Fairy Dust, Secrets, and the Real World

S.W. SMITH
Dartmouth College

In an invited talk at a recent security conference, a noted member of our field's old guard derided cryptography as "fairy dust" that we sprinkle on our protocols and designs, hoping it magically will make them secure. While I am not sure I share his conclusions in general, I do share some of his cynicism. Too many of us believe breaking RSA is at least as hard as factoring when it's the other way around: breaking RSA is no harder than factoring, (and it might even be easier). Furthermore, when designing security protocols and systems that use these properties, we depend too often on fairy dust: critical properties that we uncritically assume to be unassailable.

One of these critical assumptions is that secrets remain secret. This assumption underlies designs that make basic use of cryptography, such as an e-commerce server armed with an SSL private key, a user's desktop mail client equipped with S/MIME private keys for signed and encrypted email, or even a simple identity card that electronically authorizes its holder to receive some service, such as dormitory or lab access.

We design complex security architectures that begin with the assumption that if an entity uses a particular secret in a certain way, then that entity must be authentic: Only the SSL server www.foo.bar should be using the www.foo.bar's private key in an SSL handshake; only Alice's email client should sign with Alice's private key. But making this assumption hold in the real world requires that the actual device that is the server or the mail client store and use the private key, without revealing it to adversaries.

More esoteric architectures use "trustable" entities in attempts to build secure protocols for problems associated with multiple parties and conflicts of interest. For example, flexible electronic auctions might be made secure by having each party use a private channel to inject a bidding strategy into a trusted mediator, which then plays these strategies against each other to reveal the highest bid. Practical private information retrieval might require that a small trusted component participate in the shuffling.

For such applications, the trusted entity building block starts by assuming a *secure coprocessor* platform—and architectures for secure coprocessors usually start from the assumption that the hardware ensures that critical private keys remain inaccessible to anyone except a well-defined computational entity within an untampered platform. In particular, the machine's owner, with direct access to the hardware, should not be able to get the keys—otherwise, the auctioneer can cheat, and the private information server can spy on the private queries. Indeed, the "trustability" of these trusted third parties critically depends on secrets remaining secret.

Thus, on many levels, it all comes down to keeping secrets. When designing and deploying security architectures and systems, it's easy to think about many other things: protocols, algorithms, key lengths, interoperability with current software, maybe even the user interface. But, at the foundation, something needs to store and use secrets. This storage and usage of secrets needs to be instantiated in the real world, as devices and computational processes. This jump from concept to reality introduces threats that designers can easily overlook. In this article, we look at some of these issues.

Storing secrets

To begin, how do we securely store secrets in a device? Storing them in a general-purpose computing environment is loaded with issues. It is risky to depend on a general-purpose operating system to protect an application's memory, given the historical tendency of OSs—particularly those highly integrated with desktop applications—to provide many avenues of access for malicious code. What does the OS do when it reclaims a page frame, or a disk block? Protection through careful coding also might not work, as CryptLib architect Peter Gutmann (http://online.securityfocus.com/archive/82/297827) recently pointed out, because common development compilers can easily optimize away a careful programmer's attempt to clear vulnerable memory. Further, should adversaries have the opportunity to wander through memory, they can clearly distinguish cryptographic keys

Opening move

In the Threats Perspective department, we want to overcome the temptation to think about security in terms of a narrow set of bad actions against a narrow set of machines. Instead, we'll look at security issues that challenge practitioners working in specialized application domains and technology areas. In particular, we want to expose readers to threat perspectives that may not be readily apparent from typical points of view. Please send contributions and suggestions to me at sws@cs.dartmouth.edu.

–S.W. Smith

S.W. Smith is a Security & Privacy *task force member.* *See page 8.*

as byte strings with significantly higher entropy. Adi Shamir and Nicko van Someren pointed this out in a paper presented at the Financial Cryptography Conference in 1999.

Alternatively, instead of a general-purpose machine, we should use a specialized device to hide our secrets. In the 1990s, we witnessed the merging of several such device families: independent *personal tokens* that users (or other mobile entities) can carry with them; *cryptographic accelerators* to improve performance by offloading cryptographic computation from a host; and *secure coprocessors* to improve security by offloading sensitive computation from a host. My own exposure to the need for devices that actually keep secrets at a high level of assurance came from many years of designing applications and architectures for secure coprocessors. (For a thorough overview of architecture and applications of secure coprocessing, see www.research.ibm.com/secure_systems/scop.htm).

Unfortunately, boundaries among specialized devices blurred because of overlapping requirements: personal tokens needed to be secure against thieves, or perhaps the users themselves; coprocessors and tokens ended up requiring cryptography; and accelerators needed physical security and programmability. (Of course, putting a general-purpose computational environment within an armored box reintroduces the general-purpose issues already discussed. As Gutmann would say, "lather, rinse, repeat.")

Besides, if we're hiding secrets in physical devices, how easy is it to just open them up and take a look? Way back in 1996, Ross Anderson and Markus Kuhn[1] showed how easy it was to explore single-chip devices such as smart cards. Exploitable weaknesses included UV radiation to return chips to a "factory mode" where secrets are accessible; fuming nitric acid to remove potting compounds but not the electronics; microprobes and laser cutters; and focused ion beams to modify circuits.

Steve Weingart,[2] a longtime hardware security architect and my occasional collaborator at IBM, followed up with a longer laundry list of attacks from his own experience. These extend to larger devices with more elaborate physical protections that try to zeroize sensitive memory before adversaries can reach it.

In addition to various probing approaches, careful machining (including using tools as mundane as sand, water, and hands) can be quite effective in removing barriers without triggering zeroization. Shaped-charge explosives can create *plasma lances* to separate sensitive memory from its destruction circuitry before the latter can do its work.

Even if the physical protections work, environmental factors such as extremely cold temperatures and radiation can cause SRAM to imprint and safely retain its secrets for an adversary to inspect after an attack. Even long-term storage of the same bits in SRAM can cause such imprinting.

Anderson and Kuhn continue with their work; a recent result involves using flash bulbs and a microscope to probe smart card electronically erasable programmable read-only memory.[3] Other ongoing efforts in this spirit include penetrating physical protections in the XBox gaming device (www.xenatera.com/bunnie/proj/anatak/xboxmod.html). Weingart's work, however, focused primarily on defense (as we will discuss shortly). These researchers remain active in the area.

Using secrets

Even if we could hide a secret effectively, our device must occasionally perform operations with it. Using a secret is not a clean, abstract process; it must occur in real physical and computational environments. An adversary can observe and manipulate these environments with surprising effectiveness.

For a classic example of this approach, let's look back to password checking in the Tenex operating system, an early 1970s timesharing system for the PDP-10. At first glance, the number of attempts necessary to guess a secret password is exponential to the password's length. Tenex made it much easier to gain access because it checked a guess one character at time, and stopped at the first mismatch. By lining up a guess across the boundary between a resident page and a nonresident page and observing whether a page fault occurred when the system checked the guess, an adversary could verify whether a

specific prefix of a guess was correct.

The fact that the secret password comparison occurred on a real machine led to an observable property that turned an intractable exponential search into a feasible linear one.

If we fast forward to 1995, the same basic problem—an observable artifact lets an adversary verify the prefix of a guess—emerged for more abstract cryptographic devices.[4] Let's consider modular exponentiation with a secret exponent, the core of the RSA cryptosystem.

The time that the modular operation takes depends on the exponent and the operand, and is well understood. Suppose an adversary guessed a secret exponent, calculated the time for that guess on an operand, and then actually measured that time. If only the first k bits of the guess were correct, then the adversary's model would be correct for the first k bits, but wrong for the remainder.

However, over enough samples, the difference between predicted and real times would form a distribution with variance proportional to n-k, so the adversary could confirm the correctness of a guessed k-bit prefix of the secret by calculating the variance. With enough samples, this artifact of the physical implementation of RSA (and other cryptosystems) turns an infeasible exponential search into a feasible linear one. Instantiating the cryptography in the real world leads to threats that do not always show up on a programmer's white board.

Paul Kocher's timing attack triggered an avalanche of *side-channel analysis* (www.research.ibm.com/intsec) in the open world, by Kocher and others.

In addition to the time-of-operation approach, physical devices have other observable physical characteristics that depend on hidden secrets. One natural characteristic is power. When complementary metal-oxide semiconductors switch, they consume power; an adversary could measure this consumption and attempt to deduce things about the operation. With *simple power analysis*, an adversary tries to draw conclusions from a simple power trace. SPA can be quite effective; a co-worker of mine managed to extract a DES key from a commercial device with a single power trace—an initial parity check led to a mix of 56 spikes, some short, some tall, one for each bit. More advanced *differential power analysis* looks at more subtle statistical correlations between the secret bits and power consumption.

Other types of observable physical properties exploited by researchers in the last few years to reveal hidden secrets include EMF radiation and even the changes in room light from CRT displays. (In the classified world, the use of such emanations from the real-world instantiations of algorithms has fallen under the Tempest program, some of which has become declassified.)

In attacks described earlier, an adversary learns information by observing physical properties of the device doing the computation. However, an adversary can sometimes learn more by carefully inducing errors in the computation.

In their physical attacks on smart cards, Anderson and Kuhn observed that carefully timed voltage spikes on a processor could disrupt its current instruction. Applying these spikes at critical points during an operation on a secret—such as when comparing the number of DES iterations completed against the number desired—can transform a designer's intended cryptographic operation into one much more amenable to cryptanalysis. (This provided a practical demonstration of speculative work on such *differential fault analysis*.)

In the last two years, researchers have focused attention on the API level of these devices.[5,6] Besides physical properties, instantiation of abstract ideas in the real world also can lead to feature creep. As Mike Bond paraphrases Needham, clean abstract designs tend to become "Swiss Army knives." In particular, cryptographic accelerators have found major commercial application in banking networks: for ATM and credit card processing, devices need to transmit, encode, and verify PINs. However, the accumulation of usage scenarios leads to a transaction set complexity that permits many clever ways to piece together program calls that disclose sensitive PINs and keys. Jolyon Clulow, in particular, discusses many amusing attacks possible from exploiting error behavior resulting from devious modifications of legitimate transaction requests.

Traditional defenses

How do we build devices that actually retain their secrets? Weingart[2] gives a good overview of the multiple defense components:

- tamper evidence—ensuring that tamper causes some observable consequence
- tamper resistance—making it hard to tamper with the device
- tamper detection—having the device able to sense when tamper is occurring
- tamper response—having the de-

In their physical attacks on smart cards, Anderson and Kuhn observed that carefully timed voltage spikes on a processor could disrupt its current instruction.

vice take some appropriate countermeasure

Weingart also presents the operating envelope concept he evolved in his earlier work. If certain aspects of the defense mechanism require certain environmental properties (such as voltage and temperature) to function, then the device should treat departures from that envelope as a tamper event.

Our team put these principles into practice. Our coprocessor architecture paper[7] provides a more detailed discussion of a practical defense instantiation that our team designed and built at IBM. We dispensed with tamper evidence, because tamper evidence is worthless without an audit trail; our usage model did not guarantee one. Instead, we focused on resistance (the device was housed in two metal cans filled with resin); detection (the resin contained a conductive mesh, chemically and physically similar to the resin); and response (changes to the mesh—or other detected tamper events—triggered direct zeroization of the sensitive SRAM).

To enforce the operating envelope, we detected anomalies in voltage, temperature, and radiation from the moment the device left the factory. We also ensured that sensitive memory was regularly, randomly inverted to avoid imprinting. The large form factor created Faraday cages to protect against electromagnetic radiation, and power circuitry sophisticated enough to protect against SPA and DPA.

To date, we know of no successful attack on the basic coprocessor platform or on the security configuration software that controls what the box does. We make such claims hesitantly, however, because we cannot prove the absence of flaws, just the absence of successful penetrations up to some point in time.

Starting with protection of secrets, we then made design choices in a product's lifecycle, software configuration, and other hardware protection that actually yields a secure coprocessor platform suitable for some trusted third-party applications.

Unfortunately, this was a somewhat Pyrrhic victory—the primary commercial use of our box was to house cryptographic accelerator software vulnerable to the API-level attacks discussed earlier.

New directions

Secure interaction in the distributed, heterogeneous, mutually suspicious environment that is today's Internet appears to require that we have trustable places to keep and wield secrets. Security designs start out by assuming these places exist. Our quick look at history suggests that it can be risky to assume that current computing technology can live up to this task. So what do we do? How do we keep secrets in the real world?

Looking at the weaknesses of traditional computing technologies, some researchers are exploring fundamentally different approaches. As an example, effectively hiding information in storage bits or circuit structure of a consumer-level chip has been difficult—an adversary always seems to find a way to peel off protections and probe. In response, researchers at MIT have proposed—and prototyped—*silicon physical unknown functions*[8] that use the random *delays* in circuit elements as a secret. In theory, these delays are byproducts of the manufacturing process, and the only way to measure the delay is to use the circuit. If these hypotheses hold up, this technology could enable some interesting applications. Many researchers (including myself) are looking into this.

Other researchers have been moving away from silicon circuits to other physical forms of storage and computation. Ravi Pappu and his colleagues have built *physical one-way functions*[9] from optical epoxy tokens containing tiny glass spheres. The "secret" is how this exact arrangement of spheres scatters laser light sent in from different angles. The researchers believe that determining the arrangement might be possible, but cloning a token, even from this information, is intractable.

Another novel direction is the use of *microelectromechanical systems* for infosecurity (www.cs.dartmouth.edu/~brd/Research/MEMS/ISTS/memms.html). MEMS are physical devices—levers, gears, springs—that are *small*: feature size less than 1 micron, and total size typically between 10 and 100 microns. Such small physical devices might not be as susceptible to circuit-based side-channels such as EMF and power analysis, and (for the foreseeable future) fabrication cost places a very high threshold for adversaries.

The preceding new approaches start to blur our original problem. Do we want to hide a secret we choose? Is it sufficient to hide a random secret that we cannot choose? Is it sufficient to have a device with a unique property that cannot be physically duplicated?

This blurring indicates another new line of thinking: If technology cannot support our assumption of a trustable place, perhaps we can make do with a weaker assumption. In this direction, researchers have considered alternate trust models, and protocols to support these models.

On a high design level, examples include *general multi-party computation*, *encrypted functions*, and *threshold cryptography*, which transform a sensitive computation into a larger computation among many parties, but which is more resilient to various trust compromises. Harnessing such transformation in practical ways is an area of ongoing research. Some examples include distributed systems, mobile agent security, and ad-hoc PKI. Another new trust model is *white-box cryptography* (see www.scs.carleton.ca/~paulv for some representative papers): encoding computation in such a way that it can hide its secret even if runs on an adversary's computer.

Other recent work tries to bridge

these theoretical results with more practical systems views by reducing the role (and required trust) of the trustable place. Some examples include recent work in practical private information retrieval and mobile code security (where a trusted third party is involved only in a small part of the operation), and the commodity model of server-assisted cryptography and privacy protocols (where only partially trusted third parties participate, and do so before the protocol starts).

To some extent, recent commercial efforts to incorporate some hardware security into common desktops (the multi-vendor Trusted Computing Platform Alliance, and also Microsoft's Palladium initiative) fall into this category.[10] Of course, given the history of sound ideas proving leaky when they become real devices in the real world, several aspects of these notes could have relevance for these new efforts.

Computation must exist in the physical world. Security designs that require secrets must hide and use them in the real world. Unfortunately, the real world offers more paths to secret storage and more observable computational artifacts than these security designs anticipate. Careful integration of physical defenses and security architecture can sometimes succeed against the adversary class designers consider. However, in the long term, we hope for either a quantum leap in physically defensible technology—or a significant reduction in the properties that designs force us to assume about our computers. It would be nice to have fairy dust that works. □

References

1. R. Anderson and M. Kuhn, "Tamper Resistance—A Cautionary Note," *Proc. 2nd Usenix Workshop on Electronic Commerce*, Usenix Assoc., 1996, p. 1–11.
2. S. Weingart, "Physical Security Devices for Computer Subsystems: A Survey of Attacks and Defenses," C. K. Koc and C. Paar, eds., *Proc. 2nd Workshop on Cryptographic Hardware and Embedded Systems (CHES 00)*, Springer-Verlag LNCS 1965, 2000, pp. 302–317.
3. S. Skorobogatov and R. Anderson, "Optical Fault Induction Attacks," to be published in *Proc. 4th Workshop on Cryptographic Hardware and Embedded Systems (CHES 02)*, 2002.
4. P. Kocher, "Timing Attacks on Implementations of Diffie-Hellman, RSA, DSS, and Other Systems," N. Koblitz, ed., *Advances in Cryptology (Crypto 96)*, Springer-Verlag LNCS 1109, pp. 104–113.
5. M. Bond and R. Anderson, "API-Level Attacks on Embedded Systems," *Computer*, vol. 29, October, 2001, pp. 67–75.
6. J. Clulow, "I Know Your PIN," RSA Conference 2002, Europe, 2002.
7. S.W. Smith and S. Weingart, "Building a High-Performance, Programmable Secure Coprocessor," *Computer Networks*, vol. 31, April, 1999, pp. 831–860
8. B. Gassend et al., "Silicon Physical Random Functions," *Proc. 9th ACM Conference on Computer and Communications Security*, ACM Press, 2002, pp. 148–160.
9. R. Pappu et al, "Physical One-Way Functions," *Science*, vol. 297, Sept., 2002, pp. 2026–2030.
10. P. England and M. Peinado, "Authenticated Operation of Open Computing Devices," L. Baten and J. Seberry eds., *Proc. 7th Australasian Conf. Information Security and Privacy (ACISP 2002)*, Springer-Verlag LNCS 2384, June, 2002, pp. 346–361.

***S.W. Smith** is currently an assistant professor of computer science at Dartmouth College. Previously, he was a research staff member at IBM Watson, working on secure coprocessor design and validation, and a staff member at Los Alamos National Laboratory, doing security reviews and designs for public-sector clients. He received his BA in mathematics from Princeton and his MSc and PhD in computer science from Carnegie Mellon University. He is a member of ACM, Usenix, the IEEE Computer Society, Phi Beta Kappa, and Sigma Xi. Contact him at sws@cs.dartmouth.edu or through his home page, www.cs.dartmouth.edu/~sws/.*