

---

# INSPECTION-RESISTANT MEMORY ARCHITECTURES

---

THE AUTHORS EXAMINE THE RELATIONSHIP BETWEEN SECURITY, AREA, AND EFFICIENCY IN SEVERAL NOVEL MEMORY ARCHITECTURES AND QUANTITATIVELY EXAMINE THE RESULTING SYSTEMS THROUGH CRYPTOGRAPHIC ANALYSIS AND MICROARCHITECTURAL IMPACT. THEY DISCOVER AN EFFICIENT SCHEME IN WHICH, EVEN IF AN ATTACKER CAN INSPECT A STORED BIT'S VALUE WITH A PROBABILISTIC ERROR OF ONLY 5 PERCENT, THE SYSTEM CAN PREVENT THAT ADVERSARY FROM LEARNING ANY INFORMATION ABOUT THE ORIGINAL UNCODED BITS WITH 99.999999999 PERCENT PROBABILITY.

**Jonathan Kaveh Valamehr**  
University of California,  
Santa Barbara

**Melissa Chase**  
**Seny Kamara**  
**Andrew Putnam**  
**Daniel Shumow**  
Microsoft Research

**Vinod Vaikuntanathan**  
University of Toronto

**Timothy Sherwood**  
University of California,  
Santa Barbara

..... Computer architects must balance performance with all the other design aspects, including reliability, power consumption, cost, ease of use, and, of course, security. Just as we've developed architecture-level schemes to help manage circuit-level problems to achieve reliability (dealing with soft-error susceptibility and early wear-out,<sup>1-3</sup> for example), we need new architecture-level schemes to deal with the circuit-level security problem of information leakage through hardware examination.

Underlying many, if not most, modern security schemes is the idea of a *key*—a small set of bits whose secrecy ensures the overarching policy's effectiveness. There are many architecture-level techniques for keeping these bits secret. Processors enforce the operating system's memory-access policies, information-flow-aware hardware can prevent secret data from escaping at runtime,<sup>4-8</sup> and many side channels can be mitigated through randomization or partitioning.<sup>9-11</sup> However, providing secrecy becomes even more challenging when the device in question is portable, such as a smart card or a

cellphone, or when the keys are shared across many different devices, as in many non-network-based attestation schemes. In these cases, we must consider that an adversary might be able to physically dismantle and inspect the system, bypassing all of our traditional security mechanisms.

If the secrets are of sufficient value, a determined attacker can use many intensive methods to learn these bits from even minute physical differences imprinted on the chip by that bit's storage. With tools available for rent at any major university or fabrication facility, a memory array can be carefully sliced apart with a focused-ion beam and inspected under an electron microscope. Our goal is to architect memory structures that present the same interface as a simple memory, yet are hardened against these direct and destructive physical attacks in their most general form. If an attacker can perfectly deconstruct and read (with no measurement error) every bit in the system, we can do little to protect the secret key. However, the attacker's role is not so easy. Often there are errors inherent to the measurement process, and as these

errors are made and some fraction of the bits are incorrectly identified, the attacker loses information.

Our core idea is that instead of storing the secret key in its original form on the chip, we can encode the keys (or other secret data) using additional bits, increasing the number of bits that an attacker must identify to recover the secret key. Ideally, this will force the attacker to successfully identify a large number of the encoded bits to be able to learn even a single bit of information about the original secret key.

## Background and motivation

We begin by discussing our motivation for investigating how to store secrets on a device where the attacker has full access. We also provide some background on memory remanence attacks and other physical attacks on cryptography, as well as the scenarios in which they are problematic in real life, and a formalization of our threat model.

### Physical inspection attacks

Storing a secret in a device when the attacker has full physical access to that device is extremely difficult. At a fundamental level, attacks that attempt to infer a set of bits from a device can be divided into two classes: *passive attacks*, in which the system's interface is probed for either timing or electrical differences, and *intrusive attacks*, those that actually breach the package's boundary, allowing the attacker to scan, probe, or alter the physical hardware itself. Although still a topic of further research, passive attacks and their countermeasures have been discussed extensively in prior work. In this article, which is based on our ISCA 2012 paper,<sup>12</sup> we concern ourselves primarily with the latter, more intrusive style of attacks.

Independent of whether active countermeasures have been applied, we specifically consider unpowered attacks in which attackers are free to slice, cut, and examine the silicon however they desire. Specifically, in the threat model we consider, an attacker has physical access to a dead device on which a secret key is stored or has been stored. Rather than select one specific memory technology and a specific mode of attack, we abstract

this problem to one in which an adversary, after performing an attack, correctly identifies a fraction  $p$  of the bits, and learns nothing about the remaining  $1 - p$  fraction of bits. This model fits several device technologies:

- *Antifuse*. Antifuses are a nonvolatile write-once memory formed by creating an electrical connection through the application of high current across a thin channel. The actual connections are electrically stable, yet they're difficult to examine even under a scanning electron microscope. Researchers have primarily analyzed their security properties within the context of field-programmable gate array (FPGA) reverse engineering.<sup>13,14</sup>
- *Eeprom and Flash*. Flash memory and electrically erasable programmable read-only memory (Eeprom) are similar in structure, with both storing charges on floating gates. When Eeprom and Flash cells are overwritten, some residue of the previous bits remains within the substrate as bias, and differences in the threshold voltage or gate voltage can be measured to detect a cell's state. This effect is particularly noticeable in infrequently programmed cells,<sup>15</sup> which is likely the case for cells holding a secret key. Prior work has shown that this bias can survive even tens of redundant "clean-up" writes, making complete erasure of the information difficult within the time dictated by a countermeasure.<sup>16</sup>
- *SRAM*. Even volatile memories are subject to analysis in many cases. Static RAM (SRAM), while storing a bit for even as little as a half a second, can develop tell-tale signs due to differences in the electromigration at the bit-cell level.<sup>17</sup> Because of this, even volatile memories that hold secret keys must be protected.

Although physical analysis gives an attacker incredible access to the internals of the system, we believe these analyses cannot be 100 percent accurate. However, as we will show in the next section, even if these

techniques were to be 95 percent accurate, it is possible to create reasonable architectures that prevent even a single bit of the key from leaking.

### Architectural goal and attack model

When facing an adversary who might employ any number of attacks on these memory technologies and correctly identify any bits of a secret, we need a hardware component that will

- act like a standard memory, allowing random access to the stored keys with no special restrictions;
- allow an adversary to learn some distribution of bits without giving away any information about the secret keys; and
- be efficient enough that the end system can be included in a real design.

Our secret-store memory keeps some number of keys, each of length  $k$ . Each key is stored in an encoded form that takes some larger number of bits  $c$ ; we'll call this the "encoded key." Attackers examine the  $c$  bits of the encoded key through whatever methods are at their disposal, and they make a best guess as to the state of each bit of the encoded key. The attacker, when dissecting and analyzing each bit, has a probability  $p$  of learning the bit's correct value, and a probability  $1 - p$  of learning nothing about the bit. In particular, as a running example, we will consider the cases where  $p = 90$  percent, which, by this reasoning, is at least as strong as saying that the adversary guesses correctly with probability 95 percent and incorrectly with probability 5 percent. In other words, if  $p = 90$  percent, the adversary will learn the value of 90 percent of the bits, and nothing about the others. However, the adversary is still free to guess, and these guesses will be correct 50 percent of the time. Thus  $p = 90$  percent is the same as 95 percent correct guesses.

In this article, we don't estimate a reasonable value for  $p$ , but rather present results and the architectural impact across the spectrum of possibilities. We will be conservative and consider an attack successful if the attacker can infer anything about the secret, even a single bit. Although a single-bit

leakage isn't enough to break any reasonable cryptographic scheme (because an attacker could have just tried both possible options of 0 and 1), if other copies of the key are known to exist, an attacker might combine information from many such attacks to reduce the keyspace far enough that it can be searched exhaustively.

### Architectures

There are several different architectural options here, and we begin with the simplest, and work our way quantitatively toward the best possible options. We examined four design options for our architecture: secret sharing, random-matrix encoding, a hybrid scheme, or dynamic matrix creation.

#### Option 1: Apply the idea of secret sharing

The first, and easiest to understand, option for hiding a bit of information is secret sharing. A simple and efficient secret sharing scheme consists of XORing the secret  $x$  with  $s$  random bits ( $x_1, \dots, x_s$ ), and setting the shares to be  $[x_1, x_2, \dots, x_s, x_{s+1} = x \oplus (x_1 \oplus x_2 \oplus \dots \oplus x_s)]$ . For example, to share a 0 bit three ways, we first take two random bits. We then set the third bit such that the parity of the three bits is 0. A change (or, measurement error) in any of the three bits will result in a change (or, measurement error) in value stored. In a sense, the fragility of the parity bit used to catch errors in one scheme is used to help hide data here.

How do we store a  $k$ -bit key using this method? The simple option is for each individual bit of the key to be shared across  $s + 1$  stored bits, increasing the total number of bits to be stored per key to  $c = (s + 1)k$ . To be successful, an attacker must compromise all of the bits of at least one  $(s + 1)$ -bit share block. The probability of compromising a share block is now  $p^{s+1}$ , and our attacker has  $k$  independent trials to succeed, yielding a total probability of success  $P_{\text{succ}} = 1 - (1 - p^{s+1})^k$ . Interestingly, there is a steep drop in the probability of a successful attack as we grow from 40 to 80 shares per bit. Clearly, replicating every bit in the key by a factor of 80 is a steep overhead to endure, but as we increase  $s$ , we can drive  $P_{\text{succ}}$  arbitrarily close to zero.

## Option 2: Encode with a random matrix

Intuitively, a simple secret-sharing-based scheme has a large overhead in storage because we encode each bit separately. This raises the possibility of encoding all the bits together using a matrix  $T$  and achieving much better parameters. A coding-based scheme is a natural extension of the secret sharing scheme, yet it shares many of its theoretical properties. At a high level, the coding-based scheme works as follows. We will encode the key as  $s$  random bits plus  $k$  data bits. To do this, we fix random subsets  $T_1, T_2, \dots, T_k \subseteq \{1, 2, \dots, s\}$  in advance. The choice of these subsets doesn't depend on the data to be encoded; in fact, in the implementation, they will be chosen at random once and for all, and stored on chip. To encode a sequence of  $k$  bits  $x_1, \dots, x_k$ , we choose  $s$  random bits  $r_1, \dots, r_s$  (as before) and compute the  $k$  data bits  $r_{s+j}$  for  $j = 1, \dots, k$  as  $r_{s+j} = x_j \oplus (\bigoplus_{i \in T_j} r_i)$ . In other words, we compute the XOR of a subset of the  $s$  random bits and XOR the result to the bit that we wish to encode. Thus, encoding a string of  $k$  bits consists of choosing  $s$  random bits, computing a matrix vector product of these  $s$  bits (written out as a vector) with a fixed  $s$ -by- $k$  matrix, and XORing the result with the  $k$ -bit key to be protected. A downside of this scheme is that the matrix  $T$  must be chosen at random and must be stored on chip. However, the storage required for the matrix can be amortized if we store a large number of keys, because we can reuse the same matrix for all the keys, and  $T$  can be completely public!

## Option 3: A novel combination

Both the options presented so far—the secret-sharing-based solution and the coding-based solution—have pros and cons. Although researchers have proposed both those schemes for other applications, we are the first to quantify their actual implementation overhead and the first to apply them to the specific problem of inspection resistance in memory. However, we go beyond these basic applications to design a hybrid scheme that achieves the best of both worlds. The idea is to use a code-based secret sharing scheme with a small matrix, and then secret share those pre-encoded bits.

To do this, we first encode the key's individual bits using a code-based secret sharing scheme with a smaller matrix  $T$ . Roughly speaking, the intent is to reduce the attack's severity from a per-bit leakage probability of  $p$  to a smaller number  $p'$ . We then further encode the resulting string using the secret-sharing-based scheme. The upshot is that the matrix needed for the coding-based scheme is smaller because it only needs to be strong enough to reduce the attack's severity from  $p$  to  $p'$ . As Figure 1 shows, the combined scheme outperforms both the coding-based and the secret-sharing-based schemes when the matrix  $T$  is stored on chip. Figure 2 shows how the storage overhead varies as a function of the key's length. Although the increase in storage overhead with the key length is disturbing, it is predominantly due to the need to store the large matrix  $T$ , and will be removed by our improved solution.

Figure 3 shows that the overhead caused by storing the matrix can be amortized by the number of keys we store in the system. This is simply because we can use the same matrix  $T$  to encode many different keys. Figure 4 shows a plot of the area required to store the encoded key (together with the auxiliary information such as the matrix  $T$ ) as a function of the adversary's success probability. Obviously, if we let the adversary succeed with probability 1, very little storage is required. The storage jumps to a certain number as soon as we demand the adversary's success probability to be less than, say, 0.1, and stays more or less constant from then on. This phenomenon can be explained by the (roughly) logarithmic dependency of the storage on the success probability. In other words, reading Figure 4 from right to left roughly gives a logarithmic curve.

## Option 4: Dynamic matrix creation

In the last section, we showed a way to ameliorate the effect of storing the huge matrix  $T$  in the coding-based scheme. We now show a way to eliminate this overhead almost entirely, using a cryptographic hash function (such as SHA-2) that's specifically designed to produce output that looks random in a strong sense but which can be generated deterministically from a very small input.

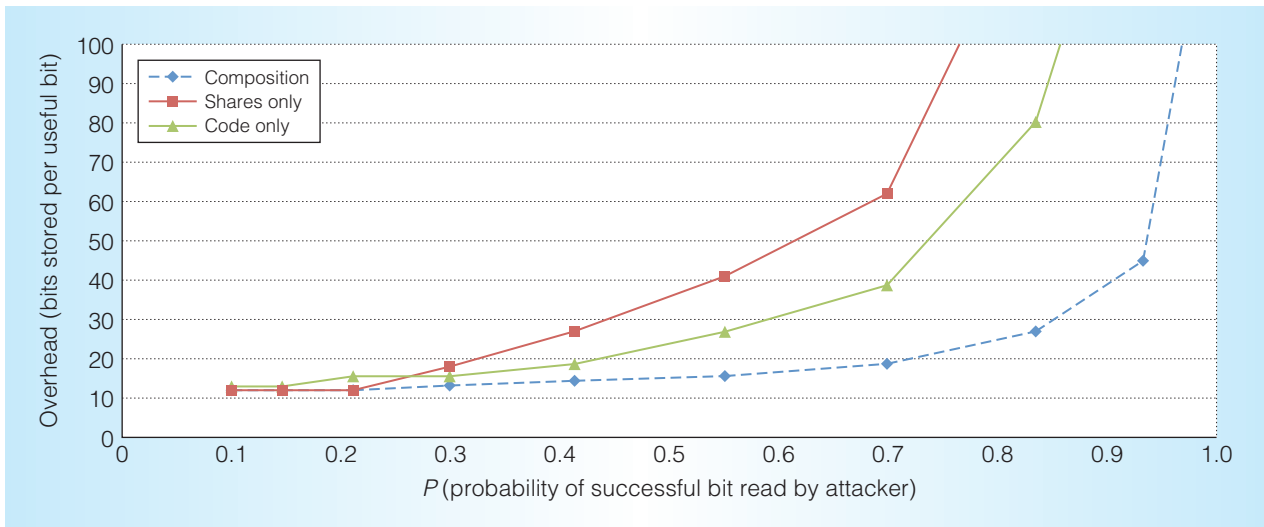


Figure 1. The efficiency of different schemes as a function of the attacker’s power to read individual bits. The x-axis is the probability  $p$  that an attacker can correctly determine the value of each stored bit in the scheme. The y-axis is the size of the code (as a ratio) needed to ensure that the probability that an attacker can learn at least one bit is below one in a billion. Although the code scheme scales better as the attacker becomes more powerful, our approach outperforms either individual approach when the matrix must be stored on chip.

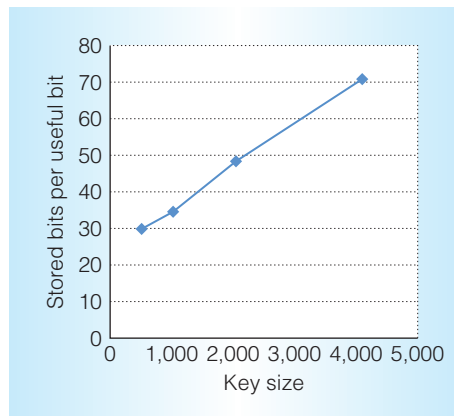


Figure 2. The combined scheme’s efficiency as a function of the key size. As the key grows beyond 1,024 bits, the scheme’s efficiency decreases (as the number of stored bits required per key bit increases linearly).

In particular, we note that modern cryptographic hash functions are designed to behave like truly random functions (in which each output is chosen independently and uniformly at random) as much as possible. In fact, evidence of any nontrivial application in which a hash function behaved significantly differently from a random function

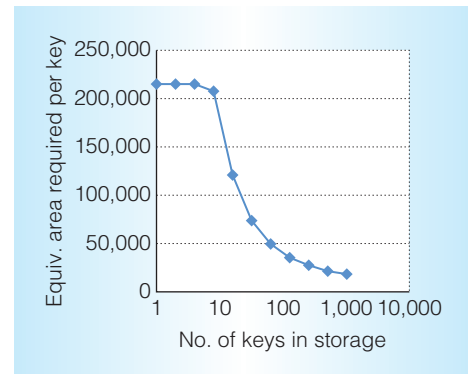


Figure 3. The optimally combined scheme’s efficiency as a function of the number of keys stored in the architecture. With very few keys stored, the cost of the matrix can’t be amortized, and the best scheme uses secret sharing only. However, once we get beyond on the order of 10 keys stored, the scheme’s efficiency improves drastically as we can amortize the cost of larger matrices.

would be considered a major weakness, and finding such a weakness would be a major result.

Our approach is to use the SHA-2 family of hash functions, which consists mainly of

two functions, SHA-256 and SHA-512 (where the numbers indicate the hash function's output length), to dynamically generate the matrix  $T$ . Specifically, we'll choose a short random string  $seed$ , and compute the  $i$ th column as:  $SHA2(seed \circ i \circ 1) \circ SHA2(seed \circ i \circ 2) \circ \dots \circ SHA2(seed \circ i \circ l)$ , where  $l = s/256$ ,  $seed$  is a 256-bit string,  $seed \circ i \circ j$  is encoded as a 448-bit string to be input to the SHA-256 hash function, and  $\circ$  denotes concatenation of strings.

To ground the microarchitectural impact of this fourth option, consider a key length  $k = 1,024$ , and the probability of successfully attacking a single stored bit  $p = 90$  percent. This fourth option, outlined in Figure 5, requires 35 bits of additional storage per bit of the key. At the 65-nm process node, the cost of securing a single key using this method is  $0.589 \text{ mm}^2$ . Storing additional keys helps amortize the cost of the secure hash algorithm (SHA) generator and the computation logic. Storing the 128 keys requires  $6.18 \text{ mm}^2$ , still well within the area bounds for embedded and consumer devices. An additional advantage to this approach is that SHA-2 is widely used, and optimized hardware implementations are already available. In particular, most of the next-generation SHA implementations require on the order of 20,000 gates to implement, which is quickly amortized over the entirety of the key storage architecture.

To help ground this work in a real-world context, consider the problem of networkless attestation for console gaming systems. To help prevent cheating as well as unauthorized and malicious knock-offs, a console system might wish to ensure that it talks only with certified devices. Well-known cryptographic methods, such as public key authentication, allow this process to occur without requiring the transmission of any secrets. However, each system must, in some form or another, physically possess and use that secret. The problem is that the cost of leaking that key is now much larger. Once the secret is leaked, any number of new systems can be created containing that same key and the scheme is completely broken. Such "break-once run-anywhere" attacks provide attractive and

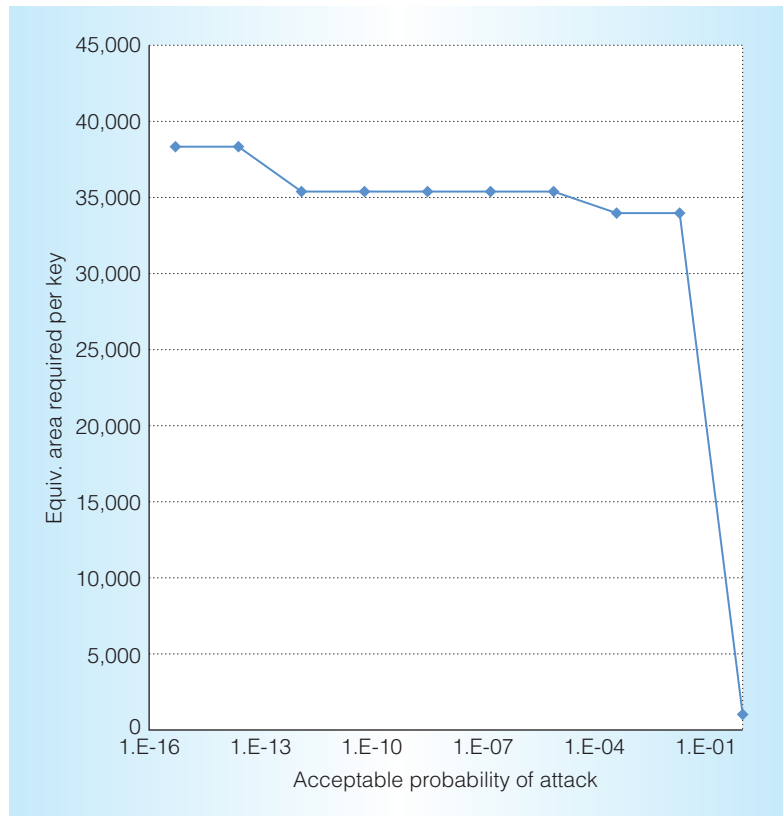


Figure 4. Core size requirement as a function of acceptable attack probability. On the x-axis, we plot the risk probability that one might consider acceptable. On the y-axis is the size of the hardware required (in bits) for the best scheme. When we're willing to accept 100 percent probability of breaking, then clearly very little space is needed. Although there is a large jump as soon as we demand even a 1 in 10 probability of breaking, we can then demand up to 1 in 1,016 (reading the graph from right to left) with only incremental increases in hardware overhead.

potentially lucrative targets for attackers. Attackers are thus often willing to bring highly specialized equipment and training to bear. Although our approach's applicability isn't limited to such extremely high value data, it is certainly one motivating instance.

The minute physical differences in the memory circuitry (caused by wear or by improper or insufficient clearing) and the minute variations used to store the bits themselves are a complex new form of information leakage, which no previous work has addressed. Computer architecture has a long history of being at the forefront of technology, helping to bridge the hardware/software divide. Consider that prior architecture contributions have explored the ability of designs



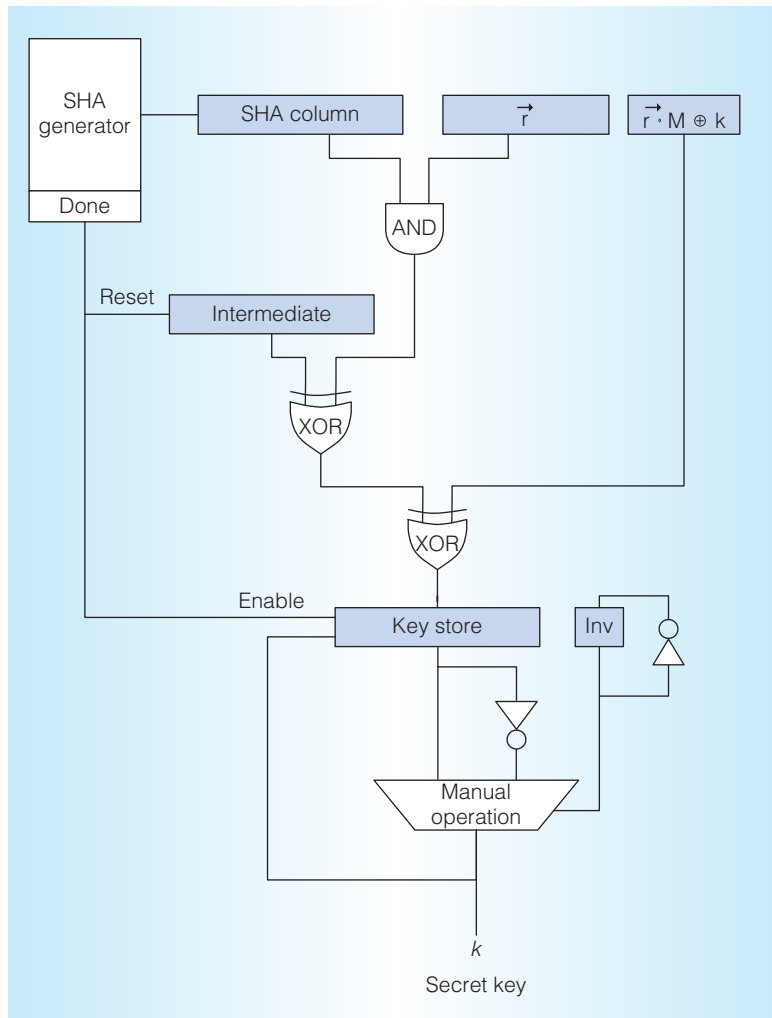


Figure 5. The best performing scheme combines the matrix multiplication operation of the code scheme, but does not store the actual bits of the matrix on chip directly; rather, it generates them dynamically through the use of a cryptographic hash function. Most of the next generation of secure hash algorithm (SHA) implementations require on the order of 20,000 logic gates to implement, which will quickly be amortized over the entirety of the key storage architecture. The key store static RAM (SRAM) is inverted on every cycle to prevent any signs of electromigration that the attacker could use to directly identify bits of the secret key.

to cope with hardware failures, memory hierarchy errors, and even early wear-out. We continue this tradition by evaluating a novel set of hardware methods capable of abstracting away a new class of problems: physical memory inspection. Of course, this isn't the only way in which the bits might leak; they might be exposed through timing, power, software exploits, or electromagnetic radiation variations in addition to these

more intrusive attacks. Our methods should be used as part of a comprehensive strategy to manage these different channels, and although this article concentrates specifically on special inspection-resistant memories, we see both our analysis methods and the memory block as being an important step toward more general-purpose inspection-resistant architectures.

Ultimately, we argue that resistance to physical attacks is a fundamentally new design constraint for computer architects, and examining the tradeoffs in this space requires a truly interdisciplinary approach. There's no question that this article relies on cryptographic techniques that are outside the background of many computer architects, and one might be tempted to ask if this article's contributions are really "computer architecture." However, although the tools are new, the goal is old—to create a new hardware abstraction, encapsulate complexity, and provide a building block for new software and systems to grow around. MICRO

## Acknowledgments

We thank the anonymous reviewers for their insightful comments. This work was funded in part by grants CNS-1239567 and CNS-1162187. The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of the sponsoring agencies.

## References

1. M.K. Qureshi, V. Srinivasan, and J.A. Rivers, "Scalable High Performance Main Memory System Using Phase-Change Memory Technology," *Proc. 36th Ann. Int'l Symp. Computer Architecture (ISCA 09)*, ACM, 2009, pp. 24-33.
2. A. Tiwari and J. Torrellas, "Facelift: Hiding and Slowing Down Aging in Multicores," *Proc. 41st Ann. IEEE/ACM Int'l Symp. Microarchitecture*, ACM, 2008, 129-140.
3. U.R. Karpuzcu, B. Greskamp, and J. Torrellas, "The Bubblewrap Many-Core: Popping Cores for Sequential Acceleration," *Proc. 42nd Ann. IEEE/ACM Int'l Symp. Microarchitecture*, ACM, 2009, pp. 447-458.

4. G.E. Suh et al., "Secure Program Execution via Dynamic Information Flow Tracking," *Proc. 11th Int'l Conf. Architectural Support for Programming Languages and Operating Systems (ASPLOS 04)*, ACM, 2004, pp. 85-96.
5. N. Vachharajani et al., "Rifle: An Architectural Framework for User-Centric Information-Flow Security," *Proc. 37th IEEE/ACM Int'l Symp. Microarchitecture*, IEEE CS, 2004, pp. 243-254.
6. M. Dalton, H. Kannan, and C. Kozyrakis, "Raksha: A Flexible Information Flow Architecture for Software Security," *Proc. 34th Int'l Symp. Computer Architecture (ISCA 07)*, ACM, 2007, pp. 482-493.
7. G. Venkataramani et al., "FlexiTaint: A Programmable Accelerator for Dynamic Taint Propagation," *Proc. 14th Int'l Symp. High Performance Computer Architecture (HPCA 08)*, ACM, 2008, pp. 196-206.
8. O. Ruwase et al., "Parallelizing Dynamic Information Flow Tracking," *Proc. 20th Ann. Symp. Parallelism in Algorithms and Architectures (SPAA 08)*, ACM, pp. 35-45.
9. Z. Wang and R. Lee, "New Cache Designs for Thwarting Cache-Based Side Channel Attacks," *Proc. 34th Int'l Symp. Computer Architecture (ISCA 07)*, ACM, 2007, pp. 494-505.
10. Z. Wang and R. Lee, "A Novel Cache Architecture with Enhanced Performance and Security," *Proc. 41st IEEE/ACM Int'l Symp. Microarchitecture*, IEEE CS, 2008, pp. 83-93.
11. M. Tiwari et al., "Crafting a Usable Microkernel, Processor, and I/O System with Strict and Provable Information Flow Security," *Proc. 38th Int'l Symp. Computer Architecture (ISCA 11)*, IEEE CS, 2011, pp. 189-199.
12. J. Valamehr, "Inspection Resistant Memory: Architectural Support for Security from Physical Examination," *Proc. 39th Ann. Int'l Symp. Computer Architecture (ISCA 12)*, IEEE CS, 2012, pp. 130-141.
13. "Understanding Actel Antifuse Device Security," white paper, Actel, Jan. 2004.
14. T. Wollinger and C. Paar, "New Algorithms, Architectures and Applications for Reconfigurable Computing," *Security Aspects of FPGAs in Cryptographic Applications*, Springer, 2005, pp. 265-278.
15. A. Kolodny et al., "Analysis and Modeling of Floating-Gate Eeprom Cells," *IEEE Trans. Electron Devices*, June 1986, pp. 835-844.
16. S. Haddad et al., "Degradations Due to Hole Trapping in Flash Memory Cells," *IEEE Electron Device Letters*, Mar. 1989, pp. 117-119.
17. M. Shatzkes and Y. Huang, "Characteristic Length and Time in Electromigration," *J. Applied Physics*, Dec. 1993, pp. 6609-6614.

**Jonathan Kaveh Valamehr** is a PhD candidate in the Department of Electrical and Computer Engineering at the University of California, Santa Barbara. His research interests include using hardware and emerging technologies in computer architecture to develop secure microprocessors. Valamehr has an MS in electrical and computer engineering from the University of California, Santa Barbara. He is a student member of IEEE and the ACM.

**Melissa Chase** is a researcher in the Cryptography Group at Microsoft Research, focusing on provably secure privacy. Her research interests include cryptography, including anonymous credentials, electronic cash, attribute-based encryption, and re-encryption, as well as more theoretical topics such as zero knowledge proofs and size-hiding computation. Chase has a PhD in computer science from Brown University.

**Seny Kamara** is a researcher in the Cryptography group at Microsoft Research. His research interests include cloud computing, security, cryptography, and privacy. Kamara has a PhD in computer science from Johns Hopkins University.

**Andrew Putnam** is a senior research hardware development engineer in the eXtreme Computing Group at Microsoft Research. His research interests include field-programmable gate arrays (FPGAs), reconfigurable computing, and computer architecture. Putnam has a PhD in computer science from the University of Washington.

**Daniel Shumow** is a senior software development engineer specializing in cryptography in the eXtreme Computing Group at Microsoft Research. His research interests



include side-channel resistant algorithm design and implementation, and arithmetic geometry applied to cryptography. Shumow has an MS in mathematics from the University of Washington.

**Vinod Vaikuntanathan** is an assistant professor in the Computer Science Department at the University of Toronto. His research interests include cryptography and privacy. Vaikuntanathan has a PhD in computer science from the Massachusetts Institute of Technology.

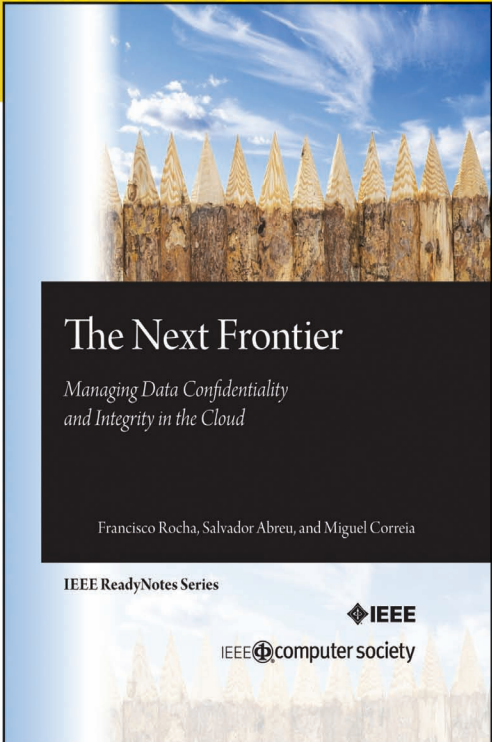
**Timothy Sherwood** is a professor in the Computer Science Department at the University of California, Santa Barbara. His


research interests include high-performance and high-assurance embedded systems. Sherwood has a PhD in computer science and engineering from the University of California, San Diego.

Direct questions and comments about this article to Jonathan Valamehr, University of California, Santa Barbara, Harold Frank Hall, Room 5120A, Santa Barbara, CA 93106-5110; valamehr@ece.ucsb.edu.



*Selected CS articles and columns are also available for free at <http://ComputingNow.computer.org>.*



**NEW** from  **CSPress**

**THE NEXT FRONTIER**  
**Managing Data Confidentiality  
 and Integrity in the Cloud**  
*by Francisco Rocha, Salvador Abreu,  
 and Miguel Correia*

CS authors present the architecture, main mechanisms, and challenges of their proposed defense against malicious insiders in the cloud.

ISBN 978-0-7695-4978-1 • 7" x 10" • 58 pp.

Order .PDF (\$15):  
<http://bit.ly/12Rk6gP>

Order Paperback (\$19):  
<http://bit.ly/166DuZr>