How to Protect Yourself without Perfect Shredding*

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

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

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

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

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

1 Introduction

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

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

Examples where perfect erasures have been used extensively include the areas of *proactive security* [7,17,19,22,30,36], *forward security* [1,12,20] and *intrusion*

* Full version of paper available at `http://eprint.iacr.org/` as [5]

L. Aceto et al. (Eds.): ICALP 2008, Part II, LNCS 5126, pp. 511–523, 2008.

resilience [27], and *adaptive security* [2,6,28,37]. Whereas erasures merely simplify the design of adaptively secure protocols, some form of erasures is provably necessary for achieving proactive security and even for defining the task of forward security as we explain below.

The goal of *Proactive Security* introduced in [36] is to achieve secure multiparty computations where some fraction of the parties are faulty. The identity of faulty parties are decided by a *mobile* adversary who can corrupt a different set of players in different *time periods* (here the protocols assume time is divided into well-defined intervals called time periods) subject to an upper bound on the total number of corrupted players per time period. At the heart of the solutions pursued in the literature are secret sharing methods in which in every time period, the old shares held by players are first replaced by new shares and then perfectly erased. It is easy to prove that secret sharing would be impossible to achieve without some form of erasures: otherwise a mobile adversary which is able to corrupt every single player in some time period or another, can eventually recover all old shares for some single time period and recover the secret. Forward security [1,12,20]. is an approach taken to tackle the *private key exposure* problem, so that exposure of long-term secret information does not compromise the security of previous sessions. Again, the lifetime of the system is divided into time periods. The receiver initially stores secret key SK_0 and this secret key "evolves" with time: at time period i, the receiver applies some function to the previous key SK_{i-1} to derive the current key SK_i and then key SK_{i-1} is perfectly erased. The public (encryption) key remains fixed throughout the lifetime of the scheme. A forward-secure encryption scheme guarantees that even if an adversary learns the secret key available at time i, SK_i, messages encrypted during all time periods prior to i remain secret. Intrusion Resilience is a strengthening of forward security [27] which can be viewed as combination of forward and backward security. Obviously, erasures are essential to define (and solve) both the forward security and intrusion resilience problems.

Finally, an example of a different flavor of the utility of erasures to guard against adversaries that can choose which future parties to corrupt as the protocol proceeds, based on information already gathered. Erasures are useful in this context since they limit the information the adversary sees upon corrupting a party. Protocols designed without erasures (although possible in this context), tend to be much more complex than those that rely on data erasures [2,6,28,37].

Unfortunately, perfect erasures of data are hard to achieve in practice and are thus problematic as a security assumption, as pointed out by Jarecki and Lysyanskaya [28] in their study of adaptive adversaries versus static adversaries in the context of threshold secret sharing.

Some of the difficulty in implementing perfect erasures is illustrated in the works of Hughes and Coughlin, Garfinkel, and Vaarala [18,24,25,39]. The root cause of this difficulty is that systems are actually designed to preserve data, rather than to erase it. Erasures present difficulties at both the hardware level (e.g. due to physical properties of the storage media) and at the software level (e.g. due to the complications with respect to system bookkeeping and backups).

At the hardware level, e.g. for hard drives, the Department of Defense recommends overwriting with various bit patterns [35]. This takes the order of days per 100GB, and is not fully effective because modern hard drives use block replacement and usually employ some form of error correction. For main memory, due to "ion migration", previous states of memory can be determined even after power off. At the software level, many operating systems detect and remap bad sectors of the hard drive on the fly, but original data can remain in the bad sectors and be recoverable.

1.1 This Paper

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

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

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

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

The function h is assumed to be a function only of the storage contents to be erased. Furthermore, for simplicity we assume that h is fixed in advance – our schemes remain secure without any modification even if the adversary chooses a new h_i prior to each new erasure. This choice seems to adequately capture erasures that are only partially successful due to the physical properties of the storage media[1]. However, this may not adequately capture situations where the failure to erase comes from interactions with an operating system, for

[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.

instance memory swapping, and caching. In order to capture this sort of erasure failures, one might want to let h to be a function of some information other than the contents to be erased, or alternatively to be determined adaptively as the computation evolves.

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

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

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

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

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

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

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

Our technique for the case of a single secret is inspired by results in the *bounded storage model*, introduced by Maurer [32,33]. Work by Lu [29] casted results in the bounded storage model in terms of *extractors* [34], which are functions that when given a source with some randomness, purifies and outputs an almost random string.

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

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

There is however a snag in the above description: in order to employ this scheme, one has to evaluate the extractor Ext without leaving any trace of the intermediate storage used during the evaluation. Recall that 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 [31]. It would seem superficially that locally computable strong extractors [40] can be used, but unfortunately they cannot (proof deferred to full version [5]).

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

For maximum generality we formulate our results in the *Universally Composable (UC)* framework. In particular we use the notion of *UC emulation* [3], 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, in [5] 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 $1 < c < 2$ is a constant.

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

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

Remark 1. As we elaborate in [5], a side benefit of using using our constructions is that it can be resistant to a practical class of physical attacks [21] that involves freezing RAM and recovering secrets from it.

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

1.2 Related Work

The Bounded Storage Model (BSM). The Bounded Storage Model (BSM) proposed by Maurer [32,33], considers computationally unbounded but storage limited adversaries. This enables novel approaches to the secure communication problem as follows. The communicating parties begin with a short initial secret key k. In the first phase they use this key k and access to a long public random string R to derive a longer key X. The storage bounded adversary computes an arbitrary length-shrinking function on R. In the second phase, R "disappears", and the parties will use X as a one-time pad to communicate privately.

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

However, conceptually the settings of the BSM and partial erasures are fundamentally different. In the BSM model possibility is proved by putting limitations on the **adversary** (storage), where as in our work possibility is proved

inspite of putting limitation on the **honest parties** (erasing capability). Thus, although some of techniques are similar the setup is entirely different. From a technical point of view there are two differences we emphasize as well. Firstly, the extractors that we use must be computable with constant sized memory), whereas in the BSM the extractors are not necessarily computable with constant-sized memory. Secondly, in the BSM, it is assumed that the adversary's storage bound remains the same as time goes by, namely a constant fraction ϕ of the public randomness R. The same assumption is used in the bounded retrieval model [11,10,14,15,16]. For instance [16] constructs intrusion resilient secret sharing schemes by making the shares large and assuming that the adversary will never be able to retrieve any share completely. For partial erasures this bound is unreasonable, and we allow the adversary to get ϕ fraction of R_i for each erasure operation.

Exposure Resilient Functions. Exposure-Resilient Functions, or ERFs, were introduced by Canetti et al. [4,13]. An ℓ-ERF is a function with a random input, such that an adversary which learns all but ℓ bits of the input, cannot distinguish the output of the function from random.

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

Encryption as Deletion. As Di Crescenzo et al. [9] noted, one simple but inefficient way to implement erasable memory can be obtained by using the crypto-paging concept of Yee [41]. Assume that some amount of storage that is linear in the security parameter is available that is perfectly erasable, and some other poly storage is persistent. To make the persistent memory effectively erasable, pick an encryption scheme and keep the key in the erasable part. Always encrypt the contents to be kept on the persistent storage. Then erasing the key is as good as erasing the contents.

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

2 How to Make Secrets Partially Erasable

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

number of bits than to perfectly erase a small number. Furthermore, we show in 2.1 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 1 (Statistical Distance). *Suppose that A and B are two distributions over the same finite set Σ. The statistical distance between A and B is defined as* $\Delta(A;B) := \frac{1}{2}\sum_{\sigma\in\Sigma}\left|\Pr_A[\sigma]-\Pr_B[\sigma]\right|$.

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

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

Definition 3 (Partially Erasable (or Expanded) Form of a Secret). *Let* $\mathsf{Exp}(\circ,\circ)$ *be the "expansion" function taking the secret s to be expanded as the first input and randomness as the second, and* Con *be the "contraction" function taking the output of* Exp *as the input. Let* $h_i^s := h(\mathsf{Exp}(s,\$_i))$, *where* $\$_i$ *are independent randomness. We say that* $(\mathsf{Exp},\mathsf{Con})$ *is (ℓ,α,ϕ)-partially erasable form of a secret if $\forall s\in\{0,1\}^n$, for any h with leakage fraction ϕ,*

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

Remark 3. We require both Exp and Con to be computable with constant memory to ensure that intermediate computations can be kept in the registers which are perfectly erasable.

Remark 4. We require indistinguishability for many (ℓ above) erasures to account for the fact that many computations may be done during the protocol (directly or indirectly) on the secret, from which the adversary might gain more information on a secret. Generally, an adversary may have many partially erasable forms of the same secret (i.e adversary can see $h(\mathsf{Exp}(s,\$_i))$ s.t for each i adversary knows a 1-1 and onto correspondence q_i from $\mathsf{Exp}(s,\$_i)$ to s).

An example of an expanded form of a secret which can be partially erased and satisfies correctness and secrecy (in the above definition) would be to use a universal hash function family $\{H_R\}$ as follows: expand s to (v,R,k) s.t. $s = H_R(k)\oplus v$. By using the leftover hash lemma [26], 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).

Let us first focus on bounding $d(H_R(k)|R,h(k))$.

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

The proof of this theorem and the next is deferred to the full version [5].

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

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

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

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

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

$$\Leftrightarrow m \geq \frac{3\log((\ln 2)\ell) - 4 + 6(\alpha+1) + (1+\beta)n}{(1-\phi)} \quad (2)$$

Let us make a few observations. Inequality 1 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 2 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.

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[2].

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

Our preferred partial erasable form will be to use Toeplitz hashing instead (whose universality is proven in [31]). A random Toeplitz matrix $R \in \{0,1\}^{n\times m}$

[2] Which makes our results stronger than required by the definition.

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

It is easy to see how to implement $(\mathsf{Exp}, \mathsf{Con})$ corresponding to Toeplitz hash that is computable with constant memory, bit by bit. From the triangle inequality, for all $s, s' \in \{0,1\}^n$, $\Delta\Big(H_R(k) \oplus s, R, h(k); H_R(k) \oplus s', R, h(k)\Big) \leq 2d(H_R(k)|R, h(k))$. 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.*

2.1 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 (proof in [5]):

Theorem 3 (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: $\Psi \geq \frac{1}{1-\phi}\left(1 - \frac{n+\alpha-1}{n\ell 2^{\alpha-1}}\right)$.*

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

Efficiency. Let us see how tight our construction is to the lower bound. If a completely random R is used and $H_R := R \cdot k$ (whose universality is proven in [8]), 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 2, we see that this bound is a (growing) factor of n away from the optimal.

If a Toeplitz matrix R is used instead, then the corresponding expanded form will be $R \in \{0,1\}^{n \times m}, k \in \{0,1\}^m$ and $x \in \{0,1\}^n$ such that $R \cdot k \oplus x = s$. In this case, R requires $n+m-1$ bits to specify (since it is Toeplitz), and k requires m bits and x requires n bits respectively. So in this case, n bits get expanded into $2m+2n-1$ bits, and Ψ is $2\frac{m}{n} + 2 - \frac{1}{n}$. Plugging in inequality 2, 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 3. 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[3].

[3] In the full version [5] we discuss two other partially erasable forms: 1. ϵ-almost universal hashing which, provided that $\phi > 1/4$, gives roughly $\Psi \geq \frac{c}{1-\phi}$ for some constant $1 < c < 2$, and 2. strong extractors computable with constant memory (e.g. those in NC^0), which, provided that the extractor is near optimal, would achieve the lower bound. Unfortunately we do not know of such extractors.

3 A General Construction

3.1 Computing on Partially Erasable Secrets at the Gate Level

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

To evaluate a gate, the two corresponding input bits are reconstructed from their expanded forms in the registers (using Con). The gate is evaluated, resulting in an output bit b in the registers. This output bit is expanded into the partially erasable form and output to main memory[4], by using Exp. This can be done with constant memory. Note that if we just store the values of the wires naïvely, i.e. by individually expanding the 1-bit value of each wire to a Ψn size secret, then the overhead of our scheme will not even be constant. So we must amortize the cost: group the wires of the circuit into groups of size t (i.e., there are t wires in each group), where t is such that secrets of size t are expanded into m-bit strings. Now, when we write the values of the wires in an expanded form, we expand all the t values into a single m-bit string. This will make sure that the overhead of the general compiler will still be the same as the overhead for the scheme described in section 2.

The above is an informal description of COMPUTE-IN-REGISTER$(g, \mathsf{Exp}(s, \$))$, which makes sure that the computation of $g(s)$ is done properly without leaking intermediate computation (through expressing them in expanded form). The proof of the following lemma is in [5].

Lemma 1. *Let $s \in \{0,1\}^n$ be any secret, g be the function to be computed on s, where each bit i of output of $g(s)$ is computed by a poly(n)-sized circuit C_{g_i}, consisting of gates $\{X_i^j\}_j$. Let v_i denote the number of partially erased intermediate computations while computing the i-th output bit (*COMPUTE-IN-REGISTER$(g, \mathsf{Exp}(s, \$))$*), where $\mathsf{Exp}(s, \$)$ is the expansion function of a (ℓ, α, ϕ)-partially erasable form and $v := \sum_i v_i \leq \ell$.*

Then, the adversary cannot distinguish the case of s versus any $s' \in \{0,1\}^n$ being partially erased, by more than $2^{-\alpha}$ probability, i.e.:

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

Starting with any protocol that uses perfect erasures, we replace all computations on the secrets to use COMPUTE-IN-REGISTER instead. The result that we get is (proof in [5]):

[4] 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.

Theorem 4. *For any protocol Π_{org} that requires perfect erasures (for security), the protocol $\Pi_{new} = \text{COMPILER}(\Pi_{org})$ UC-emulates Π_{org}, and tolerates (maintains security even with) imperfect/partial erasures in the register model.*

References

1. Anderson, R.: Two remarks on public key cryptology invited lecture. In: Acm-Ccs 1997 (1997)
2. Beaver, D.: Plug and play encryption. In: Kaliski Jr., B.S. (ed.) CRYPTO 1997. LNCS, vol. 1294. Springer, Heidelberg (1997)
3. Canetti, R.: Universally composable security: A new paradigm for cryptographic protocols. In: Proc. 42nd IEEE Symp. on Foundations of Comp. Science, pp. 136–145 (2001)
4. Canetti, R., Dodis, Y., Halevi, S., Kushilevitz, E., Sahai, A.: Exposure-resilient functions and all-or-nothing transforms. In: Preneel, B. (ed.) EUROCRYPT 2000. LNCS, vol. 1807, pp. 453–469. Springer, Heidelberg (2000)
5. Canetti, R., Eiger, D., Goldwasser, S., Lim, D.-Y.: How to protect yourself without perfect shredding (full version) (2008), `http://eprint.iacr.org/`
6. Canetti, R., Feige, U., Goldreich, O., Naor, M.: Adaptively secure computation (1995)
7. Canetti, R., Gennaro, R., Herzberg, A., Naor, D.: Proactive security: Long-term protection against break-ins. In: CryptoBytes (1) (1999)
8. Carter, J.L., Wegman, M.N.: Universal classes of hash functions. JCSS 18 (1979)
9. Di Crescenzo, G., Ferguson, N., Impagliazzo, R., Jakobsson, M.: How to forget a secret. In: Meinel, C., Tison, S. (eds.) STACS 1999. LNCS, vol. 1563, pp. 500–509. Springer, Heidelberg (1999)
10. Di Crescenzo, G., Lipton, R.J., Walfish, S.: Perfectly secure password protocols in the bounded retrieval model. In: Theory of Cryptography Conference, pp. 225–244 (2006)
11. Dagon, D., Lee, W., Lipton, R.J.: Protecting secret data from insider attacks. In: Financial Cryptography, pp. 16–30 (2005)
12. Diffie, W., Van-Oorschot, P.C., Weiner, M.J.: Authentication and authenticated key exchanges. In: Designs, Codes, and Cryptography, pp. 107–125 (1992)
13. Dodis, Y.: Exposure-Resilient Cryptography. PhD thesis. MIT, Cambridge (2000)
14. Dziembowski, S.: Intrusion-resilience via the bounded-storage model. In: Theory of Cryptography Conference, pp. 207–224 (2006)
15. Dziembowski, S.: On forward-secure storage. In: Dwork, C. (ed.) CRYPTO 2006. LNCS, vol. 4117, pp. 251–270. Springer, Heidelberg (2006)
16. Dziembowski, S., Pietrzak, K.: Intrusion-resilient secret sharing. In: FOCS 2007, Washington, DC, USA, pp. 227–237. IEEE Computer Society, Los Alamitos (2007)
17. Frankel, Y., Gemmel, P., MacKenzie, P.D., Yung, M.: Proactive rsa. In: Kaliski Jr., B.S. (ed.) CRYPTO 1997. LNCS, vol. 1294, pp. 440–454. Springer, Heidelberg (1997)
18. Garfinkel, S.L.: Design Principles and Patterns for Computer Systems That Are Simultaneously Secure and Usable. PhD thesis. MIT, Cambridge (2005)
19. Gennaro, R., Jarecki, S., Krawczyk, H., Rabin, T.: Robust threshold Dss signatures. In: Maurer, U.M. (ed.) EUROCRYPT 1996. LNCS, vol. 1070, pp. 354–371. Springer, Heidelberg (1996)

20. Günther, C.G.: An identity-based key-exchange protocol. In: Proc. EUROCRYPT 1989, pp. 29–37 (1989)
21. Halderman, J.A., Schoen, S.D., Heninger, N., Clarkson, W., Paul, W., Calandrino, J.A., Feldman, A.J., Appelbaum, J., Felten, E.W.: Lest We Remember: Cold Boot Attacks on Encryption Keys (April 2008), http://citp.princeton.edu/memory/
22. Herzberg, A., Jakobsson, M., Jarecki, S., Krawczyk, H., Yung, M.: Proactive public key and signature systems. In: ACM Conference on Computers and Communication Security (1997)
23. Herzberg, A., Jarecki, S., Krawczyk, H., Yung, M.: Proactive secret sharing, or: How to cope with perpetual leakage. In: Coppersmith, D. (ed.) CRYPTO 1995. LNCS, vol. 963, pp. 339–352. Springer, Heidelberg (1995)
24. Hughes, G., Coughlin, T.: Tutorial on hard drive sanitation (2006), http://www.tomcoughlin.com/
25. Hughes, G., Coughlin, T.: Secure erase of disk drive data, pp. 22–25 (2002)
26. Impagliazzo, R., Levin, L.A., Luby, M.: Pseudo-random generation from one-way functions. In: STOC 1989, pp. 12–24 (1989)
27. Itkis, G., Reyzin, L.: Sibir: Signer-base intrusion-resilient signatures. In: Yung, M. (ed.) CRYPTO 2002. LNCS, vol. 2442, pp. 499–514. Springer, Heidelberg (2002)
28. Jarecki, S., Lysyanskaya, A.: Adaptively secure threshold cryptography: Introducing concurrency, removing erasures. In: Preneel, B. (ed.) EUROCRYPT 2000. LNCS, vol. 1807, pp. 221–243. Springer, Heidelberg (2000)
29. Lu, C.-J.: Encryption against storage-bounded adversaries from on-line strong extractors. In: Proc. CRYPTO 2002, pp. 257–271 (2002)
30. Lysyanskaya, A.: Efficient threshold and proactive cryptography secure against the adaptive adversary (extended abstract) (1999)
31. Mansour, Y., Nisan, N., Tiwari, P.: The computational complexity of universal hashing. In: Proc. 22nd ACM Symp. on Theory of Computing (2002)
32. Maurer, U.: A provably-secure strongly-randomized cipher. In: Damgård, I.B. (ed.) EUROCRYPT 1990. LNCS, vol. 473, pp. 361–373. Springer, Heidelberg (1991)
33. Maurer, U.: Conditionally-perfect secrecy and a provably-secure randomized cipher, pp. 53–66 (1992)
34. Nisan, N., Zuckerman, D.: Randomness is linear in space. Journal of Computer and System Sciences 52(1), 43–52 (1996)
35. Department of Defense. DoD 5220.22-M: National Industrial Security Program Operating Manual (1997)
36. Ostrovsky, R., Yung, M.: How to withstand mobile virus attacks, pp. 51–61 (1991)
37. Damgård, I., Nielsen, J.: Improved non-committing encryption schemes based on a general complexity assumption. In: Bellare, M. (ed.) CRYPTO 2000. LNCS, vol. 1880, Springer, Heidelberg (2000)
38. Shannon, C.E.: Communication theory of secrecy systems. Bell System Technical Journal, 656–715
39. Vaarala, S.: T-110.5210 cryptosystems lecture notes (2006)
40. Vadhan, S.P.: Constructing locally computable extractors and cryptosystems in the bounded-storage model. J. Cryptol. 17(1), 43–77 (2004)
41. Yee, B.: Using secure coprocessors. PhD thesis (May 1994)