Editors: Peter Gutmann, pgut001@cs.auckland.ac.nz
David Naccache, david.naccache@ens.fr
Charles C. Palmer, ccpalmer@us.ibm.com

Finding Faults

David Naccache
Département d'Informatique, École Normale Supérieure, Paris

Imagine that a fictional foreign embassy on US soil imports various objects via diplomatic pouch. Intelligence sources indicate that each incoming pouch contains either harmless archive boxes (3 Kg each) or forbidden bomb detonators (5 Kg each). Because officials can't legally open the pouch (and it's opaque to x-ray technology), the FBI decides to discreetly intercept one and weigh it. If the pouch weighs 5 Kg, federal agents will require embassy officials to open it in front of unbiased observers; should the pouch weigh 3 Kg, nothing will happen.

Unfortunately, the pouch weighs 15 Kg, which means it could contain five archive boxes ($5 \times 3 = 15$) or three detonators ($3 \times 5 = 15$). Assuming the embassy has a 50 percent probability of being involved in terrorist activity, information theorists would say the pouch features one (binary) bit of entropy (uncertainty)—namely, it can be in one of two states with equal probability. Is there a diplomatic way of inferring the pouch's contents without making potentially unfounded accusations or running the risk of allowing bomb parts into the country? Actually, yes: FBI agents could simply hit the pouch strongly enough to partially damage its contents, then sit back and see what happens. If the embassy subsequently receives a replacement pouch weighing 3, 6, 9, or 12 Kg, it means the original pouch contained archive boxes. If, however, the replacement pouch weighs 5 or 10 Kg, there is reason to worry.

This scenario describes how the injection of an error can, in fact, cause the leakage of confidential information. Often, the only way to attack strong cryptographic implementations is to attack the infrastructure upon which they are built. This infrastructure is most often the underlying operating system or middleware, but attacks can also be mounted directly against the hardware upon which the cryptographic implementation is being run. This issue's Crypto Corner describes some of the methods used to induce faults in systems and explains how such faults can be exploited to reveal secret information.

Fault history

In the 1970s, chip manufacturers noticed that radioactive particles produced by elements naturally present in packaging material introduced faults in computer chips. Specifically, when the Uranium-235, Uranium-238, and Thorium-230 residues present in the packaging decay to Lead-206, they release α particles. These particles create charges in sensitive chip areas, causing bits to flip. Although these elements were only present in two or three parts per million, this concentration was sufficient to affect overall chip behavior.

Subsequent research by organizations such as NASA and Boeing simulated the effects of cosmic rays on semiconductors; these rays are very weak at ground level, due to the Earth's atmosphere, but their effect becomes more pronounced in the upper atmosphere and outer space. This problem is further compounded by the fact that the more RAM a computer has, the higher the chance a fault will occur. As a result, the aerospace industry devotes considerable engineering efforts to "hardening" electronic devices designed to operate in harsh environments. To get a fuller picture, researchers typically use simulators to model the circuits and study the effects of randomly induced faults.

In 2001, Dan Boneh, Richard DeMillo, and Richard Lipton published in the *Journal of Cryptology* the first attack that used faults to derive secret information.[1] This paper didn't report any practical experiments and targeted the RSA public-key cryptosystem. The fault revealed the two secret prime factors that constitute the RSA modulus. This paper was followed by theoretical attacks on other cryptographic algorithms, but as practical experience would later reveal, fault injection proves much more efficient when targeting assembler *instructions* instead of algorithms.

Deliberate electrical glitches

In a glitch attack, an attacker deliberately generates a malfunction that causes one or more flip-flops to transition into a wrong state. The aim is usually to replace a sin-

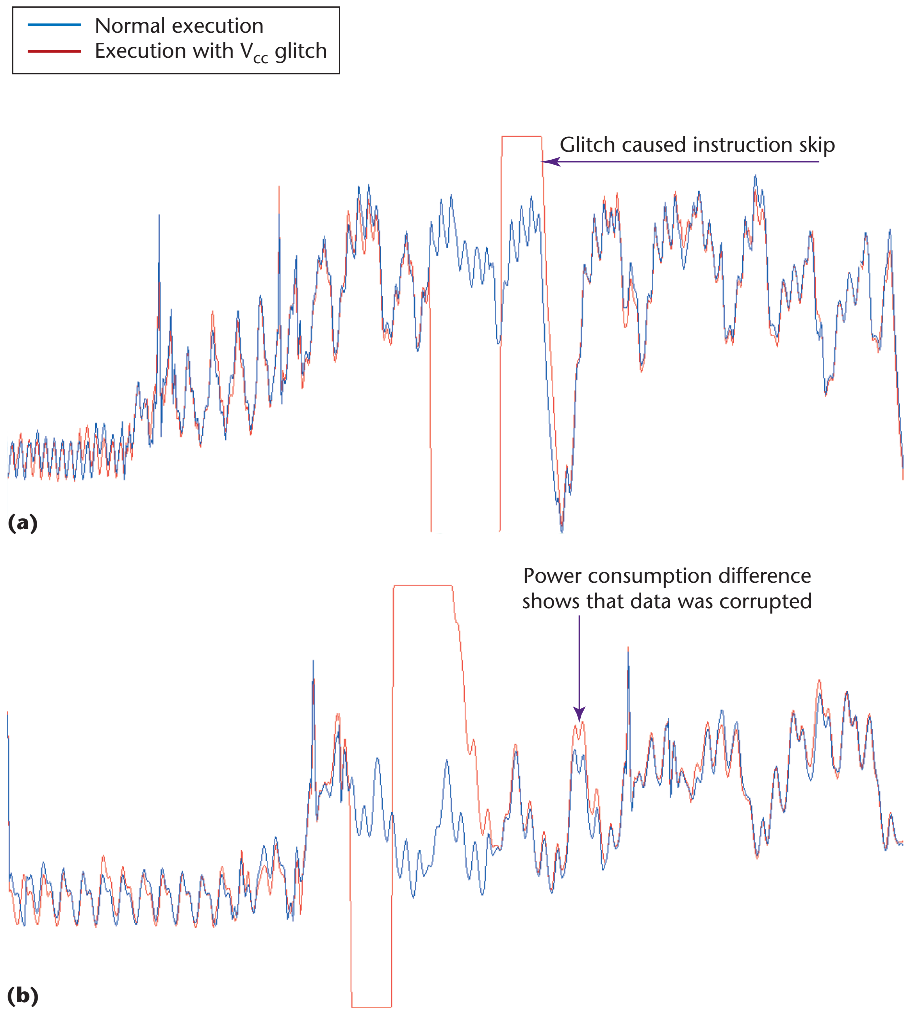

Figure 1. Glitch effects. By playing with the glitch's parameters, we see two types of behavior: (a) skipping an instruction and (b) skipping an instruction and corrupting data.

gle critical machine instruction with an arbitrary one. Glitch attacks can also attempt to corrupt data values as the data moves between registers and memory. Three main glitch techniques can create fairly reliable malfunctions that affect a small number of machine cycles in embedded processors: clock-signal transients, power-supply transients, and external electrical field transients.

Particularly interesting instructions that an attacker might want to glitch include conditional instructions or the test instructions that precede them. Conditional instructions create a window of vulnerability in many security applications' processing stages that lets attackers bypass sophisticated cryptographic barriers by simply preventing the execution of the code that detects unsuccessful authentication attempts. Instruction glitches can extend loop runtimes—for instance, in serial port output routines—to make visible more of the memory after output buffer. They can also reduce loop runtimes, thereby transforming an iterated block-cipher into an easy-to-break single-round variant.

Clock-signal glitches are currently the simplest and most practical glitch attack. They temporarily increase the clock frequency for one or more half-cycles, so that some flip-flops in faster-reacting portions of the circuit will respond in time, whereas others won't, leaving their state unchanged. An attacker uses power analysis first to monitor how far a program has progressed and then to launch a fault as soon as it recognizes a specific instruction's power consumption curve profile. The attacker can use this profile to determine when the microprocessor, for example, is about to execute a branch instruction. A more rapid clock cycle at this point (a clock-signal glitch) gives the processor insufficient time to write the jump address to the program counter, thereby annulling the branch operation.

Because of the different number of gate delays in various signal paths and the varying parameters of the chip's circuits, the attacker's clock glitch affects only some signals. By varying the glitch's timing and duration, the attacker can fool some CPUs into executing several completely different—and wrong—instructions. These vary from one instance of the chip to another, but can generally be found via a systematic search using specialized hardware.

The technical reason why a glitch causes a malfunction is that the program counter logic in a CPU is generally much simpler than the instruction unit (IU): by providing two clock pulses in the space of one, it's possible to increase the PC logic. The IU logic doesn't have time to react, thus it skips an instruction. An attacker can typically use this skip to break out of a loop that checks a password or a PIN.

Figure 1 illustrates the different effects glitches can have. Here, the attacker causes a power drop from V_{cc} to 0 V over a few nanoseconds. By carefully playing with the glitch's parameters (duration,

How to avoid faults

We can employ a variety of mechanisms to try and counter fault attacks. The chip's programmer, for example, can implement checksums and critical variable duplications in the software itself. If the order in which operations in an algorithm are executed is randomized, it's difficult to predict what the machine does at any given cycle. Further increasing this difficulty, execution redundancy repeats algorithms and compares the results to verify that the correct result is generated. Baits are the small (< 10-byte) code fragments that perform an operation and test its result—typically write, read, and compare data—and perform XORs, additions, and other operations whose results can be easily checked. When a bait detects an error, it increments a non-volatile memory (NVM) counter and resets the chip; when this NVM counter exceeds a tolerance limit (usually three), the device under attack definitely ceases to function.

Chip manufacturers implement hardware protections, which can be subdivided into two categories: active and passive. In active protections, light detectors detect changes in light gradient, and supply voltage detectors react to abrupt variations in the applied potential, continuously ascertaining that the voltage is within the circuit's tolerance thresholds. Frequency detectors impose an interval of operation outside which the electronic circuit will reset itself.

Active shields are metal meshes that cover the entire chip and have data passing continuously through them. If this mesh is disconnected or modified, the chip won't operate anymore. Although this is primarily a countermeasure against probing, it also helps protect against faults by making it harder to find the location of specific blocks in a circuit. Hardware redundancy, on the other hand, is a duplication of hardware blocks followed by a test and a comparator. When the two blocks' results don't match, the comparator circuitry sends an alert signal to a decision block. The device can then implement two types of reaction: a hardware reset or the activation of an interruption that triggers dedicated countermeasures.

Passive protections involve things like running dummy random cycles during code processing, which makes fault synchronization harder. Bus and memory encryption scramble the bus and RAM contents, which makes targeting a specific memory cell useless because successive computations with identical data use different memory cells. Passive shields cover sensitive chip parts and make it harder to identify them. Finally, unstable internal frequency generators protect against faults that must be synchronized with certain events because the clock's duty cycle changes in time.

falling edge, amplitude, and so on), two types of behavior emerge. Under the first set of conditions (Figure 1a), the processor skips several instructions and resumes normal execution several microseconds after the glitch. This fault allows the selective execution of instructions in a program. Under a second set of conditions (Figure 1b), not only does the processor skip the instructions, but it can also precisely manipulate the data value. This is visually reflected in Figure 1b's power curves.

Fault attacks on Eprom

Erasable programmable read-only memory (Eprom) stores information by entrapping electrical charges in a gate's insulator. In essence, during the information-writing process, charges are forced into the insulator using a phenomenon called Fowler-Nordheim tunneling. During programming, charges are forced until the insulator's static voltage exceeds a reference voltage, calculated as a fraction of V_{cc} (say, $V_{cc}/2$). The presence or absence of charges in the insulator is interpreted as information. To read information from a cell, an external comparator circuit compares the cell's static voltage to a reference-detection voltage, V_{det} (usually $V_{det} = V_{cc}/2$).

Consequently, if we program under the lowest tolerable voltage, we force fewer particles into the cell. Then, if during reading, V_{cc} is increased to the highest value tolerated by the circuit, V_{det} is artificially boosted, causing data to be interpreted as zero regardless of its actual value. To attack an n byte key, an attacker can simply subject the circuit to $n - 1$ power glitches to get the encryption of a known plaintext under a vulnerable key:

00 00 … 00 00 XX 00 00 … 00 00

The attacker will then exhaust the value XX and move the glitch's position to successively scan the entire key. This attack was implemented in the late 1990s.

Similarly, a laser impact on a specific chip's bus during information transfer has the effect of reading the value 255 (0xFF), regardless of the transferred information's actual value.

Faults and Java

The Java sandbox is an environment in which applets run without direct access to the computer's resources, the idea being that an applet need not be trusted because it's incapable of running malicious code. Java programs typically appear on the Internet, where a PC downloads and executes an applet to achieve a given effect on a Web page.

A couple of years ago, Sudhakar Govindavajhala and Andrew Appel reported fault attacks on PCs that forced Java Virtual Machines to execute arbitrary code.[2] They showed that an attacker could use a light bulb to heat up the PC's RAM to the point at which a fault occurs (in this case, a bit flip). The expected fault was that the address of a function a called by the applet would have one bit changed, so that the address called was $a \pm 2^i$,

How to inject faults

Just as we have many options for avoiding faults, we also have several mechanisms for injecting them. Variations in supply voltage during execution, for example, can cause a processor to misinterpret or skip instructions. The smart-card industry researches and practices this method behind closed doors, but it doesn't often appear in the open literature. Variations in the external clock can also cause data misread (the circuit tries to read a value from the data bus before the memory has time to return the asked value) or an instruction miss (the circuit starts executing instruction $n + 1$ before the microprocessor finishes executing instruction n).

Circuit manufacturers define upper and lower temperature thresholds within which their circuits will function correctly. The attacker's goal here is to vary temperature by using an alcoholic cooler until the chip exceeds the threshold's bounds. This creates two effects: the random modification of RAM cells due to heating and the exploitation of the fact that read and write temperature thresholds don't coincide in most non-volatile memories (NVMs). By setting the chip's temperature to values at which write works but read doesn't (or vice versa), several attacks can be mounted.

All electric circuits are sensitive to light because of photo-electric effects, thus photon-induced currents can introduce faults if a circuit is exposed to intense light for a brief time period. This type of fault induction is inexpensive, too. Figure A shows a light fault injector.

Similarly, lasers can simulate the faults induced by particle accelerators. The effect produced is comparable to white light, but a laser's advantage is directionality, which lets it target a small circuit area precisely (see Figure B).

X-rays and ion beams can also serve as fault sources (although they're less common). They have the advantage of allowing fault attacks, but without depackaging the chip.

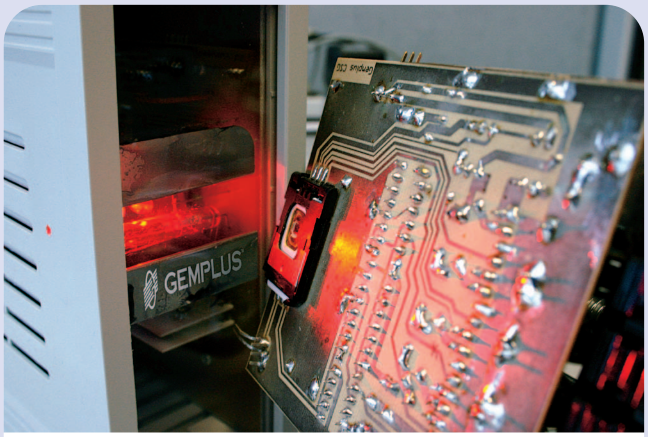

Figure A. Light fault injector based on a camera-flash-like lamp.

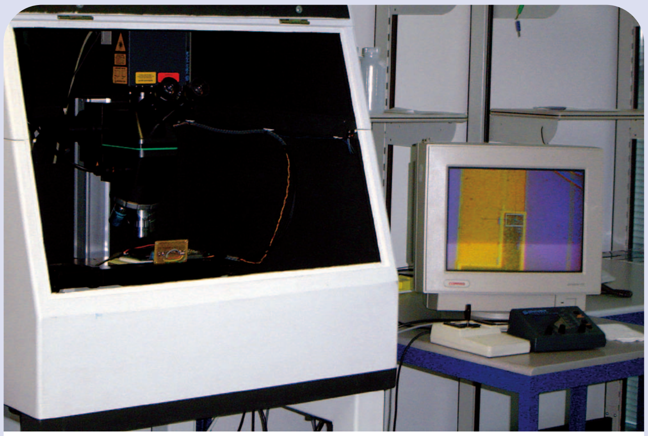

Figure B. Laser fault setup. Note the joystick and the control screen, allowing the user to target a precise chip area.

where $0 \leq i \leq 31$ (the computer's word size). The attacker arranges to have a function present at this address that will return a variable of a type not expected by the calling function—causing for instance, an integer to be interpreted as a pointer. The attacker can then use this fault to read/write to arbitrary addresses in the computer's memory. One of the possible uses of such a fault would be to change fields in the Java runtime system's security manager to grant the applet illegal rights.

Alongside hardware-based attacks, pure message format errors can also reveal secret keys. A particularly instructive example is an attack reported back in 1998 on the encryption standard PKCS#1 (PKCS has, of course, since been updated).[3] Daniel Bleichenbacher, a researcher at Bell Labs, discovered that he could decrypt PKCS#1 ciphertexts by sending erroneously crafted messages to a targeted server and analyzing the returned error codes.

We can also classify buffer overflows as fault attacks. A buffer overflow occurs when a program tries to store (typically, maliciously crafted) long data in a short buffer. The overflowing data overwrites the computer's executable code and starts being executed instead. The continuous stream of updated defenses and correspondingly updated attacks that have appeared since the late 1990s illustrate just how hard it is to defend against this type of problem.

As this article goes to press, cryptographers and hardware specialists are gathering for an international workshop in Edinburgh, Scotland. The conference, entitled "Fault Diagnosis and Tolerance in Cryptography" (FDTC 2005; www.elet.polimi.it/conferences/FDTC05/), is entirely devoted to the security implications of faults in embedded electronic devices. This area promises to provide fruitful employment for cryptographers for some years to come. ▢

References

1. D. Boneh, R.A. DeMillo, and R.J. Lipton, "On the Importance of Eliminating Errors in Cryptographic Computations," *J. Cryptology*, vol. 14, no. 2, 2001, pp. 101–119.
2. S. Govindavajhala and A.W. Appel, "Using Memory Errors to Attack a Virtual Machine," *IEEE Symp. Security and Privacy*, IEEE CS Press, 2003, pp. 154–165.
3. D. Bleichenbacher, "Chosen Ciphertext Attacks Against Protocols Based on the RSA Encryption Standard PKCS #1," *Advances in Cryptology*, LNCS 1462, Springer-Verlag, 1998, pp. 1–12.

David Naccache *is a researcher at the École Normale Supérieure's Complexity and Cryptography Group and a professor at the University of Paris II. His research interests include public-key cryptography and mobile code security. Naccache has a PhD in cryptology from the École Nationale Supérieure des Télécommunications Paris. He is a member of IACR, the ACM, and the IEEE. Contact him at david.naccache@ens.fr.*

ADVERTISER / PRODUCT INDEX SEPTEMBER/OCTOBER 2005

Advertiser	Page Number
Addison Wesley Professional	**5**
Hindawi Publishing Corp.	**13**
Infosecurity 2005	**3**
LISA 2005	**Cover 3**
Morgan Kaufmann	**9**
Retail Security Forum 2005	**Cover 2**
U.S Navy Reserve	**Cover 4**

Boldface denotes advertisements in this issue.

Advertising Personnel

Marion Delaney
IEEE Media, Advertising Director
Phone: +1 212 419 7766
Fax: +1 212 419 7589
Email: md.ieeemedia@ieee.org

Marian Anderson
Advertising Coordinator
Phone: +1 714 821 8380
Fax: +1 714 821 4010
Email: manderson@computer.org

Sandy Brown
IEEE Computer Society,
Business Development Manager
Phone: +1 714 821 8380
Fax: +1 714 821 4010
Email: sb.ieeemedia@ieee.org

Advertising Sales Representatives

Mid Atlantic (product/recruitment)
Dawn Becker
Phone: +1 732 772 0160
Fax: +1 732 772 0161
Email: db.ieeemedia@ieee.org

New England (product)
Jody Estabrook
Phone: +1 978 244 0192
Fax: +1 978 244 0103
Email: je.ieeemedia@ieee.org

New England (recruitment)
Robert Zwick
Phone: +1 212 419 7765
Fax: +1 212 419 7570
Email: r.zwick@ieee.org

Connecticut (product)
Stan Greenfield
Phone: +1 203 938 2418
Fax: +1 203 938 3211
Email: greenco@optonline.net

Midwest (product)
Dave Jones
Phone: +1 708 442 5633
Fax: +1 708 442 7620
Email: dj.ieeemedia@ieee.org
Will Hamilton
Phone: +1 269 381 2156
Fax: +1 269 381 2556
Email: wh.ieeemedia@ieee.org
Joe DiNardo
Phone: +1 440 248 2456
Fax: +1 440 248 2594
Email: jd.ieeemedia@ieee.org

Southeast (recruitment)
Thomas M. Flynn
Phone: +1 770 645 2944
Fax: +1 770 993 4423
Email: flynntom@mindspring.com

Southeast (product)
Bill Holland
Phone: +1 770 435 6549
Fax: +1 770 435 0243
Email: hollandwfh@yahoo.com

Midwest/Southwest (recruitment)
Darcy Giovingo
Phone: +1 847 498-4520
Fax: +1 847 498-5911
Email: dg.ieeemedia@ieee.org

Southwest (product)
Josh Mayer
Phone: +1 972 423 5507
Fax: +1 972 423 6858
Email: jm.ieeemedia@ieee.org

Northwest (product)
Peter D. Scott
Phone: +1 415 421-7950
Fax: +1 415 398-4156
Email: peterd@pscottassoc.com

Southern CA (product)
Marshall Rubin
Phone: +1 818 888 2407
Fax: +1 818 888 4907
Email: mr.ieeemedia@ieee.org

Northwest/Southern CA (recruitment)
Tim Matteson
Phone: +1 310 836 4064
Fax: +1 310 836 4067
Email: tm.ieeemedia@ieee.org

Japan
Tim Matteson
Phone: +1 310 836 4064
Fax: +1 310 836 4067
Email: tm.ieeemedia@ieee.org

Europe (product)
Hilary Turnbull
Phone: +44 1875 825700
Fax: +44 1875 825701
Email: impress@impressmedia.com