How to Protect Yourself without Perfect Shredding*

Ran Canetti[1], Dror Eiger[2], Shafi Goldwasser[3] and Dah-Yoh Lim[4]

[1] IBM T. J. Watson Research Center
[2] Google, Inc. (work done at Weizmann Institute of Science)
[3] MIT and Weizmann Institute of Science
[4] MIT

Abstract. Erasing old data and keys is an important tool in cryptographic protocol design. It is useful in many settings, including proactive security, adaptive security, forward security, and intrusion resilience. Protocols for all these settings typically assume the ability to *perfectly erase* information. Unfortunately, as amply demonstrated in the systems literature, perfect erasures are hard to implement in practice.

We propose a model of *partial erasures* where erasure instructions leave almost all the data erased intact, thus giving the honest players only a limited capability for disposing of old data. Nonetheless, we provide a general compiler that transforms any secure protocol using perfect erasures into one that maintains the same security properties when only partial erasures are available. The key idea is a new redundant representation of secret data which can still be computed on, and yet is rendered useless when partially erased. We prove that any such a compiler must incur a cost in additional storage, and that our compiler is near optimal in terms of its storage overhead.

Key words: mobile adversary, proactive security, adaptive security, forward security, intrusion resilience, universal hashing, partial erasures, secure multiparty computation, randomness extractors

1 Introduction

As anyone who has ever tried to erase an old white board knows, it is often easier to erase a large amount of information imperfectly, than to erase a small amount of information perfectly.

In cryptographic protocol design, *perfect erasures*, namely the complete disposal of old and sensitive data and keys, is an important ability of honest players in fighting future break-ins, as this leaves no trace of sensitive data for the adversary to recover.

Examples where perfect erasures have been used extensively include the areas of *proactive security* [58,30,28,37,12,50], *forward security* [33,20,2] and *intrusion resilience* [43], and *adaptive security* [44,11,7,59]. Whereas erasures merely simplify the design of adaptively secure protocols, some form of erasures is provably necessary for achieving proactive security and even for defining the task of forward security as we explain below.

The goal of *Proactive Security* introduced in [58] is to achieve secure multi-party computations where some fraction of the parties are faulty. The identity of faulty parties are decided by a *mobile* adversary who can corrupt a different set of players in different *time periods* (here the protocols assume time is divided into well-defined intervals called time periods) subject to an upper bound on the total number of corrupted players per time period. At the heart of the solutions pursued in the literature are secret sharing methods in which in every time period, the old shares held by players are first replaced by new shares and then perfectly erased. It is easy to prove that secret sharing would be impossible to achieve without some form of erasures: otherwise a mobile adversary which is able to corrupt every single player in some time period or another, can eventually recover all old shares for some single time period and recover the secret. Forward security [33,20,2]. is an approach taken to tackle the *private key exposure* problem, so that exposure of long-term secret information does not compromise the security of previous sessions. Again, the lifetime of the system is divided into time periods. The receiver initially stores secret key SK_0 and this secret key "evolves" with time: at time period

* This is the full version of the paper appearing in ICALP08 under the same title.

i, the receiver applies some function to the previous key SK_{i-1} to derive the current key SK_i and then key SK_{i-1} is perfectly erased. The public (encryption) key remains fixed throughout the lifetime of the scheme. A forward-secure encryption scheme guarantees that even if an adversary learns the secret key available at time i, SK_i, messages encrypted during all time periods prior to i remain secret. Intrusion Resilience is a strengthening of forward security [43] which can be viewed as combination of forward and backward security. Obviously, erasures are essential to define (and solve) both the forward security and intrusion resilience problems.

Finally, an example of a different flavor of the utility of erasures to guard against adversaries that can choose which future parties to corrupt as the protocol proceeds, based on information already gathered. Erasures are useful in this context since they limit the information the adversary sees upon corrupting a party. Protocols designed without erasures (although possible in this context), tend to be much more complex than those that rely on data erasures [44,11,7,59].

Unfortunately, perfect erasures of data are hard to achieve in practice and are thus problematic as a security assumption, as pointed out by Jarecki and Lysyanskaya [44] in their study of adaptive adversaries versus static adversaries in the context of threshold secret sharing.

Some of the difficulty in implementing perfect erasures is illustrated in the works of Hughes and Coughlin, Garfinkel, and Vaarala [40,39,29,62]. The root cause of this difficulty is that systems are actually designed to preserve data, rather than to erase it. Erasures present difficulties at both the hardware level (e.g. due to physical properties of the storage media) and at the software level (e.g. due to the complications with respect to system bookkeeping and backups). At the hardware level, e.g. for hard drives, the Department of Defense recommends overwriting with various bit patterns [57]. This takes the order of days per 100GB, and is not fully effective because modern hard drives use block replacement and usually employ some form of error correction. For main memory, due to "ion migration", previous states of memory can be determined even after power off. At the software level, many operating systems detect and remap bad sectors of the hard drive on the fly, but original data can remain in the bad sectors and be recoverable.

1.1 This paper

In light of the above difficulties, we propose to study protocols that can guarantee security even when only *imperfect* or *partial* erasures are available.

The first question to be addressed is how to *model* erasures that are only partially effective. One option is to simply assume that each erasure operation succeeds with some probability. However, such a modeling does not capture all the difficulties described above. In particular, it allows obtaining essentially perfect erasures by applying the erasure operation several times on a memory location; therefore such a model is unlikely to yield interesting or effective algorithms. In addition, such modeling does not take into account potential dependencies among information in neighboring locations.

The model of Partial Erasures We thus take a much more conservative approach. Specifically, we model partial erasures by a length-shrinking function $h : \{0,1\}^m \mapsto \{0,1\}^{\lfloor \phi m \rfloor}$, that shrinks stored information by a given fraction $0 \leq \phi \leq 1$. We call ϕ the leakage fraction. When $\phi = 0$ then we get the perfect erasures case; when $\phi = 1$ nothing is ever erased. For the rest of this work we think of ϕ being a value close to 1 (namely, the size of what remains after data is partially-erased is close to the original size). Note that we do not require ϕ to be a constant – for instance, for reasonable settings of the parameters, it may be $\frac{1}{poly(\alpha)}$ close to 1, where α is a security parameter of the protocol in question.

The shrinking function may be chosen adversarially. In particular, it is not limited to outputting exact bits, and any length-shrinking function (efficiently computable or not) on the inputs is allowed. This modeling captures the fact that the remaining information may be a function of multiple neighboring bits rather than on a single bit. It also captures the fact that repeated erasures may not be more effective than a single one.

The function h is assumed to be a function only of the storage contents to be erased. Furthermore, for simplicity we assume that h is fixed in advance – our schemes remain secure without any modification even if the adversary chooses a new h_i prior to each new erasure. This choice seems to adequately capture erasures

that are only partially successful due to the physical properties of the storage media[5]. However, this may not adequately capture situations where the failure to erase comes from interactions with an operating system, for instance memory swapping, and caching. In order to capture this sort of erasure failures, one might want to let h to be a function of some information other than the contents to be erased, or alternatively to be determined adaptively as the computation evolves.

We treat m, the input length of h, as a system parameter. (For instance, m might be determined by the physical properties of the storage media in use.) One can generalize the current model to consider the cases where h is applied to variable-length blocks, and where the block locations are variable.

Our Memory Model. We envision that processors participating in protocols can store data (secret and otherwise) in main memory as well as cache, hard drives, and CPU registers. We assume all types of storage are partially erasable, except a constant number of constant size CPU registers which are assumed to be perfectly erasable. We emphasize that the constant size of the registers ensures that we do not use this to effectively perfectly erase main memory and thus circumvent the lack of perfect erasures in main memory, since at no time can the registers hold any non-negligible part of the secret. We call this the *register model*.

We shall use these registers to perform intermediate local computations during our protocols. This will allow us to ignore the traces of these computations, which would otherwise be very messy to analyze.

Results and Techniques. Our main result is a compiler that on input any protocol that uses perfect erasures, outputs one that uses only partial erasures, and preserves both the functionality and the security properties of the original protocol. Our transformation only applies to the storage that needs to be erased.

The idea is to write secrets in an *encoded form* so that, on the one hand, the secret can be explicitly extracted from its encoded form, and on the other hand loss of even a small part of the encoded form results in loss of the secret.

Perhaps surprisingly, our encoding results in *expanding* the secret so that the encoded information is longer than the original. We will prove that expanding the secret is *essential* in this model (see more discussion below). This expansion of secrets seems a bit strange at first, since now there is more data to be erased (although only partially). However, we argue that it is often easier to erase a large amount of data imperfectly than to erase even one bit perfectly.

We describe the compiler in two steps. First we describe a special case where there is only a single secret to be erased. Next we describe the complete compiler.

Our technique for the case of a single secret is inspired by results in the *bounded storage model*, introduced by Maurer [52,53]. Work by Lu [49] casted results in the bounded storage model in terms of *extractors* [56], which are functions that when given a source with some randomness, purifies and outputs an almost random string.

At a high level, in order to make an n-bit secret s partially erasable, we choose random strings R, X and store $R, X, \mathrm{Ext}(R, X) \oplus s$, where Ext is a strong extractor that takes R as seed and X as input, and generates an n-bit output such that $(R, \mathrm{Ext}(R, X))$ is statistically close to uniform as long as the input X has sufficient min-entropy. To erase s, we apply the imperfect erasure operation on X. Both R and $\mathrm{Ext}(R, X) \oplus s$ are left intact.

For sake of analysis, assume that $|X| = m$, where m is the input length for the partial erasure function h. Recall that erasing X amounts to replacing X with a string $h(X)$ whose length is ϕm bits. Then, with high probability (except with probability at most $2^{-(1-\phi)m/2}$), X would have about $(1-\phi)m/2$ min-entropy left given $h(X)$. This means that, as long as $(1-\phi)m/2 > n$, the output of the extractor is roughly $2^{-(1-\phi)|X|/2}$-close to uniform even given the seed R and the partially erased source, $h(X)$. Consequently, s is effectively erased.

There is however a snag in the above description: in order to employ this scheme, one has to evaluate the extractor Ext without leaving any trace of the intermediate storage used during the evaluation. Recall that

[5] Indeed, physical properties of the storage are mostly fixed at the factory; from then on the behavior of the hardware only depends on what is written.

our model the size of the perfectly erasable memory is constant independently of n, the length of the secret. This means that Ext should be computable with constant amount of space, even when the output length tends to infinity. We identify several such extractors, including ϵ-almost universal hashing, strong extractors in NC^0, and Toeplitz Hashing [51]. It would seem superficially that locally computable strong extractors [63] can be used, but unfortunately they cannot (see theorem 3.3).

Now let us move on to describe the general compiler. Suppose we want to compute some function g (represented as a circuit) on some secret s, only now, s is replaced by a representation that is partially erasable, and we would like to make sure that we can still compute $g(s)$. We are going to evaluate the circuit in a gate-by-gate manner where the gate inputs are in expanded form. The inputs are reconstructed in the registers, and the gate is evaluated to get an output, which is in turn expanded and stored in main memory. Even though some small (negligible) amount of information is leaked at each partial erasure, we show that as long as the number of erasure operations is sub-exponential, the overall amount of information gathered by the adversary on the erased data is negligible.

For maximum generality we formulate our results in the *Universally Composable (UC)* framework. In particular we use the notion of *UC emulation* [9], which is a very tight notion of correspondence between the emulated and emulating protocols. Our analysis applies to essentially any type of corruption – adaptive, proactive, passive, active, etc. That is, we show:

Theorem (informal): For any protocol Π_{org} that requires perfect erasures (for security), the protocol $\Pi_{new} = \text{COMPILER}(\Pi_{org})$ UC-emulates Π_{org}, and tolerates (maintains security even with) imperfect/partial erasures in the register model. For leakage fraction of ϕ, if Π_{org} uses n bits of storage then Π_{new} uses about $\frac{2}{1-\phi}n$ bits of storage.

Optimality of the scheme. One of the main cost parameters of such compilers is the *expansion factor,* the amount by which they increase the (erasable part of the) storage. That is, a compiler has expansion factor Ψ if whenever Π_{org} uses n bits of storage, Π_{new} uses at most Ψn bits of storage. It can be seen that our compiler has expansion factor $\Psi \leq \frac{2}{1-\phi} + \nu(n)$ where ν is a negligible function. In addition, we show that if ϵ-almost universal hashing is used and $\phi > 1/4$, then our compiler would have an expansion factor of about $\frac{c}{1-\phi}$, where $c = \frac{7}{3} - \frac{4}{3}\phi$. For $1/4 \leq \phi < 1$, we have that $2 \geq c > 1$.

We show that our construction is at most twice the optimal in this respect. That is, we show that *any* such compiler would necessarily have an expansion of roughly $\Psi \geq \frac{1}{1-\phi}$. This bound holds even for the simplest case of compiling even a single secret into one that is partially erasable. Roughly speaking, the argument is as follows. If we do not want to leak any information on a secret of n bits the function h must shrink the expanded version of s by at least n bits. In our model, h shrinks by $(1-\phi)\Psi n$ bits and therefore, $(1-\phi)\Psi n \geq n \Rightarrow \Psi \geq \frac{1}{1-\phi}$.

Some specific solutions. In addition to the general compiler, in 5 we describe some special-tailored solutions to two specific cases. One case is where the function to be evaluated is computable by NC^0 circuits. The second case is the case for all known proactive secret sharing schemes. These solutions are computationally more efficient since they do not require running the compiler on a gate by gate basis. In particular, in the case of proactive secret sharing we can apply our expanded representation directly to the secret and its shares and the instructions which modify the shares (to accordingly modify the new representations) and leave the rest of the protocol intact. Note that this greater efficiency also translates into tighter security – for instance if the original protocol assumed some timing guarantees, then the new protocol need not assume a timing that is much looser than the original.

Remark 1. As we elaborate at the end of section 4, a side benefit of using using our constructions is that it can be resistant to a practical class of physical attacks [34] that involves freezing RAM and recovering secrets from it.

Remark 2. Note that because we prove statistical security, our schemes are "everlastingly secure" in the sense that even if the adversary stores all the partially erased information, whatever happens in the future will not help him, e.g. even if it turns out that $P = NP$.

1.2 Related Work

The Bounded Storage Model (BSM). The Bounded Storage Model (BSM) proposed by Maurer [52,53], considers computationally unbounded but storage limited adversaries. This enables novel approaches to the secure communication problem as follows. The communicating parties begin with a short initial secret key k. In the first phase they use this key k and access to a long public random string R to derive a longer key X. The storage bounded adversary computes an arbitrary length-shrinking function on R. In the second phase, R "disappears", and the parties will use X as a one-time pad to communicate privately.

We will use the same kind of length-shrinking function to capture the act of partially erasing old shares of a secret.

However, conceptually the settings of the BSM and partial erasures are fundamentally different. In the BSM model possibility is proved by putting limitations on the **adversary** (storage), where as in our work possibility is proved *in spite of* putting limitation on the **honest parties** (erasing capability). Thus, although some of techniques are similar the setup is entirely different. From a technical point of view there are two differences we emphasize as well. Firstly, the extractors that we use must be computable with constant sized memory, whereas in the BSM the extractors are not necessarily computable with constant-sized memory. Secondly, in the BSM, it is assumed that the adversary's storage bound remains the same as time goes by, namely a constant fraction ϕ of the public randomness R. The same assumption is used in the bounded retrieval model [19,17,23,24,25]. For instance [25] constructs intrusion resilient secret sharing schemes by making the shares large and assuming that the adversary will never be able to retrieve any share completely. For partial erasures this bound is unreasonable, and we allow the adversary to get ϕ fraction of R_i for each erasure operation.

Exposure Resilient Functions. Exposure-Resilient Functions, or ERFs, were introduced by Canetti et al. [10,21]. An ℓ-ERF is a function with a random input, such that an adversary which learns all but ℓ bits of the input, cannot distinguish the output of the function from random.

At a high level the ERF objectives seem very similar to partial erasures. However, the settings are different. In particular, ERFs only deal with the leakage of exact bits whereas we deal with the leakage of general information. (We remark that this limitation of ERFs is inherent in their model: It is easy to see that there do not exist ERFs that resists arbitrary leakage functions).

Encryption as Deletion. As Di Crescenzo et al. [16] noted, one simple but inefficient way to implement erasable memory can be obtained by using the crypto-paging concept of Yee [64]. Assume that some amount of storage that is linear in the security parameter is available that is perfectly erasable, and some other poly storage is persistent. To make the persistent memory effectively erasable, pick an encryption scheme and keep the key in the erasable part. Always encrypt the contents to be kept on the persistent storage. Then erasing the key is as good as erasing the contents.

By combining these ideas with ours, it is possible to have an increase in storage that is linear in the security parameter, while using only a *constant* amount of perfectly erasable memory.

2 The Model of Partial Erasures

As sketched in the introduction, we model partial erasures by an arbitrary length-shrinking function $h : \{0,1\}^m \to \{0,1\}^{\lfloor \phi m \rfloor}$, where $0 \leq \phi \leq 1$ is called the *leakage fraction*. By *partially erasing* some information x, we mean the operation of applying such an h to x, leaving $h(X)$. If $|X| > m$, then we consider a predefined partition of x into blocks of size m, and h is then applied separately to each block. We stress that the protocol has no access whatsoever to (partially) erased data, and the partial leakage is only made available to the adversary.

More precisely, we need a suitable model of computation for Turing Machines that partially erase.

Definition 1 (Partially Erasable (PE) Tape). *A tape of a Turing Machine is called partially erasable (PE) if:*

1. *It may be read multiple times, i.e. the read head is free to move both forward and backward.*

2. *It is write once, i.e. the write head always moves forward.*
3. *It is partially erasable, i.e. there is a partial-erase head that has a separate control logic (state transition function) that performs the partial erase operation as follows. We think of the tape as pre-partitioned into m-bit blocks (where m is a parameter of the Turing Machine). The partial erase operation on input a block number, replaces that block x by $\perp^m$ (so as far as the Turing Machine is concerned the information is erased), and writes $h(x)$ along with the block number to a special "shadow" tape. This shadow tape is unreadable by the Turing Machine, but will be given to the adversary upon corruption.*

Note that we can see the computation of h as being done by an oracle machine O, i.e. the state transition function for computing h is separate from the erasing Turing Machine and can be arbitrary, and in particular (to the advantage of the adversary) the computation of h is not limited by the resources of the erasing Turing Machine.

Definition 2 (Externally Writable (EW) Tape). *A tape of a Turing Machine is called externally writable (EW) if it may be written to by other partially erasable interactive Turing Machines. We assume that all EW tapes are write-once, so the write head only moves in one direction.*

The only differences in our model of computation from the interactive Turing Machine of [9] is that:

1. The work tape, input tape, and subroutine output tape are PE instead of EW.
2. The random tape is read once, i.e. the read head always moves forward. It is also hidden from the adversary upon corruption.

The first point addresses the fact that overwriting is not allowed (it is a form of perfect erasures), only partial erasures is.

The second point addresses the fact that if the Turing Machine wants to use the random bits, it has to copy them to the computation tape, and thus can only be partially erased. Upon corruption of a party, we assume that the adversary does not gain access to the random tape, but sees all the rest of the tapes. This is reasonable because firstly, this does not provide a "backdoor" – if the Turing Machine wants to use the random bits it has to copy them to the computation tape, which the adversary will be allowed to see anyways. Secondly, since the encoding of the secret has to be randomized, if the adversary is allowed to see all the random coins then this essentially allows the adversary to see everything, and therefore this is the same as having no erasures. Thirdly, this just boils down to assuming that we have a random (one) bit generator in the CPU that is perfectly erasable.

In order to have a concrete model, here is the full description of our model of computation, by incorporating the change described above into the model of interactive Turing Machines (ITMs) given by [9]:

Definition 3 (Partially Erasable Interactive Turing Machine (PEITM)). *A partially erasable interactive Turing Machine (PEITM) is a Turing Machine with the following tapes:*

- *An EW identity tape.*
- *An EW security parameter tape.*
- *An PE input tape.*
- *An EW incoming communication tape.*
- *An PE subroutine output tape.*
- *An output tape.*
- *A read and write one-bit activation tape.*
- *A read once random tape that is hidden from the adversary upon corruption.*
- *A PE work tape.*

We refer the reader to [9] for the definitions of a system of ITMs and probabilistic polynomial time ITMs, which can be extended straightforwardly to deal with PEITMs instead. We stress again that in bounding the resources of the ITMs, the time taken to partially erase (i.e. compute h) should be ignored.

Alternatively, one could apply the modifications to the ITM model of [32], rather than to the ITM model of the UC framework [9]. We choose to do the latter since our results are later stated in the UC framework.

The partial erasures model represents several sources of information leakage on data that is intended to be erased – let us discuss a few. One difficulty with implementing perfect erasures is that it is highly dependent on the physical characteristics of the devices in use. For instance, as described in the introduction, for main memory, due to *ion migration* there is a cumulative remanence effect – the longer a bit pattern sits in the RAM, the more "burnt in" it becomes and the longer it takes to erase. Furthermore, even different RAM batches exhibit different physical characteristics, much less different memory technology. Security proofs that assume perfect erasures fall as long as even one bit of information remains; to ensure security, one would have to somehow find out the "maximum" time it takes for the RAM to be erased. Partial erasures also provides security even when the adversary breaks into the parties when they are only "half-way" done in erasing their secrets., no matter how the adversary chooses the timing of the corruptions (as long as some information has been erased).

We note that our schemes remain secure without any modification even if the adversary is allowed to choose a new h_i whenever an erasure is to be done; in particular, no independence is assumed on the different erasures done on different memory locations. However, this choice must be done independently of the current data to be erased, i.e. the choice of h_i is fixed before the data to be erased by h_i is written in memory. This is certainly enough for modeling the imperfect erasures due to physical limitations.

As mentioned in 1.1, we relax the model a bit and allow a constant number of constant size registers to be perfectly erasable, whereas the other parts of the storage (including cache, main memory, and hard drives) can only be partially erased. We will refer to this as the *register model*. Actually, these perfectly erasable registers can in principle be encoded in the finite control state of the Turing Machine, and need not be written on any tape at all. In particular, the model need not explicitly allow for perfectly erasable storage at all.

Remark 3. As we mentioned in the beginning of section 1.1, we only deal with storage that would need to be erased, so not everything that is normally done in the CPU registers is going to be included in the constant-size registers in our model – the addressing of main memory being an example (otherwise, in order to be able to access polynomial sized storage, the registers need to be of logarithmic size).

3 How to Make Secrets Partially Erasable

To change a protocol using perfect erasures to one that uses only partial erasures, the first step is to store secrets in a partially erasable way, which is what this section focuses on. The second step is to make sure computations can still be done on the secrets without leaking too much information, and will be discussed in the next section.

The high level idea is that instead of having a piece of secret $s \in \{0,1\}^n$ directly in the system, we let the parties store it in expanded form. At the cost of more storage, this gives the ability to effectively erase a secret even when only partial erasures are available. In the end, the number of bits that have to be partially erased might be more than the number of bits that have to be perfectly erased. This is still reasonable because it is often much easier to partially erase a large number of bits than to perfectly erase a small number. Furthermore, we show in 3.4 that such expansion is inherent in the model.

We write $U_{\mathcal{X}}$ to denote an r.v. uniformly random on some set $\mathcal{X}$.

Definition 4 (Statistical Distance). *Suppose that A and B are two distributions over the same finite set Σ. The statistical distance between A and B is defined as $\Delta(A;B) := \frac{1}{2}\sum_{\sigma\in\Sigma}\left|\Pr_A[\sigma] - \Pr_B[\sigma]\right|$.*

Definition 5 (Statistical Distance from the Uniform). *Let $d(A) := \Delta(A;U_{\mathcal{A}})$. Also define $d(A|B) := \sum_b d(A|B=b)\cdot\Pr_B[b] = \sum_b \Pr_B[b]\frac{1}{2}\sum_a |\Pr_{A|B=b}[a] - \frac{1}{|\mathcal{A}|}|$. We say that a random variable A is ϵ-close to the uniform given B to mean that $d(A|B) \le \epsilon$.*

Due to lack of space we refer the reader to the introduction for the definitions of a *partial erasure function* $h : \{0,1\}^m \mapsto \{0,1\}^{\lfloor \phi m \rfloor}$ and a *leakage fraction* $0 \leq \phi \leq 1$.

Definition 6 (Partially Erasable (or Expanded) Form of a Secret). *Let* $\mathsf{Exp}(\circ,\circ)$ *be the "expansion" function taking the secret* s *to be expanded as the first input and randomness as the second, and* Con *be the "contraction" function taking the output of* Exp *as the input. Let* $h_i^s := h(\mathsf{Exp}(s, \$_i))$*, where* $\$_i$ *are independent randomness. We say that* $(\mathsf{Exp}, \mathsf{Con})$ *is* (ℓ, α, ϕ)*-partially erasable form of a secret if* $\forall s \in \{0,1\}^n$*, for any* h *with leakage fraction* ϕ*,*

1. *(Correctness)* $\mathsf{Con}(\mathsf{Exp}(s,r)) = s$ *for all* $r \in \{0,1\}^{poly(n)}$.
2. *(Secrecy)* $\forall s' \in \{0,1\}^n, \Delta\big(h_1^s, ..., h_\ell^s; h_1^{s'}, ..., h_\ell^{s'}\big) \leq 2^{-\alpha}$.
3. *(With Constant Memory) Both* $\mathsf{Exp}, \mathsf{Con}$ *are computable with constant memory.*

Remark 4. We require both Exp and Con to be computable with constant memory to ensure that intermediate computations can be kept in the registers which are perfectly erasable.

Remark 5. We require indistinguishability for many (ℓ above) erasures to account for the fact that many computations may be done during the protocol (directly or indirectly) on the secret, from which the adversary might gain more information on a secret. Generally, an adversary may have many partially erasable forms of the same secret (i.e adversary can see $h(\mathsf{Exp}(s, \$_i))$ s.t for each i adversary knows a 1-1 and onto correspondence q_i from $\mathsf{Exp}(s, \$_i)$ to s).

An example of an expanded form of a secret which can be partially erased and satisfies correctness and secrecy (in the above definition) would be to use a universal hash function family $\{H_R\}$ as follows: expand s to (v, R, k) s.t. $s = H_R(k) \oplus v$. By using the leftover hash lemma [41], for any constant ϕ such that $0 < \phi < 1$, for any arbitrary partial erasure function h with leakage fraction ϕ, for any universal hash function family $\{H_R\}$, $H_R(k)$ can be made negligibly close to uniform given R and $h(k)$ (so $H_R(k)$ is as good as a one time pad).

3.1 Using Universal Hashing

Let us first focus on bounding $d(H_R(k)|R, h(k))$. We need the leftover hash lemma:

Lemma 1 (The Leftover Hash Lemma [36]). *Let* $\mathcal{H}$ *be a universal family of hash functions from* $\mathcal{X}$ *to* $\mathcal{Y}$*. Let* H *denote a random variable with the uniform distribution on* $\mathcal{H}$*, and let* X *denote a random variable taking values in* $\mathcal{X}$*, with* H, X *independent. Then* $(H, H(X))$ *is* δ*-uniform on* $\mathcal{H} \times \mathcal{Y}$*, where*

$$\delta \leq \sqrt{|\mathcal{Y}| \max_x \mathbf{P}(X)}/2.$$

Theorem 1 (Security for a Single Erasure using Universal Hash). *Let* $\{H_R\}$ *be a universal family of hash functions. Let* $(R, h(k))$ *be a tuple such that* $R \in \{0,1\}^{n \times m}$*,* $k \in \{0,1\}^m$*, and* $h(k) \in \{0,1\}^{\phi m}$*, where* R *picks out a random function out of* $\{H_R\}$*, and* k *is random. Then* [6]*,*

$$d(H_R(k)|R, h(k)) \leq \frac{3}{2^{2/3}} \cdot 2^{-\frac{1}{3}(1-\phi)m + \frac{n}{3}}. \tag{1}$$

Proof. The big picture is that, since $h(\circ)$ is a length shrinking function, with good probability k should still have high min entropy given $h(k)$. This implies that if we apply a strong extractor (in particular, here we use universal hashing) on k, the result should still be close to random when given the hash function and $h(k)$.

First, let us show that with high probability, k still has high min entropy given $h(k)$. Intuitively, rare events give more information to the adversary. Accordingly, let λ be a real number such that $0 < \lambda < 1 - \phi$, and define $\mathcal{B}$, the "bad" set, to be the set of realizations y of $h(k)$ such that $\mathbf{P}_k(h(k) = y) \leq 2^{-(1-\lambda)m}$.

[6] Yevgeniy Dodis pointed out to us that by using their generalized leftover hash lemma in [22], we can save a root, to get $d(H_R(k)|R, h(k)) \leq \frac{3}{2^{2/3}} \cdot 2^{-\frac{1}{2}(1-\phi)m + \frac{n}{2}}$.

So, for $y_0 \notin \mathcal{B}$,

$$\begin{aligned}\mathbf{P}(k = k_0 | h(k) = y_0) &= \frac{\mathbf{P}(k = k_0 \cap h(k) = y_0)}{\mathbf{P}(h(k) = y_0)} \\ &\leq \frac{\mathbf{P}(k = k_0)}{\mathbf{P}(h(k) = y_0)} \\ &\leq 2^{-m} \cdot 2^{(1-\lambda)m} \\ &= 2^{-\lambda m}.\end{aligned}$$

Therefore, for $h(k) \notin \mathcal{B}$, $H_\infty(k|h(k)) \geq \lambda m$, and by the leftover hash lemma 1, for $h(k) \notin \mathcal{B}$, $d(H_R(k)|R, h(k)) \leq \sqrt{2^n \cdot 2^{-\lambda m}}$, and we can bound the statistical distance of $H_R(k)$ from uniform:

$$\begin{aligned}d(H_R(k)|R, h(k)) &= \sum_{r,y} \mathbf{P}(R = r \cap h(k) = y) d(H_R(k)|R = r, h(k) = y) \\ &= \sum_r \mathbf{P}(R = r) \Big(\sum_{y \in \mathcal{B}} \mathbf{P}(h(k) = y) d(H_R(k)|R = r, h(k) = y) \\ &\qquad + \sum_{y \notin \mathcal{B}} \mathbf{P}(h(k) = y) d(H_R(k)|R = r, h(k) = y) \Big) \\ &\leq \sum_r \mathbf{P}(R = r) \Big(\sum_{y \in \mathcal{B}} \mathbf{P}(h(k) = y) \cdot 1 \\ &\qquad + \sum_{y \notin \mathcal{B}} \mathbf{P}(h(k) = y) d(H_R(k)|R = r, h(k) = y) \Big) \\ &\leq \sum_r \mathbf{P}(R = r) \left(|h(k)| \cdot 2^{-(1-\lambda)m} + \sum_{y \notin \mathcal{B}} \mathbf{P}(h(k) = y) \sqrt{2^n \cdot 2^{-\lambda m}} \right) \\ &\leq \sum_r \mathbf{P}(R = r) \left(2^{\phi m} \cdot 2^{-(1-\lambda)m} + \sqrt{2^n \cdot 2^{-\lambda m}} \right) \\ &= 2^{-((1-\phi)-\lambda)m} + 2^{-\lambda m/2 + n/2}.\end{aligned}$$

We could minimize the right hand side over $0 < \lambda < 1 - \phi$ by differentiating wrt λ and setting to zero. After some simplification we get the result. □

To prove security for multiple erasures, we need the following two lemmas (refer to [15] for the definition of mutual information, I):

Lemma 2 (Difference in Unconditional and Conditional Mutual Information is Symmetric). *For random variables X, Y, and Z,*

$$I(X;Y) - I(X;Y|Z) = I(X;Z) - I(X;Z|Y) = I(Y;Z) - I(Y;Z|X). \tag{2}$$

Proof.

$$\begin{aligned}I(X;Y) - I(X;Y|Z) &= (H(X) - H(X|Y)) - (H(X|Z) - H(X|Y,Z)) \\ &= (H(X) - H(X|Z)) - (H(X|Y) - H(X|Y,Z)) \\ &= I(X;Z) - I(X;Z|Y).\end{aligned}$$

The symmetric equality, reversing the roles of X and Y, can be obtained by noting that the proof above does not use anything particular about X, Y, or Z. □

Lemma 3 (Statistical Distance and Entropy). *If X is a random variable taking values in $\mathcal{X} := \{0,1\}^n$, and $d(X) \leq 1/4$, then*

$$n - H(X) \leq 2d(X) \log \frac{2^n}{2d(X)}, \tag{3}$$

and

$$\frac{2}{\ln 2} d(X)^2 \leq n - H(X). \tag{4}$$

Proof. This lemma is a special Case of Theorem 16.3.2 and Lemma 16.3.1 of [15]. □

Theorem 2 (Security for Multiple Erasures using Universal Hash). *Let $\{H_R\}$ be a family of universal hash functions. Let $(R_1, h(k_1)), ..., (R_\ell, h(k_\ell))$ be ℓ tuples such that $R_i \in \{0,1\}^{n \times m}, k_i \in \{0,1\}^m$, and $h(k_i) \in \{0,1\}^{\phi m}$, where R_i picks out a random function out of $\{H_R\}$, k_i is random, and q_i are public 1-1 correspondences such that $s = q_i(H_{R_i}(k_i))$. Then, for any $\beta > 0$, m poly in n, and sufficiently large n,*

$$d(H_{R_i}(k_i) | R_1, h(k_1), ..., R_\ell, h(k_\ell)) \leq \sqrt{\frac{\ln 2}{2} \ell 2^{-\frac{1}{3}(1-\phi)m + \frac{(\beta+1)n}{3} - \frac{1}{3}}}.$$

Proof. Since the family of universal hash functions that we consider, $\{H_R\}$, is always of the form $H_R(k) = R \cdot k$, we will just write $R \cdot k$ below, to prevent confusion from $H()$, the entropy function.

$$\begin{aligned}
I\big(R_i \cdot k_i; (R_i, h(k_i))\big) &= H(R_i \cdot k_i) - H(R_i \cdot k_i | R_i, h(k_i)) \\
&\leq n - H(R_i \cdot k_i | R_i, h(k_i)) \\
&\overset{(3)}{\leq} 2d(R_i \cdot k_i | R_i, h(k_i)) \log \frac{2^n}{2d(R_i \cdot k_i | R_i, h(k_i))} \\
&\overset{(1)}{\leq} 3 \cdot 2^{-\frac{1}{3}(1-\phi)m + \frac{n}{3} + \frac{1}{3}} \log \left(2^n \frac{1}{3} \cdot 2^{\frac{1}{3}(1-\phi)m - \frac{n}{3} - \frac{1}{3}} \right) \\
&= 3 \cdot 2^{-\frac{1}{3}(1-\phi)m + \frac{n}{3} + \frac{1}{3}} \left(\log \left(2^{\frac{1}{3}(1-\phi)m + \frac{2n}{3} - \frac{1}{3}} \right) + \log \frac{1}{3} \right) \\
&\leq 3 \cdot 2^{-\frac{1}{3}(1-\phi)m + \frac{n}{3} + \frac{1}{3}} \left(\frac{1}{3}(1-\phi)m + \frac{2n}{3} - \frac{1}{3} \right) \\
&= 2^{-\frac{1}{3}(1-\phi)m + \frac{n}{3} + \frac{1}{3}} \left((1-\phi)m + 2n - 1 \right) \\
&\leq 2^{-\frac{1}{3}(1-\phi)m + \frac{(\beta+1)n}{3} + \frac{1}{3}}.
\end{aligned} \tag{5}$$

The second inequality also requires the fact that for δ and ϵ such that $0 \leq \delta \leq \epsilon \leq 2^{-\frac{1}{\ln 2}}$ (which the right hand side of equation (1) satisfies for sufficiently large m), we have that $-\delta \log \delta \leq -\epsilon \log \epsilon$ [7]. The last inequality follows because, given ϕ and n, for any $0 < \beta$ and m poly in n, for sufficiently large n,

$$((1-\phi)m + 2n - 1) \leq 2^{\frac{\beta}{3}n}. \tag{6}$$

By chain rule for mutual information,

$$\begin{aligned}
&I\big(R_i \cdot k_i; (R_1, h(k_1)), ..., (R_\ell, h(k_\ell))\big) \\
&= \sum_{i=1}^{\ell} I\big(R_i \cdot k_i; (R_i, h(k_i)) \big| (R_1, h(k_1)), ..., (R_{i-1}, h(k_{i-1}))\big)
\end{aligned} \tag{7}$$

[7] This is because the function $f(x) := -x \log x$ is concave and positive, and has its maximum at $-x \cdot \frac{1}{x \ln 2} - \log x = 0 \Leftrightarrow x = 2^{-\frac{1}{\ln 2}}$, and so before this maximum, for $0 \leq \delta \leq \epsilon \leq 2^{-\frac{1}{\ln 2}}$, the function $-x \log x$ is monotonically increasing.

Also, because the difference in unconditional and conditional mutual information is symmetric in the random variables, (theorem 2 on page 9)

$$I(Y;X) - I(Y;X|Z) = I(X;Z) - I(X;Z|Y).$$

Setting $Y = R_i \cdot k_i, X = (R_i, h(k_i))$ and $Z = (R_1, h(k_1)), ..., (R_{i-1}, h(k_{i-1}))$, we have

$$\begin{aligned}
&I\big(R_i \cdot k_i; (R_i, h(k_i))\big) - I\big(R_i \cdot k_i; (R_i, h(k_i))\big|(R_1, h(k_1)), ..., (R_{i-1}, h(k_{i-1}))\big) \\
&= I\big((R_i, h(k_i)); (R_1, h(k_1)), ..., (R_{i-1}, h(k_{i-1}))\big) \\
&\quad - I\big((R_i, h(k_i)); (R_1, h(k_1)), ..., (R_{i-1}, h(k_{i-1}))\big|R_i \cdot k_i\big) \\
&= I\big((R_i, h(k_i)); (R_1, h(k_1)), ..., (R_{i-1}, h(k_{i-1}))\big) \geq 0,
\end{aligned}$$

where the last equality uses the fact that q_is are public bijections of the $R_i \cdot k_i$s to one another.

Therefore, $I\big(R_i \cdot k_i; (R_i, h(k_i))\big|(R_1, h(k_1)), ..., (R_{i-1}, h(k_{i-1}))\big) \leq I\big(R_i \cdot k_i; (R_i, h(k_i))\big)$.

Substituting into equation 7, we get

$$\begin{aligned}
I\big(R_i \cdot k_i; (R_1, h(k_1)), ..., (R_\ell, h(k_\ell))\big) &\leq \sum_{i=1}^{\ell} I\big(R_i \cdot k_i; (R_i, h(k_i))\big) \\
&\overset{(5)}{=} \ell 2^{-\frac{1}{3}(1-\phi)m + \frac{(\beta+1)n}{3} + \frac{1}{3}}.
\end{aligned} \tag{8}$$

Also,

$$\begin{aligned}
&n - H(R_i \cdot k_i) \overset{(3)}{\leq} 2d(R_i \cdot k_i) \log \frac{2^n}{2d(R_i \cdot k_i)} \\
\Leftrightarrow\ &H(R_i \cdot k_i) \geq n - 2d(R_i \cdot k_i) \log \frac{2^n}{2d(R_i \cdot k_i)} \\
\Leftrightarrow\ &H(R_i \cdot k_i) \geq n - 2d(R_i \cdot k_i)n - 2d(R_i \cdot k_i) \log \frac{1}{2d(R_i \cdot k_i)} \\
\Rightarrow\ &H(R_i \cdot k_i) \geq n - 2d(R_i \cdot k_i|R_i, h(k_i))n - 2d(R_i \cdot k_i|R_i, h(k_i)) \log \frac{1}{2d(R_i \cdot k_i|R_i, h(k_i))} \\
\Rightarrow\ &H(R_i \cdot k_i) \overset{(1)}{\geq} n - 2\frac{3}{2^{2/3}} \cdot 2^{-\frac{1}{3}(1-\phi)m + \frac{n}{3}} n - 2\frac{3}{2^{2/3}} \cdot 2^{-\frac{1}{3}(1-\phi)m + \frac{n}{3}} \log \frac{1}{2\frac{3}{2^{2/3}} \cdot 2^{-\frac{1}{3}(1-\phi)m + \frac{n}{3}}} \\
\Leftrightarrow\ &H(R_i \cdot k_i) \geq n - 2^{-\theta}(n + \theta) \\
\Leftrightarrow\ &n \leq H(R_i \cdot k_i) + 2^{-\theta}(n + \theta),
\end{aligned} \tag{9}$$

where $\theta := \frac{1}{3}(1-\phi)m - \frac{n}{3} - \frac{1}{3} - \log_2(3)$.

Putting it all together,

$$\begin{aligned}
& d\big(R_i \cdot k_i \big| (R_1, h(k_1)), ..., (R_\ell, h(k_\ell))\big) \\
\overset{(4)}{\leq} & \sqrt{\frac{\ln 2}{2}\Big(n - H\big(R_i \cdot k_i \big| (R_1, h(k_1)), ..., (R_\ell, h(k_\ell))\big)\Big)} \\
\overset{(9)}{\leq} & \sqrt{\frac{\ln 2}{2}\Big(H(R_i \cdot k_i) + 2^{-\theta}(n+\theta) - H\big(R_i \cdot k_i \big| (R_1, h(k_1)), ..., (R_\ell, h(k_\ell))\big)\Big)} \\
= & \sqrt{\frac{\ln 2}{2}\Big(I\big(R_i \cdot k_i; (R_1, h(k_1)), ..., (R_\ell, h(k_\ell))\big)\Big) + 2^{-\theta}(n+\theta)} \\
\overset{(8)}{\leq} & \sqrt{\frac{\ln 2}{2}\ell 2^{-\frac{1}{3}(1-\phi)m + \frac{(\beta+1)n}{3} + \frac{1}{3}} + 2^{-\theta}(n+\theta)} \\
\overset{(6)}{\leq} & \sqrt{\frac{\ln 2}{2}\ell 2^{-\theta + \frac{\beta n}{3} - \log_2(3)} + 2^{-\theta + \frac{\beta n}{3}}} \\
\leq & \sqrt{(\ln 2)\ell 2^{-\frac{1}{3}(1-\phi)m + \frac{(\beta+1)n}{3} + \frac{1}{3}}},
\end{aligned}$$

where the last inequality holds for large n and m. □

Note that to get $2^{-(\alpha+1)}$ security when the adversary gets ℓ partially erased tuples, we need:

$$\begin{aligned}
& \sqrt{\frac{\ln 2}{2}\ell 2^{-\frac{1}{3}(1-\phi)m + \frac{(\beta+1)n}{3} - \frac{1}{3}}} \leq 2^{-(\alpha+1)} \\
\Leftrightarrow \ell \leq & \frac{2^{\frac{4}{3} - 2(\alpha+1) - \frac{1}{3}(1+\beta)n + \frac{1}{3}m(1-\phi)}}{\ln 2} \qquad (10) \\
\Leftrightarrow m \geq & \frac{3\log\left((\ln 2)\ell\right) - 4 + 6(\alpha+1) + (1+\beta)n}{(1-\phi)} \qquad (11)
\end{aligned}$$

Let us make a few observations. Inequality 10 shows that if h has leakage fraction ϕ, how many times can you partially erase a secret (or computations on the secret) without leaking too much information. Rearranging, and fixing the other parameters, we can also see that the fraction that needs to be erased, $(1-\phi)$, has to be at least logarithmic in ℓ. Inequality 11 on the other hand lower bounds m, which as we will see shortly, translates into a statement about the space efficiency of using universal hashing to get partially erasable forms.

Note that in the case where the secret s we are dealing with is a random one, we can save n-bits in the expansion and just use $\mathsf{Exp}(s, \$) := (R, k)$ where R and k are random s.t. $R \cdot k = s$ [8]. This can be done easily for a random R or even a random Toeplitz R. The theorem applies even to this case, i.e. however way the secrets are chosen to be dependent on one another, any secret itself is still close to random. Note that we give the adversary bijections that link one secret to another, and thus all the pairs (R_i, k_i) are dependent (which in turn implies that the pairs $(R_i, h(k_i))$ are dependent too), and in particular we cannot just do a union bound.

Let us now consider two partially erasable forms based on universal hashing which satisfy correctness, secrecy and moreover can be computed with constant size memory. The expansions we consider can be thought of as having two parts, R, k, each serving different purposes. Furthermore, only one part, k, needs to be partially erased[9].

The first expanded form of s is random matrix $R \in \{0,1\}^{n \times m}$ and vector $k \in \{0,1\}^m$ subject to the constraint that $R \cdot k = s$. Only the vector k needs to be erased. However, this simple construction is highly randomness inefficient.

[8] In fact this type of expansion is what we are going to use later in section 5.1 when we give a special compiler for all known proactive secret sharing schemes.

[9] which makes our results stronger than required by the definition

Our preferred partial erasable form will be to use Toeplitz hashing instead (whose universality is proven in [51]). A random Toeplitz matrix $R \in \{0,1\}^{n\times m}$ which selects a random hash function out of the Toeplitz family $\{H_R\}$, where $H_R : \{0,1\}^m \mapsto \{0,1\}^n$. In order to store an n-bit secret s in a partially erasable manner, we choose random Toplitz matric $R \in \{0,1\}^{n\times m}, k \in \{0,1\}^m$ (where R is fully specified by a random string of $n+m-1$ bits), and store $R, k, R\cdot k \oplus s$ instead. Again, only the k part needs to be erased. In this case $\mathsf{Exp}(s,r)$ uses r to form a random Toeplitz R and a random k, and outputs $R, k, R\cdot k \oplus s$, and $\mathsf{Con}(R,k,x)$ recovers s by computing $R\cdot k \oplus x$.

It is easy to see how to implement $(\mathsf{Exp}, \mathsf{Con})$ corresponding to Toeplitz hash that is computable with constant memory, bit by bit. From the triangle inequality, for all $s, s' \in \{0,1\}^n$,

$$\Delta\Big(H_R(k)\oplus s, R, h(k); H_R(k)\oplus s', R, h(k)\Big) \le 2d(H_R(k)|R, h(k)). \tag{12}$$

Combining these with theorem 2 proves that:

Corollary 1 (Toeplitz Hashing gives a Partially Erasable Form). *Toeplitz hashing yields a partially erasable form of a secret.*

3.2 Using $\frac{1}{2^n} + \epsilon$-almost Universal Hash

An $\frac{1}{2^n} + \epsilon$-almost universal hash is one in which the collision probability is upper bounded by $\frac{1}{2^n} + \epsilon$:

Definition 7 ($\frac{1}{2^n} + \epsilon$-almost Universal Hash Functions). *We say that $\mathcal{H}$ is an $\frac{1}{2^n} + \epsilon$-almost universal family of hash functions from $\mathcal{X}$ to $\mathcal{Y}$ if for all $x, x' \in \mathcal{X}$ such that $x \neq x'$, and H uniform from $\mathcal{H}$, we have that*

$$\mathbf{P}_H\left(H(x) = H(x')\right) \le \frac{1}{2^n} + \epsilon.$$

Lemma 4 (Leftover Hash Lemma for $\frac{1}{2^n}+\epsilon$-almost Universal Hash Functions). *Let $\mathcal{H}$ be an $\frac{1}{2^n}+\epsilon$-almost universal family of hash functions from $\mathcal{X}$ to $\mathcal{Y}$. Let H denote a random variable with the uniform distribution on $\mathcal{H}$, and let X denote a random variable taking values in $\mathcal{X}$, with H, X independent. Then $(H, H(X))$ is δ-uniform on $\mathcal{H}\times\mathcal{Y}$, where*

$$\delta \le \sqrt{|\mathcal{Y}|\kappa(X) + |\mathcal{Y}|(\frac{1}{2^n}+\epsilon) - 1}/2.$$

Proof. The proof is analogous to that of theorem 1. □

Definition 8 (ϵ-biased Distributions [55]). *Let X be a distribution on $\{0,1\}^q$. Let $\langle x, y\rangle$ denote the inner product mod 2 of $x \in \{0,1\}^q$ and $y \in \{0,1\}^q$. Then,*

1. *X is said to pass the linear test y with bias ϵ if $|\mathbf{P}_{x\leftarrow X}(\langle x, y\rangle) = 1) - \frac{1}{2}| \le \epsilon$.*
2. *X is said to be an ϵ-biased distribution if it passes all linear tests $a \neq 0$ with bias ϵ.*

The following theorem, implied by theorem 14 of [46], proves that if the Toeplitz matrix is generated not from $n+m-1$ random bits but just r random bits which give a ϵ-biased distribution, where $\epsilon = \frac{n+m-1}{2^{r/2}}$, then the resulting hash function family is $\frac{1}{2^n} + \epsilon$-almost universal.

Theorem 3 (ϵ-biased Generation of Toeplitz Matrices of Dimension $n\times m$ gives $\frac{1}{2^n} + \epsilon$-almost Universal Hash Functions [46]). *Consider any construction that would generate $\epsilon = \frac{n+m-1}{2^{r/2}}$-biased distributions on sequences of length $n+m-1$, using r initial random bits [1]. The Toeplitz matrix that corresponds to these $n+m-1$ bits gives rise to an $\frac{1}{2^n}+\epsilon$-almost universal hash function family from $\{0,1\}^m$ to $\{0,1\}^n$.*

Theorem 4 (Secrecy for Single Erasures using $\frac{1}{2^n}+\epsilon$-almost Universal Hash). *Let $s \in \{0,1\}^n$ be any n bit string. Let $(R, h(k))$ be a partially erased tuple such that $R \in \{0,1\}^{n\times m}, k \in \{0,1\}^m$, and $h(k) \in \{0,1\}^{\phi m}$, where R is an $\epsilon = \frac{n+m-1}{2^{r/2}}$-biased generated Toeplitz matrix (using r initial random bits). Then,*

$$d(R\cdot k|R, h(k)) \leq \frac{3}{2^{2/3}} \cdot 2^{-\frac{1}{3}(1-\phi)m+\frac{n}{3}} + \sqrt{n+m-1}\; 2^{\frac{1}{2}(n-r/2)}. \tag{13}$$

Proof. The proof idea is similar to the one using universal hash; differences will be pointed out below.

First, let us show that with high probability, k still has high min entropy given $h(k)$. Intuitively, rare events give more information to the adversary. Accordingly, let λ be a real number such that $0 < \lambda < 1-\phi$, and define $\mathcal{B}$, the "bad" set, to be the set of realizations y of $h(k)$ such that $\mathbf{P}_k(h(k) = y) \leq 2^{-(1-\lambda)m}$.

So, for $y_0 \notin \mathcal{B}$,

$$\begin{aligned}
\mathbf{P}(k = k_0 | h(k) = y_0) &= \frac{\mathbf{P}(k = k_0 \cap h(k) = y_0)}{\mathbf{P}(h(k) = y_0)} \\
&\leq \frac{\mathbf{P}(k = k_0)}{\mathbf{P}(h(k) = y_0)} \\
&\leq 2^{-m} \cdot 2^{(1-\lambda)m} \\
&= 2^{-\lambda m}.
\end{aligned}$$

Therefore, for $h(k) \notin \mathcal{B}$, $H_\infty(k|h(k)) \geq \lambda m$, and by theorem 4 (instead of 1) and lemma **??**, for $h(k) \notin \mathcal{B}$, $d(R\cdot k|R, h(k)) \leq \sqrt{2^n \cdot 2^{-\lambda m} + 2^n \cdot (\frac{1}{2^n} + \frac{n+m-1}{2^{r/2}}) - 1}$, and we can bound the statistical distance of $R \cdot k$ from uniform:

$$\begin{aligned}
d(R\cdot k|R, h(k)) &= \sum_{r,y} \mathbf{P}(R = r \cap h(k) = y) d(R\cdot k|R = r, h(k) = y) \\
&= \sum_r \mathbf{P}(R = r) \Big(\sum_{y\in\mathcal{B}} \mathbf{P}(h(k) = y) d(R\cdot k|R = r, h(k) = y) \\
&\qquad + \sum_{y\notin\mathcal{B}} \mathbf{P}(h(k) = y) d(R\cdot k|R = r, h(k) = y) \Big) \\
&\leq \sum_r \mathbf{P}(R = r) \Big(\sum_{y\in\mathcal{B}} \mathbf{P}(h(k) = y) \cdot 1 \\
&\qquad + \sum_{y\notin\mathcal{B}} \mathbf{P}(h(k) = y) d(R\cdot k|R = r, h(k) = y) \Big) \\
&\leq \sum_r \mathbf{P}(R = r) \Big(|h(k)| \cdot 2^{-(1-\lambda)m} \\
&\quad + \sum_{y\notin\mathcal{B}} \mathbf{P}(h(k) = y) \sqrt{2^n \cdot 2^{-\lambda m} + 2^n \cdot \frac{n+m-1}{2^{r/2}}} \Big) \\
&\leq \sum_r \mathbf{P}(R = r) \left(2^{\phi m} \cdot 2^{-(1-\lambda)m} + \sqrt{2^n \cdot 2^{-\lambda m} + 2^n \cdot \frac{n+m-1}{2^{r/2}}} \right) \\
&= 2^{-((1-\lambda)-\phi)m} + \sqrt{2^n \cdot 2^{-\lambda m} + 2^n \cdot \frac{n+m-1}{2^{r/2}}}
\end{aligned}$$

Minimizing the right hand side over $0 < \lambda < 1-\phi$ by differentiating wrt λ and setting to zero, involves solving a quartic polynomial, resulting in a formula that can barely fit in two pages. So for simplicity we are going to cut some slack before optimizing:

$$d(R\cdot k|R,h(k)) \le 2^{\phi m}\cdot 2^{-(1-\lambda)m} + \sqrt{2^n\cdot 2^{-\lambda m} + 2^n\cdot\frac{n+m-1}{2^{r/2}}}$$
$$\le 2^{-((1-\lambda)-\phi)m} + \sqrt{2^n\cdot 2^{-\lambda m}} + \sqrt{2^n\cdot\frac{n+m-1}{2^{r/2}}}$$
$$= 2^{-((1-\lambda)-\phi)m} + 2^{\frac{1}{2}(n-\lambda m)} + \sqrt{n+m-1}\;2^{\frac{1}{2}(n-r/2)}$$

The second inequality holds since $\sqrt{a+b}\le\sqrt{a}+\sqrt{b}$ for $a,b\ge 0$.

Minimizing the right hand side over $0<\lambda<1-\phi$ by differentiating wrt λ and setting to zero we get the same λ^* as in theorem 1, because the extra term does not involve λ^*. So we get:

$$\lambda^* = \frac{2}{3}(1-\phi)+\frac{n}{3m}-\frac{2}{3m}.$$

Therefore,

$$d(R\cdot k|R,h(k)) \le \frac{3}{2^{2/3}}\cdot 2^{-\frac{1}{3}(1-\phi)m+\frac{n}{3}} + \sqrt{n+m-1}\;2^{\frac{1}{2}(n-r/2)}.$$

□

The second term, $\sqrt{n+m-1}\;2^{\frac{1}{2}(n-r/2)}$, can be seen as an overhead to using an $\epsilon=\frac{1}{2^n}+\frac{n+m-1}{2^{r/2}}$-almost universal hash instead of a universal one. This extra term will make it difficult to upper-bound the multi-period statistical distance, so let us simplify it first.

This second term can be upper-bounded by

$$2^{\frac{1}{2}((1+\beta)n-r/2)},$$

for any $\beta>0$ and n sufficiently large.

Intuitively, the value of r that would minimize 13 is one where the two terms are roughly equal. If we equate $\frac{3}{2^{2/3}}\cdot 2^{-\frac{1}{3}(1-\phi)m+\frac{n}{3}}$ and $2^{\frac{1}{2}((1+\beta)n-r/2)}$, we get that

$$r = 2(\frac{1}{3}+\beta)n+\frac{4}{3}(1-\phi)m-4\log(\frac{3}{2^{2/3}}). \tag{14}$$

Since r is the number of bits used to generate a Toeplitz of dimension $n\times m$, we know that $r\le n+m-1$.

Therefore,

$$2(\frac{1}{3}+\beta)n+\frac{4}{3}(1-\phi)m-4\left(\log(\frac{3}{2^{2/3}})\right)\le n+m-1$$
$$\Leftrightarrow 2(-\frac{1}{3}+\beta)n+\frac{1}{3}(1-4\phi)m-4\left(\log(\frac{3}{2^{2/3}})-\frac{3}{4}\right)\le 0,$$

which, for n and m growing (and β small), can only hold if $\phi>\frac{1}{4}$. In this case, we get the simplified bound of

$$d(R\cdot k|R,h(k)) \le 3\cdot 2^{-\frac{1}{3}(1-\phi)m+\frac{n}{3}+\frac{1}{3}}. \tag{15}$$

When compared to the case when universal hash functions are used, this represents a factor of 2 loss in security; or to put it differently, m has to be larger to achieve the same level of security.

Theorem 5 (Secrecy for Multiple Erasures using $\frac{1}{2^n}+\epsilon$-almost Universal Hash). *Let $(R_1,h(k_1)),...,(R_\ell,h(k_\ell))$ be ℓ tuples such that $R_i\in\{0,1\}^{n\times m}, k_i\in\{0,1\}^m$, and $h(k_i)\in\{0,1\}^{\phi m}$, where each R_i is an $\epsilon=\frac{n+m-1}{2^{r/2}}$-biased generated Toeplitz matrix (using r initial random bits). Let q_i be public 1-1 correspondences such that $R_1\cdot k_1 = q_i(R_i\cdot k_i)$. Then, for any $\beta>0, m$ poly in n, and sufficiently large n,*

$$d(R_i\cdot k_i|R_1,h(k_1),...,R_\ell,h(k_\ell)) \le \sqrt{(\ln 2)\ell 2^{-\frac{1}{3}(1-\phi)m+\frac{(\beta+1)n}{3}+\frac{4}{3}}}. \tag{16}$$

Proof. The proof is analogous to that of theorem 2. □

Therefore, to get $2^{-(\alpha+1)}$ security when the adversary gets ℓ partially erased tuples, it is sufficient that

$$\sqrt{(\ln 2)\ell 2^{-\frac{1}{3}(1-\phi)m+\frac{(\beta+1)n}{3}+\frac{4}{3}}} \leq 2^{-(\alpha+1)}$$

$$\Leftrightarrow \ell \leq \frac{2^{-\frac{4}{3}-2(\alpha+1)-\frac{1}{3}(1+\beta)n+\frac{1}{3}m(1-\phi)}}{\ln 2} \tag{17}$$

$$\Leftrightarrow m \geq \frac{3\log\left((\ln 2)\ell\right)-2+6(\alpha+1)+(1+\beta)n}{(1-\phi)}. \tag{18}$$

Calling the bounds on ℓ and m in the case of using universal hash functions ℓ' and m' respectively, we see that

$$\ell \leq 2\ell' \tag{19}$$

$$\Leftrightarrow m \geq m' + \frac{3}{(1-\phi)}. \tag{20}$$

Corollary 2 ($\frac{1}{2^n}+\epsilon$-almost Universal Hashing gives a Partially Erasable Form). *Consider a $\frac{1}{2^n}+\epsilon$-almost universal family of hash functions $\{H_R\}$. Consider $\mathsf{Exp}(s,\$) := \left(R, k, H_R(k)\oplus s\right)$ and $\mathsf{Con}(R,k,x) := H_R(k)\oplus x$, where $R\in\{0,1\}^{n\times m}$, $k\in\{0,1\}^m$, and $s\in\{0,1\}^n$. Then $(\mathsf{Exp},\mathsf{Con})$ is a partially erasable form. More precisely, given $0\leq\phi<1$, any $\alpha>0$, any $\beta>0$, any $\ell>0$, and any m that satisfies 18, $(\mathsf{Exp},\mathsf{Con})$ is an (ℓ,α,ϕ)-partially erasable form.*

Proof. The proof is analogous to the proof of corollary 1. □

3.3 Using Strong Extractors

Let us first review the some definitions. Extractors were introduced by Nisan and Zuckerman [56], and have played a unifying role in the theory of pseudorandomness.

Definition 9 (Extractors). *A function $\mathsf{Ext}:\{0,1\}^n\times\{0,1\}^d\mapsto\{0,1\}^m$ is a (k,ϵ)-extractor if for any k-source X over $\{0,1\}^n$, the distribution $\mathsf{Ext}(X,U_d)$ (the extractor's output on an element sampled from X and a uniformly chosen d bit string) is ϵ-close to U_m.*

Definition 10 (Strong Extractors). *A (k,ϵ)-extractor is strong if for any k-source X over $\{0,1\}^n$, the distribution $(\mathsf{Ext}(X,U_d)\circ U_d)$ (obtained by concatenating the seed to the output of the extractor) is ϵ-close to U_{m+d}.*

The following theorems from [48] give extractors that are simultaneously optimal (up to constant factors) in both seed length and output length.

Theorem 6 (Optimal Extractors for Constant ϵ [48]). *For any constants $\alpha,\epsilon>0$, every n and every $k\leq n$, there is an explicit (k,ϵ)-extractor $\mathsf{Ext}:\{0,1\}^n\times\{0,1\}^d\mapsto\{0,1\}^m$, with $d=O(\log n)$ and $m=(1-\alpha)k$.*

Theorem 7 (Optimal Extractors for $\epsilon>\exp(-k/2^{O(\log^* k)})$ [48]). *For any constant $\alpha\in(0,1), c\in\mathbb{N}$, for every k, and every $\epsilon\in(0,1)$ where $\epsilon>\exp(-k/2^{O(\log^* k)})$, there are explicit (k,δ)-extractors $\mathsf{Ext}:\{0,1\}^n\times\{0,1\}^d\mapsto\{0,1\}^m$, with each one of the following parameters:*

- *$d=O(\log n), m=(1-\alpha)k$, and $\delta=(1/n)^{1/\log^{(c)} n}$.*
- *$d=O((\log^* n)^2\log n+\log(1/\delta)), m=(1-\alpha)k$, and $\delta=\epsilon$.*
- *$d=O(\log(n/\delta)), m=\Omega(k/\log^{(c)} n)$, and $\delta=\epsilon$.*

The following theorem from [60] shows that every non-strong extractor can be transformed into one that is strong with essentially the same parameters.

Theorem 8 (Strong Extractors from Extractors [60]). *Any explicit (k,ϵ)-extractor $\mathsf{Ext} : \{0,1\}^n \times \{0,1\}^d \mapsto \{0,1\}^m$ can be transformed into an explicit strong $(k, O(\sqrt{\epsilon}))$-extractor $\mathsf{Ext}' : \{0,1\}^n \times \{0,1\}^{O(d)} \mapsto \{0,1\}^{m-d-\Delta-1}$, where $\Delta = 2\log(1/\epsilon) + O(1)$.*

Combining theorem 8 with theorems 6 and 7, we get the following.

Theorem 9 (Near Optimal Strong Extractors for Constant ϵ). *For any constants $\alpha, \epsilon > 0$, every n and every $k \leq n$, there is an explicit strong $(k, O(\sqrt{\epsilon}))$-extractor $\mathsf{Ext} : \{0,1\}^n \times \{0,1\}^d \mapsto \{0,1\}^m$, with $d = O(\log n)$ and $m = (1-\alpha)k - d - O(1)$.*

Theorem 10 (Near Optimal Strong Extractors for $\epsilon > \exp(-k/2^{O(\log^* k)})$). *For any constant $\alpha \in (0,1), c \in \mathbb{N}$, for every k, and every $\epsilon \in (0,1)$ where $\epsilon > \exp(-k/2^{O(\log^* k)})$, there are explicit strong (k,δ)-extractors $\mathsf{Ext} : \{0,1\}^n \times \{0,1\}^d \mapsto \{0,1\}^m$, with each one of the following parameters:*

- *$d = O(\log n), m = (1-\alpha)k - d - \frac{k}{2^{O(\log^* k)}} - O(1)$, and $\delta = (1/n)^{1/\log^{(c)} n}$.*
- *$d = O((\log^* n)^2 \log n + \log(1/\delta)), m = (1-\alpha)k - d - \frac{k}{2^{O(\log^* k)}} - O(1)$, and $\delta = \epsilon$.*
- *$d = O(\log(n/\delta)), m = \Omega(k/\log^{(c)} n) - d - \frac{k}{2^{O(\log^* k)}} - O(1)$, and $\delta = \epsilon$.*

Now let us look at the secrecy part of a candidate partially erasable form based on strong extractors.

Theorem 11 (Secrecy for a Single Erasure using Strong Extractors). *Let $s \in \{0,1\}^n$ be any string. Let $(R, h(k))$ be a partially erased tuple such that $R \in \{0,1\}^{\mathcal{D}}, k \in \{0,1\}^{\mathcal{N}}$, and $h(k) \in \{0,1\}^{\phi\mathcal{N}}$, where R is the seed of a $(\mathcal{K},\delta)$ strong extractor $\mathsf{Ext} : \{0,1\}^{\mathcal{N}} \times \{0,1\}^{\mathcal{D}} \mapsto \{0,1\}^{\mathcal{M}}$, $0 < \mathcal{K} < (1-\phi)\mathcal{N}$. Then,*

$$d(R \cdot k | R, h(k)) \leq 2^{-((1-\phi)\mathcal{N}-\mathcal{K})} + \delta. \tag{21}$$

Proof. The proof is analogous to that of theorem 1 on page 8. □

Theorem 12 (Secrecy for Multiple Erasures using Strong Extractors). *Let $s \in \{0,1\}^n$ be any string. Let $(R_1, h(k_1)), ..., (R_\ell, h(k_\ell))$ be ℓ tuples such that $R_i \in \{0,1\}^{\mathcal{D}}, k_i \in \{0,1\}^{\mathcal{N}}$, and $h(k_i) \in \{0,1\}^{\phi\mathcal{N}}$, where R_i is the seed of a $(\mathcal{K},\delta)$ strong extractor $\mathsf{Ext} : \{0,1\}^{\mathcal{N}} \times \{0,1\}^{\mathcal{D}} \mapsto \{0,1\}^{\mathcal{M}}$, $0 < \mathcal{K} < (1-\phi)\mathcal{N}$. Let q_i be public 1-1 correspondences such that $\mathsf{Ext}(R_1, k_1) = q_i(\mathsf{Ext}(R_i, k_i))$. Then, for any $\beta > 0, m$ poly in n, and sufficiently large n,*

$$d(\mathsf{Ext}(R_i, k_i) | R_1, h(k_1)), ..., (R_\ell, h(k_\ell)) \leq \sqrt{(\ln 2)\ell 2^{\beta n + 1 - \log 3} 2^{-((1-\phi)\mathcal{N}-\mathcal{K})} + \delta}. \tag{22}$$

Proof. The proof is analogous to that of theorem 2 on page 10. □

Corollary 3 (Strong Extractors in NC^0 gives a Partially Erasable Form). *Consider a $(\mathcal{K},\delta)$-strong extractor $\mathsf{Ext} : \{0,1\}^{\mathcal{D}} \times \{0,1\}^m \mapsto \{0,1\}^n$ that is in NC^0. Consider $\mathsf{Exp}(s, \$) := \big(R, k, \mathsf{Ext}(R,k) \oplus s\big)$ and $\mathsf{Con}(R, k, x) := \mathsf{Ext}(R,k) \oplus x$. This is a partially erasable form computable with constant memory.*

Proof. The proof is straightforward using theorem 12 and inequality 12, and noting that, for a function in NC^0, each output bit depends only on a constant number of input bits. □

We note that requiring an extractor to be in NC^0 is sufficient but not necessary, and also that locally computable strong extractors [63] cannot be used here.

Theorem 13 (Locally Computable Strong Extractors does not give a Partially Erasable Form). *Consider a $(\mathcal{K},\delta)$-strong extractor $\mathsf{Ext} : \{0,1\}^{\mathcal{D}} \times \{0,1\}^m \mapsto \{0,1\}^n$ that is t local. Consider $\mathsf{Exp}(s, \$) := (R, k, \mathsf{Ext}(R,k) \oplus s)$ and $\mathsf{Con}(R, k, x) := \mathsf{Ext}(R,k) \oplus x$, as a candidate partially erasable form of any secret $s \in \{0,1\}^n$, using the t local strong extractor Ext. This is not a partially erasable form because both Exp and Con) are not computable with constant memory; in particular, for our setting, t has to be non-constant.*

Proof. Consider Ext, a t-local strong $(\mathcal{K}, \delta)$-extractor, and say the adversary chooses h to be equally distributed - for example, it chooses to keep only the first ϕm bits of k. Clearly, in that case $H_\infty(k \mid h(k)) = (1-\phi)\cdot m$. Thus, we need Ext to be a strong $((1-\phi)\cdot m, \delta)$-extractor. According to corollary 9.2 of [63], if Ext is a t-local strong $(\epsilon m, \delta)$-extractor, then $t \geq (1-\delta-2^{-n})\cdot(1/\epsilon)\cdot n$. In our case, $\epsilon = 1-\phi$, so it is required that $t \geq (1-\delta-2^{-n})\cdot(1/(1-\phi))\cdot n$. Even if ϕ were a constant, t would still be greater than some constant c times n, contrary to our demand of a fixed, constant length t. □

3.4 Space Efficiency

Lower Bound. Say that an expansion function Exp is Ψ-expanding if for any r we have $|\mathsf{Exp}(s,r)| \leq \Psi|s|$. One parameter we would like to minimize is Ψ, the storage overhead, whose lower bound is given below:

Theorem 14 (Lower Bound on the Storage Expansion Ψ). *For any Ψ-expanding, (ℓ, α, ϕ)-partially erasable expansion function* Exp *that is applied to inputs of length n we have: $\Psi \geq \frac{1}{1-\phi}\left(1 - \frac{n+\alpha-1}{n\ell 2^{\alpha-1}}\right)$.*

Proof. In our model, to compile a scheme using perfect erasures to one with partial erasures, we replace each secret of n bits by the expanded form of it, of Ψn bits. This is partially erased to give $\phi\Psi n$ bits, i.e. $(1-\phi)\Psi n$ bits of information are erased.

If no information is to be gained by the adversary, the number of bits of information erased has to be at least n, so $\Psi \geq \frac{1}{(1-\phi)}$.

On the other hand if we allow the adversary to get small amounts of information, we need the following from [15]: for a random variable X taking values in $\{0,1\}^n$, and $d(X) \leq 1/4$ we have:

$$n - H(X) \leq -2d(X)\log\frac{2d(X)}{2^n} \tag{23}$$

Since the function $-y\log y$ is concave and positive for $0 \leq y \leq 1$, and attains its maximum at $y = 2^{-\frac{1}{\ln 2}}$, we know that before this maximum, $-y\log y$ is monotonically increasing. Therefore, as long as $2d(X) \leq 2^{-\alpha+1} \leq 2^{-\frac{1}{\ln 2}}$ (or equivalently that $\alpha \geq 1 + \frac{1}{\ln 2}$), a necessary condition for $d(X) \leq 2^{-\alpha}$ is that:

$$\begin{aligned} n - H(X) &\leq -2d(X)\log 2d(X) + 2d(X)n \\ &\leq -2^{-\alpha+1}(-\alpha+1) + 2^{-\alpha+1}n \\ &= (n+\alpha-1)2^{-\alpha+1}, \end{aligned}$$

or,

$$H(X) \geq n(1-2^{-\alpha+1}) - (\alpha-1)2^{-\alpha+1}$$

In other words, to achieve $2^{-\alpha}$ security when the adversary is given ℓ partially erased tuples, we need:

$$\begin{aligned} &\ell(1-\phi)\Psi n \geq n(\ell - 2^{-\alpha+1}) - (\alpha-1)2^{-\alpha+1} \\ \Leftrightarrow\ &\Psi \geq \frac{1}{1-\phi}\left(1 - \frac{n+\alpha-1}{n\ell 2^{\alpha-1}}\right). \end{aligned}$$

□

For typical settings of the parameters, where both α and ℓ are polynomial in n, we get that $\Psi \geq \frac{1}{1-\phi}(1-\mathrm{neg}(\alpha))$.

Efficiency. Let us see how tight our construction is to the lower bound.

Random R If a completely random R is used and $H_R := R \cdot k$ (whose universality is proven in [14]), then the expansion factor Ψ of the storage would be (size of R + size of k) $\cdot \frac{1}{n}$, which is $(n+1)m \cdot \frac{1}{n} = (1+\frac{1}{n})m$. Plugging this into inequality 11, we see that this bound is a (growing) factor of n away from the optimal.

Random Toeplitz R If a Toeplitz matrix R is used instead, then the corresponding expanded form will be $R \in \{0,1\}^{n \times m}, k \in \{0,1\}^m$ and $x \in \{0,1\}^n$ such that $R \cdot k \oplus x = s$. In this case, R requires $n+m-1$ bits to specify (since it is Toeplitz), and k requires m bits and x requires n bits respectively. So in this case, n bits get expanded into $2m+2n-1$ bits, and Ψ is $2\frac{m}{n} + 2 - \frac{1}{n}$. Plugging in inequality 11, we see that for $\alpha = O(n)$ and $\ell = 2^{O(n)}$, then this bound is a constant factor away from the optimal given in theorem 14. If ℓ is subexponential in n and α is sublinear in n, then the bound we get is about $\Psi \geq \frac{2}{1-\phi} + 2$, so it is essentially a factor of 2 away from the optimal bound.

ϵ-biased Toeplitz R If an $\frac{1}{2^n} + \epsilon$ universal hash is used, the total number of bits required to store an n bit secret in a partially erasable manner is $r+m+n$, which is:

$$\begin{aligned} r+m+n &\overset{(14)}{=} 2(\frac{1}{3}+\beta)n + \frac{4}{3}(1-\phi)m - 4\log(\frac{3}{2^{2/3}}) + m + n \\ &= (\frac{5}{3}+2\beta)n + (\frac{7}{3}-\frac{4}{3}\phi)m - 4\log\frac{3}{2^{2/3}}. \end{aligned}$$

Therefore, if ℓ is subexponential in n and α sublinear in n, then the bound we get on the expansion factor is:

$$\begin{aligned} \Psi &= (\frac{5}{3}+2\beta) + (\frac{7}{3}-\frac{4}{3}\phi)\frac{m}{n} - \frac{4}{n}\log\frac{3}{2^{2/3}} \\ &\approx (\frac{7}{3}-\frac{4}{3}\phi)\frac{m}{n} \\ &\overset{(18)}{\geq} (\frac{7}{3}-\frac{4}{3}\phi)\frac{1}{1-\phi} \end{aligned}$$

However, if $1 > \phi > 1/4$, we have that $1 < (\frac{7}{3} - \frac{4}{3}\phi) < 2$. Therefore the bound we obtain here is about $\Psi \geq \frac{c}{1-\phi}$, where $c < 2$, compared to $\Psi \geq \frac{2}{1-\phi}$ for universal hashing. If ϕ is very close to 1, then c is close to 1 to and the storage expansion is thus very close to optimal.

For the other case, $\phi \leq \frac{1}{4}$, to achieve the same level of security while minimizing $d(R \cdot k | R, h(k))$, the best we can do is to set $r = n+m-1$. In this case we do not gain anything in terms of the storage efficiency. Therefore, if $\phi \leq \frac{1}{4}$, the only reason to use the ϵ-biased construction instead of the universal one is to have the flexibility to tradeoff security with the space efficiency.

Near Optimal Strong Extractors in NC^0 Plugging the near optimal extractor given in theorem 10 on 17 and working through the algebra, we get that the expansion factor using strong extractors is about $(n+m+\log m)\frac{1}{n}$, which is about $\frac{1}{1-\phi}$, modulo the $\frac{\log m}{n}$ term, so this essentially achieves the optimum. Unfortunately, we do not know of any optimal strong extractors that are in NC^0, nor do we know of any optimal strong extractors that can be implemented directly in the register model.

4 A General Construction

Now that we know how to store secrets in a partially erasable way, the second step is to make sure computations can still be done on the secrets without leaking too much information.

4.1 Computing on Partially Erasable Secrets at the Gate Level

Let $s \in \{0,1\}^n$ be the secret involved, and let $(\mathsf{Exp}, \mathsf{Con})$ be the partially erasable form of s. Consider any efficient computation on s, which can be modeled as a poly(n)-sized circuit. Without loss of generality, we consider gates with fan-in of two and fan-out of one, and consider each output bit separately as being computed by a polynomial-sized circuit.

To evaluate a gate, the two corresponding input bits are reconstructed from their expanded forms in the registers (using Con). The gate is evaluated, resulting in an output bit b in the registers. This output bit is expanded into the partially erasable form and output to main memory[10], by using Exp. This can be done with constant memory. Note that if we just store the values of the wires naïvely, i.e. by individually expanding the 1-bit value of each wire to a Ψn size secret, then the overhead of our scheme will not even be constant. So we must amortize the cost: group the wires of the circuit into groups of size t (i.e., there are t wires in each group), where t is such that secrets of size t are expanded into m-bit strings. Now, when we write the values of the wires in an expanded form, we expand all the t values into a single m-bit string. This will make sure that the overhead of the general compiler will still be the same as the overhead for the scheme described in section 3.

The above is an informal description of COMPUTE-IN-REGISTER$(g, (\mathsf{Exp}, \mathsf{Con}), \mathsf{Exp}(s, \$))$, which makes sure that the computation of $g(s)$ is done properly without leaking intermediate computation (through expressing them in expanded form).

COMPUTE-IN-REGISTER$(g, (\mathsf{Exp}, \mathsf{Con}), \mathsf{Exp}(s, \$))$

1 ▷ For some efficiently computable function g, computes $g(s)$ by proceeding gate by gate,
2 **for** $i \leftarrow 1, ..., n$ ▷ For each bit of the output
3 **do**
4 **for** $X_{ij} \leftarrow$ each gate in C_{g_i} (level by level)
5 **do**
6 Compute the required two bits of input, in $Reg[1]$ and $Reg[2]$, using $\mathsf{Con}(\circ)$.
7 Evaluate $X_{ij}\big(Reg[1], Reg[2]\big)$ to get an intermediate output bit b_{ij}, in $Reg[3]$.
8 **Output** $\mathsf{Exp}(Reg[3], \$)$ ▷ Output in expanded form; amortization not shown.
9 **EraseReg**

Note that this algorithm COMPUTE-IN-REGISTER$(g, (\mathsf{Exp}, \mathsf{Con}), \mathsf{Exp}(s, \$))$ can be easily modified into one that handles multiple inputs and/or outputs (i.e. multiple inputs: just compute the bits needed from either of the inputs; multiple outputs: if $g(s)$ outputs $\{y_i\}_i$, COMPUTE-IN-REGISTER outputs $\{\mathsf{Exp}(y_i, \$)\}_i$); it can also be easily modified if (some parts of) the input or output is not required (and thus not modified) to be partially erasable (for such an input, just compute on it as usual; for such an output, just skip the expansion and directly output to main memory).

In the algorithms below, let S be the set of secrets in the original protocol that has to be perfectly erased, and **PartialErase** be the command for partially erasing the cache, memory, and hard drives.

In addition, for $(\mathsf{Exp}, \mathsf{Con})$ a partially erasable form, let us break it down into $\mathsf{Exp}_i(\circ)$, the algorithm for expanding each bit of the secret, and $\mathsf{Con}_i(\circ)$, the algorithm for contracting each bit of the secret (this can be done since these functions are computable with constant memory). This is straightforward for all the partially erasable forms we considered. For completeness we briefly describe how this can be done for Toeplitz hashing. The goal is to come up with R, k, x such that $R \cdot k \oplus x = s$, in a bit by bit fashion. In other words the goal is to come up with R row by row and x bit by bit, such that $R_i \cdot k \oplus x[i] = s[i]$. Across different i s, $\mathsf{Exp}_i(\circ)$ needs to share R and k as global variables (or keep state; here we choose the former). First k is chosen at random in $\{0,1\}^m$. For $i = 1$, R_i is just random, in $\{0,1\}^m$. For $i > 1$, R_i is just one extra random bit concatenated with the first $n-1$ bits of R_{i-1} (this makes sure R is Toeplitz). For each i, $x[i]$ can be calculated from R_i, k, and $s[i]$, in the registers and then perfectly erased.

[10] Note that even if in practice, storage locations holding expanded forms of the intermediate computations may be overwritten, for analyzing security we can think of all the expanded forms as being written in a new memory location. Put another way, overwriting is just one form of the imperfect erasures we are capturing.

GEN-PARTIAL-ERASABLE-SECRET$((\mathsf{Exp}, \mathsf{Con}), n)$

▷ Generates a random secret of length n bit by bit in a partial erasable way,
▷ using the partially erasable form $(\mathsf{Exp}, \mathsf{Con})$.
$k \xleftarrow{\$} \{0,1\}^m$
for $i \leftarrow 1, ...n$
    do
        $Reg[1] \xleftarrow{\$} \{0,1\}$
        $(R_i, x[i]) \leftarrow \mathsf{Exp}_i(Reg[1], \$)$
EraseReg
return (R, k, x)

COMPILER$((\mathsf{Exp}, \mathsf{Con}), \Pi_{org})$

▷ Compile Π_{org} requiring perfect erasures for security,
▷ into Π_{new} that tolerates partial erasures, using $(\mathsf{Exp}, \mathsf{Con})$ as the expanded form.
for all $s \in S$ of length n that are to be selected at random by parties
    do
        replace by the following code:
        "GEN-PARTIAL-ERASABLE-SECRET$((\mathsf{Exp}, \mathsf{Con}), n)$"
for all computations of primitives g involving $s \in S$ (or rather, $\mathsf{Exp}(s, \$)$ s.t. $s \in S$)
    do
        replace by the following code:
        "COMPUTE-IN-REGISTER$(g, (\mathsf{Exp}, \mathsf{Con}), \mathsf{Exp}(s, \$))$"
for all instructions "Perfectly Erase"
    do
        replace by the following code:
        "**PartialErase**"
return modified protocol Π_{new}

We will need the notion of UC emulation:

Definition 11 (UC Emulation [9]). *Protocol Π_{new}* UC emulates *protocol Π_{org} if for any adversary $\mathcal{A}$ there exists an adversary $\mathcal{S}$ (of complexity polynomial in that of $\mathcal{A}$) such that, for any environment $\mathcal{Z}$ and on any input, the probability that $\mathcal{Z}$ outputs 1 after interacting with $\mathcal{A}$ and parties running Π_{new} differs by at most a negligible amount from the probability that $\mathcal{Z}$ outputs 1 after interacting with $\mathcal{S}$ and parties running Π_{org}.*

If $\mathcal{A}$ and $\mathcal{Z}$ are both limited to probabilistic polynomial time, then the emulation captures computational security. If they are unbounded then the emulation captures statistical security. If in addition, the distinguishing probability of $\mathcal{Z}$ is 0, then the emulation captures perfect security.

Note that a protocol Π_{new} UC-emulating another protocol Π_{org}, means that Π_{new} preserves the security properties of the original protocol Π_{org}, and does not require that Π_{org} be UC-secure. This notion of emulation is the strongest notion of it's kind. In particular, our result applies to static/adaptive adversaries, byzantine/honest-but-curious adversaries, 2 party or multi-party protocols, etc. In particular, this does not require that the original protocol Π_{org} be "UC secure".

In our case, in the models of protocol execution, instead of having perfect erasures (so on corruption, the adversary expects to see only the current information), we have partial erasures, where the adversary chooses a length-shrinking h and on corruption, expects to see the current information plus the partially erased past (using h).

In modeling the corruptions, in order to capture repeated adaptive corruptions as in for instance proactive security, the adversary can write a "recover" message on the incoming communication tape of any previously corrupted party, after which it relinquishes control of the party. The adversary is allowed to corrupt parties over and over again, with a sequence of "corrupt, recover, corrupt..." commands.

Now we are ready to present our main theorem. Starting with any protocol that uses perfect erasures, we replace all computations on the secrets to use COMPUTE-IN-REGISTER instead. The result that we get is:

Theorem 15. *Let* $(\mathsf{Exp}, \mathsf{Con})$ *be an* (ℓ, α, ϕ)*-partially erasable form. For any protocol* Π_{org} *that requires perfect erasures (for security), the protocol* $\Pi_{new} = \text{COMPILER}((\mathsf{Exp}, \mathsf{Con}), \Pi_{org})$ *UC-emulates* Π_{org}*, and tolerates (maintains security even with) imperfect/partial erasures in the register model.*

Proof. We wish to prove that for all adversaries $\mathcal{A}_{\Pi_{new}}$ against Π_{new} with output distribution $D_{\Pi_{new}}$, there exists an adversary $\mathcal{A}_{\Pi_{org}}$ against Π_{org} (with running time poly related to $\mathcal{A}_{\Pi_{new}}$) with output distribution $D_{\Pi_{org}}$, such that no environment $\mathcal{Z}$ can distinguish between the output distributions $D_{\Pi_{new}}$ and $D_{\Pi_{org}}$.

We construct the adversary $\mathcal{A}_{\Pi_{org}}$ as follows. Whenever it is supposed to show some partially erased secret $h\big(\mathsf{Exp}(s, \$)\big)$ to $\mathcal{A}_{\Pi_{new}}$, $\mathcal{A}_{\Pi_{org}}$ simply generates an s' that is iid to s [11], using GEN-PARTIAL-ERASABLE-SECRET$((\mathsf{Exp}, \mathsf{Con}), n)$ to get $\mathsf{Exp}(s', \$)$, and then partially erase s' to get $h\big(\mathsf{Exp}(s', \$)\big)$. The adversary $\mathcal{A}_{\Pi_{org}}$ does the same thing for the intermediate computations.

Regardless of what computations were done on the secrets, the worst case (for security) is that the past and current secrets are actually correlated given $\mathcal{A}_{\Pi_{org}}$'s view (for instance this would be the case for proactive secret sharing, since the one secret s it protects remains the same, and therefore the other secrets, which represent shares of s, are actually correlated). The worst case of this correlation is that the adversary knows 1-1 and onto functions that relate all the secrets (or rather the partially erased secrets). Essentially, then, there is only one secret the adversary is after, as in the proactive secret sharing case.

If Exp is a (ℓ, α, ϕ)-partially erasable form, then as long as the total number of erasures is less than ℓ, then

$$\Delta\big(h(\mathsf{Exp}(s, \$_1)), ..., h(\mathsf{Exp}(s, \$_\ell)); h(\mathsf{Exp}(s', \$'_1)), ..., h(\mathsf{Exp}(s', \$'_\ell))\big) \leq 2^{-\alpha}.$$

Therefore, no environment $\mathcal{Z}$ can distinguish between the output distributions $D_{\Pi_{new}}$ and $D_{\Pi_{org}}$ with probability better than $2^{-\alpha}$. □

This means that when designing protocols, perfect erasures can be assumed and used freely to simplify protocol design and proof, and then later weakened into partial erasures by simply invoking our compiler. In particular, if the original protocol is adaptively secure, forward secure, intrusion resilient, or proactively secure, then the resulting protocol is as well. The price that we pay for this is a Ψ factor blow up in storage requirements.

If the original protocol assumes secure channels (which the adversary cannot eavesdrop on), then it is possible that some secrets are exchanged through these channels. Only in cases like this will the compiled protocol actually change the communication complexity.

A Practical Note Throughout, we describe the construction as a general compiler, but it could be implemented as a transparent layer between existing code and the CPU hardware. This transparent layer could for instance be implemented in hardware, to get a commodity processor that automatically makes all the secrets partially erasable.

A Side Benefit Also, as a side benefit of using our compiler, attacks like the "cold reboot" attacks employed by [34] could effectively be ruled out. Their attack is a recent example of exploiting the remanence effect on DRAM to break popular disk encryption schemes.

First, they experimentally characterize the extent and predictability of memory remanence and report that remanence times can be increased dramatically with simple techniques. They observed that at room temperature, the time until complete data loss (in various memory technology they tested) varies between approximately 2.5 seconds and 35 seconds. By discharging inverted cans of "canned air" duster spray directly

[11] again, in the algorithm GEN-PARTIAL-ERASABLE-SECRET that was presented, we only showed the case for which the secrets were generated randomly, but this can be extended to any efficiently samplable distribution, taking care to do it bit by bit – if some state needs to be kept across the different bits, this state has to be stored in a partially erasable form

onto the chips, they obtained surface temperatures of approximately -50°C. At these temperatures, it was found that about 0.1% of bits decayed after 60 seconds and fewer than 1% of bits decayed even after 10 minutes without power. However, this decay rate increases dramatically with time: even if the RAM modules were submerged in liquid nitrogen (ca. -196 °C), roughly 0.17% would have decayed after 60 minutes out of the computer.

Second, they offer new algorithms for finding cryptographic keys in memory images and for correcting errors caused by bit decay.

Third, combining their first two techniques of increasing remanence time and key extraction, they show that their attack is practical by mounting attacks on popular disk encryption systems – BitLocker, FileVault, dm-crypt, and TrueCrypt – using no special devices or materials. They exploited the remanence effect to attack these disk encryption schemes by recovering the key (permanently residing on a "secure", inaccessible part of the disk) that was loaded into main memory (while the user was using the disk).

Fourth, they conclude that the memory remanence effect severely limits the ability of an operating system to protect cryptographic key material from an attacker with physical access, and discuss several strategies for partially mitigating these risks, noting that there is no simple remedy that would eliminate them.

If our techniques are used to store the key in a partially erasable way, the the system would be "resistant" to the freezing attacks in the following sense. Decide how long you are willing to wait around your computer after power off - say 10 seconds, to prevent the adversary from physically accessing it before 10 seconds have elapsed. Experimentally characterize (or look up data) what the decay rate r for your memory technology is after 10 seconds. (We stress that [34] observed that newer memory technology exhibits higher decay rates and those that they tested completely decayed after 2.5 to 35 seconds.) Then, choosing the parameters for the expanded form such that $(1 - \phi) \leq r$ would effectively rule out the attack. In other words, once 10 seconds have passed, the partial erasures that occurred "naturally" (due to the physical characteristics of RAM), would in effect be as good as perfect erasures.

Notice one subtle point here: that this does not require the honest parties to explicitly issue an "erase" command on the RAM. This is certainly good for security since otherwise, the component issuing the "erase" command would then itself be susceptible to attack/circumvention. For instance, if the BIOS is coded so that it zeros out the RAM before the system boots, then the attacker can just code a modified BIOS which does not perform zeroization and move the RAM chips to that system; if the OS scrubs memory before shutting down, then the attacker can briefly cut power to the machine, then restore the power and boot a custom kernel.

5 More Efficient Constructions

In this section we look at more efficient constructions for two cases. The first is the case of all known proactive secret sharing schemes, and the second is the case where all computations on the secrets to be erased are computable in NC^0.

5.1 Our Direct Solution to the PSS with Partial Erasures

For simplicity, in this section we will only focus on the following simple expanded form of s: (R, k) where $R \in \{0,1\}^{n \times m}, k \in \{0,1\}^m$ such that both R and k are randomly selected subject to the constraint $R \cdot k = s$.

In the proactive (c, p) threshold secret sharing, the goal is to proactively share the secret, against an adversary that is mobile and can corrupt up to $c - 1$ parties in each time period. To maintain correctness and privacy, in between time periods the parties enter a *refreshing phase*. At the end they get new shares. If the adversary breaks into a party for the first time at time t, then we want to say that whatever information the adversary can see will not allow it to infer the old shares, those that the party holds for time 1 to $t - 1$. Perfect erasures is one way to guarantee that. As we will show, weaker forms of erasures are also good enough.

We refer the reader to [58,38] for the definition of a mobile adversary and the definition of a proactive secret sharing scheme. Here we give a modified definition of proactive (c,p) threshold secret sharing scheme with partial erasures.

Definition 12 (View of the Adversary). *Let $VIEW^T$ denote the view of the adversary E up to time period T, i.e. the concatenation of $VIEW^{T-1}$ and all the public information that E sees as well as the information seen when breaking into at most $c-1$ parties in time T. $VIEW^0$ is defined to be the empty set.*

Definition 13 (Privacy of a proactive (c,p) threshold secret sharing scheme with partial erasures). *We say that a proactive (c,p) threshold secret sharing scheme with partial erasures $\Pi = (S,(S_1,\ldots,S_p),R,D)$ is (τ,α,ϕ)-secure, if for all time periods $T \leq \tau$, for all adversaries E that breaks into at most $c-1$ parties per time period, for all partial erasing functions h with leakage fraction ϕ, and for all secret $\in S, d(secret|VIEW^T)$ is at most $2^{-\alpha}$.*

Definition 14 (Robustness of a proactive (c,p) threshold secret sharing scheme with partial erasures). *Like the perfect erasure scheme, we say a proactive (c,p) threshold secret sharing scheme with partial erasures $\Pi = (S,(S_1,\ldots,S_p),R,D)$ is robust, if privacy and correctness are maintained in the presence of less than c malicious parties.*

In subsection 5.1 we present a method to modify any existing proactive secret sharing scheme that requires perfect erasures under general conditions, into a new scheme that requires only partial erasures, and will maintain every other property of the original scheme.

The conditions are:

1. The scheme consists of 3 phases:
   (a) Dealing - when the parties receive their shares.
   (b) Refreshing - when the parties negotiate to create new shares, after every time period.
   (c) Reconstruction - when some group of parties reconstruct the secret.
2. The shares and refresh data are uniformly distributed over GF (2^n).
3. The refresh data can be computed sequentially, for one party at a time.
4. All the computations can be done directly under our register model.

The proactive secret sharing schemes of [13,38] satisfy these conditions (under some minor modifications to the schemes).

For concreteness, in subsection 5.1 we present a particular proactive (p,p) secret sharing scheme, in the honest but curious mobile adversary setting.

In subsection 5.1 a proactive (c,p) threshold secret sharing scheme, for $c < \frac{1}{3}p$ robust against malicious mobile adversary is presented.

The scheme modification method We give a high level design of the modified protocol:

Say Π is a (c,p) proactive secret sharing scheme which requires perfect erasures, and whose shares are uniformly chosen from GF (2^n), and meets all the other conditions stated above. We think of n as the security parameter. We define hereby the modified proactive secret sharing scheme PESS (Π).

Notation: Let

1. Π call a subroutine to generate the refresh data, namely REFRESH $\left(i,j,\left\{s_{ij'}^t\right\}_{1\leq j'<j}\right)$ (accepts as arguments the source party i, the target party j, and the refresh data sequentially generated to the other parties so far).
2. Π call a subroutine which uses the refresh data data received to update the shares, namely UPDATE $\left(s_i^t,\left\{s_{ji}^t\right\}_{1\leq j\leq p}\right)$ (accepts as arguments the old share, and all the refresh data it received).

PESS (Π) will use altered versions of these two subroutines, REFRESH$'$ and UPDATE$'$.
The difference is that:

1. REFRESH′ and UPDATE′ take as input the so-called pseudo-shares (defined below) instead of the shares, but essentially do the same computation.
2. Our altered subroutines will do their computation locally (that is, by using only the fixed length register to store valuable data) bit by bit, i.e. for any bit number $\alpha \in \{1 \dots m\}$, REFRESH′$_\alpha$ will compute the α^{th} bit of the refresh data to be sent by party i to party j, according to Π, and likewise UPDATE′$_\alpha$. By conditions 3 and 4, this is possible.

Our protocol calls subroutines named $\text{GEN}^1_\Pi()$, $\text{GEN}^2_\Pi()$ and $\text{GEN}^3_\Pi()$ in order to generate the share parts. The subroutine are called in the dealing and refreshing phases. Detailed code for $\text{GEN}^1_\Pi()$, $\text{GEN}^2_\Pi()$ and $\text{GEN}^3_\Pi()$, and some supporting algorithms follows.

Notation The following algorithms will follow this notation:

1. Reg is the fixed length register. All other variables are in the main memory.
2. $Var[\alpha]$ in the code signify the α^{th} bit of the variable Var. For instance, $Reg[0]$ is the first cell of the register. For a matrix R such two parameters are used, for instance $R[\alpha, \beta]$.
3. $[Val]_\alpha$ in the comments signify the α^{th} coordinate (bit) of the computable value Val. A different notation is used, since the whole value isn't being computed but only the specified bit.

Note 1. All the parameters (e.g. n, m, c, p) are assumed global.

Note 2. In the context of calling these algorithms, i should be defined to indicate P_i, which is the main party involved. The current time t should also be defined.

Note 3. These algorithms do not handle communication between parties.

RANDOM-SPLIT(FUNC_α, arg)

```
▷ for each α ← 1, ..., n,
▷    split FUNC_α(arg) into random R[α,:] ∈ {0,1}^m (row vector) and k ∈ {0,1}^m (col vector) s.t.
▷    R[α,:] · k = FUNC_α(arg) (here, R[α,:] denotes the α^th row of the matrix R)
▷ As the main text explains, we use FUNC_α because it is sensitive info and
▷    we want to compute it bit by bit in the perfectly erasable registers.
▷ The error probability is (1 − 1/2^n)(1/2^m), when for some α ∈ [1,n], FUNC_α(arg) ≠ 0, and k = 0^m.
k ←$ {0,1}^m
for α ← 1, ..., n ▷ for each row of R
    do
        Reg[0] ← 0
        for β ← 1, ..., m ▷ for each column of R
            do
                if k[β] = 1 and ∀l ∈ {β+1, ..., m}.k[l] = 0 ▷ if β is the last column that matters
                  then
                        Reg[0] ← Reg[0] ⊕ FUNC_α(arg)
                        R[α,β] ← Reg[0]
                  else
                        R[α,β] ←$ {0,1}
                        if k[β] = 1 and R[α,β] = 1 ▷ if R[α,β] matters to the product
                          then
                                Reg[0] ← Reg[0] ⊕ R[α,β]
return (R, k)
```

$\text{ID}_\alpha(s)$

```
▷ Identity function, to wrap constant vectors s.t. we can use RANDOM-SPLIT to split them
return s[α]
```

$\text{GEN}^1_{\Pi}\left(s_i^1\right)$
1 ▷ Dealer's randomly splitting of the initial share for party P_i at time 1.
2 $R_i^1, k_i^1 \leftarrow \text{RANDOM-SPLIT}\left(\text{ID}_\alpha, s_i^1\right)$
3 **return** (R_i^1, k_i^1)

$\text{GEN}^2_{\Pi}()$
1 ▷ Party P_i's refreshing data computation at time t.
2 ▷ Generate p refreshing pseudo-shares, one for P_j, that correspond to $\text{REFRESH}\left(i, j, \left\{s_{ij'}^t\right\}_{1\le j'<j}\right)$
3 **for** $j \leftarrow 1, \ldots, p$
4 **do**
5 $\left(\rho_{ij}^t, \kappa_{ij}^t\right) \leftarrow \text{RANDOM-SPLIT}\left(\text{REFRESH}'_\alpha, \left(i, j, \left\{\rho_{ij'}^t, \kappa_{ij'}^t\right\}_{1\le j'<j}\right)\right)$
6 **return** $\left\{\rho_{ij}^t, \kappa_{ij}^t\right\}_{1\le j\le p}$

$\text{GEN}^3_{\Pi}\left(\left(R_i^t, k_i^t\right), \left\{\left(\rho_{ji}^t, \kappa_{ji}^t\right)\right\}_{1\le j\le p}\right)$
1 ▷ Party P_i's share updating computation.
2 $R_i^{t+1}, k_i^{t+1} \leftarrow \text{RANDOM-SPLIT}\left(\text{UPDATE}'_\alpha, \left\{\left(\rho_{ji}^t, \kappa_{ji}^t\right)\right\}_{1\le j\le p}\right)$
3 **return** $\left(R_i^{t+1}, k_i^{t+1}\right)$

We are finally ready to describe the transformation of Π.

The Compiler PESS(Π) For simplicity, we will only state and prove the general transformation for protocols secure against honest but curious adversaries, but it can be extended to one against malicious adversaries. In particular, we will show the transformation for malicious adversaries for [38] in section 5.1.

Dealing In original scheme Π, the dealer privately sends each party i its share s_i^1. In the modified scheme PESS(Π), for every share $s_i^1 \in \{0,1\}^n$, the dealer will generate random $k_i^1 \in \{0,1\}^m$ and $R_i^1 \in \{0,1\}^{n\times m}$, such that $s_i^1 = R_i^1 \cdot k_i^1$ using subroutine $\text{GEN}^1_{\Pi}\left(s_i^1\right)$.
She will then privately send $\left(R_i^1, k_i^1\right)$ to the party instead. To prevent confusion we will be calling $\left(R_i^1, k_i^1\right)$ a *pseudo-share*.

Refreshing In Π, at the end of every time period t, in order to refresh their shares, party i may send party j refresh data, s_{ij}^t.
In PESS(Π), the sending party i will instead generate a random matrix ρ_{ij}^t and a random vector κ_{ij}^t, such that $s_{ij}^t = \rho_{ij}^t \cdot \kappa_{ij}^t$, using subroutine $\text{GEN}^2_{\Pi}()$. This subroutine calls the scheme-specific $\text{REFRESH}'_\alpha$. [If robustness against malicious adversary is required, an application of a VSS scheme will be needed here. For further elaboration on this see subsection 5.1.]
Say party i accepts the refresh data from all the other parties. Then party i generates random $\left(R_i^t, k_i^t\right)$ such that $R_i^{t+1} \cdot k_i^{t+1} = g\left(\rho_{ji}^t, \kappa_{ji}^t, R_i^t, k_i^t\right)$, for some 1-to-1 and onto function g, using subroutine $\text{GEN}^3_{\Pi}\left(\left(R_i^t, k_i^t\right), \left\{\left(\rho_{ji}^t, \kappa_{ji}^t\right)\right\}_{1\le j\le}\right.$
This subroutine calls the scheme-specific UPDATE'_α.
Finally, party i partially erase all the k_i^t and κ_{ij}^t, i.e. only $h\left(k_i^t\right)$ and $h\left(\kappa_{ij}^t\right)$ remain from them, respectively.

Reconstruction Any group of parties which could reconstruct the secret in the original scheme, may in the modified scheme compute $R_i^t \cdot k_i^t$, and use it to compute the secret.

Remark 6. The computation of the new shares is done, as mentioned above, bit by bit using a fully erasable register of constant length, without storing any full length intermediate values. Even though it might not be the most efficient way to compute new shares, the adversary learns no data from the computation itself.

Remark 7. If robustness against a malicious dealer or malicious parties is required, a VSS scheme may be applied to the pseudo-shares sent by the dealer, and to the refresh data sent by each of the parties at the beginning of each time period. Indeed, a VSS scheme was used to achieve robustness in [38] in this precise

way. In our context, one must ensure that computations done in the VSS must be implementable using the fixed length register model. It is unclear whether the VSS scheme used by [38] can be implemented in the fixed register model. Instead, we show how to do this for the scheme defined by [8], or rather its simplified version, defined by [27]. We will elaborate on this in subsection 5.1.

Remark 8. We assume that the function h (to be used by party i) is adversary chosen in the worst case manner, and may change from time period to time period. More precisely, an adversary may choose h in an arbitrary manner, differently for every participating party, and anew before each time period (i.e. prior to refreshing).

Theorem 16. *Let ϕ be the leakage fraction of the partial erasure function h. Let $(\mathsf{Exp}, \mathsf{Con})$ be a (ℓ, α, ϕ)-partially erasable form, where $\mathsf{Exp}(s, \$) := (R, k)$ and $\mathsf{Con}(R, k) := R \cdot k$. Then PESS($\Pi$) described above is a $\left(\frac{\ell}{\rho}, \alpha, \phi\right)$-secure proactive (c, p) threshold secret sharing scheme with partial erasures as per definition 13, where $\rho := 2(c-1)p + p - c + 1$.*

Proof. Again, for simplicity, we will only prove the case of robustness against a malicious adversary for a specific protocol in subsection 5.1, where the robustness is added explicitly by the use of a VSS scheme. Here we prove the general transformation under an honest but curious adversary.

Correctness is easy to see, since by construction, the product of any pair (R_i^t, k_i^t) is the share under Π, and similarly the product of any pair (ρ_i^t, κ_i^t) is the refresh share.

According to the definition of Π, and thus also for PESS (Π), the mobile adversary may corrupt at most $c-1$ parties at any time period. Without loss of generality, consider the case where the adversary corrupts exactly $c-1$ parties in each time period, since by corrupting less she can only gain less information.

There are two types of information, which is perfectly erased in the perfect erasure model, and only partially erased here: old pseudo-shares and refresh data.

Old pseudo-shares:
At any time period t, the adversary misses $p-c+1$ pseudo-shares, which are then partially erased. Since the pseudo-shares are generated by $\text{GEN}_{\Pi}^1()$ and $\text{GEN}_{\Pi}^3()$ are random up to the product of the pseudo-share parts, which in itself is also random. Therefore, these are partially erased tuples. Moreover, since the adversary knows $c-1$ of the pseudo-shares of time period t, she knows a 1-to-1 and onto function between each of the other pseudo-shares and s. Thus, in each time period t she learns $p-c+1$ partially erased tuples related to s.

Refresh data:
Between any pair of adjacent time periods t and $t+1$, the adversary may switch between parties. We will define the following subgroups of the p parties:

- S_R is the group of parties in which she remains.
- S_L is the group of parties which she leaves in time period t, and does not corrupt in time period $t+1$.
- S_A is the group of parties to which she arrives in time period $t+1$. Clearly, $|S_A| = |S_L|$.
- S_I is the group of parties which she ignores. That is, these parties are not corrupted in either of the two time periods.

For instance, if the adversary stays in the same $c-1$ parties, then $|S_R| = c-1$ and $S_L = S_A = \emptyset$. If the adversary changes all the parties, then $S_R = \emptyset$ and $|S_A| = |S_L| = c-1$.
Let us look at the refresh data used by each of the sets:

- S_R: The adversary knows the refresh data generated between the two time periods, so the partial erasure of them does not add any new information.
- S_I: For each party $P_i \in S_I$, the adversary knows some 1-to-1 and onto functions f and g such that $s = f\left(R_i^t \cdot k_i^t\right)$ and $s = g\left(R_i^{t+1} \cdot k_i^{t+1}\right)$. Therefore, $R_i^{t+1} \cdot k_i^{t+1} = g^{-1}\left(f\left(R_i^t \cdot k_i^t\right)\right)$, so the valuable information in the refresh data is all known, and the partial erasure of them does not add any new information.

– S_L, S_A: For any party P_i in one of these two sets, the adversary can potentially gain information from the refresh data which was received from any party P_j that is not in S_R (since the refresh data sent by them is already known). Each of the useful refresh tuples is generated by $\text{GEN}_{\Pi}^{2}()$ as random up to the product of the tuple parts, which in itself is also random. Therefore, these are partially erased tuples. For simplicity, let us assume the adversary can learn information about the secret s from each tuple independently. That is, the adversary knows a 1-to-1 and onto function between each tuple and s. Clearly such a function exists, though the adversary's knowledge of it is over estimated, since she actually knows only the relation between all the tuples used to refresh P_i's pseudo-share and s, and the relation between all the tuples sent by each party. Therefore, according to our over estimated assumption, between each two adjacent time periods t and $t+1$ she learns at most $2(c-1)p$ partially erased tuples related to s.

Therefore, over τ time periods, all the adversary may learn is at most $\tau(2(c-1)p+p-c+1)$ partially erased tuples related to s. Since the partially erasable form we are using is $\left(\ell, \alpha, \phi\right)$-partially erasable, as long as

$$\tau(2(c-1)p+p-c+1) \leq \ell \Leftrightarrow \tau \leq \frac{\ell}{(2(c-1)p+p-c+1)} = \frac{\ell}{\rho},$$

then we have that $d(s|VIEW^{\ell}) \leq 2^{-\alpha}$, and so $\text{PESS}(\Pi)$ is a $\left(\frac{\ell}{\rho}, \alpha, \phi\right)$-secure proactive (c,p) threshold secret sharing scheme with partial erasures as per definition 13. □

Proactive (p,p) secret sharing with partial erasures Now, let us apply the general modification method discussed in the previous section to the trivial (p,p) secret sharing scheme secure against honest but curious adversaries.

The following algorithms will be used.

$\text{ADD}\,(\delta, R, k)$

▷ Add $[R \cdot k]_{\delta}$ to $Reg[1]$.
▷ Result is returned in $Reg[1]$.
for $\beta \leftarrow 1, ..., m$
    do
        if $R[\delta, \beta] = 1$ and $k[\beta] = 1$
            then $Reg[1] = Reg[1] \oplus 1$
return

$\text{REFRESH}'_{\alpha}\left(i, j, \left\{\rho_{ij'}^{t}, k_{ij'}^{t}\right\}_{1 \leq j' < j}\right)$

▷ Compute the α^{th} bit of the refreshing pseudo-share from P_i to P_j for refreshing time t to $t+1$.
$Reg[1] \leftarrow 0$
if $j \neq p$
    then $Reg[1] \overset{\$}{\leftarrow} \{0,1\}$
    else
        for $j' \leftarrow 1, ..., p-1$
            do
                $\text{ADD}\left(\alpha, \rho_{ij'}^{t}, \kappa_{ij'}^{t}\right)$ ▷ Add $\left[\rho_{ij'}^{t} \cdot \kappa_{ij'}^{t}\right]_{\alpha}$ to $Reg[1]$
return $Reg[1]$

$\text{UPDATE}'_{\alpha}\left((R_i^t, k_i^t), \left\{\left(\rho_{ji}^{t}, \kappa_{ji}^{t}\right)\right\}_{1 \leq j \leq p}\right)$

▷ Compute the α^{th} bit of the new, time $t+1$ pseudo-share of P_i.
$Reg[1] \leftarrow 0$
$\text{ADD}(\alpha, R_i^t, K_i^t)$ ▷ Calculate $\left[R_i^t \cdot k_i^t\right]_{\alpha}$ into the register $Reg[1]$
for $j \leftarrow 1, ..., p$
    do
        $\text{ADD}\left(\alpha, \rho_{ji}^{t}, \kappa_{ji}^{t}\right)$ ▷ Add $\left[\rho_{ji}^{t} \cdot \kappa_{ji}^{t}\right]_{\alpha}$ to the register $Reg[1]$
return $Reg[1]$

Dealing On input (s, r) where s is the secret of length n and r random string of length $poly(n, p)$:

1. Generate from r random $\left(k_1^1, ..., k_p^1\right) \in \{0,1\}^m$ and $\left(R_1^1, ..., R_p^1\right) \in \{0,1\}^{n \times m}$, such that $\bigoplus_{i=1}^p R_i^1 \cdot k_i^1 = s$, using GEN_{Π}^1.
2. Send $\left(R_i^1, k_i^1\right)$ to P_i privately.

Note: in the original scheme, the shares s_i are uniform, and therefore so are the pseudo-shares (R_i^1, k_i^1).

Refreshing In between each adjacent time periods t and $t+1$, each party P_i does the following, to refresh its old pseudo-share (R_i^t, k_i^t) into a new one $\left(R_i^{t+1}, k_i^{t+1}\right)$.

1. Generate random $\left(\kappa_{i1}^t, ..., \kappa_{ip}^t\right) \in \{0,1\}^m$ and $\left(\rho_{i1}^t, ..., \rho_{ip}^t\right) \in \{0,1\}^{n \times m}$, such that $\bigoplus_{i=1}^p \rho_{ij}^t \cdot \kappa_{ij}^t = 0$, using GEN_{Π}^2.
2. Privately send $\left(\rho_{ij}^t, \kappa_{ij}^t\right)$ to party P_j, for all $j \in [1, p]$.
3. Privately receive $\left(\rho_{ji}^t, \kappa_{ji}^t\right)$ from party P_j, for all $j \in [1, p]$.
4. Generate random k_i^{t+1} and R_i^{t+1}, such that $R_i^{t+1} \cdot k_i^{t+1} = R_i^t \cdot k_i^t \oplus \bigoplus_{j=1}^p \rho_{ji}^t \cdot \kappa_{ji}^t$, using GEN_{Π}^3.
5. Partially erase all the k_i^t and κ_{ij}^t, i.e. $h\left(k_i^t\right)$ and $h\left(\kappa_{ij}^t\right)$ remain from them, respectively.

Remark 9. Note that in GEN_{Π}^2 and GEN_{Π}^3 we call $\text{REFRESH}'_\alpha$ and UPDATE'_α, respectively, which are protocol specific.

Reconstruction All Parties pull together their current pseudo-share (R_i^t, k_i^t), and compute the secret $s = \bigoplus_{i=1}^p R_i^t \cdot k_i^t$.

Theorem 17. *Let ϕ be the leakage fraction of the partial erasure function h. Let $(\mathsf{Exp}, \mathsf{Con})$ be a (ℓ, α, ϕ)-partially erasable form, where $\mathsf{Exp}(s, \$) := (R, k)$ and $\mathsf{Con}(R, k) := R \cdot k$. Then the proactive (p, p) secret sharing scheme described above is a $\left(\frac{\ell}{\rho}, \alpha, \phi\right)$-secure proactive (c, p) threshold secret sharing scheme with partial erasures as per definition 13, where $\rho := 2(c-1)p + p - c + 1$.*

Proof. This is a direct application of the modification method, so this theorem derives directly from theorem 16. □

Proactive (c, p) threshold secret sharing with partial erasures Next, we'll modify the efficient (c, p) threshold proactive secret sharing protocol, secure against a malicious adversary, as defined in [38] to one which uses only partial erasures. According to their protocol, which enhances Shamir's secret sharing [61], each party holds the value of a random c degree polynomial in $\mathbb{Z}_q$ at the party's id, and the polynomial's value at 0 is the secret s. The shares are refreshed, by each party sending its neighbors their value of a random polynomial whose value at 0 is 0. Each party adds the received values to its current secret and deletes the old one, as well as the update values.

Their scheme [38] is robust against malicious adversaries (that control up to $c - 1 < p/2$ parties), under the discrete log assumption. Each party broadcasts g^{a_i}s, where the a_is are his polynomial coefficients. These can then be used by each party to verify their value is correct, yet not gain any information about the refresh shares of the others.

Instead of this verification scheme, we will use the VSS scheme defined by [27], which do not rely on a cryptographic assumption such as the irreversibility of discrete log, and contrary to [26] used in [38], it is easy to fit into the memory model.

As commonly done in VSS, we assume that broadcast channels are available.

The following algorithms will be used.

$\text{ADD}(\delta, R, k)$ is the same as that of the (p, p) case.

$\text{REFRESH}'_\alpha\left(i, j, \left\{\rho^t_{ij'}, k^t_{ij'}\right\}_{1\le j'<j}\right)$

1 ▷ Compute the α^{th} bit of the refreshing pseudo-share from P_i to P_j for refreshing time t to $t+1$.
2 $Reg[2] \leftarrow 0$
3 **if** $j < c$
4 **then** $Reg[2] \overset{\$}{\leftarrow} \{0,1\}$
5 **else** ▷ According to Lagrange interpolation $s^t_{ij} = \sum_{j'=1}^{c-1} s^t_{ij'} \cdot L_{j'}(j)$, where $L_{j'}(j) = \prod_{v\neq j'} \frac{j-v}{j'-v}$
6 $L^\gamma_{j'}(j) \leftarrow 2^\gamma \prod_{v\neq j'} \frac{j-v}{j'-v}$ ▷ These are constants with no relation to the secrets, so they can
▷ be computed and stored in main memory, without revealing info
7 **for** $j' \leftarrow 1, ..., c-1$
8 **do** ▷ Add $\left[s^t_{ij'} \cdot L_{j'}(j)\right]_\alpha$ to the register $Reg[1]$ by going bit by bit of $s^t_{ij'}$
9 **for** $\gamma \leftarrow 1, ..., m$
10 **do**
11 $Reg[1] \leftarrow 0$
12 $\text{ADD}\left(\gamma, \rho^t_{ij'}, \kappa^t_{ij'}\right)$ ▷ Compute $\left[s^t_{ij'}\right]_\gamma$ into $Reg[1]$
13 **if** $Reg[1] = 1$
14 **then** $Reg[2] = Reg[2] \oplus L^\gamma_{j'}(j)[\alpha]$
15 **return** $Reg[2]$

$\text{UPDATE}'_\alpha\left((R^t_i, k^t_i), \left\{\left(\rho^t_{ji}, \kappa^t_{ji}\right)\right\}_{1\le j\le p}\right)$ is the same as that of the (p,p) case.

$\text{VER}_i\left(R^t_i, k^t_i, \bar{R}^{t(j)}_i, \bar{k}^{t(j)}_i\right)$

1 ▷ Party P_i's verification at time t that for a given j his share is valid.
2 ▷ Note that the use of this algorithm in the dealing and the refreshing phases are in a sense reversed.
3 ▷ In particular, dealing uses VER_i for P_i to verify the shares dealt to it by the dealer,
4 ▷ and refreshing uses VER_j for P_j to verify P_i's actions.
5 **for** $\alpha \leftarrow 1, ..., n$
6 **do** ▷ Check for every bit α
7 $Reg[1] \leftarrow 0$
8 $\text{ADD}\left(\alpha, \bar{R}^{t(j)}_i, \bar{k}^{t(j)}_i\right)$ ▷ Compute $\left[\bar{R}^{t(j)}_i \cdot \bar{k}^{t(j)}_i\right]_\alpha$ into the register $Reg[1]$
9 **if** $Q_j = 1$
10 **then** $\text{ADD}\left(\alpha, R^t_i, k^t_i\right)$ ▷ Add $\left[R^t_i \cdot k^t_i\right]_\alpha$ to the register $Reg[1]$
11 **if** $\left[g'^{t(j)}(i)\right]_\alpha \neq Reg[1]$
12 **then return** *invalid*
13 **return** *valid*

Dealing On input (s, r) where s is the secret of length n and r random string of length $poly(n,p)$:

1. Generate a random c degree polynomial $f(\cdot)$ in GF (2^n) (as nothing in the original protocol limits us to $\mathbb{Z}_q$ in particular), whose value at 0 is s (that is, the free coefficient is s). Generate from r random $\left(k^1_1, \ldots, k^1_p\right) \in \{0,1\}^m$ and $\left(R^1_1, \ldots, R^1_p\right) \in \{0,1\}^{n\times m}$, such that $f(i) = R^1_i \cdot k^1_i$, using $\text{GEN}^1_{\Pi'}$.
2. Send (R^1_i, k^1_i) to P_i privately.
3. **Verifying:** For robustness, the following will be used:
   (a) Generate $V = poly(n)$ more c degree polynomials $g^{1(1)}(\cdot), \ldots, g^{1(V)}(\cdot)$. For every $j \in \{1 \ldots V\}$, generate random $\left(\bar{k}^{1(j)}_1, \ldots, \bar{k}^{1(j)}_p\right) \in \{0,1\}^m$ and $\left(\bar{R}^{1(j)}_1, \ldots, \bar{R}^{1(j)}_p\right) \in \{0,1\}^{n\times m}$, such that $g^{1(j)}(i) = \bar{R}^{1(j)}_i \cdot \bar{k}^{1(j)}_i$, using $\text{GEN}^1_{\Pi'}$.
   (b) For each polynomial $g^{1(j)}$, send each pseudo-share $(\bar{R}^{1(j)}_i, \bar{k}^{1(j)}_i)$ to P_i privately.
   (c) Each party i picks $Q_{\lceil \frac{V(i-1)}{p}+1\rceil}, \ldots, Q_{\lceil \frac{Vi}{p}\rceil} \overset{\$}{\leftarrow} \{0,1\}$ and broadcasts them.
   (d) For every j, if $Q_j = 0$, broadcast $g'^{1(j)} = g^{1(j)}$. Otherwise, broadcast $g'^{1(j)} = g^{1(j)} + f$.
   (e) Each party i will run VER to verify for every j that his values are valid (and declare if so).

Note: in the original scheme, the shares $f(i)$ are uniform, and therefore so are the pseudo-shares (k_i^1, R_i^1). Likewise for $g^{1(j)}(i)$.

Refreshing In between each adjacent time periods t and $t+1$, each party P_i does the following, to refresh its old pseudo-share (R_i^t, k_i^t) into a new one (R_i^{t+1}, k_i^{t+1}).

1. Generate random $\left(\kappa_{i1}^t, ..., \kappa_{ip}^t\right) \in \{0,1\}^m$ and $\left(\rho_{i1}^t, ..., \rho_{ip}^t\right) \in \{0,1\}^{n\times m}$, such that the $\rho_{ij}^t \cdot \kappa_{ij}^t$s will define a c degree polynomial f_i^t whose value at 0 is 0 (that is, the free coefficient is 0), using GEN_{Π}^2.
2. Privately send $(\rho_{ij}^t, \kappa_{ij}^t)$ to party P_j, for all $j \in [1,p]$.
3. Privately receive $(\rho_{ji}^t, \kappa_{ji}^t)$ from party P_j, for all $j \in [1,p]$.
4. **Verifying:** For robustness, follow practically the same method used above by the dealer, in order to allow validation of the refresh data:
   (a) For every $v \in \{1 \dots V\}$, generate random $\left(\bar{\kappa}_{i1}^{t(v)}, ..., \bar{\kappa}_{ip}^{t(v)}\right) \in \{0,1\}^m$ and $\left(\bar{\rho}_{i1}^{t(v)}, ..., \bar{\rho}_{ip}^{t(v)}\right) \in \{0,1\}^{n\times m}$, such that the $\bar{\rho}_{ij}^{t(v)} \cdot \bar{\kappa}_{ij}^{t(v)}$s will define a c degree polynomial $g_i^{t(v)}$ with free coefficient 0, using GEN_{Π}^2.
   (b) For every $v \in \{1 \dots V\}$, privately send $(\bar{\rho}_{ij}^{t(v)}, \bar{\kappa}_{ij}^{t(v)})$ to party P_j, for all $j \in [1,p]$.
   (c) Each party j picks $Q_{\lceil \frac{V(j-1)}{p}+1 \rceil}, \dots, Q_{\lceil \frac{Vj}{p} \rceil} \xleftarrow{\$} \{0,1\}$ and broadcasts them.
   (d) For every v, if $Q_v = 0$, calculate and broadcast $g_i'^{t(v)} = g_i^{t(v)}$. Otherwise, calculate and broadcast $g_i'^{t(v)} = g_i^{t(v)} + f_i^t$.
   (e) Each party j will run VER to verify for every v that $g_i'^{t(v)}(0) = 0$ (in main memory), and that his values are valid (and declare if so).
5. Generate random k_i^{t+1} and R_i^{t+1}, such that $R_i^{t+1} \cdot k_i^{t+1} = R_i^t \cdot k_i^t + \sum_{j=1}^p \rho_{ji}^t \cdot \kappa_{ji}^t$, using GEN_{Π}^3.
6. Partially erase all the k_i^t and κ_{ij}^t, and they become $h(k_i^t)$ and $h(\kappa_{ij}^t)$, respectively.

Remark 10. Note that in GEN_{Π}^2 and GEN_{Π}^3 we call $\text{REFRESH}'_\alpha$ and UPDATE'_α, respectively, which are protocol specific.

Reconstruction Any c of the parties pull together their current pseudo-share (R_i^t, k_i^t), and compute the polynomial and by that the secret.

Theorem 18. *Let ϕ be the leakage fraction of the partial erasure function h. Let $(\mathsf{Exp}, \mathsf{Con})$ be a (ℓ, α, ϕ)-partially erasable form, where $\mathsf{Exp}(s, \$) := (R, k)$ and $\mathsf{Con}(R, k) := R \cdot k$. Then the proactive (c,p) secret sharing scheme described above is a robust and $\left(\frac{\ell}{V \cdot p + \rho}, \alpha, \phi\right)$-secure proactive (c,p) threshold secret sharing scheme with partial erasures as per definition 13, where $\rho := 2(c-1)p + p - c + 1$.*

Proof. Correctness in the presence of honest but curious adversaries is easy to see, since by construction, the product of any pair (R_i^t, k_i^t) is the trivial share, and the product of any pair (ρ_i^t, κ_i^t) is the trivial refresh share.

For the same reason, the correctness in the presence of malicious adversaries is trivial, as it has been proven in [27], and since the verification polynomials of the refresh data have free coefficient being 0, receiving parties can also easily verify the value of the polynomial at 0 is 0 with probability $1 - 2^{-V}$.

As for privacy in the presence of malicious adversaries:
VSS is added, which may give the adversary more data, in the form of partially erased verification tuples $(\bar{\rho}_{ij}^{t(v)}, \bar{\kappa}_{ij}^{t(v)})$, sent in step 4e (of the refresh phase, or step 3e in the dealing phase). Clearly, any of these can only be used when $g_i'^{t(v)} = g_i^{t(v)} + f_i^t$ was broadcasted in step 4d (giving the adversary full knowledge of it), making a tolerable relation between the tuple and s. On the worst case, all the verification tuples are related to s. Since the amount of verification tuples sent after each time period is Vp, and in total less than ℓVp.

Therefore, since the partially erasable form is (ℓ, α, ϕ)-secure, this means that PESS(Π) described above is $(\frac{\ell}{V \cdot p + \rho}, \alpha, \phi)$-secure as per definition 13. □

5.2 More Efficient Compiler for NC^0

Our general compiler had to work at the gate level so that we can compute any efficient function on the secrets just using the constant size registers; this results in a blow up in computation proportional to the size of the circuit (computing some function on the secret). However, if all the computations on the secret can be done with constant output locality (where each bit of the output depends only on a constant number of input bits), then we do not need to go to the gate level.

A summary of the results as to which cryptographic primitives can be computed in a constant output locality fashion (assuming corresponding assumptions [12]) is provided in 5.2. We give the more efficient compiler in 5.2.

Restricting the computations on the secret to be of constant output locality is stronger than required. In fact, in the next chapter, chapter 5.1, we give a proactive secret sharing protocol using partial erasures that is more efficient than the general compiler, which *does not* require the computations to be of constant output locality.

Summary of Some Results in Parallel Time Complexity The parallel time complexity of cryptographic primitives has been studied in various works, including [35,31,18,45,54,3,6]. They try to minimize the parallel time complexity of basic cryptographic primitives (or prove impossibility results). Recall that NC^i is the class of functions that can be computed by $O((\log n)^i)$ depth circuits with bounded fan-in. In particular, an NC^0 function is one where each bit of the output depends on a constant number of input bits (i.e. constant *output locality*); this class is also called constant parallel time, for if we had a polynomial number of processors, an NC^0 function would take constant time to evaluate (in the straightforward manner). We write NC^0_c for one where each bit of the output depends on at most c input bits.

The existence of one way functions and pseudo random generators in NC^1 is a relatively mild assumption, implied by most number-theoretic or algebraic intractability assumptions commonly used in cryptography. On the other hand, until the work of Applebaum, Ishai and Kushilevitz [3], there has been no convincing theoretical answer to the question of whether there are instances of basic cryptographic primitives that can be computed in constant parallel time (NC^0). The main result in [3] is that, every "moderately easy" OWF (resp., PRG), say computable in NC^1 (or even $\oplus L/poly$), can be compiled into a corresponding OWF (resp., PRG) in which each output bit depends on at most 4 input bits, i.e. in NC^0_4. On the other hand the existence of OWF and PRG in NC^1 is a relatively mild assumption, implied by most number theoretic or algebraic intractability assumptions commonly used in cryptography, such as the intractability of factoring, discrete logarithms and lattice problems. They give a similar compiler for other cryptographic primitives like one-way permutations, encryption, signatures, commitment, and collision resistent hashing. Improvements to these, and some additional results were made in [4,6].

Tables 1 and 2 presents a quick summary of the results relevant to us (AC^0 is a similar class as NC^0 except that the fan-in of each gate can be unbounded; by "standard" assumptions we mean intractability of factoring, discrete logs, lattice problems, etc.).

Negative Results	
Result	**Reference**
PRF $\notin AC^0$	[47]
PRG $\notin NC^0_2$	[18]
PRG with superlinear stretch $\notin NC^0_3$	[18]
PRG with superlinear stretch $\notin NC^0_4$	[54]

Table 1. Negative Results in Parallel Time Complexity

[12] We stress that, for our results, the assumptions do not relate to security but rather only to the efficiency of the compiled protocol, since we can always forget about the assumptions and just use the general compiler.

For our purposes, the implication of these negative results for us is: for protocols that use the corresponding primitives on the secrets, it might be difficult to replace perfect erasures by partial erasures in the register model, more efficiently than through using the general compiler.

Positive Results		
Assumption	**Result**	**Ref**
Subset-sum related	$\exists$ PRG with sublinear stretch $\in AC^0$	[42]
Intractability in Decoding Random Linear Code	$\exists$ OWF $\in NC^0_3$	[3]
Intractability in Decoding "Sparsely Generated" Code	$\exists$ linear stretch PRG $\in NC^0$	[5]
$\exists$ TDP whose function evaluator $\in \oplus L/poly$	$\exists$ TDP whose function evaluator $\in NC^0_4$	[3]
$\exists$ 1 bit stretching PRG $\in$ uniform-$\oplus L/poly$	$\exists$ PRG with sublinear stretch $\in NC^0_4$	[4]
$\exists$ 1 bit stretching PRG $\in$ uniform-$\oplus L/poly$	(Priv./Pub.) Encryption Algorithm $\in NC^0_4$	[4]
$\exists$ 1 bit stretching PRG $\in$ uniform-$\oplus L/poly$	$\exists$ Signatures $\in NC^0_4$	[4]
$\exists$ 1 bit stretching PRG $\in$ uniform-$\oplus L/poly$	$\exists$ MACs $\in NC^0_4$	[4]
$\exists$ 1 bit stretching PRG $\in$ uniform-$\oplus L/poly$	(Non-interactive) Commitment $\in NC^0_4$	[4]
$\exists$ 1 bit stretching PRG $\in$ uniform-$\oplus L/poly$	(Non-interactive) ZK Prover $\in NC^0_4$	[4]
$\exists$ 1 bit stretching PRG $\in$ uniform-$\oplus L/poly$	Constant round ZK Prover for $NP \in NC^0_4$	[4]
$\exists$ 1 bit stretching PRG $\in$ uniform-$\oplus L/poly$	(Comp. Secure) MPC of any P-time $f \in NC^0_4$	[4]

Table 2. Positive Results in Parallel Time Complexity

Remark 11. Most of the assumptions above (except for the subset-sum related and the intractability of decoding codes) are implied by standard number-theoretic or algebraic intractability assumptions, including intractability of factoring, discrete logarithms, and lattice problems.

Remark 12. For encryption schemes, the above result only applies to the encryption algorithm and not the decryption algorithms. In [3] it is argued that in many settings, decryption in NC^0 is impossible. However, if the scheme is restricted to a single message of bounded length, or maintains state, then decryption can also be done in NC^0. Since [3] adapts the construction used for encryption schemes to commitments and zero-knowledge schemes, they point out that in general, the interactive cases of these primitives there is an analogous problem: the transformation results in an NC^0 sender but does not promise anything regarding the parallel complexity of the receiver. For more details the reader is referred to [3,4].

Remark 13. Based on the intractability of some problems from the domain of error correcting codes, [6] obtained further improvements for some of the results, e.g. a PRG that has output and input locality (how many bits of output each bit of input influences) of 3.

More Efficient Compiler for NC^0 It is straightforward to get a more efficient compiler Compiler_{NC^0}: instead of using Compute-in-register$(g, ...)$, we use g that has constant output locality directly, which exists assuming the appropriate assumptions. In other words, instead of going gate by gate as in Compute-in-register$(g, ...)$, for each output bit, reconstruct the constant number of bits that this bit depends on in the registers, compute the function in "one-shot," and output in partially erasable form.

Theorem 19 (Compiler for NC^0). *For any protocol Π_{org} that relies on perfect erasures for security, and in which all the computations on the secrets can be done in NC^0, we have that $\Pi_{new} = \text{Compiler}_{NC^0}$ UC emulates Π_{org}, and tolerates (maintains security even with) imperfect/partial erasures in the register model.*

References

1. N. Alon, O. Goldreich, H. Hastad, and R. Peralta. Simple constructions of almost k-wise independent random variables. In *Proc. 31st IEEE Symp. on Foundations of Comp. Science*, pages 544–553, St. Louis, 1990. IEEE. 13
2. R. Anderson. Two remarks on public key cryptology. invited lecture, acm-ccs '97, 1997. 1
3. B. Applebaum, Y. Ishai, and E. Kushilevitz. Cryptography in nc^0. In *Proc. 45th IEEE Symp. on Foundations of Comp. Science*. 32, 33
4. B. Applebaum, Y. Ishai, and E. Kushilevitz. Computationally private randomizing polynamials and their applications. In *Proc. 20th IEEE Conference on Computational Complexity (CCC)*, 2005. 32, 33
5. B. Applebaum, Y. Ishai, and E. Kushilevitz. On pseudorandom generators with linear stretch in nc^0. In *Proc. of 10th International Workshop on Randomization and Computation*, 2006. 33
6. B. Applebaum, Y. Ishai, and E. Kushilevitz. Cryptography with constant input locality. In *Proc. 27th Crypto*, 2007. 32, 33
7. D. Beaver. Plug and play encryption. In *Proc. CRYPTO 97*, 1997. 1, 2
8. M. Ben-Or, S. Goldwasser, and A. Wigderson. Completeness theorems for non-cryptographic fault-tolerant distributed computation. In *Proc. 20th ACM Symp. on Theory of Computing*, pages 1–10. 27
9. R. Canetti. Universally composable security: A new paradigm for cryptographic protocols. In *Proc. 42nd IEEE Symp. on Foundations of Comp. Science*, pages 136–145, 2001. 4, 6, 7, 21
10. R. Canetti, Y. Dodis, S. Halevi, E. Kushilevitz, and A. Sahai. Exposure-resilient functions and all-or-nothing transforms. In *Proc. EUROCRYPT 2000*, pages 453–469, 2000. 5
11. R. Canetti, U. Feige, O. Goldreich, and M. Naor. Adaptively secure computation. 1, 2
12. R. Canetti, R. Gennaro, A. Herzberg, and D. Naor. Proactive security: Long-term protection against break-ins. In *CryptoBytes(1)3, 1997*. 1
13. R. Canetti and A. Herzberg. Maintaining security in the presence of transient faults. In *Proc. CRYPTO 94*, pages 425–438, 1994. 24
14. J. L. Carter and M. N. Wegman. Universal classes of hash functions. *JCSS*, 18. 19
15. T. Cover and J. Thomas. *Elements of Information Theory*. John Wiley & Sons, 1991. 9, 10, 18
16. Giovanni Di Crescenzo, Niels Ferguson, Russell Impagliazzo, and Markus Jakobsson. How to forget a secret. In Christoph Meinel and Sophie Tison, editors, *STACS*, volume 1563 of *Lecture Notes in Computer Science*, pages 500–509. Springer, 1999. 5
17. Giovanni Di Crescenzo, Richard J. Lipton, and Shabsi Walfish. Perfectly secure password protocols in the bounded retrieval model. In *Theory of Cryptography Conference*, pages 225–244, 2006. 5
18. M. Cryan and P. B. Miltersen. On pseudorandom generators in nc^0. In *Proc. 26th MFCS*, 2001. 32
19. David Dagon, Wenke Lee, and Richard J. Lipton. Protecting secret data from insider attacks. In *Financial Cryptography*, pages 16–30, 2005. 5
20. W. Diffie, P. C. Van-Oorschot, and M. J. Weiner. Authentication and authenticated key exchanges. In *Designs, Codes, and Cryptography*, pages 107–125, 1992. 1
21. Y. Dodis. *Exposure-Resilient Cryptography*. PhD thesis, MIT, 2000. 5
22. Yevgeniy Dodis, Rafail Ostrovsky, Leonid Reyzin, and Adam Smith. Fuzzy extractors: How to generate strong keys from biometrics and other noisy data. Cryptology ePrint Archive, Report 2003/235, 2003. http://eprint.iacr.org/. 8
23. Stefan Dziembowski. Intrusion-resilience via the bounded-storage model. In *Theory of Cryptography Conference*, pages 207–224, 2006. 5
24. Stefan Dziembowski. On forward-secure storage. In *Proc. CRYPTO 2006*, pages 251–270, 2006. 5
25. Stefan Dziembowski and Krzysztof Pietrzak. Intrusion-resilient secret sharing. In *FOCS '07: Proceedings of the 48th Annual IEEE Symposium on Foundations of Computer Science (FOCS'07)*, pages 227–237, Washington, DC, USA, 2007. IEEE Computer Society. 5
26. P. Feldman. A practical scheme for non-interactive verifiable secret sharing. In *Proc. 28th IEEE Symp. on Foundations of Comp. Science*, pages 427–437, 1987. 29
27. P. Feldman and S. Micali. Optimal algorithms for byzantine agreement. In *Proc. 20th ACM Symp. on Theory of Computing*, pages 148–161, 1988. 27, 29, 31
28. Y. Frankel, P. Gemmel, P. D. MacKenzie, and M. Yung. Proactive rsa. In *Proc. CRYPTO 97*, pages 440–454, 1997. 1
29. S. L. Garfinkel. *Design Principles and Patterns for Computer Systems That Are Simultaneously Secure and Usable*. PhD thesis, MIT, 2005. 2

30. R. Gennaro, S. Jarecki, H. Krawczyk, and T. Rabin. Robust threshold dss signatures. In *Proc. EUROCRYPT 96*, pages 354–371, 1996. 1
31. O. Goldreich. Candidate one way functions based on expander graphs. In *ECCC*, volume 7(090), 2000. 32
32. S. Goldwasser, S. Micali, and C. Rackoff. The knowledge complexity of interactive proof systems. *SIAM J. Comput.*, 18(1):186–208, 1989. 7
33. C. G. Günther. An identity-based key-exchange protocol. In *Proc. EUROCRYPT 89*, pages 29–37, 1989. 1
34. J. Alex Halderman, Seth D. Schoen, Nadia Heninger, William Clarkson, William Paul, Joseph A. Calandrino, Ariel J. Feldman, Jacob Appelbaum, and Edward W. Felten. 4, 22, 23
35. J. Håstad. One-way permutations in nc^0. In *Information Processing Letters*, volume 26. 32
36. J. Hastad, R. Impagliazzo, L. Levin, and M. Luby. Construction of pseudorandom generator from any one-way function, 1993. 8
37. A. Herzberg, M. Jakobsson, S. Jarecki, H. Krawczyk, and M. Yung. Proactive public key and signature systems. In *ACM Conference on Computers and Communication Security*, 1997. 1
38. A. Herzberg, S. Jarecki, H. Krawczyk, and M. Yung. Proactive secret sharing, or: How to cope with perpetual leakage. In *Proc. CRYPTO 95*, pages 339–352, 1995. 24, 26, 27, 29
39. G. Hughes and T. Coughlin. Tutorial on hard drive sanitation. 2
40. G. Hughes and T. Coughlin. Secure erase of disk drive data. pages 22–25, 2002. 2
41. R. Impagliazzo, L. A. Levin, and M. Luby. Pseudo-random generation from one-way functions. In *Proc. 21st ACM Symp. on Theory of Computing*, pages 12–24, 1989. 8
42. R. Impagliazzo and M. Naor. Efficient cryptographic schemes provably as secure as subset sum. 9. 33
43. G. Itkis and L. Reyzin. Sibir: Signer-base intrusion-resilient signatures. In *Proc. CRYPTO 2002*, pages 499–514, 2002. 1, 2
44. S. Jarecki and A. Lysyanskaya. Adaptively secure threshold cryptography: Introducing concurrency, removing erasures. In *Proc. EUROCRYPT 2000*, pages 221–243, 2000. 1, 2
45. M. Krause and S. Lucks. On the minimal hardware complexity of pseudorandom function generators. 32
46. H. Krawczyk. New hash functions for message authentication. In *Proc. EUROCRYPT 95*, pages 301–310, 1995. 13
47. N. Linial, Y. Mansour, and N. Nisan. Constant depth circuits, fourier transform, and learnability. 32
48. C. Lu, O. Reingold, S. Vadhan, and A. Wigderson. Extractors: Optimal up to constant factors, 2003. 16
49. C-J. Lu. Encryption against storage-bounded adversaries from on-line strong extractors. In *Proc. CRYPTO 2002*, pages 257–271, 2002. 3
50. Anna Lysyanskaya. Efficient threshold and proactive cryptography secure against the adaptive adversary (extended abstract), 1999. 1
51. Y. Mansour, N. Nisan, and P. Tiwari. The computational complexity of universal hashing. In *Proc. 22nd ACM Symp. on Theory of Computing*. 4, 13
52. U. Maurer. A provably-secure strongly-randomized cipher. In *Proc. EUROCRYPT 90*, pages 361–373, 1990. 3, 5
53. U. Maurer. Conditionally-perfect secrecy and a provably-secure randomized cipher. pages 53–66, 1992. 3, 5
54. E. Mossel, A. Shpilka, and L. Trevisan. On ϵ-baised generators in nc^0. In *Proc. 43rd IEEE Symp. on Foundations of Comp. Science*, pages 136–145, 2003. 32
55. J Naor and M Naor. Small-bias probability spaces: Efficient constructions and applications. In *Preliminary version in Proc. STOC '90*, 1993. 13
56. Noam Nisan and David Zuckerman. Randomness is linear in space. *Journal of Computer and System Sciences*, 52(1):43–52, 1996. 3, 16
57. Department of Defense. *DoD 5220.22-M: National Industrial Security Program Operating Manual*, 1997. 2
58. R. Ostrovsky and M. Yung. How to withstand mobile virus attacks. pages 51–61, 1991. 1, 24
59. I. Damgård and J. Nielsen. Improved non-committing encryption schemes based on a general complexity assumption. In *Proc. CRYPTO 2000*. 1, 2
60. Omer Reingold, Ronen Shaltiel, and Avi Wigderson. Extracting randomness via repeated condensing. In *IEEE Symposium on Foundations of Computer Science*, pages 22–31, 2000. 17
61. Adi Shamir. How to share a secret. In *Communications of the ACM*, pages 612–613, 1979. 29
62. S. Vaarala. T-110.5210 cryptosystems lecture notes, implementation issues, 2006. 2
63. Salil P. Vadhan. Constructing locally computable extractors and cryptosystems in the bounded-storage model. *J. Cryptol.*, 17(1):43–77, 2004. 4, 17, 18
64. Bennet Yee. *Using secure coprocessors*. PhD thesis, May 1994. 5