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, canetti@csail.mit.edu
[2] Google, Inc. (work done at Weizmann Institute of Science), droreiger@gmail.com
[3] MIT and Weizmann Institute of Science, shafi@theory.csail.mit.edu
[4] MIT, dylim@theory.csail.mit.edu

Abstract. Erasing old data and keys is an important capability of honest parties in cryptographic protocols. It is useful in many settings, including proactive security in the presence of a mobile adversary, adaptive security in the presence of an adaptive adversary, forward security, and intrusion resilience. Some of these settings, such as achieving proactive security, is provably impossible without some form of erasures. Other settings, such as designing protocols that are secure against adaptive adversaries, are much simpler to achieve when erasures are allowed. Protocols for all these contexts 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 imperfect or *partial erasures* where erasure instructions are only partially effective and leave almost all the data erased intact, thus giving the honest parties only a limited capability to dispose old data. Nonetheless, we show how to design protocols for *all* of the above settings (including proactive security, adaptive security, forward security, and intrusion resilience) for which this weak form of erasures suffices.

We show how to automatically modify protocols relying on perfect erasures to ones for which partial erasures suffices. Stated most generally, we provide a general compiler that transforms any protocol using perfect erasures into one that maintains the same security properties even 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 compiler must incur a cost in additional storage, and that our compiler is near optimal in terms of its storage overhead.

We also give computationally more efficient compilers for a number of special cases: (1) when all the computations on secrets can be done in constant parallel time (NC^0); (2) for a class of proactive secret sharing protocols where we leave the protocol intact except for changing the representation of the shares of the secret and the instructions that modify the shares (to correspondingly modify the new representation instead).

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, sensitive data and keys, is an important ability of honest parties 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:

[⋆] This is the full version of the paper appearing in ICALP08 under the same title.
[†] Supported by the Weizmann Chais Fellows Program for New Scientists.
[‡] Supported by NSF Grants CCF-0514167, CCF-0635297, NSF-0729011, the RSA Professor Chair, and by the Weizmann Chais Fellows Program for New Scientists.
[§] Supported in part by NSF grant CCF-0635297.

- **Proactive Security:** One example where some form of erasures is provably necessary is in the setting of *proactive security* [61]. Here time is split into fixed size intervals, or *time periods*, and a *mobile adversary* is considered. A mobile adversary can corrupt different parties in different time periods, subject to an upper bound on the total number of corrupted parties per time period. Namely, the identity of the corrupted parties may change from one time period to the next. Ostrovsky and Yung [61] studied the question of achieving general secure computation and presented an information theoretic solution robust against mobile adversaries. At the heart of their solution (as in all subsequent papers on proactive security, for instance [32,30,39,13,52]) is a secret sharing method in which at the end of every time period, the old shares held by parties are first replaced by new shares and then erased. It is easy to see that secret sharing would be impossible to achieve without some form of erasures: otherwise a mobile adversary which is able to corrupt every single party in some time period or another, can eventually recover all old shares for some single time period and recover the secret.
- **Forward Security:** Erasures are essential to even define (as well as solve) *forward security* [35,21]. Forward security is an approach taken to tackle the *key exposure* problem, so that exposure of long-term secret information does not compromise the security of previous sessions. This notion was first proposed in the context of key-exchange protocols by Günther [35], Diffie, Van-Oorschot, and Weiner [21], and by Anderson [2] for the challenging setting of non-interactive public-key encryption. Again, the lifetime of the system is divided into N intervals (or time periods). The receiver initially stores the secret key SK_0 and this secret key "evolves" with time: at the beginning of time period i (or, one can think of it as the end of time period $i-1$), the receiver applies some function to the previous key SK_{i-1} to derive the current key SK_i; the key SK_{i-1} is then erased and SK_i is used for all secret cryptographic operations during period i. To make such a scheme viable, the public (encryption) key remains fixed throughout the lifetime of the scheme. A forward-secure encryption scheme guarantees that even if an adversary learns SK_i (for some i), messages encrypted during all time periods prior to i remain secret. Obviously, this is provably impossible to do without some form of erasures.
- **Intrusion-Resilience:** Intrusion resilience is a strengthening of forward security [45] which can be viewed as combination of forward and backward security, i.e. an adversary that attacks the system and gets the current key at time t cannot compromise the security of sessions either before t (forward security) or after (backward security). Again as in the case of forward security it is straight forward to see that it is impossible to do without some form of erasures.
- **Adaptive Security:** An example of a different flavor of the utility of erasures is in adaptive security, which guards 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. Indeed, although protocols have been designed which remain secure even without erasures, these tend to be much more complex than those that rely on data erasures [46,12,8,63].

Unfortunately, perfect erasures of data are hard to achieve in practice and thus it is problematic to assume that they are available, as pointed out by Jarecki and Lysyanskaya [46] in their study of adaptive adversaries versus static adversaries in the context of threshold secret sharing.

Some of the difficulties in implementing perfect erasures are illustrated in the works of Hughes and Coughlin, Garfinkel, and Vaarala [41,42,31,67]. The root cause of the difficulties is that systems are actually designed to preserve data, rather than to erase them. 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 [60]. This takes the order of days to complete 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", there is the memory remanence effect – previous states of memory can be determined even after power off. At the software/operating systems level, many operating systems detect and remap bad sectors of the hard drive on the fly. Such a remapping is stored in the drive's file system, thus providing a software layer of hard disk defect management. Original data can remain in the bad sectors and be recoverable to a certain extent, even if these sectors are labeled "bad" and are not considered part of the storage anymore. Erasures also complicates the management of the virtual memory system: sensitive data in memory might be paged onto the hard drive. In fact the entire memory contents will be saved onto the hard drive when a system hibernates.

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* and h the *partial-erasing function*. 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[1]. 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 or block 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 the CPU registers, as well as in the cache, main memory (RAM), and hard drives. We assume all types of storage are only partially erasable with the exception of a constant number of constant size CPU registers, which are assumed to be perfectly erasable. We emphasize that having a constant number of constant size registers being perfectly erasable just means that we have perfect erasures only for some constant space. This limitation ensures that we cannot use the registers to (effectively) perfectly erase all other types of storage and thus circumvent the lack of perfect erasures for them, since at no time can the registers hold any non-negligible part of the secret. We call this the *register model*.

We remark that it seems very reasonable to assume that a constant number of constant size CPU registers can be effectively erased, whereas main memory, hard drives, etc. cannot.

[1] 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.

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.

Since we only (need to) focus on the part of the storage that needs 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).

Results and Techniques. Our main result is a compiler that on input any protocol that relies on perfect erasures for its security (proofs), outputs a protocol with the same functionality that remains secure even if the erasures are only partial. We assume that the original protocol uses two types of storage – one that is (perfectly) erasable, and one that is persistent (never going to be erased). Of course, this persistent part need not be changed, so in the sequel we will ignore this part of the storage, and our transformation applies only to the erasable part. (In particular, in the case of a single secret, the erasable part is a single secret.) Consequently, when we focus on the erasable part of the storage and analyze the storage and computation costs of our transformation, we will be overstating the costs: in reality, a large part of the storage is persistent, and there is no storage or computation blow up for this part.

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 [54,55]. Work by Lu [50] casted results in the bounded storage model in terms of *extractors* [59], 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 $(R, X, \mathsf{Ext}(R, X) \oplus s)$, where Ext is a strong extractor that takes R as seed and X as input, and generates an n-bit output such that $(R, \mathsf{Ext}(R, X))$ 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 $\mathsf{Ext}(R, X) \oplus s$ are left intact.

For the 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 in 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 [53]. It would seem superficially that locally computable strong extractors [68] can be used, but unfortunately they cannot, as we show in theorem 3.3 on page 23.

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 (see Figure 1). 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.

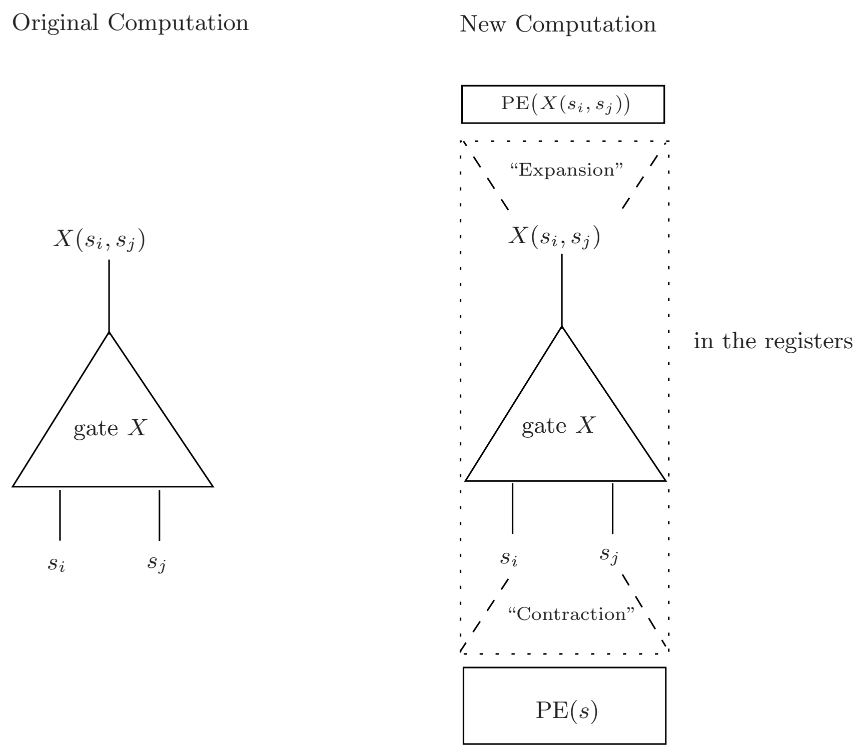


Fig. 1. Computations Involving Secret $s = s_1, ..., s_n$; PE($\circ$) is the partially erasable form of its argument

For maximum generality we formulate our results in the *Universally Composable (UC)* framework. In particular we use the notion of *UC-emulation* [10], 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 $\Pi_{new} := \textsc{Compiler}(\Pi_{org})$ 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 Sections 5 and 6 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 first case, since each output bit can be computed with only a constant number of input bits, we can afford to keep all the input bits required in the registers and compute each output bit in "one-shot". In the second case of proactive secret sharing, we can apply our expanded representation directly to the secret and its shares, and correspondingly modify the instructions that operate on the shares, leaving the rest of the protocol intact. In both cases we avoid partially erasable computation at the gate level, and thus the resulting protocols are more efficient computation wise than via the general compiler. 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 (Everlasting Security). 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

Side Channel Attacks. We can view partial erasures as a special case in the more general framework of side channel attacks which takes into account the physical nature of computation, as opposed to an idealized, abstract computation model like Turing Machines (which can, among other things, overwrite data). See [56] for a nice model of computation which explicitly captures physical attacks, and for schemes that are provably secure against all physically observable attacks. We stress that in this work we focus only on partial erasures. In particular, the adversary is not allowed physical access to the parties before corruption, and so is not allowed to measure say the time or power taken to do some computation, or to monitor the electromagnetic radiation. If these side channel attacks are a concern, then they have to be dealt with in other ways – for instance using provably secure schemes given in [56], or using heuristic countermeasures like shielding the hardware to reduce radiation, power filtering and balancing, and injecting noise to the side channel.

On the other hand, we note that even though we can view partial erasures as a special case of side channel attacks, the "practical implications" go beyond just partial erasures. In particular, as we elaborate at the end of Section 4, a side benefit of using our constructions is that it can be resistant to a practical class of physical attacks [36] that involves freezing RAM and recovering secrets from it.

The Bounded Storage Model (BSM) Much of modern cryptography assumes that the adversary is computationally bounded. The Bounded Storage Model (BSM) proposed by Maurer [54,55], makes in contrast assumptions on the storage capabilities of the adversary. These assumptions enables novel approaches to the secure communication problem as follows.

As usual communicating parties A and B begin with a short initial secret key k. They use this key k and access to a very long public random string R in order to derive a longer key X, which will enable them to communicate privately over the public channel by simply using X as a one-time pad. The key (or one-time pad) derivation protocol takes place in two phases.

Phase I: During this phase all parties (A, B, and the adversary) have access to the long random string R. A and B (non-interactively) apply a public *key-deriving function* f to (k, R) to derive the long key X that they can use as a one-time pad. The adversary which has access to R is space bounded, and cannot store all of R. This is formalized by modeling the adversary's storage with a length-shrinking *storage function* h_ϕ, i.e. a function $h_\phi : \{0,1\}^* \rightarrow \{0,1\}^*$ s.t. $\forall x, \frac{|h_\phi(x)|}{|x|} \leq \phi$, where ϕ is a constant $0 \leq \phi < 1$ which is called the *storage fraction* of the adversary (or of function h_ϕ). The best the adversary can do at this phase is to store $h_\phi(R)$.

Phase II: The common random string disappears. In this phase, the honest parties use their derived key X to encrypt their communication. The question is, how much information can the adversary find about X? A sequence of works including [55,7,26] culminated in showing that, for ϕ arbitrarily close to 1, there exists an explicit key-deriving function f such that, $X = \mathsf{f}(k, R)$ is ϵ-close to uniform given $k, h_\phi(R)$. Namely, even if the adversary is given the initial secret key k at this point (in phase II), on top of $h_\phi(R)$ that it stored in phase I, it still cannot distinguish X apart from random with probability more than ϵ.

The ideas emerging from the BSM research inspire a way to capture some weak form of erasures. In particular, the information about R (the long common random string) stored by the bounded-space adversary in the BSM model was captured by computing an arbitrary length-shrinking function applied to R. In the partial erasures setting we will use the same kind of length-shrinking function to capture the act of partially erasing old shares of a secret.

However, the settings of the BSM and partial erasures are fundamentally different. In the BSM possibility is proved by putting limitations on the *adversary* (storage), whereas in our work possibility is achieved in spite of putting limitations on the *honest parties* (erasing capability). Thus, although some of techniques are similar the underlying 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 the constant number of constant size registers, whereas in the BSM the extractors are not necessarily computable with constant size 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 r, the length of R_i. Whenever the adversary gets access to new information (e.g. R_{i+1}), it has to update the information Q_i kept in its storage (e.g. by applying some update function $g : \{0,1\}^r \times \{0,1\}^{\phi r} \mapsto \{0,1\}^{\phi r}$, on (R_{i+1}, Q_i) to get Q_{i+1}). The same assumption holds for results in the bounded retrieval model [20,18,24,25,27]. For instance [27] constructs intrusion resilient secret sharing schemes by making the secret shares large and assuming that the adversary will never be able to retrieve any piece completely. On the other hand, for partial erasures this bound on the storage is unreasonable, and we allow the adversary's storage to grow with time, namely he gets ϕ fraction of some R_i for each erasure operation.

Exposure Resilient Functions (ERF) Exposure-resilient functions, or ERFs, were introduced by Canetti et al. [11,22]. An ℓ-ERF is a threshold function with a random input, which if the adversary learns all but ℓ bits of the input, cannot distinguish the output of the function from random. Formally, a function $f : \{0,1\}^m \to \{0,1\}^n$ is ℓ-ERF if for any choice of $m - \ell$ bits and for $x \overset{\$}{\leftarrow} \{0,1\}^m$ and $u \overset{\$}{\leftarrow} \{0,1\}^n$, given the $m - \ell$ bits of x, $f(x)$ and u are indistinguishable, either perfectly, statistically or computationally.

The ERF objectives seem very similar to partial erasures. However, the settings are different. It turns out that in partial erasures we would think about our inputs as consisting of two parts, so to aid comparison, below we use the same notation, (R, k), to denote the input.
In ERFs:

(a) There is a fixed known function f.
(b) The adversary chooses up to $m - \ell$ bits of (R, k).
(c) The adversary sees his choice of the bits of (R, k).
(d) The adversary attempts to guess $f(R, k)$, using his knowledge of (R, k).

In partial erasures:

(a) There is a fixed known function f.
(b) The adversary chooses a shrinking function $h(\cdot)$ to apply on k, leaving up to $m - \ell$ bits.
(c) The adversary sees the result of function $h(\cdot)$ applied on k, *and also sees independently chosen new data* R.
(d) The adversary attempts to guess $f(R, k)$, using his knowledge of $h(k)$ and R.

Notice that in (b) for ERFs, it is crucial that the adversary only gets to choose exact bits of (R, k) to see, otherwise he can just compute $f(R, k)$ and store (parts of) it.

On the other hand, in (b) for partial erasures, the adversary can get to store $h(k)$, which represents arbitrary $m - \ell$ bits of information about k. This is because R will only be seen by the adversary later (in (c)), and will thus be independent of $h(\circ)$ and k regardless of whether $h(\circ)$ outputs only exact bits of its input or not.

Therefore, due to the difference in settings, ERFs can only tolerate adversaries knowing $m - \ell$ exact bits, whereas partial erasures can tolerate leakage of $m - \ell$ arbitrary bits of information.

Encryption as Erasures. One straightforward way of achieving secure erasures is to keep the data in encrypted form and the key on some secure part of the storage that is assumed to be perfectly erasable. When the data is to be erased, erasing the key suffices. Without the key, the encrypted data is useless, and so in this sense the data has been erased.

In more detail, as Di Crescenzo et al. [17] noted, one simple but inefficient way to implement erasable memory can be obtained by using the crypto-paging concept of Yee [69]. Assume that we have a linear amount of perfectly erasable memory $Perfect$, and we want to make the other part of the memory $Persistent$ (poly sized) into an erasable one, despite the fact that what is stored in $Persistent$ remains there forever. The preprocessing stage consists of choosing a secret key k_1 for an encryption scheme (Gen, Enc, Dec), storing it in $Perfect$, and using it to encrypt the contents (say C_1, C_2) to be stored in $Persistent$. When C_1 is to be erased, $Persistent = Enc_{k_1}(C_1, C_2)$ is decrypted and a new key k_2 is chosen (replacing k_1) and used to encrypt C_2, which is then stored back to $Persistent$. If the adversary breaks in at this time, he gets $Enc_{k_1}(C_1, C_2)$, $Enc_{k_2}(C_2)$, and k_2, so he recovers C_2. However, without k_1, C_1 is effectively erased. Di Crescenzo et al. improve on the efficiency of this scheme.

To achieve statistical security, Shannon's classic result [66] implies that this secure part of the storage has to be of size at least as large as the other part, so such a solution is not too interesting. On the other hand if only computational security is desired, then this secure part of the storage only needs to be linear in the security parameter, and the other part of the storage can be polynomial.

In contrast, using our compiler, we achieve statistical security only assuming that constant sized registers are perfectly erasable. Even without using our general compiler, we can directly apply our ideas to the "encrypted erasures" described above, to get computationally secure erasures. The advantage of using our ideas here is that the assumption is weakened (from having perfect erasures for a linear storage to having perfect erasures for a constant sized storage).

2 The Model of Partial Erasures

As sketched in the introduction, we model partial erasures by an arbitrary length-shrinking function $h : \{0,1\}^m \rightarrow \{0,1\}^{\lfloor \phi m \rfloor}$, 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 (Partial Erasing Function, Leakage Fraction, Block Length). *A* partial-erasing function *h is a length-shrinking function $h : \{0,1\}^m \mapsto \{0,1\}^{\lfloor \phi m \rfloor}$, with a positive integer m as its* block length *and a real number ϕ s.t. $0 \leq \phi \leq 1$ as its* leakage fraction.

Definition 2 (Partially Erasable (PE) Tape). *Let h be a partial-erasing function with m as its block length and ϕ as its leakage fraction. A tape of a Turing Machine is said to be* partially erasable (PE) *if:*

– ***It has two sides****:*
  – a primary side *pre-partitioned into m-bit blocks, and*
  – a secondary, “shadow” side*, pre-partitioned into m-bit blocks.*
– ***The primary side****:*
  – may be read multiple times*, i.e. the corresponding read head is free to move both forward and backward,*
  – may be written once*, i.e. the corresponding write head always moves forward, and*
  – has an “erase” write head *that is free to move both forward and backward, but can only write $\perp$, the empty or blank symbol, to replace a block by $\perp^m$* [2].
– ***The secondary, “shadow” side****:*
  – cannot be read*, and*
  – has a write head *with a separate control logic (state transition function) which computes the function h. This write head is used to implement the partial erasure operation.*
– ***It supports the partial erasure operation****: on input a block number i, let $x \in \{0,1\}^m$ be the contents of the i-th m-bit block on the primary side, and the Turing Machine:*
  – *computes $h(x)$ and writes it on the i-th block on the shadow side* [3]*, and*
  – *overwrites the i-th block of the primary tape, currently containing x, with $\perp^m$ (so the Turing Machine no longer gets any access to the data).*

Remark 2 (The Computation of h is Unbounded). In the above definition we specifically introduced a separate control logic for computing h to address the fact that whereas the complexity of the Turing Machine may be restricted, e.g. to polynomial time, the complexity of h is not. Alternatively, one may view the computation as an oracle call.

Definition 3 (Externally Writable (EW) Tape). *A tape of a Turing Machine is said to be* 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 forward.*

The only differences between our model of computation and the interactive Turing Machine of [10] are 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.

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 (or rather, encode them in some way and keep the encoded version on the computation tape), and thus they 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. See remark 3.

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 [10]:

Definition 4 (Partially Erasable Interactive Turing Machine (PEITM)). *Let h be a partial-erasing function with m as its block length and ϕ as its leakage fraction. A* partially erasable interactive Turing Machine (PEITM) *is a Turing Machine with the following tapes:*

– *An EW identity tape (here the identity of the party is written).*

[2] The reason we have two different write heads is that overwriting old data is a form of erasures, and we want to make it clear whenever erasures occur. Forcing the ordinary write head to always move forward means it cannot overwrite or erase, which is left to the special head.

[3] More precisely it passes control to the control logic computing h, which writes $h(x)$ of size $\phi m < m$ to the corresponding block on the hidden side in some canonical way, for instance by always starting from the beginning of the block.

- *An EW security parameter tape.*
- *A PE input tape.*
- *A PE work tape.*
- *An EW incoming communication tape.*
- *A read once random tape.*
- *An output tape.*
- *A PE subroutine output tape (this and the next tape are used in the UC framework).*
- *A read and write one-bit activation tape.*

Definition 5 (Adversary View). *When the adversary corrupts a party, it gains access to all the tapes of the party's TM, including the shadow sides of all the PE tapes (and thus gains access to the partially erased data), with the* ***exception*** *of the random tape.*

Remark 3 (Randomness). We think of the randomness as coin tosses rather than as a tape of random bits prepared in advance. Essentially, this just boils down to assuming that we have a random (one) bit generator in the CPU that is perfectly erasable. The reason for thinking about randomness this way is the following. The encoding of the secret has to be randomized, and anyone with access to all of the randomness can always decode and get the secret back. Therefore if upon corruption the adversary gets to see all the randomness, then effectively this is the same as having no erasures. On the other hand, to prevent the Turing Machine from using the random tape to trivially implement erasures, we require that the random tape be un-writable and read once. In particular, if the Turing Machine wants to use the random bits, it has to encode them somehow and keep the encoded version on the computation work tape, which (upon corruption) the adversary will be allowed to see anyways.

Remark 4 (When does the adversary choose h). 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 certainly suffices for modeling the imperfect erasures due to physical limitations.

Remark 5 (How to model ITMs). We refer the reader to [10] 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 to the advantage of the adversary, in bounding the resources of the ITMs, the time taken to partially erase (i.e. compute h) should be ignored.

Alternatively, instead of the ITM model of the UC framework [10], one could apply the modifications (of changing some tapes to PE instead of EW, and making the random tape read once and hidden from the adversary) to the ITM model of [34]. We choose to use the ITM model of the UC framework [10] since our results are later stated in the UC framework.

Remark 6 (Boolean circuits versus Turing machines). Furthermore, we will actually be using boolean circuits as our model of computation instead of Turing Machines. The reason is that boolean circuits already operate at the bit level, allowing us to explicitly talk about leveraging the constant number of constant size registers so that the computations would not leak too much information. For instance, for a computation on a secret, the input bits to the gates at the first level of the corresponding circuit are actually bits of the secret, and the output bits of gates represent various stages of the output; these input and output bits need to be kept secret, which is where the perfectly erasable registers come in. Therefore, instead of using Turing Machines, we will be using circuits and the register model. This is without loss of generality since for any TM running in time $T(n)$, there exists a family of circuits $\{Cn\}$ of size $|C_n| = T(n) \times polylog(T(n))$ computing the same function [62]. The same holds for any PEITM.

2.1 The Memory Model

We envision that processors participating in protocols can store data (secret and otherwise) in the CPU registers, as well as in the cache, main memory (RAM), and hard drives. We assume all types of storage are only partially erasable with the exception of a constant number of constant size CPU registers, which are assumed to be perfectly erasable. We call this memory model the *register model*. We remark that it seems a very reasonable assumption that a constant number of constant size CPU registers can be effectively erased, whereas the main memory, hard drives, etc. cannot.

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 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.

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 7 (The Registers). 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 registers of 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).

2.2 Discussion on the Partial Erasures Model

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 fail 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. In contrast, 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).

2.3 Alternative Models of Partial Erasures

The function h we consider is a deterministic length-shrinking function, that can be different each time erasures is to be done, but this choice has to be done independent of the data to be erased. It is assumed to be a function only of the storage contents to be erased. In Section 1.1 and above we argued that this suffices for capturing imperfect erasures due to physical limitations. We discuss alternative models in the conclusion.

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 and a small computation overhead, 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 Section 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 6 (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 $\Delta(A;B) := \frac{1}{2}\sum_{\sigma\in\Sigma}\Big|\Pr_A[\sigma]-\Pr_B[\sigma]\Big|$.*

Definition 7 (Statistical Distance from 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 uniform given B *to mean that $d(A|B)\le\epsilon$.*

Definition 8 (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 an (ℓ,α,ϕ)*-partially erasable form *if $\forall s\in\{0,1\}^n$, for any h with leakage fraction ϕ,*

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

In correctness, we require that the expansion function be non-trivial and not lose information about the secret. It does not mean that whenever we need bits of the secret s, we would actually compute the whole s from the expanded form in one shot, since once the "bare" secret appears outside of the registers, there is nothing much that can be done: bits of the secret would be leaked even after partially erasing (see the elaboration in the paragraph following the next).

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

We also require partially erasable forms to be computable with constant memory (i.e. in the register model), bit-by-bit, so that no bit of the secret will be leaked in expanding and contracting to and from the secret (or more precisely, the functions work on some constant number of bits of the secret at a time). This will be discussed in more detail in the next chapter. For $(\mathsf{Exp},\mathsf{Con})$ a partially erasable form, we introduce further notation. let $\mathsf{Exp}_i(s,\$)$ denote the algorithm for expanding the i-th bit of the secret, and $\mathsf{Con}_i(\mathsf{Exp}_i(s,\$))$ denote the algorithm for contracting the i-th bit of the secret. We overload the notation so that for instance, $\mathsf{Con}_i(\circ)$ takes as input both $\mathsf{Exp}(s,\$)$, the expanded form of the whole secret, and $\mathsf{Exp}_i(s,\$)$, the expanded form of just the i-th bit of the secret.

Note that $\mathsf{Exp}_i(s,\$)$ is not necessarily independent from $\mathsf{Exp}_j(s,\$)$ (i.e. the coins used are not necessarily independent), and in fact for space efficiency purposes we would like them to be dependent– but of course there is a tradeoff between efficiency and security. For completeness, on page 19 immediately before corollary 1, we briefly describe this dependency (for partially erasable forms based on Toeplitz hashing). As we will see in the next section, the storage costs can be amortized so that it no longer grows with m. Regarding computational efficiency, since our general compiler works at the gate level, there is a computational overhead incurred to do the contraction and expansion before and after each gate computation, respectively. In Sections 5 and 6 we give special compilers that decrease this blow up in computational cost.

Whenever it is clear from the context we write "the partially erasable (or expanded) form of s" to mean $\mathsf{Exp}(s,\$)$ instead of $(\mathsf{Exp},\mathsf{Con})$, especially since the Con functions we consider are all straightforward given

Exp. 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 [43], 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 [38]). *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* [4],

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

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 $\mathbf{P}_k(h(k) = y) \leq 2^{-(1-\lambda)m}$.

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

[4] Yevgeniy Dodis pointed out to us that by using their generalized leftover hash lemma in [23], we can save a root, to get $d(H_R(k)|R, h(k)) \leq \frac{3}{2^{2/3}} \cdot 2^{-\frac{1}{2}(1-\phi)m + \frac{n}{2}}$.

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

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. □

Dealing with Relations Between Secrets What happens when the secrets are used to compute some new secrets? How would the adversary's information add up as he sees more partially erased information? Consider the following scenario. In the original protocol (before making the secrets partially erasable), for some secrets s_j and some functions g_j, say that $s_i = g_i(s_{i-1})$ and later s_i is used to do some cryptography. Now, if we made the secrets partially erasable (not worrying about how we would compute the g_is for the moment), how random is s_i, given partially erased $s_j, j<i$? In the following, we consider a generalization of the above scenario where for secret s_i, the adversary knows a 1-1 and onto function $q_i(\circ)$ which relates it to s_1. Or rather, for each tuple $\big(R_i, h(k_i), H_i(k_i)\oplus s_i\big)$, the adversary knows 1-1 and onto functions which relate $H_{R_i}(k_i)$ and $H_{R_1}(k_1)$.

There are two things that this 1-1 and onto function is trying to capture (see Figure 2). First, consider the case in which the secret s remains the same, and multiple, say two, partial erasures were done on the expanded form. In this case, let us look at how the view of the adversary allows him to add up his information. The first partial erasure gives him $\big(R_1, h(k_1), H_{R_1}(k_1)\oplus s\big)$, and the second gives him $\big(R_2, h(k_2), H_{R_2}(k_2)\oplus s\big)$. This means he gets $H_{R_1}(k_1)\oplus H_{R_2}(k_2)$, and therefore any information he gains on $H_{R_1}(k_1)$ is as good as information on $H_{R_2}(k_2)$ (and vice versa). So the 1-1 and onto function relating $H_{R_i}(k_i)$ and $H_{R_1}(k_1)$ is just the XOR operator.

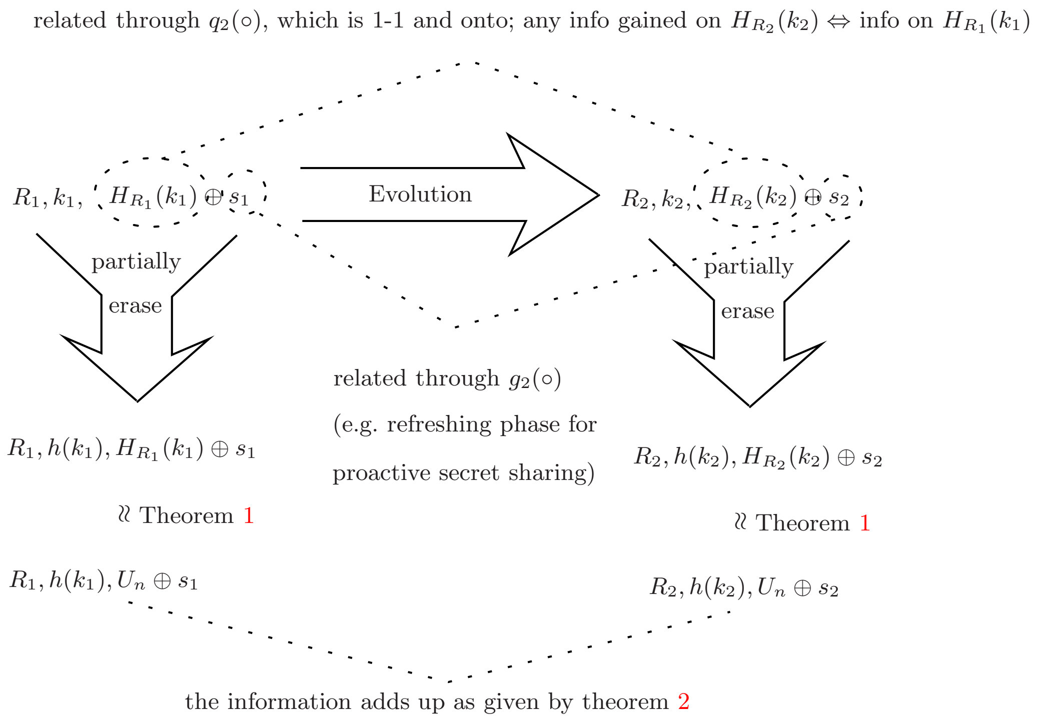

related through $q_2(\circ)$, which is 1-1 and onto; any info gained on $H_{R_2}(k_2) \Leftrightarrow$ info on $H_{R_1}(k_1)$
$R_1, k_1, H_{R_1}(k_1) \oplus s_1$
Evolution
$R_2, k_2, H_{R_2}(k_2) \oplus s_2$
partially erase
partially erase
$R_1, h(k_1), H_{R_1}(k_1) \oplus s_1$
related through $g_2(\circ)$ (e.g. refreshing phase for proactive secret sharing)
$R_2, h(k_2), H_{R_2}(k_2) \oplus s_2$
$\wr\wr$ Theorem 1
$\wr\wr$ Theorem 1
$R_1, h(k_1), U_n \oplus s_1$
$R_2, h(k_2), U_n \oplus s_2$
the information adds up as given by theorem 2

Fig. 2. Secrecy for Multiple Erasures

Second, consider the case in which the secret s evolves over time: at time 1 it is s_1 and at time 2, it evolves to $s_2 = g_2(s_1)$. The first partial erasure gives him $R_1, h(k_1), H_{R_1}(k_1) \oplus s_1$, and the second gives him $R_2, h(k_2), H_{R_2}(k_2) \oplus s_2$. This means he gets $H_{R_1}(k_1) \oplus H_{R_2}(k_2) \oplus s_1 \oplus s_2$. If in addition, he knows any of $H_{R_1}(k_1), H_{R_2}(k_2), s_1$, or s_2, then (through $g_2(\circ)$ which we can, to the advantage of the adversary, assume is public, 1-1 and onto), he knows all of them. For instance, say that he gets $H_{R_1}(k_1)$. Then he can recover s_1 through $H_{R_1}(k_1) \oplus s_1$, compute $s_2 = g_2(s_1)$, and then recover $H_{R_2}(k_2)$ through $H_{R_2}(k_2) \oplus s_2$. So in this case the 1-1 and onto function relating the one-time pads $H_{R_1}(k_1)$ and $H_{R_2}(k_2)$, also takes the evolution of the secrets into account.

To prove security for multiple erasures, we need the following two lemmas (refer to [16] 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 $d(X) \le 1/4$, then:*

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

and

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

Proof. This lemma is a special case of theorem 16.3.2 and lemma 16.3.1 of [16]. □

The following theorem proves that for secrets that possibly evolve over time, the adversary cannot gain too much information even after seeing multiple erasures.

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 $q_i(\circ)$ are public 1-1 and onto functions such that $s = q_i(H_{R_i}(k_i))$. Then, for any $\beta > 0$, m poly in n, and sufficiently large n,*

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

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 $H(\circ)$, the entropy function.

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

The second inequality also requires the fact that for δ and ϵ such that $0 \le \delta \le \epsilon \le 2^{-\frac{1}{\ln 2}}$ (which the right hand side of equation (1) satisfies for sufficiently large m), we have that $-\delta \log \delta \le -\epsilon \log \epsilon$ [5]. The last inequality follows because, given ϕ and n, for any $0 < \beta$ and m poly in n, for sufficiently large n,

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

[5] 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 $0 \le \delta \le \epsilon \le 2^{-\frac{1}{\ln 2}}$, the function $-x \log x$ is monotonically increasing.

By chain rule for mutual information,

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

Also, because the difference in unconditional and conditional mutual information is symmetric in the random variables, (theorem 2 on page 15)

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

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 $q_i(\circ)$s are public, 1-1 and onto functions 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:

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

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 just use $\mathsf{Exp}(s, \$) := (R, k)$ where R and k are random s.t. $R \cdot k = s$ [6]. This can be done easily for a random R or even a random Toeplitz R. The theorem applies even to this case, i.e. regardless of how the secrets are chosen to be dependent on one another, any secret itself is still close to random. Note that we give the adversary 1-1 and onto functions 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[7].

The first expanded form of s is a pair: a 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, which translates into space inefficiency (since the parties have to store them).

[6] In fact this type of expansion is what we are going to use later in Section 5 when we give a special compiler for all known proactive secret sharing schemes.

[7] 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 [53]). 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 a random Toeplitz matrix $R \in \{0,1\}^{n\times m}, k \in \{0,1\}^m$ (where R is fully specified by a random string of $n+m-1$ bits), and store $R, k, R\cdot k \oplus s$ instead (so this triplet is the expanded form of s). 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 $R\cdot k \oplus x$.

Let us describe how the expansion and contraction (using Toeplitz hashing) can be done in the register model. 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 $R_i \cdot k \oplus x_i = s_i$ (where R_i denotes the i-th row of R). As we remarked after definition 8 and above, for efficiency purposes the expansions have to be dependent. Here we need to "reuse" k, and make sure that the rows R_i are generated one by one s.t. the final matrix R is a Toeplitz matrix. This is straightforward to do:. 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.

We have just described how to implement $(\mathsf{Exp}, \mathsf{Con})$ based on Toeplitz hashing in the register model. Furthermore, 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

As we will see in Section 3.4, the randomness efficiency of the hash translates into the space efficiency of our expanded form. The Toeplitz universal hash uses $n+m-1$ bits of randomness, whereas almost universal hash can use just $r < n+m-1$ bits. In this section we examine expanded forms based on almost universal hashing.

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 9 ($\frac{1}{2^n}+\epsilon$-almost Universal Hash Functions). *We say that $\mathcal{H}$ is a $\frac{1}{2^n}+\epsilon$-*almost universal family of hash functions *from $\mathcal{X}$ to $\mathcal{Y}$ if for all $x, x' \in \mathcal{X}$ such that $x \ne x'$, 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 10 (ϵ-biased Distributions [58]). *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 [48], 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 [48]). *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 $(R, h(k))$ 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 lemma 1), 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:

$$\begin{aligned}
d(R \cdot k|R, h(k)) &= \sum_{r,y} \mathbf{P}(R = r \cap h(k) = y) d(R \cdot k|R = r, h(k) = y) \\
&= \sum_{r} \mathbf{P}(R = r) \Big(\sum_{y \in \mathcal{B}} \mathbf{P}(h(k) = y) d(R \cdot k|R = r, h(k) = y) \\
&\qquad + \sum_{y \notin \mathcal{B}} \mathbf{P}(h(k) = y) d(R \cdot k|R = r, h(k) = y) \Big)
\end{aligned}$$

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

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:

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

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 the righthand side of equation 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 $r \leq n+m-1$. Therefore,

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

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

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

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 $q_i(\circ)$ be public 1-1 and onto functions 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,*

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

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

$$\sqrt{(\ln 2)\ell 2^{-\frac{1}{3}(1-\phi)m + \frac{(\beta+1)n}{3} + \frac{4}{3}}} \leq 2^{-(\alpha+1)}$$
$$\Leftrightarrow \ell \leq \frac{2^{-\frac{4}{3} - 2(\alpha+1) - \frac{1}{3}(1+\beta)n + \frac{1}{3}m(1-\phi)}}{\ln 2} \tag{17}$$
$$\Leftrightarrow m \geq \frac{3\log((\ln 2)\ell) - 2 + 6(\alpha+1) + (1+\beta)n}{(1-\phi)}. \tag{18}$$

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 $\mathsf{Exp}(s,\$) := (R, k, H_R(k) \oplus s)$ 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 inequality 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

At this point, one might think that any strong extractor would suffice, and therefore we might be able to use optimal strong extractors and correspondingly, get the most space-efficient construction of partially erasable forms. Unfortunately, as we will see in this section, even though any expanded form based on any strong extractor would satisfy the secrecy requirement, the constant memory requirement is not easy to satisfy.

Let us first review the some definitions. Extractors were introduced by Nisan and Zuckerman [59], and have played a unifying role in the theory of pseudo-randomness.

Definition 11 (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 12 (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 [51] give extractors that are simultaneously optimal (up to constant factors) in both seed length and output length.

Theorem 6 (Optimal Extractors for Constant ϵ [51]). *For any constants* $\alpha, \epsilon > 0$*, every* n *and every* $k \le n$*, 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 $\epsilon > \exp(-k/2^{O(\log^* k)})$ [51]). *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 [64] 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 [64]). *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* $k \le n$*, 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 $(R, h(k))$ 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 13. □

Theorem 12 (Secrecy for Multiple Erasures using Strong Extractors). *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 $q_i(\circ)$ be public 1-1 and onto functions 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 16. □

Locally computable extractors is a class of strong extractors used to construct private-key cryptosystems in the bounded storage model [50,68]. These extractors can be computed by reading only "a few" bits from the random source. We show in the following theorem that being locally computable is not sufficient for our purposes – in particular "a few" bits, while small, would need to be non-constant.

Theorem 13 (Locally Computable Strong Extractors do not give Partially Erasable Forms). *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 [68], 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 gain some information, then the analysis is a bit more complex. We need the following from [16]: 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 $\Psi \geq \frac{1}{1-\phi}\left(1 - \text{neg}(\alpha)\right)$.

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 [15]), 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 this into 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 23 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.

In this section we present a general compiler that on input a protocol that relies on perfect erasures for its security (proofs), outputs a protocol with the same functionality that remains secure even if the erasures are only partial. The input protocol could be one that is adaptively secure, forward secure, intrusion resilient, proactively secure, etc.

We assume that all protocol computations are given to us as boolean circuits consisting of gates with fan-in 2 and fan-out 1. We remark that any TM running in time $T(n)$ can be converted to a circuit of size $T(n) \times polylog(T(n))$ computing the same function [62].

We apply the technique described in the previous section at the gate level in Section 4.1 (so that computations on the secrets are partially erasable too), and (based on this) present our general compiler in Section 4.2.

In the next section two sections following this we give special-tailored compilers that are more efficient than the general one.

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 a partially erasable form. Consider any efficient computation g on s, which can be modeled as a poly(n)-sized circuit. Without loss of generality and to establish notation, we consider:

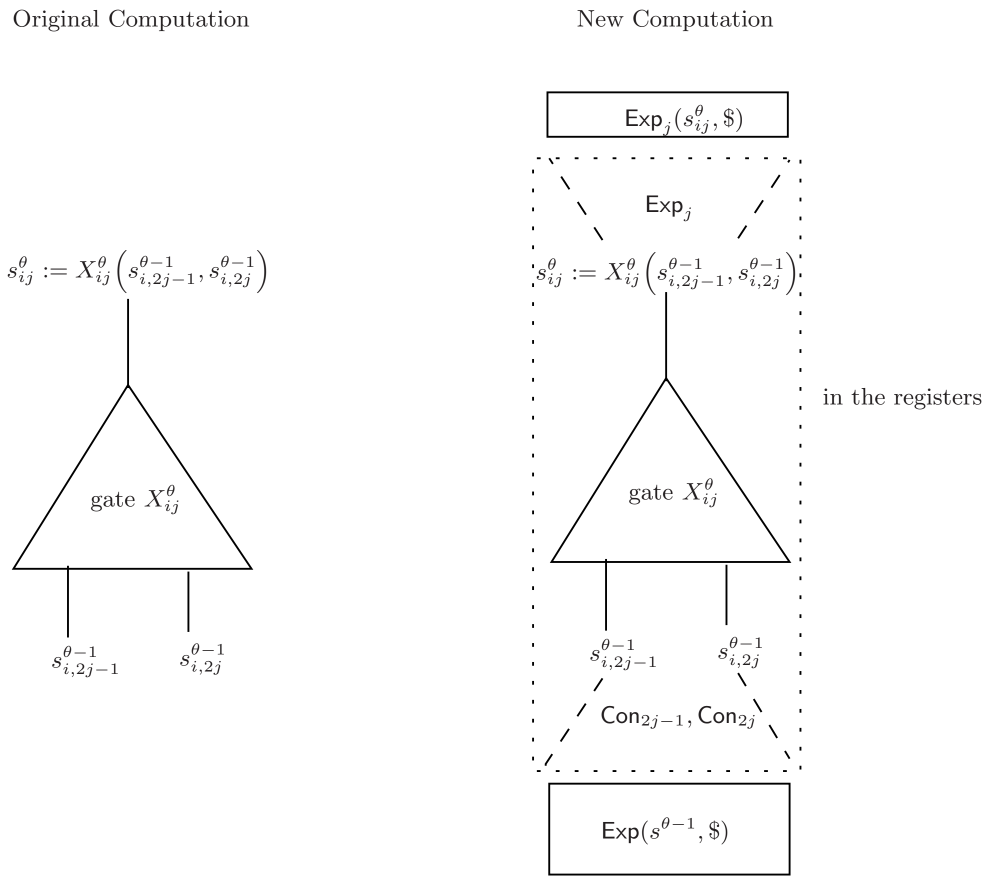
- g to be length preserving (outputs n bits),
- each output bit g_i separately as being computed by polynomial-sized circuits C_{g_i},
- each circuit C_{g_i} to be of depth $\Theta = \Theta(n)$ for some polynomial Θ, with the highest level being the single gate that outputs g_i,
- each level θ of each circuit C_{g_i} to consist of gates X^{θ}_{ij} with fan-in of two and fan-out of one, and
- each gate X^{θ}_{ij} (the j-th gate at the θ-th level of the circuit computing the i-th bit of g) to have $s^{\theta-1}_{i,2j-1}$ and $s^{\theta-1}_{i,2j}$ as its inputs and s^{θ}_{ij} as its output, so the output bits from the previous level are always read in order – in particular, the first level of each of the circuits C_{g_i} uses bits of the original secret $s^0_{ij} := s_j$ in order.

Now let us describe how we can do the computation g on s (or rather, on $\mathsf{Exp}(s, \$)$), without leaking too much information.

To evaluate a gate X^{θ}_{ij}, the two corresponding input bits $s^{\theta-1}_{i,2j-1}$ and $s^{\theta-1}_{i,2j}$ are reconstructed from their expanded forms in the registers (using Con_{2j-1} and Con_{2j}). The gate is evaluated, resulting in an output bit

s^θ_{ij} in the registers. This output bit is expanded into the partially erasable form and output to main memory[8], by using Exp_j. The comparison of the original versus the new gate level computation is summarized in the Figure 3, which is a more detailed version of Figure 1 in the introduction.

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 output wires of each level of each circuit into groups of size up to n (i.e., there are up to n wires in each group), and expand these outputs bits into a single Ψn-bit string. More precisely, by "group" we mean that the output bits should be individually expanded by Exp_i *dependently.* This will make sure that the overhead of the general compiler will be as claimed. For simplicity of presentation, this amortization will not be shown explicitly.



Original Computation
New Computation
$s^\theta_{ij} := X^\theta_{ij}\left(s^{\theta-1}_{i,2j-1}, s^{\theta-1}_{i,2j}\right)$
gate X^θ_{ij}
$s^{\theta-1}_{i,2j-1}$
$s^{\theta-1}_{i,2j}$
$\mathsf{Exp}_j(s^\theta_{ij}, \$)$
Exp_j
$s^\theta_{ij} := X^\theta_{ij}\left(s^{\theta-1}_{i,2j-1}, s^{\theta-1}_{i,2j}\right)$
in the registers
gate X^θ_{ij}
$s^{\theta-1}_{i,2j-1}$
$s^{\theta-1}_{i,2j}$
$\mathsf{Con}_{2j-1}, \mathsf{Con}_{2j}$
$\mathsf{Exp}(s^{\theta-1}, \$)$

Fig. 3. Original versus New Computation of a Gate

The above is an informal description of Compute-in-register$\big(g, (\mathsf{Exp}, \mathsf{Con}), \mathsf{Exp}(s, \$)\big)$, which computes $g(s)$ by going gate-by-gate, expanding and contracting as needed. In other words the computation of $g(s)$ is done without leaking the secret or the intermediate computations. See Figures 4 and 5.

[8] 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.

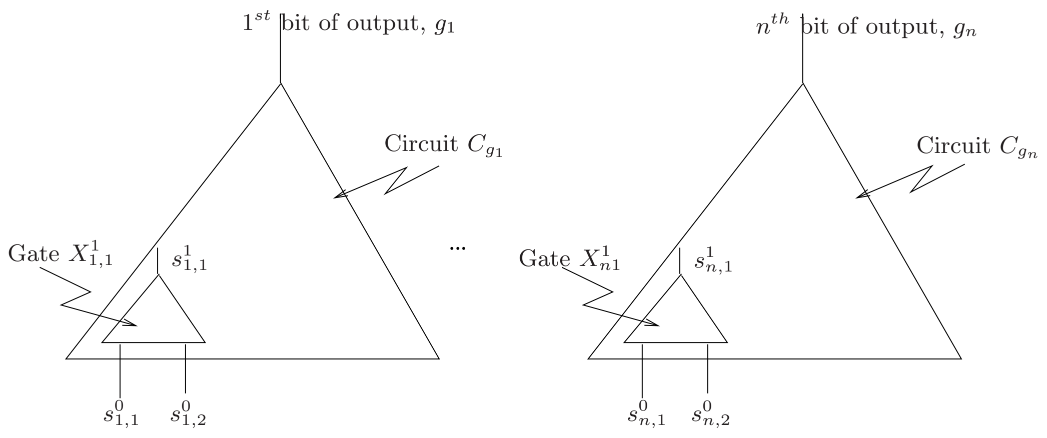


Fig. 4. Original circuit for computing a function $g(s)$

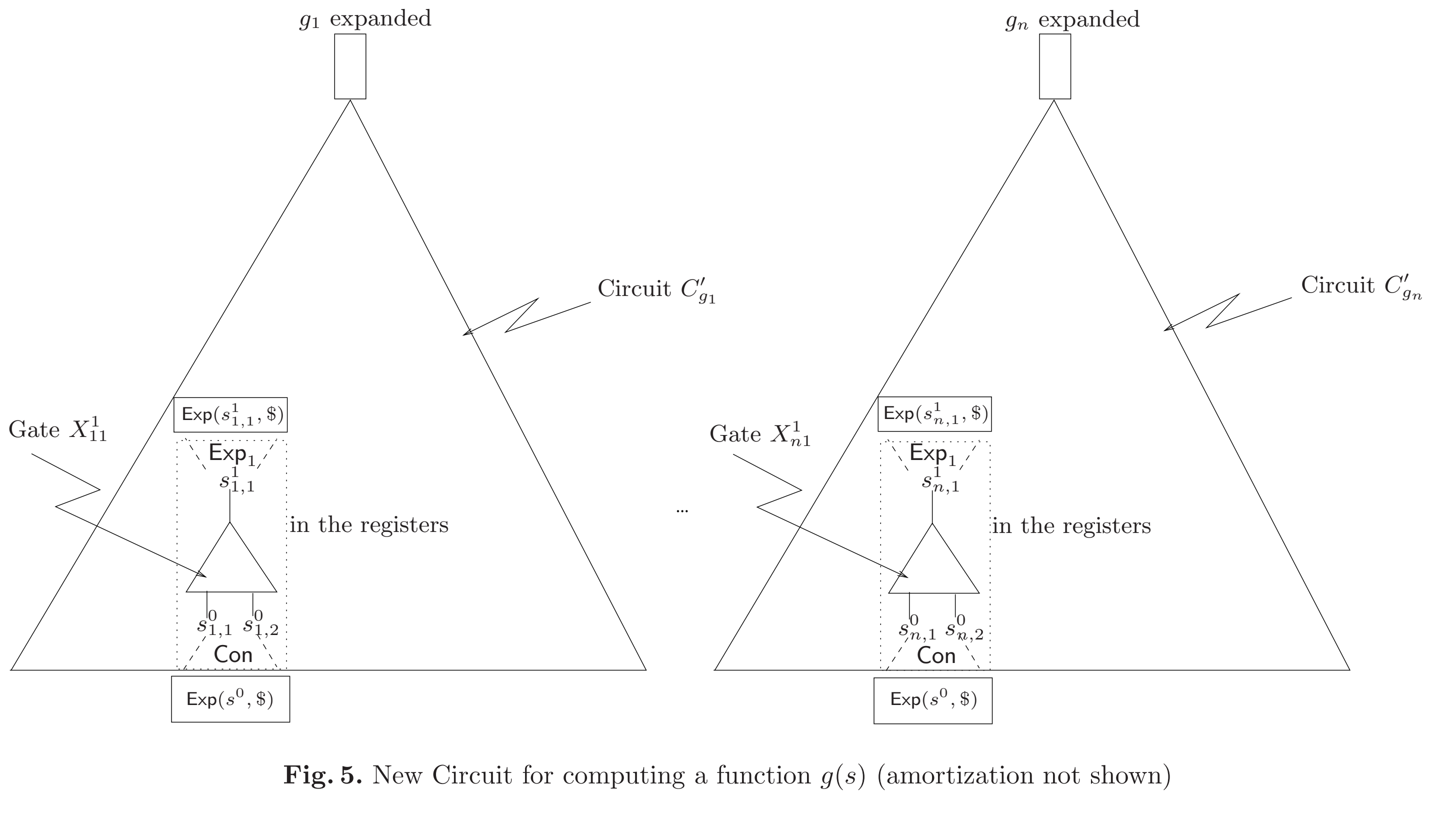


Fig. 5. New Circuit for computing a function $g(s)$ (amortization not shown)

Let **EraseReg** be the operation that perfectly erases all the registers, and $Reg[i]$ denote the i-th bit of the registers.

```
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 (or each circuit $C_{g_i}$)
3      do
4          for $j \leftarrow 1, ..., n$
5              do
6                  $s_{ij}^0 \leftarrow s_j$ ▷ Initialize the "zero-th" level outputs to be the secret...
7                  $e_{ij}^0 \leftarrow \mathsf{Exp}_j(s_{ij}^0, \$)$ ▷ ...in the expanded form.
8
9          for $\theta \leftarrow 1, ..., \Theta$ ▷ For each level of the circuit
10             do
11                 for $X_{ij}^\theta \leftarrow$ each gate in level $\theta$ of $C_{g_i}$
12                     do
13                         $Reg[1] \leftarrow \mathsf{Con}_{2j-1}\left(e_{i,2j-1}^{\theta-1}\right)$ ▷ Compute the required two bits of input.
14                         $Reg[2] \leftarrow \mathsf{Con}_{2j}\left(e_{i,2j}^{\theta-1}\right)$
15                         $Reg[3] \leftarrow X_{ij}^\theta\big(Reg[1], Reg[2]\big)$
16                         Output to main memory $e_{ij}^\theta \leftarrow \mathsf{Exp}_j(Reg[3], \$)$ ▷ Amortization implicit.
17                         EraseReg
18 return
```

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. To handle multiple inputs, just compute the bits needed from either of the inputs; to handle 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).

4.2 The Overall Compiler

Before giving the compiler, we need to describe how a secret is generated in the first place. GEN-PARTIAL-ERASABLE-SECRET generates a uniform secret in a partially erasable way. For secrets to be generated non-uniformly (by computing some function $gen(\circ)$ in input uniform coins), we can just do the computation in a partially erasable way, by using COMPUTE-IN-REGISTER on the output of GEN-PARTIAL-ERASABLE-SECRET.

```
GEN-PARTIAL-ERASABLE-SECRET($(\mathsf{Exp}, \mathsf{Con}), n$)
1  ▷ Generates a random secret of length $n$ bit-by-bit in a partial erasable way,
2  ▷ ... using the partially erasable form $(\mathsf{Exp}, \mathsf{Con})$.
3  $k \xleftarrow{\$} \{0,1\}^m$
4  for $i \leftarrow 1, ...n$
5      do
6          $Reg[1] \xleftarrow{\$} \{0,1\}$
7          $(R_i, x[i]) \leftarrow \mathsf{Exp}_i(Reg[1], \$)$
8          EraseReg
9  return $(R, k, x)$ ▷ Output to main memory and return.
```

In the compiler 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.

```
COMPILER($(\mathsf{Exp}, \mathsf{Con}), \Pi_{org}$)
1  ▷ Compile $\Pi_{org}$ requiring perfect erasures for security...
2  ▷ ... into $\Pi_{new}$ that tolerates partial erasures, using $(\mathsf{Exp}, \mathsf{Con})$ as the expanded form.
3  for all $s \in S$ of length $n$ that are to be generated using some algorithm $gen(\circ)$ on input a random string,
4      do
5          replace by the following code:
6          "$e \leftarrow$GEN-PARTIAL-ERASABLE-SECRET$((\mathsf{Exp}, \mathsf{Con}), n)$"
7          "$s \leftarrow$COMPUTE-IN-REGISTER$(gen, (\mathsf{Exp}, \mathsf{Con}), e)$"
8  for all computations of primitives $g$ involving $s \in S$ (or rather, $\mathsf{Exp}(s, \$)$ s.t. $s \in S$)
9      do
10         replace by the following code:
11         "COMPUTE-IN-REGISTER$(g, (\mathsf{Exp}, \mathsf{Con}), \mathsf{Exp}(s, \$))$"
12 for all instructions "Perfectly Erase"
13     do
14         replace by the following code:
15         "PartialErase"
16 return modified protocol $\Pi_{new}$
```

We will need the notion of UC-emulation:

Definition 13 (UC Emulation [10]). *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 the following interactions differ by at most a negligible amount:*
(1) interacting with $\mathcal{A}$ and parties running Π_{new}, and
(2) 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 its 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.

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 $\Pi_{new} =$ 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 any adversary $\mathcal{A}_{\Pi_{new}}$ there exists an adversary $\mathcal{A}_{\Pi_{org}}$ (with running time poly related to $\mathcal{A}_{\Pi_{new}}$) such that no environment $\mathcal{Z}$ can tell with non-negligible advantage whether it is interacting with $\mathcal{A}_{\Pi_{new}}$ and Π_{new}, or with $\mathcal{A}_{\Pi_{org}}$ and Π_{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 $\mathcal{A}_{\Pi_{new}}$, $\mathcal{A}_{\Pi_{org}}$ sets $s' = 0$ and then partially erases s' to get $h\big(\mathsf{Exp}(s', \$)\big)$. The adversary $\mathcal{A}_{\Pi_{org}}$ does the same thing for the intermediate computations.

It follows from definition 8 that, if Exp is an (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, as long as at most ℓ erasures are made, the environment views of the two interactions are statistically close to each other. □

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 blow up in space by Ψ, and a blow up in computational cost proportional to the time required for two (bit) contractions and one (bit) expansion (done before and after each gate computation).

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 the 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 [36] 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 and 35 seconds. By discharging inverted cans of "canned air" duster spray directly 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 systems 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 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 [36] 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 Our Direct Solution to the PSS with Partial Erasures

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

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 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 between time 1 to $t-1$. Perfect erasures is one way to guarantee that. As we will show, partial erasures are also good enough.

We refer the reader to [61,40] 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 14 (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 15 (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, $\Pi = \big(S, (S_1, \ldots, S_p), R, D\big)$ 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 16 (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 Section 5.1 we present a method to modify any existing proactive secret sharing scheme (satisfying some general conditions) that uses perfect erasures, into a new scheme that preserves the functionality and uses only partial erasures.

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, at the end of every time period.
   (c) Reconstruction – when some group of parties reconstruct the secret.
2. The shares and refresh data are uniformly distributed over $\mathrm{GF}(2^n)$.
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 [14,40] satisfy these conditions (under some minor and straightforward modifications to the schemes).

For concreteness, in Section 5.2 we present a particular proactive (p,p)-threshold secret sharing scheme, against honest-but-curious mobile adversaries.

In Section 5.3 we present a proactive (c,p)-threshold secret sharing scheme, for $c < \frac{1}{3}p$, robust against malicious mobile adversaries.

5.1 More Efficient Compiler for Proactive Secret Sharing

The high level idea of the more efficient compiler is the following. We start with Π, a proactive (c,p)-threshold secret sharing scheme which requires perfect erasures, that meets all the conditions stated above (in particular, the shares are uniformly chosen from $\mathrm{GF}(2^n)$, where n is the security parameter). Let:

1. Π call $\textsc{refresh}\left(i, j, \left\{s^t_{ij'}\right\}_{1\le j'<j}\right)$ to generate the refresh data ($\textsc{refresh}$ 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 $\textsc{update}\left(s^t_i, \left\{s^t_{ji}\right\}_{1\le j\le p}\right)$ to update the shares using the refresh data received ($\textsc{update}$ accepts as arguments the old share, and all the refresh data party i received).

The compiled proactive secret sharing scheme $\mathrm{PESS}(\Pi)$ will use altered versions of these two algorithms, $\textsc{refresh}'$ and $\textsc{update}'$. The differences are:

1. $\textsc{refresh}'$ and $\textsc{update}'$ take as input partially erasable forms of the shares instead of the shares themselves, but essentially do the same computation.
2. Our altered algorithms will do their computation bit-by-bit (that is, by using only the constant number of constant size registers to store valuable data). In other words, for any bit number $\alpha \in \{1 \dots m\}$, $\textsc{refresh}'_\alpha$ will compute the α^{th} bit of the refresh data to be sent by party i to party j, according to Π, and likewise for $\textsc{update}'_\alpha$. By conditions 3 and 4, this is possible.

We use auxiliary algorithms named $\textsc{gen}^1_\Pi()$, $\textsc{gen}^2_\Pi()$ and $\textsc{gen}^3_\Pi()$ in order to generate the share parts. These algorithms are called in the dealing and refreshing phases. Detailed code for $\textsc{gen}^1_\Pi()$, $\textsc{gen}^2_\Pi()$ and $\textsc{gen}^3_\Pi()$, and some supporting algorithms follow.

Notation The following algorithms will follow this notation:

1. Reg is the variable corresponding to the constant number of constant size registers. All other variables are in the main memory (and in particular might be paged onto the hard drive).
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 we are not computing the whole value but only the specified bit.

```
Random-Split($\text{func}_\alpha, arg$)
1  ▷ for each $\alpha \leftarrow 1, \ldots, n$,
2  ▷    split $\text{func}_\alpha(arg)$ into random $R[\alpha, :] \in \{0,1\}^m$ (row vector) and $k \in \{0,1\}^m$ (col vector) s.t.
3  ▷    $R[\alpha, :] \cdot k = \text{func}_\alpha(arg)$ (here, $R[\alpha, :]$ denotes the $\alpha^{th}$ row of the matrix $R$)
4  ▷ As the main text explains, we use $\text{func}_\alpha$ because the output is sensitive info
5  ▷    and we want to compute it bit-by-bit in the perfectly erasable registers.
6  ▷ The error probability is $(1 - \frac{1}{2^n})\frac{1}{2^m}$, when for some $\alpha \in [1, n], \text{func}_\alpha(arg) \neq 0$, and $k = 0^m$.
7  $k \xleftarrow{\$} \{0,1\}^m$
8  for $\alpha \leftarrow 1, \ldots, n$ ▷ for each row of $R$...
9        do
10           $Reg[0] \leftarrow 0$
11           for $\beta \leftarrow 1, \ldots, m$ ▷ for each column of $R$...
12                do
13                   if $k[\beta] = 1$ and $\forall l \in \{\beta + 1, \ldots, m\}, k[l] = 0$ ▷ if $\beta$ is the last column that matters.
14                     then
15                          $Reg[0] \leftarrow Reg[0] \oplus \text{func}_\alpha(arg)$
16                          $R[\alpha, \beta] \leftarrow Reg[0]$
17                     else
18                          $R[\alpha, \beta] \xleftarrow{\$} \{0,1\}$
19                          if $k[\beta] = 1$ and $R[\alpha, \beta] = 1$ ▷ $R[\alpha, \beta]$ matters to the product.
20                            then
21                                 $Reg[0] \leftarrow Reg[0] \oplus R[\alpha, \beta]$
22 return $(R, k)$
```

```
Id$_\alpha(s)$
1  ▷ Identity function, to wrap constant vectors s.t. we can use random-split to split them.
2  return $s[\alpha]$
```

```
gen$^1_\Pi$ $(s^1_i)$
1  ▷ Dealer's randomly splitting of the initial share for party $P_i$ at time 1.
2  $R^1_i, k^1_i \leftarrow$ random-split $(\text{Id}_\alpha, s^1_i)$
3  return $(R^1_i, k^1_i)$
```

```
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 refresh $\left(i, j, \{s^t_{ij'}\}_{1 \le j' < j}\right)$
3  for $j \leftarrow 1, \ldots, p$
4        do
5           $(\rho^t_{ij}, \kappa^t_{ij}) \leftarrow$ random-split $\left(\text{refresh}'_\alpha, \left(i, j, \{\rho^t_{ij'}, \kappa^t_{ij'}\}_{1 \le j' < j}\right)\right)$
6  return $\{\rho^t_{ij}, \kappa^t_{ij}\}_{1 \le j \le p}$
```

```
gen$^3_\Pi$ $\left((R^t_i, k^t_i), \{(\rho^t_{ji}, \kappa^t_{ji})\}_{1 \le j \le p}\right)$
1  ▷ Party $P_i$'s share updating computation.
2  $R^{t+1}_i, k^{t+1}_i \leftarrow$ random-split $\left(\text{update}'_\alpha, \{(\rho^t_{ji}, \kappa^t_{ji})\}_{1 \le j \le p}\right)$
3  return $(R^{t+1}_i, k^{t+1}_i)$
```

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.

We are finally ready to describe the more efficient compiler.

The Compiler $\text{PESS}(\Pi)$ For simplicity, we will only state and prove the compiler 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 the specific scheme of [40] in Section 5.3.

Dealing In original scheme Π, the dealer privately sends each party i its share s_i^1. In the modified scheme $\text{PESS}(\Pi)$, 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 $\text{GEN}_{\Pi}^1\left(s_i^1\right)$.
She will then privately send $\left(R_i^1, k_i^1\right)$ to the party instead. To prevent confusion we will call $\left(R_i^1, k_i^1\right)$ a *pseudo-share*.

Refreshing In Π, at the end of every time period t, in order to refresh their shares, party i sends party j refresh data, s_{ij}^t.
In $\text{PESS}(\Pi)$, 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 $\text{GEN}_{\Pi}^2()$. This algorithm calls the scheme-specific $\text{REFRESH}'_\alpha$.
Say party i accepts the refresh data from all the other parties. Then it generates random (R_i^t, k_i^t) s.t. $R_i^{t+1} \cdot k_i^{t+1} = g(\rho_{ji}^t, \kappa_{ji}^t, R_i^t, k_i^t)$, for some 1-1 and onto function g, using $\text{GEN}_{\Pi}^3\left(\left(R_i^t, k_i^t\right), \left\{\left(\rho_{ji}^t, \kappa_{ji}^t\right)\right\}_{1\le j\le p}\right)$.
This algorithm calls the scheme-specific UPDATE'_α.
Finally, party i partially erase all the k_i^t and κ_{ij}^t, resulting in $h\left(k_i^t\right)$ and $h\left(\kappa_{ij}^t\right)$ 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 8 (Computation of New Shares). The computation of the new shares is done, as mentioned above, bit-by-bit using the perfectly erasable registers, 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 9 (Robustness). 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 [40] in this precise way. In our context, one must ensure that computations done in the VSS be implementable in the register model. It is unclear whether the VSS scheme used by [40] can be directly implemented in the register model (we can of course always do so indirectly by using COMPUTE-IN-REGISTERS as in the general compiler, going gate-by-gate). Instead, we show how to do this direct implementation for the scheme defined by [9], or rather its simplified version, defined by [29]. We will elaborate on this in Section 5.3.

Remark 10 (h is Adversarially Chosen). 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 an (ℓ, α, ϕ)-partially erasable form, where $\mathsf{Exp}(s, \$) := (R, k)$ and $\mathsf{Con}(R, k) := R \cdot k$. Then $\text{PESS}(\Pi)$ 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 15, where $\rho := 2(c-1)p + p - c + 1$.*

Proof. Again, for simplicity, we will only prove the case of robustness against malicious adversaries for a specific protocol in Section 5.3, where the robustness is added explicitly by the use of a VSS scheme. Here we prove the compiler for protocols secure against honest-but-curious adversaries.

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.

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, 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. The pseudo-shares generated by $\text{GEN}^1_\Pi()$ and $\text{GEN}^3_\Pi()$ are random up to the product of the pseudo-share parts, which in itself is also random. Moreover, since the adversary knows $c-1$ of the pseudo-shares of time period t, she knows 1-1 and onto functions 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 that she remains in.
- S_L is the group of parties that she leaves (i.e. relinquishes control at the end of time period t, and does not corrupt again in time period $t+1$).
- S_A is the group of parties that she arrives at (i.e. newly corrupts at the beginning of time period $t+1$). Clearly, $|S_A| = |S_L|$.
- S_I is the group of parties that she ignores (i.e. these parties are not corrupted in either of the two time periods).

For instance, if the adversary remains in the same $c-1$ parties, then $|S_R| = c-1$ and $S_L = S_A = \emptyset$. If the adversary corrupts disjoint parties across the time periods, then $S_R = \emptyset$ and $|S_A| = |S_L| = c-1$.
Let us look at the refresh data in each subgroup:

- 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 1-1 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 $\text{GEN}^2_\Pi()$ as random up to the product of the tuple parts, which in itself is also random. For simplicity, let us assume the adversary can learn information about the secret s from each tuple independently. That is, the adversary knows a 1-1 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, 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 (ℓ, α, ϕ)-partially erasable, as long as:

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

then we have that $d(s|VIEW^\ell) \le 2^{-\alpha}$, and so 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 15. □

5.2 Proactive (p,p)-Threshold Secret Sharing with Partial Erasures

Now, let us apply the compiler PESS discussed in the previous section to the trivial (p,p)-threshold secret sharing scheme secure against honest-but-curious adversaries.

The following algorithms will be used.

```
ADD $(\delta, R, k)$
$\triangleright$ Add $[R \cdot k]_\delta$ to $Reg[1]$.
$\triangleright$ 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
```

```
REFRESH$'_\alpha \left(i, j, \{\rho^t_{ij'}, k^t_{ij'}\}_{1 \leq j' < j}\right)$
$\triangleright$ Compute the $\alpha^{th}$ 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
                ADD$(\alpha, \rho^t_{ij'}, \kappa^t_{ij'})$ $\triangleright$ Add $\left[\rho^t_{ij'} \cdot \kappa^t_{ij'}\right]_\alpha$ to $Reg[1]$.
return $Reg[1]$
```

```
UPDATE$'_\alpha \left((R^t_i, k^t_i), \{(\rho^t_{ji}, \kappa^t_{ji})\}_{1 \leq j \leq p}\right)$
$\triangleright$ Compute the $\alpha^{th}$ bit of the new, time $t+1$ pseudo-share of $P_i$.
$Reg[1] \leftarrow 0$
ADD$(\alpha, R^t_i, K^t_i)$ $\triangleright$ Calculate $\left[R^t_i \cdot k^t_i\right]_\alpha$ into $Reg[1]$.
for $j \leftarrow 1, ..., p$
    do
        ADD$(\alpha, \rho^t_{ji}, \kappa^t_{ji})$ $\triangleright$ Add $\left[\rho^t_{ji} \cdot \kappa^t_{ji}\right]_\alpha$ to $Reg[1]$.
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 $\left(k^1_1, ..., k^1_p\right) \in \{0,1\}^m$ and $\left(R^1_1, ..., R^1_p\right) \in \{0,1\}^{n \times m}$, such that $\bigoplus_{i=1}^p R^1_i \cdot k^1_i = s$, using GEN$^1_\Pi$.
2. Send $\left(R^1_i, k^1_i\right)$ to P_i privately.

Note: in the original scheme, the shares s_i are uniform, and therefore so are the pseudo-shares (R^1_i, k^1_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^t_i, k^t_i) into a new one $\left(R^{t+1}_i, k^{t+1}_i\right)$.

1. Generate random $\left(\kappa^t_{i1}, ..., \kappa^t_{ip}\right) \in \{0,1\}^m$ and $\left(\rho^t_{i1}, ..., \rho^t_{ip}\right) \in \{0,1\}^{n \times m}$, such that $\bigoplus_{i=1}^p \rho^t_{ij} \cdot \kappa^t_{ij} = 0$, using GEN$^2_\Pi$.
2. Privately send $\left(\rho^t_{ij}, \kappa^t_{ij}\right)$ to party P_j, for all $j \in [1, p]$.
3. Privately receive $\left(\rho^t_{ji}, \kappa^t_{ji}\right)$ from party P_j, for all $j \in [1, p]$.
4. Generate random k^{t+1}_i and R^{t+1}_i, such that $R^{t+1}_i \cdot k^{t+1}_i = R^t_i \cdot k^t_i \oplus \bigoplus_{j=1}^p \rho^t_{ji} \cdot \kappa^t_{ji}$, using GEN$^3_\Pi$.
5. Partially erase all the k^t_i and κ^t_{ij}, resulting in $h\left(k^t_i\right)$ and $h\left(\kappa^t_{ij}\right)$, respectively.

Remark 11 (GEN$^2_\Pi$ and GEN$^3_\Pi$ are Protocol Specific). Note that in GEN$^2_\Pi$ and GEN$^3_\Pi$ we call REFRESH$'_\alpha$ and UPDATE$'_\alpha$, respectively, which are protocol specific.

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

Theorem 17. *Let ϕ be the leakage fraction of the partial erasure function h. Let $(\mathsf{Exp}, \mathsf{Con})$ be an (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)-threshold 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 15, where $\rho := 2(c-1)p + p - c + 1$.*

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

5.3 Proactive (c, p)-Threshold Secret Sharing with Partial Erasures

Next, we will modify the efficient (c, p)-threshold proactive secret sharing protocol, secure against malicious adversaries as defined in [40], to one which uses only partial erasures. According to their protocol, which enhances Shamir's secret sharing [65], 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 perfectly erases the old one, as well as the update values.

Their scheme [40] 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 [29], which do not rely on a cryptographic assumption such as the irreversibility of discrete log, and contrary to [28] used in [40], it is easy to fit directly into the register model.

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

The following algorithms will be used.

ADD (δ, R, k) is the same as that of the (p, p) case.

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

```
1  ▷ Compute the $\alpha^{th}$ 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 $L_{j'}(j) = \prod_{v \ne j'} \frac{j-v}{j'-v}$
6          $L^\gamma_{j'}(j) \leftarrow 2^\gamma \prod_{v \ne j'} \frac{j-v}{j'-v}$ ▷ 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 $Reg[1]$ by going bit-by-bit of $s^t_{ij'}$.
9                 for $\gamma \leftarrow 1, ..., m$
10                    do
11                       $Reg[1] \leftarrow 0$
12                       ADD$(\gamma, \rho^t_{ij'}, \kappa^t_{ij'})$ ▷ 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 $Reg[2]$
```

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

```
$\textsc{ver}_i\left(R_i^t, k_i^t, \bar{R}_i^{t(j)}, \bar{k}_i^{t(j)}\right)$
1  $\triangleright$ Party $P_i$'s verification at time $t$ that for a given $j$ his share is valid.
2  $\triangleright$ Note that the use of this algorithm in the dealing and the refreshing phases are in a sense reversed.
3  $\triangleright$ In particular, dealing uses $\textsc{ver}_i$ for $P_i$ to verify the shares dealt to it by the dealer,
4  $\triangleright$    and refreshing uses $\textsc{ver}_j$ for $P_j$ to verify $P_i$'s actions.
5  for $\alpha \leftarrow 1, ..., n$ $\triangleright$ Verify for every bit $\alpha$.
6      do
7          $Reg[1] \leftarrow 0$
8          $\textsc{add}\left(\alpha, \bar{R}_i^{t(j)}, \bar{k}_i^{t(j)}\right)$ $\triangleright$ Compute $\left[\bar{R}_i^{t(j)} \cdot \bar{k}_i^{t(j)}\right]_\alpha$ into $Reg[1]$.
9          if $Q_j = 1$
10            then $\textsc{add}(\alpha, R_i^t, k_i^t)$ $\triangleright$ Add $\left[R_i^t \cdot k_i^t\right]_\alpha$ to $Reg[1]$.
11         if $\left[g^{t\prime(j)}(i)\right]_\alpha \neq Reg[1]$
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 $\left(k_1^1, \ldots, k_p^1\right) \in \{0,1\}^m$ and $\left(R_1^1, \ldots, R_p^1\right) \in \{0,1\}^{n \times m}$, such that $f(i) = R_i^1 \cdot k_i^1$, using $\textsc{gen}_\Pi^1$.
2. Send (R_i^1, k_i^1) 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 $\left(\bar{k}_1^{1(j)}, \ldots, \bar{k}_p^{1(j)}\right) \in \{0,1\}^m$ and $\left(\bar{R}_1^{1(j)}, \ldots, \bar{R}_p^{1(j)}\right) \in \{0,1\}^{n \times m}$, such that $g^{1(j)}(i) = \bar{R}_i^{1(j)} \cdot \bar{k}_i^{1(j)}$, using $\textsc{gen}_\Pi^1$.
   (b) For each polynomial $g^{1(j)}$, send each pseudo-share $(\bar{R}_i^{1(j)}, \bar{k}_i^{1(j)})$ 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 $\textsc{ver}$ 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 $\textsc{gen}_\Pi^2$.
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 $v \in \{1 \ldots V\}$, 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 $\textsc{gen}_\Pi^2$.
   (b) For every $v \in \{1 \ldots V\}$, 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 $Q_{\lceil \frac{V(j-1)}{p}+1 \rceil}, \ldots, Q_{\lceil \frac{Vj}{p} \rceil} \overset{\$}{\leftarrow} \{0,1\}$ 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 $\textsc{ver}$ 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, resulting in $h(k_i^t)$ and $h(\kappa_{ij}^t)$, respectively.

Remark 12 (GEN_{Π}^2 and GEN_{Π}^3 are Protocol Specific). 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 set it to zero to obtain the secret.

Theorem 18. *Let ϕ be the leakage fraction of the partial erasure function h. Let $(\mathsf{Exp}, \mathsf{Con})$ be an (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)-threshold 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 15, 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, correctness in the presence of malicious adversaries is trivial, as it has been proven in [29], and since the verification polynomials of the refresh data have free coefficient being 0, the 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, the difference with the honest-but-curious case is the addition of the VSS. This may give the adversary more information, 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). In the worst case, all the verification tuples are related to s. Since the amount of verification tuples sent after each time period is Vp, the total sent over τ time periods is less than τVp.

Therefore, over τ time periods, all the adversary may learn is at most $\tau(Vp + \rho)$ partially erased tuples related to s. Since the partially erasable form is (ℓ, α, ϕ)-secure, given the view of the adversary the distance of the secret from uniform will be upper bounded by $2^{-\alpha}$, as long as $\tau(Vp+\rho) \leq \ell$. In other words, PESS(Π) described above is $(\frac{\ell}{V \cdot p+\rho}, \alpha, \phi)$-secure as per definition 15. □

6 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 time required for two (bit) contractions and one (bit) expansion (done before and after each gate computation). 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 [9]) is provided in Section 6. We give the more efficient compiler in Section 6.

Restricting the computations on the secret to be of constant output locality is stronger than required. In fact, we gave in Section 5 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.

[9] 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.

Summary of Some Results in Parallel Time Complexity The parallel time complexity of cryptographic primitives has been studied in various works, including [37,33,19,47,57,3,6]. 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 the class of functions where each bit of the output depends on at most c input bits.

The existence of one-way functions (OWF) and pseudo-random generators (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. 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. 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 present 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).

Negative Results	
Result	**Reference**
PRF $\notin AC^0$	[49]
PRG $\notin NC^0_2$	[19]
PRG with superlinear stretch $\notin NC^0_3$	[19]
PRG with superlinear stretch $\notin NC^0_4$	[57]

Table 1. Some Negative Results in Parallel Time Complexity

For our purposes, the implication of the negative results is that: 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 via the general compiler.

Remark 13 (Assumptions). 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.

Remark 14 (Note on Encryption Schemes). For encryption schemes, the positive 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, for 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 15 (Further Improvements). 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.

Positive Results		
Assumption	**Result**	**Ref**
Subset-sum related	$\exists$ PRG with sublinear stretch $\in AC^0$	[44]
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. Some Positive Results in Parallel Time Complexity

More Efficient Compiler for NC^0 It is straightforward to get a more efficient compiler Compiler$_{NC^0}$: instead of going gate-by-gate as in Compute-in-register, for each output bit, reconstruct the constant number of bits that this bit depends on in the registers (which is possible assuming the appropriate assumptions), 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 (assuming the appropriate assumptions), we have that $\Pi_{new} :=$ Compiler$_{NC^0}$ UC-emulates Π_{org}, and tolerates (maintains security even with) imperfect/partial erasures in the register model.*

7 Conclusions

The ability to erase is an important capability of honest parties in cryptographic protocols. Our general compiler assures us that anywhere erasures can be leveraged to prove or simplify security, it should be used liberally; at the cost of using more storage and a small computation overhead, the erasures can be imperfect for all parts of the system, except for a constant number of constant size CPU registers.

7.1 Practical Applications

Our solution can be applied beyond cryptographic protocols to any security solution that does not assume a trusted computation path or protected execution (which provides an environment for protected cryptographic functions to execute without modification or exposing key information), even if the solution itself does not explicitly erase. This is because any security solution involves some secrets and some computations on those secrets, and unless the execution is protected, information about the secrets might leak through the computations.

Consider hard drive encryption. Typically, the key either resides on a special part of the drive itself and is not accessible without a PIN, or on another hardware, e.g. a USB key. Regardless of where the key resides, on any general computing platform it has to be loaded into the main memory for decryption, and this opens up the possibility of attacks, as shown for instance in [36].

Also, in Digital Rights Management (DRM), the owner of the computer *is* the adversary. Without trusted hardware, if the file is decrypted and is present in the RAM at some point in time, then (utilizing the attack given in [36]) the user can freeze the RAM and move it to another system to obtain the plaintext (e.g. an MP3 music file) or even the key for the DRM system. Note that in this setting where the user is the adversary, this attack is much easier to carry out.

7.2 Alternative Models of Partial Erasures

The function h is assumed to be a function only of the storage contents to be erased. Furthermore, we assume that h is fixed in advance. This choice seems to adequately capture erasures that are only partially successful due to the physical properties of the storage media. However, this modeling 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 depend on information other than the contents to be erased, or alternatively to be determined adaptively as the computation evolves.

Also, notice that under our current model, we cannot make sense of partially erasing one bit. One possible way of modifying the model might be to consider a length preserving h that is randomized. Obviously, without any extra conditions on h, no security can be attained, since $h : \{0,1\}^m \mapsto \{0,1\}^m$ can just ignore the randomness and output the entire input, which means that nothing is being erased. To this end, consider the condition that $H_\infty\big(h(k)\big) \geq (1-\phi)m$, meaning that $h(k)$ must still contain at least $(1-\phi)m$ bits of randomness. If k is random then the condition does not help, so consider any fixed k. (This models for instance the situation in which the hardware might have suffered from burnt-in or other effects that make k non-random.) However, regardless of what condition we pick, it is difficult to prove any meaningful security guarantees. At a high level, this is because for a fixed k, the adversary can always do a straightforward encoding and decoding so that the output of h would allow him to decode $2^{\phi m}$ different ks, so he breaks the scheme for all of those ks.

One can also ask why we have to split the input to several parts $\mathsf{Exp}(s, \$) := (\mathsf{Ext}(R, k) \oplus s, R, k)$, where only one part k needs to be erased, independently of the other parts. In a sense the result is stronger because only one small part needs to be partially erased. On the other hand, this implicitly relies on memory segmentation: when the honest parties erase using h adversarially designed, we need to trust the OS for memory protection. This is a pretty reasonable assumption, but should not stop us from asking whether it can be removed.

Certainly, h cannot be applied to the whole expansion without any other restrictions, since $h(\mathsf{Ext}(R, k) \oplus s, R, k)$ can be just the function that computes and outputs s. In the case that the secret s itself is random, we do not need the one-time pad mechanism, and we claimed in Section 3 that we can just use $\mathsf{Exp}(s, \$) := (R, k)$ s.t. $R \cdot k = s$. Focusing on this case, now the question becomes: can h be applied on (R, k)?

Since we are considering any length-shrinking h, we are dealing with the case in which the source has good min-entropy even given the adversary's view (of the partially erased secret). In other words we are dealing with the most general form of randomness sources; the R part that does not need to be erased corresponds to the random seed necessary for randomness extraction in this general case. In other words, without additional restrictions on h, it is impossible to extract randomness from the conditional distribution $(R, k)|h(R, k)$, since general extractors require an independent seed. Following this view point, the difference between $h(R, k)$ and $h(k)$ is that in the former, the "seed" is contaminated and no longer independent of the source k, whereas in the latter, R is still independent of the source and can be used as a seed for a general randomness extractor.

This leads us to restricting h. If we limit h to s.t. $(R, k)|h(R, k)$ is some form of weak source of randomness for which deterministic extraction is possible (without an independent seed), then (if the extraction can be easily inverted) we can construct corresponding partially erasable forms for which h can be applied on everything to be erased, and the need to rely on memory segmentation is removed.

7.3 Near Optimal Strong Extractors Computable with Constant Memory

If we could find an optimal strong extractor that is computable with constant memory, then using the partially erasable form based on such an extractor, our compiler would essentially match the lower bound on the expansion required, i.e. it is about as storage efficient as possible. Unfortunately we do not know of any such extractors.

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. 20
2. R. Anderson. Two remarks on public key cryptology. Invited lecture, ACM-CCS '97, 1997. 2
3. B. Applebaum, Y. Ishai, and E. Kushilevitz. Cryptography in NC^0. In *Proc. 45th IEEE Symp. on Foundations of Comp. Science*, pages 166–175, 2004. 41, 42
4. B. Applebaum, Y. Ishai, and E. Kushilevitz. Computationally private randomizing polynomials and their applications. In *Proc. 20th IEEE Conference on Computational Complexity (CCC)*, 2005. 41, 42
5. B. Applebaum, Y. Ishai, and E. Kushilevitz. On pseudo-random generators with linear stretch in NC^0. In *Proc. of 10th International Workshop on Randomization and Computation*, 2006. 42
6. B. Applebaum, Y. Ishai, and E. Kushilevitz. Cryptography with constant input locality. In *Proc. 27th Crypto*, 2007. 41
7. Y. Aumann and M. O. Rabin. Information theoretically secure communication in the limited storage space model. In *Proc. CRYPTO 99*, pages 65–79, 1999. 7
8. D. Beaver. Plug and play encryption. In *Proc. CRYPTO 97*, 1997. 2
9. 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, New York, NY, USA, 1988. ACM. 35
10. 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. 5, 9, 10, 30
11. R. Canetti, Y. Dodis, S. Halevi, E. Kushilevitz, and A. Sahai. Exposure-resilient functions and all-or-nothing transforms. In *Proc. EUROCRYPT 2000*, volume 1807, pages 453–469, 2000. 7
12. R. Canetti, U. Feige, O. Goldreich, and M. Naor. Adaptively secure computation. In *Proc. 28th ACM Symp. on Theory of Computing*, 1996. 2
13. R. Canetti, R. Gennaro, A. Herzberg, and D. Naor. Proactive security: Long-term protection against break-ins. In *CryptoBytes(1)3, 1997.* 2
14. R. Canetti and A. Herzberg. Maintaining security in the presence of transient faults. In *Proc. CRYPTO 94*, pages 425–438, 1994. 33
15. J. L. Carter and M. N. Wegman. Universal classes of hash functions. *Journal of Computer and System Sciences*, 18:143–154, 1979. 25
16. T. Cover and J. Thomas. *Elements of Information Theory*. John Wiley & Sons, 1991. 15, 16, 24
17. G. Di Crescenzo, N. Ferguson, R. Impagliazzo, and M. 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. 8
18. G. Di Crescenzo, R. J. Lipton, and S. Walfish. Perfectly secure password protocols in the bounded retrieval model. In *Theory of Cryptography Conference*, pages 225–244, 2006. 7
19. M. Cryan and P. B. Miltersen. On pseudo-random generators in NC^0. In *Proc. 26th MFCS*, 2001. 41
20. D. Dagon, W. Lee, and R. J. Lipton. Protecting secret data from insider attacks. In *Financial Cryptography*, pages 16–30, 2005. 7
21. W. Diffie, P. C. Van-Oorschot, and M. J. Weiner. Authentication and authenticated key exchanges. In *Designs, Codes, and Cryptography*, pages 107–125, 1992. 2
22. Y. Dodis. *Exposure-Resilient Cryptography*. PhD thesis, MIT, 2000. 7
23. Y. Dodis, R. Ostrovsky, L. Reyzin, and A. 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/`. 13
24. S. Dziembowski. Intrusion-resilience via the bounded-storage model. In *Theory of Cryptography Conference*, pages 207–224, 2006. 7
25. S. Dziembowski. On forward-secure storage. In *Proc. CRYPTO 2006*, pages 251–270, 2006. 7
26. S. Dziembowski and U. Maurer. Tight security proofs for the bounded-storage model. In *Proc. 34th ACM Symp. on Theory of Computing*, pages 341–350, 2002. 7
27. S. Dziembowski and K. 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. 7
28. 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. 38

29. P. Feldman and S. Micali. Optimal algorithms for byzantine agreement. In *Proc. 20th ACM Symp. on Theory of Computing*, pages 148–161, 1988. 35, 38, 40
30. Y. Frankel, P. Gemmel, P. D. MacKenzie, and M. Yung. Proactive RSA. In *Proc. CRYPTO 97*, pages 440–454, 1997. 2
31. S. L. Garfinkel. *Design Principles and Patterns for Computer Systems That Are Simultaneously Secure and Usable*. PhD thesis, MIT, 2005. 2
32. R. Gennaro, S. Jarecki, H. Krawczyk, and T. Rabin. Robust threshold DSS signatures. In *Proc. EUROCRYPT 96*, pages 354–371, 1996. 2
33. O. Goldreich. Candidate one-way functions based on expander graphs. In *ECCC*, volume 7(090), 2000. 41
34. S. Goldwasser, S. Micali, and C. Rackoff. The knowledge complexity of interactive proof systems. *SIAM J. Comput.*, 18(1):186–208, 1989. 10
35. C. G. Günther. An identity-based key-exchange protocol. In *Proc. EUROCRYPT 89*, volume 434, pages 29–37, 1989. 2
36. J. A. Halderman, S. D. Schoen, N. Heninger, W. Clarkson, W. Paul, J. A. Calandrino, A. J. Feldman, J. Appelbaum, and E. W. Felten. Lest we remember: Cold boot attacks on encryption keys. 2008. `http://citp.princeton.edu/memory/`. 6, 31, 32, 42
37. J. Håstad. One-way permutations in NC^0. In *Information Processing Letters*, volume 26, pages 153–155, 1987. 41
38. J. Håstad, R. Impagliazzo, L. A. Levin, and M. Luby. A pseudo-random generator from any one-way function. *SIAM J. Computing*, 28(4):1364–1396, 1999. 13
39. 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. 2
40. 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. 32, 33, 35, 38
41. G. Hughes and T. Coughlin. Secure erase of disk drive data. In *Insight*, pages 22–25, 2002. 2
42. G. Hughes and T. Coughlin. Tutorial on disk drive data sanitation, retrieved online in July 2007. `http://cmrr.ucsd.edu/people/Hughes/SecureErase.shtml`. 2
43. 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. 13
44. R. Impagliazzo and M. Naor. Efficient cryptographic schemes provably as secure as subset sum. *Journal of Cryptology*, 9:199–216, 1996. 42
45. G. Itkis and L. Reyzin. SIBIR: Signer-base intrusion-resilient signatures. In *Proc. CRYPTO 2002*, pages 499–514, 2002. 2
46. S. Jarecki and A. Lysyanskaya. Adaptively secure threshold cryptography: Introducing concurrency, removing erasures. In *Proc. EUROCRYPT 2000*, pages 221–243, 2000. 2
47. M. Krause and S. Lucks. On the minimal hardware complexity of pseudo-random function generators. In *Proc. 18th STACS*, pages 419–430, 2001. 41
48. H. Krawczyk. New hash functions for message authentication. In *Proc. EUROCRYPT 95*, pages 301–310, 1995. 20
49. N. Linial, Y. Mansour, and N. Nisan. Constant depth circuits, fourier transform, and learnability. In *Proc. 30th IEEE Symp. on Foundations of Comp. Science*, pages 574–579, Washington, DC, USA, 1989. IEEE Computer Society. 41
50. C.-J. Lu. Encryption against storage-bounded adversaries from on-line strong extractors. In *Proc. CRYPTO 2002*, pages 257–271, 2002. 4, 24
51. C.-J. Lu, O. Reingold, S. Vadhan, and A. Wigderson. Extractors: Optimal up to constant factors. In *Proc. 35th ACM Symp. on Theory of Computing*, 2003. 23
52. A. Lysyanskaya. Efficient threshold and proactive cryptography secure against the adaptive adversary (extended abstract), 1999. `http://www.citeseer.ist.psu.edu/lysyanskaya99efficient.html`. 2
53. Y. Mansour, N. Nisan, and P. Tiwari. The computational complexity of universal hashing. In *Proc. 22nd ACM Symp. on Theory of Computing*, pages 235–243, 1991. 4, 19
54. U. Maurer. A provably-secure strongly-randomized cipher. In *Proc. EUROCRYPT 90*, pages 361–373, 1990. 4, 6
55. U. Maurer. Conditionally-perfect secrecy and a provably-secure randomized cipher. *Journal of Cryptology*, pages 53–66, 1992. 4, 6, 7
56. S. Micali and L. Reyzin. Physically observable cryptography. In *Cryptology ePrint Archive: Report 2003/120*, 2003. `http://eprint.iacr.org/2003/120`. 6

57. E. Mossel, A. Shpilka, and L. Trevisan. On ϵ-biased generators in NC^0. In *Proc. 43rd IEEE Symp. on Foundations of Comp. Science*, pages 136–145, 2003. 41
58. J. Naor and M. Naor. Small-bias probability spaces: Efficient constructions and applications. In *Proc. 22nd ACM Symp. on Theory of Computing*, 1993. 19
59. N. Nisan and D. Zuckerman. Randomness is linear in space. *Journal of Computer and System Sciences*, 52(1):43–52, 1996. 4, 23
60. Department of Defense. *DoD 5220.22-M: National Industrial Security Program Operating Manual*, 1997. 2
61. R. Ostrovsky and M. Yung. How to withstand mobile virus attacks. In *Proc. 10th ACM Symp. on Principles of Distributed Computation*, pages 51–61, 1991. 2, 32
62. N. Pippenger and M. J. Fischer. Relations among complexity measures. *J. ACM*, 26(2):361–381, 1979. 10, 26
63. I. Damgård and J. Nielsen. Improved non-committing encryption schemes based on a general complexity assumption. In *Proc. CRYPTO 2000*, 2000. 2
64. O. Reingold, R. Shaltiel, and A. Wigderson. Extracting randomness via repeated condensing. In *IEEE Symposium on Foundations of Computer Science*, pages 22–31, 2000. 23
65. A. Shamir. How to share a secret. In *Communications of the ACM*, pages 612–613, 1979. 38
66. C. E. Shannon. Communication theory of secrecy systems. In *Bell System Technical Journal*, pages 656–715, 1949. 8
67. S. Vaarala. T-110.5210 cryptosystems lecture notes, implementation issues, 2007. `http://www.tml.tkk.fi/Opinnot/T-110.5210/2007/lectures.html`. 2
68. S. P. Vadhan. Constructing locally computable extractors and cryptosystems in the bounded-storage model. *J. Cryptol.*, 17(1):43–77, 2004. 4, 24
69. B. Yee. *Using secure coprocessors*. PhD thesis, Carnegie Mellon University, May 1994. 8