores the assigned leaf nodes for each array entry, that is, $\mathsf{PMAP}[i] = \mathsf{Leaf}(i)$.

Binary Tree. Whenever an array entry $\mathtt{D}[i]$ is either read or overwritten, the root-to-leaf path to $\mathsf{Leaf}(i)$ will be accessed along with the stash Stash to obtain $\mathtt{D}[i]$. Afterwards, $\mathsf{Leaf}(i)$ is re-initialized by picking amongst the n leaf nodes uniformly at random with PMAP being updated accordingly. Finally, $\mathtt{D}[i]$ is stored in Stash with its updated $\mathsf{Leaf}(i)$. To ensure Stash remains small, entries in Stash are evicted in a greedy manner whenever a root-to-leaf path is accessed. If there is space in the root node, any item in Stash may be evicted into the root node (as the root appears on every root-to-leaf node path). Generally, for any node accessed in a root-to-leaf path, any data entry $\mathtt{D}[i]$ whose leaf node $\mathsf{Leaf}(i)$ appears in the sub-tree rooted at the node may be evicted until reaching the node's capacity. Prior works proved that the Stash remains small except with negligible probability. Formally, Stash contains at most $\omega(\log n)$ items except with proba-

[9] To be fair, we note that these works appeared before recent developments leading to $O(\log n)$ overhead ORAMs [32,1]

bility negligible in n. Additionally, it has been showing that accessing the tree is oblivious.

To obtain forward secrecy, we embed the folklore FS eRAM ideas into the binary tree. Each internal node in the binary tree will additionally store a random encryption key that will be used to encrypt the contents of both children. Each time a root-to-leaf path is accessed, all children of nodes in the root-to-leaf path will also be accessed. All encryption keys of nodes in the path will be re-generated. Furthermore, all nodes will be re-encrypted using their parent's new encryption key. The newly re-encrypted nodes will be uploaded back to the server for storage. This modification guarantees forward secrecy for the data.

Position Map and Stash. Next, we consider the position map PMAP. Note that PMAP only stores relationships between entries and leaf nodes. In particular, PMAP does not store any information about the array entry contents. As a result, we only need to focus on obliviousness for PMAP. We choose to store PMAP in any oblivious RAM. If we choose the construction in [1], reading or overwriting any entry $\mathsf{PMAP}[i]$ requires only $O(\log n)$ overhead as PMAP contains only n entries. Finally, the Stash is handled by encrypting the array entries $\mathtt{D}[i]$ that it contains, along with their corresponding leaves $\mathsf{Leaf}(i)$, and storing the ciphertexts on the server. Making the natural assumption that the cell size is $\Omega(\log n)$ bits and can fit memory addresses, we can encode $\mathsf{Leaf}(i)$ using $O(\log n)$ bits, and thus additionally storing $\mathsf{Leaf}(i)$ does not incur any extra overhead.

Read and Write Algorithms. Altogether, a read or write to the oblivious forward secure encrypted RAM works as follows. To read/overwrite $\mathtt{D}[i]$, the leaf node $\mathsf{Leaf}(i)$ is queried from PMAP using ORAM operations. Next, the root-to-leaf path to $\mathsf{Leaf}(i)$, children of nodes in the root-to-leaf path and Stash are downloaded and decrypted. $\mathtt{D}[i]$ is then retrieved and updated if needed. A new uniformly random $\mathsf{Leaf}(i)$ is generated and written back to PMAP, and $\mathtt{D}[i]$ is placed into Stash with its new $\mathsf{Leaf}(i)$. Items are greedily evicted from Stash into the downloaded root-to-leaf path using their $\mathsf{Leaf}(i)$ as guidance. Each node is padded to the maximum capacity with dummies if needed. All encryption keys of internal nodes are freshly sampled and all nodes are encrypted using the parent node's encryption keys before being uploaded back to the server. The root is encrypted with a freshly generated client-stored encryption key. Finally, Stash is padded to the maximum capacity with dummies and re-encrypted using the new client key before being uploaded to the server.

Theorem 7. *For any function $f(n) = \omega(1)$, there exists a (weak) oblivious forward secret encrypted RAM with $O(\log n \cdot f(n))$ overhead, $O(n)$ server storage and $O(1)$ client storage.*

We provide a proof of this Theorem in the full version [5]. Our construction is essentially optimal except for the multiplicative $\omega(1)$ factor as we already proved that forward secure encrypted RAMs require $\Omega(\log n)$ overhead. Similarly, it is known that ORAMs also require $\Omega(\log n)$ overhead [17,26].

6 Forward Secret Memory Checkers

In this section, we combine forward secret encrypted RAMs with memory checkers (MCs) to get forward secret memory checkers (FS MCs). Memory checking is a well-studied cryptographic notion [6,31,11,15] that provides authenticity for outsourced data storage. Intuitively, MCs use some small local storage to guarantee that an adversarial server cannot alter outsourced data entries without the MC noticing (and outputting that a bug has occurred).

We will first define our combined notion of FS MCs then provide a scheme which overlays the folklore FS eRAM scheme with a tree-based MC from [6] that uses ideas from Merkle Trees [28]. As a result, we will achieve $O(1)$ secret storage, $O(n)$ remote storage, and $O(\log n)$ overhead, i.e., a scheme which provides memory checking with no additional asymptotic overhead to the folklore FS eRAM scheme.[10] Our construction is optimal with respect to both our $\Omega(\log n)$ lower bound on the overhead of FS eRAM and the best known $O(\log n)$ overhead construction for MCs (and almost optimal with respect to the $\Omega(\log n/\log\log n)$ MC lower bound of [15]).[11]

6.1 Forward Secret Memory Checker Definition

For FS MCs, we alter the syntax of FS eRAMs presented in Definition 12 to highlight the interaction between the FS MC and the potentially malicious remote server. Before reading the data from a virtual cell i, the FS MC must first receive the relevant (but possibly maliciously fabricated) public cells from the server. Additionally, to write to a cell i, the FS MC must first receive the relevant public cells from the server as above, then send new public cells to the server, which the FS MC expects to be written in place of the old public cells.

Definition 17 (Forward Secret Memory Checker). *A* Forward Secret Memory Checker (FS MC) $\Pi = (\text{init}, \text{retrieve}, \text{read}, \text{write}, \text{commit})$ *consists of the following algorithms:*[12]

- $(\mathtt{Sec}, \mathtt{Pub}) \leftarrow \text{init}(1^\lambda, n)$*, which initializes public cells* $\mathtt{Pub}$ *and secret cells* $\mathtt{Sec}$*, where n is the number of virtual cells and λ is the security parameter.*
- $S \leftarrow \text{index}(i)$*, which the server uses to identify* $S \subseteq \{1, \ldots, |\mathtt{Pub}|\}$*, a sparse subset of indices of the cells of* $\mathtt{Pub}$ *associated with virtual cell i. We refer to these cells of* $\mathtt{Pub}$ *as* $\mathtt{Pub}_{\text{index}(i)}$.
- $(\mathtt{Sec}', d) \leftarrow \text{read}(\mathtt{Sec}, \mathsf{C}, i)$*, which the FS MC uses to obtain the data d of virtual cell i, where in the case of an honest server,* C *is expected to be* $\mathtt{Pub}_{\text{index}(i)}$*. In the case that the FS MC wants to report a loss of integrity, it may output* $d \leftarrow \mathsf{bug}$.

[10] Although the solution of [38] informally provides the same guarantees with a similar construction, we provide a complete formal model and construction.

[11] For online MCs that access server memory deterministically and non-adaptively.

[12] While our definition is not fully general, i.e., does not allow for arbitrary interaction between the FS MC and server, it suffices for our optimal construction.

– $(\mathtt{Sec}', \mathsf{C}') \leftarrow \text{write}(\mathtt{Sec}, \mathsf{C}, d, i)$*, which the FS MC uses to replace the contents of virtual cell i with data d, where in the case of an honest server,* C *is expected to be* $\mathtt{Pub}_{\text{index}(i)}$*. The FS MC will provide the server with new public cells* C' *relevant to virtual cell i with which an honest server will replace cells* $\mathtt{Pub}_{\text{index}(i)}$*. In the case that the FS MC wants to report a loss of integrity, it may output* $\mathsf{C}' \leftarrow \mathsf{bug}$.

Correctness. An FS MC scheme is *correct* if for any sequence of operations executed by the FS MC and an honest server: $\Pr[(\cdot, d^*) \leftarrow \text{read}(\mathtt{Sec}, \mathtt{Pub}_{\text{index}(i)}, i) : d^* = d] = 1$, for any execution of $(\cdot, d^*) \leftarrow \text{read}(\mathtt{Sec}, \mathtt{Pub}_{\text{index}(i)}, i)$ in the sequence after an execution of $(\mathtt{Sec}', \mathsf{C}') \leftarrow \text{write}(\mathtt{Sec}, \mathtt{Pub}_{\text{index}(i)}, d, i)$, and before a subsequent execution of $(\mathtt{Sec}', \mathsf{C}') \leftarrow \text{write}(\mathtt{Sec}, \mathtt{Pub}_{\text{index}(i)}, d', i)$, where the probability is over the random coin tosses of the protocol.

Security. We now provide the security definition of FS MCs. The adversary in the game will adaptively specify the sequence of operations that the FS MC performs and have the ability to choose which cells C to provide to the FS MC challenger for read() and write() operations.

Throughout, the security game will store a dictionary $\mathtt{D}[\cdot]$ containing the correct data items at each index $i \in [n]$, corresponding to the most recent adversarial write or challenge to that index i. For every $i \in [n], \mathtt{D}[i] \leftarrow \perp$ initially. The challenger initially chooses $b \in \{0, 1\}$ uniformly at random, runs $(\mathtt{Sec}, \mathtt{Pub}) \leftarrow \text{init}(1^\lambda, n)$, sends $\mathtt{Pub}$ to the adversary $\mathcal{A}$, and deletes it. Then the adversary has access to (polynomial-many queries of) the following oracles:

– **read**(C, i): $\mathcal{A}$ sends public cells C for data cell i to the challenger who then computes $(\mathtt{Sec}', d) \leftarrow \text{read}(\mathtt{Sec}, \mathsf{C}, i)$. If $d \notin \{\mathtt{D}[i], \mathsf{bug}\}$, the game outputs win and ends. Note: d is not sent to $\mathcal{A}$.
– **write**(C, d, i): $\mathcal{A}$ sends public cells C and data d to overwrite virtual cell i with to the challenger who then computes $(\mathtt{Sec}', \mathsf{C}') \leftarrow \text{write}(\mathtt{Sec}, \mathsf{C}, d, i)$. The challenger then sends C' back to $\mathcal{A}$, and deletes it. Additionally, if $\mathsf{C}' \neq \mathsf{bug}$, the game sets $\mathtt{D}[i] \leftarrow d$.
– **chall**$(\mathsf{C}, d_0, d_1, i)$: $\mathcal{A}$ sends public cells C and data d_0, d_1 to overwrite data cell i with to the challenger who then computes $(\mathtt{Sec}', \mathsf{C}') \leftarrow \text{write}(\mathtt{Sec}, \mathsf{C}, d_b, i)$. The challenger then sends C' back to $\mathcal{A}$, and deletes it. Additionally, if $\mathsf{C}' \neq \mathsf{bug}$, the game sets $\mathtt{D}[i] \leftarrow d_b$.
– **corrupt**(): The challenger simply sends the contents of $\mathtt{Sec}$ to $\mathcal{A}$.

Finally, $\mathcal{A}$ outputs a bit b' and the game outputs win if and only if $b' = b$.

As in the FS eRAM security definition, $\mathcal{A}$ is not allowed to call **corrupt**() after a call to **chall**$(\mathsf{C}, d_0, d_1, i)$, without first a call to **write**(C, d, i) to overwrite the i-th virtual cell with some other data d, in which the challenger returns $\mathsf{C}' \neq \mathsf{bug}$, since otherwise they would trivially win.

Definition 18 (Forward Secret Memory Checker Security). *A Forward Secret Memory checker is* secure *if for every PPT adversary* $\mathcal{A}$, $\left|\Pr[\mathcal{A} \textit{ wins}] - \frac{1}{2}\right| \leq \mathtt{negl}(\lambda)$.

6.2 Forward Secret Memory Checker Construction

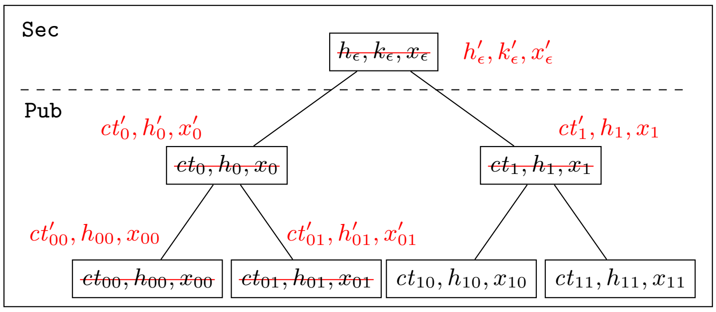

Fig. 4. Depiction of how our Forward Secret Memory Checker overwrites Sec and $\mathtt{Pub}_{\mathsf{index}(01)}$ after an execution of $\mathsf{write}(\mathtt{Sec}, \mathtt{Pub}_{\mathsf{index}(01)}, d'_{01}, 01)$. For every internal node v along the root-to-leaf path of leaf 01, as well as their siblings, the ciphertext $ct'_v = \mathbf{E}_{k'_p}(k'_v)$ is regenerated with the new keys k'_v, k'_p at v and its parent p, respectively (for siblings of path nodes, $k'_v = k_v$, the current key). For leaves 00 and 01, $ct'_{00} = \mathbf{E}_{k'_0}(d_{00}), ct'_{01} = \mathbf{E}_{k'_0}(d'_{01})$. For every node u on *only* the root-to-leaf path of leaf 01 (*not* their siblings), the hash function h'_u and corresponding hash value $x'_u = h'_u(ct'_{u.c_1}||h'_{u.c_1}||x'_{u.c_1}||ct'_{u.c_2}||h'_{u.c_2}||x'_{u.c_2})$ is regenerated, where one child $u.c_1$ or $u.c_2$, w.l.o.g., $u.c_1$, is not on the root-to-leaf path, and thus $h'_{u.c_1} = h_{u.c_1}, x'_{u.c_1} = x_{u.c_1}$ (their current values). Also, h_w and x_w are the all-zero string if w is a leaf.

Our FS MC construction is depicted in Figure 4. We overlay the folklore FS eRAM solution, which uses IND-CPA secure symmetric encryption for forward secrecy, with the MC solution of [6], which uses universal one-way hash functions (UOWHF) for integrity. Both utilize a binary tree with n leaf nodes to store the data elements in Pub, and store the root in Sec. Note: It is possible to construct a family of UOWHF given any one-way function [40], but the use of UOWHFs requires the assumption that the word size of the RAM is $\ell = n^\epsilon$, for any $\epsilon > 0$.

More specifically, each leaf node i holds $ct_i = \mathbf{E}_{k_p}(d_i)$, where k_p is the encryption key of the parent p of i. Each internal node v holds $(h_v, ct_v = \mathbf{E}_{k_p}(k_v), x_v = h_v(ct_{v.c_1}||h_{v.c_1}||x_{v.c_1}||ct_{v.c_2}||h_{v.c_2}||x_{v.c_2}))$, where here and in the following, $||$ represents concatenation, k_v, k_p are the encryption keys of v and its parent p, respectively, $h_v, h_{v.c_1}, h_{v.c_2}$ are the description of the hash functions used at v and the children $v.c_1, v.c_2$ of v, respectively (which can be described in $O(\ell)$ bits), $ct_{v.c_1}, ct_{v.c_2}$ are the ciphertexts stored at the children $v.c_1, v.c_2$, respectively, and $x_{v.c_1}, x_{v.c_2}$ are the hashes at the children $v.c_1, v.c_2$, respectively. If $v.c_1, v.c_2$ are leaves, $h_{v.c_1}, x_{v.c_1}, h_{v.c_2}, x_{v.c_2}$ are the all-zero string. The root node, stored in Sec contains $(h_r, k_r, x_r = h_r(ct_{r.c_1}||h_{r.c_1}||x_{rc_1}||ct_{r.r.c_2}||h_{r.c_2}||x_{r.c_2}))$, where k_r is the root encryption key. The FS MC algorithms are as follows:

– $\mathsf{init}(1^\lambda, n)$: Initializes a complete binary tree with n leaves in Pub, with data $\perp$ at each node, $\mathtt{Sec} \leftarrow \perp$.

– index(i): Returns the indices of the cells of the root-to-leaf path nodes of leaf i, along with all path nodes' siblings.
– read($\mathtt{Sec}, \mathsf{C}, i$): Given (what the FS MC believes to be) the root-to-leaf path of leaf i and siblings along the path in C from the server, the FS MC:
  - First checks $x_r = h_r(ct_{r.c_1}||h_{r.c_1}||x_{r.c_1}||ct_{r.c_2}||h_{r.c_2}||x_{r.c_2})$.
  - Then, for every internal node v along the path to i, the FS MC checks $x_v = h_v(ct_{v.c_1}||h_{v.c_1}||x_{v.c_1}||ct_{v.c_2}||h_{v.c_2}||x_{v.c_2})$ and decrypts $k_v \leftarrow \mathbf{D}_{k_p}(ct_v)$.
  - Finally, at leaf node i, the FS MC decrypts $d_i \leftarrow \mathbf{D}_{k_p}(ct_i)$.

  If any hash check fails, the FS MC outputs $d_i \leftarrow \mathtt{bug}$, otherwise, it outputs d_i. The FS MC does not change $\mathtt{Sec}$.
– write($\mathtt{Sec}, \mathsf{C}, d'_i, i$): Given (what the FS MC believes to be) the root-to-leaf path of leaf i and siblings along the path in C, the FS MC
  - First verifies and decrypts all of the nodes (including siblings) in C as in read() above; if the test on any node fails, the FS MC outputs $\mathsf{C}' \leftarrow \mathtt{bug}$ and does not change $\mathtt{Sec}$. Otherwise,
  - For each internal node v on the root-to-leaf path of i, except for the parent of i, the FS MC regenerates k'_v and $ct'_{v.c_1} = \mathbf{E}_{k'_v}(k'_{v.c_1}), ct'_{v.c_2} = \mathbf{E}_{k'_v}(k'_{v.c_2})$, where if $v.c_1$ is not on the root-to-leaf path, $k'_{v.c_1} = k_{v.c_1}$ (its current key), and symmetrically if $v.c_2$ is not.
  - Then for the parent p of i, the FS MC regenerates k'_p and $ct'_i = \mathbf{E}_{k'_p}(d'_i)$ and for the sibling j of i, $ct'_j = \mathbf{E}_{k'_p}(d_j)$.
  - Next, for each internal node v on the root-to-leaf path of i, the FS MC regenerates h'_v and $x'_v = h'_v(ct'_{v.c_1}||h'_{v.c_1}||x'_{v.c_1}||ct'_{v.c_2}||h'_{v.c_2}||x'_{v.c_2})$ where if $v.c_1$ is not on the root-to-leaf path, $h'_{v.c_1} = h_{v.c_1}, x'_{v.c_1} = x_{v.c_1}$ (its current values), and symmetrically if $v.c_2$ is not.

  C' is set to store the updated ciphertexts, hash functions, and hash values at their corresponding nodes, and $\mathtt{Sec}'$ is set to store the regenerated hash function h'_r, root key k'_r, and hash value x'_r.

Intuitively, the root key and hash stored in $\mathtt{Sec}$ ensure that the ciphertexts at the root's children will decrypt to the correct keys, while still ensuring privacy. Then, inductively, the decrypted keys and corresponding hashes at level i ensure the ciphertexts at level $i + 1$ decrypt to the correct keys, while still ensuring privacy. We formalize this in the following theorem, which we prove in the full version [5].

Theorem 8. *There exists a secure Forward Secret Memory Checker with $O(\log n)$ overhead, $O(n)$ public storage and $O(1)$ secret storage.*

Remark 1. In practice, a single CRHF can be used in place of a UOWHF family so that there is no need to regenerate the hash functions at the nodes of the root-to-leaf path for every write(), nor include them in the hashes at each node. We use a UOWHF family since it is possible to construct one given any OWF.

Remark 2. For the same level of privacy, but weaker integrity (i.e., integrity only before any corruption of $\mathtt{Sec}$), one can use *only* AEAD in place of *both* symmetric encryption *and* a UOWHF/CHRF. In this case also, the word size need not be polynomial. This is a generalization of a weaker MC construction from [6].

References

1. Asharov, G., Komargodski, I., Lin, W.K., Nayak, K., Peserico, E., Shi, E.: Optorama: Optimal oblivious ram. In: Annual International Conference on the Theory and Applications of Cryptographic Techniques. pp. 403–432. Springer (2020)
2. Aviram, N., Gellert, K., Jager, T.: Session resumption protocols and efficient forward security for TLS 1.3 0-RTT. In: Ishai, Y., Rijmen, V. (eds.) EUROCRYPT 2019, Part II. LNCS, vol. 11477, pp. 117–150. Springer, Heidelberg, Germany, Darmstadt, Germany (May 19–23, 2019)
3. Bajaj, S., Sion, R.: Ficklebase: Looking into the future to erase the past. In: 2013 IEEE 29th International Conference on Data Engineering (ICDE). pp. 86–97. IEEE (2013)
4. Bienstock, A., Dodis, Y., Rösler, P.: On the price of concurrency in group ratcheting protocols. In: Pass, R., Pietrzak, K. (eds.) TCC 2020, Part II. LNCS, vol. 12551, pp. 198–228. Springer, Heidelberg, Germany, Durham, NC, USA (Nov 16–19, 2020)
5. Bienstock, A., Dodis, Y., Yeo, K.: Forward secret encrypted ram: Lower bounds and applications. Cryptology ePrint Archive, Report 2021/244 (2021), `https://eprint.iacr.org/2021/244`
6. Blum, M., Evans, W.S., Gemmell, P., Kannan, S., Naor, M.: Checking the correctness of memories. In: 32nd FOCS. pp. 90–99. IEEE Computer Society Press, San Juan, Puerto Rico (Oct 1–4, 1991)
7. Boneh, D., Lipton, R.J.: A revocable backup system. In: USENIX Security Symposium. pp. 91–96 (1996)
8. Boyle, E., Naor, M.: Is there an oblivious ram lower bound? In: Proceedings of the 2016 ACM Conference on Innovations in Theoretical Computer Science. pp. 357–368 (2016)
9. Canetti, R., Garay, J., Itkis, G., Micciancio, D., Naor, M., Pinkas, B.: Multicast security: a taxonomy and some efficient constructions. In: IEEE INFOCOM '99. Conference on Computer Communications. Proceedings. Eighteenth Annual Joint Conference of the IEEE Computer and Communications Societies. The Future is Now (Cat. No.99CH36320). vol. 2, pp. 708–716 vol.2 (1999)
10. Canetti, R., Raghuraman, S., Richelson, S., Vaikuntanathan, V.: Chosen-ciphertext secure fully homomorphic encryption. In: Fehr, S. (ed.) PKC 2017, Part II. LNCS, vol. 10175, pp. 213–240. Springer, Heidelberg, Germany, Amsterdam, The Netherlands (Mar 28–31, 2017)
11. Clarke, D.E., Suh, G.E., Gassend, B., Sudan, A., van Dijk, M., Devadas, S.: Towards constant bandwidth overhead integrity checking of untrusted data. In: 2005 IEEE Symposium on Security and Privacy. pp. 139–153. IEEE Computer Society Press, Oakland, CA, USA (May 8–11, 2005)
12. Cohen, A., Holmgren, J., Nishimaki, R., Vaikuntanathan, V., Wichs, D.: Watermarking cryptographic capabilities. In: Wichs, D., Mansour, Y. (eds.) 48th ACM STOC. pp. 1115–1127. ACM Press, Cambridge, MA, USA (Jun 18–21, 2016)
13. Derler, D., Jager, T., Slamanig, D., Striecks, C.: Bloom filter encryption and applications to efficient forward-secret 0-RTT key exchange. In: Nielsen, J.B., Rijmen, V. (eds.) EUROCRYPT 2018, Part III. LNCS, vol. 10822, pp. 425–455. Springer, Heidelberg, Germany, Tel Aviv, Israel (Apr 29 – May 3, 2018)
14. Di Crescenzo, G., Ferguson, N., Impagliazzo, R., Jakobsson, M.: How to forget a secret. In: Proceedings of the 16th Annual Conference on Theoretical Aspects of Computer Science. p. 500–509. STACS'99, Springer-Verlag, Berlin, Heidelberg (1999)

15. Dwork, C., Naor, M., Rothblum, G.N., Vaikuntanathan, V.: How efficient can memory checking be? In: Reingold, O. (ed.) TCC 2009. LNCS, vol. 5444, pp. 503–520. Springer, Heidelberg, Germany (Mar 15–17, 2009)
16. Geambasu, R., Kohno, T., Levy, A.A., Levy, H.M.: Vanish: Increasing data privacy with self-destructing data. In: USENIX Security Symposium. vol. 316 (2009)
17. Goldreich, O., Ostrovsky, R.: Software protection and simulation on oblivious rams. Journal of the ACM (JACM) 43(3), 431–473 (1996)
18. Goodrich, M.T., Mitzenmacher, M.: Privacy-preserving access of outsourced data via oblivious ram simulation. In: International Colloquium on Automata, Languages, and Programming. pp. 576–587. Springer (2011)
19. Green, M.D., Miers, I.: Forward secure asynchronous messaging from puncturable encryption. In: 2015 IEEE Symposium on Security and Privacy. pp. 305–320. IEEE Computer Society Press, San Jose, CA, USA (May 17–21, 2015)
20. Günther, F., Hale, B., Jager, T., Lauer, S.: 0-RTT key exchange with full forward secrecy. In: Coron, J.S., Nielsen, J.B. (eds.) EUROCRYPT 2017, Part III. LNCS, vol. 10212, pp. 519–548. Springer, Heidelberg, Germany, Paris, France (Apr 30 – May 4, 2017)
21. Harney, H., Muckenhirn, C.: Rfc2093: Group key management protocol (gkmp) specification (1997)
22. Harney, H., Muckenhirn, C.: Rfc2094: Group key management protocol (gkmp) architecture (1997)
23. Hubáček, P., Koucký, M., Král, K., Slívová, V.: Stronger lower bounds for online oram. In: Theory of Cryptography Conference. pp. 264–284. Springer (2019)
24. Jacob, R., Larsen, K.G., Nielsen, J.B.: Lower bounds for oblivious data structures. In: Proceedings of the Thirtieth Annual ACM-SIAM Symposium on Discrete Algorithms. pp. 2439–2447. SIAM (2019)
25. Kobeissi, N., Nicolas, G., Bhargavan, K.: Noise explorer: Fully automated modeling and verification for arbitrary noise protocols. In: 2019 IEEE European Symposium on Security and Privacy (EuroS P). pp. 356–370 (2019)
26. Larsen, K.G., Nielsen, J.B.: Yes, there is an oblivious ram lower bound! In: Annual International Cryptology Conference. pp. 523–542. Springer (2018)
27. Larsen, K.G., Simkin, M., Yeo, K.: Lower bounds for multi-server oblivious rams. Theory of Cryptography Conference (to appear) (2020)
28. Merkle, R.C.: A certified digital signature. In: Brassard, G. (ed.) CRYPTO'89. LNCS, vol. 435, pp. 218–238. Springer, Heidelberg, Germany, Santa Barbara, CA, USA (Aug 20–24, 1990)
29. Micciancio, D., Panjwani, S.: Optimal communication complexity of generic multicast key distribution. In: Cachin, C., Camenisch, J. (eds.) EUROCRYPT 2004. LNCS, vol. 3027, pp. 153–170. Springer, Heidelberg, Germany, Interlaken, Switzerland (May 2–6, 2004)
30. Mittra, S.: Iolus: A framework for scalable secure multicasting. In: Proceedings of the ACM SIGCOMM '97 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication. p. 277–288. SIGCOMM '97, Association for Computing Machinery, New York, NY, USA (1997), `https://doi.org/10.1145/263105.263179`
31. Oprea, A., Reiter, M.K.: Integrity checking in cryptographic file systems with constant trusted storage. In: Provos, N. (ed.) USENIX Security 2007. USENIX Association, Boston, MA, USA (Aug 6–10, 2007)
32. Patel, S., Persiano, G., Raykova, M., Yeo, K.: Panorama: Oblivious ram with logarithmic overhead. In: 2018 IEEE 59th Annual Symposium on Foundations of Computer Science (FOCS). pp. 871–882. IEEE (2018)

33. Patel, S., Persiano, G., Yeo, K.: Lower bounds for encrypted multi-maps and searchable encryption in the leakage cell probe model. In: Annual International Cryptology Conference. pp. 433–463. Springer (2020)
34. Persiano, G., Yeo, K.: Lower bounds for differentially private rams. In: Annual International Conference on the Theory and Applications of Cryptographic Techniques. pp. 404–434. Springer (2019)
35. Peterson, Z.N., Burns, R.C., Herring, J., Stubblefield, A., Rubin, A.D.: Secure deletion for a versioning file system. In: FAST. vol. 5 (2005)
36. Pinkas, B., Reinman, T.: Oblivious ram revisited. In: Annual cryptology conference. pp. 502–519. Springer (2010)
37. Reardon, J., Basin, D., Capkun, S.: Sok: Secure data deletion. In: 2013 IEEE symposium on security and privacy. pp. 301–315. IEEE (2013)
38. Reardon, J., Ritzdorf, H., Basin, D., Capkun, S.: Secure data deletion from persistent media. In: Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security. pp. 271–284 (2013)
39. Roche, D.S., Aviv, A., Choi, S.G.: A practical oblivious map data structure with secure deletion and history independence. In: 2016 IEEE Symposium on Security and Privacy (SP). pp. 178–197. IEEE (2016)
40. Rompel, J.: One-way functions are necessary and sufficient for secure signatures. In: 22nd ACM STOC. pp. 387–394. ACM Press, Baltimore, MD, USA (May 14–16, 1990)
41. Sherman, A.T., McGrew, D.A.: Key establishment in large dynamic groups using one-way function trees. IEEE Transactions on Software Engineering 29(5), 444–458 (2003)
42. Stefanov, E., Van Dijk, M., Shi, E., Fletcher, C., Ren, L., Yu, X., Devadas, S.: Path oram: an extremely simple oblivious ram protocol. In: Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security. pp. 299–310 (2013)
43. Sun, S., Sakzad, A., Steinfeld, R., Liu, J.K., Gu, D.: Public-key puncturable encryption: Modular and compact constructions. In: Kiayias, A., Kohlweiss, M., Wallden, P., Zikas, V. (eds.) PKC 2020, Part I. LNCS, vol. 12110, pp. 309–338. Springer, Heidelberg, Germany, Edinburgh, UK (May 4–7, 2020)
44. Susilo, W., Duong, D.H., Le, H.Q., Pieprzyk, J.: Puncturable encryption: A generic construction from delegatable fully key-homomorphic encryption. In: Chen, L., Li, N., Liang, K., Schneider, S.A. (eds.) ESORICS 2020, Part II. LNCS, vol. 12309, pp. 107–127. Springer, Heidelberg, Germany, Guildford, UK (Sep 14–18, 2020)
45. Wallner, D., Harder, E., Agee, R.: Rfc2627: Key management for multicast: Issues and architectures (1999)
46. Wong, C.K., Gouda, M., Lam, S.S.: Secure group communications using key graphs. In: Proceedings of the ACM SIGCOMM '98 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication. p. 68–79. SIGCOMM '98, Association for Computing Machinery, New York, NY, USA (1998), `https://doi.org/10.1145/285237.285260`
47. Yao, A.C.C.: Should tables be sorted? Journal of the ACM (JACM) 28(3), 615–628 (1981)