

6.857 Lecture 13: Physical Attacks (aka “Losing Secrets”)

October 31, 2005

Readings:

- “Soft Tempest: Hidden Data Transmission Using Electromagnetic Emanations,” by Ross Anderson and Markus Kuhn
- “Optical Time-Domain Eavesdropping Risks of CRT Displays,” by Markus Kuhn
- Optical Emission Security FAQ, by Markus Kuhn:
<http://www.cl.cam.ac.uk/~mgk25/emsec/optical-faq.html>
- “Low Cost Attacks on Tamper Resistant Devices,” by Ross Anderson and Markus Kuhn
- “Timing Attacks on Implementations of Diffie-Hellman, RSA, DSS, and Other Systems,” by Paul Kocher
- “Protecting Smart Cards from Passive Power Analysis with Detached Power Supplies,” by Adi Shamir

In this lecture:

- Hard disk storage
- Tempest eavesdropping
- Smartcards and other tamper-resistant devices:
 - (passive) power analysis — simple and differential
 - (active) power glitches
 - timing analysis
 - tamper (non)resistance

Today we'll be looking at security in the physical, material "real world." In contrast with the many of the very important *natural* physical security risks (e.g., fire, flood, power interruption, storms, earthquakes, war, ...), we will be examining risks due to *malicious attackers*. These risks are typically less obvious and more devastating than their natural counterparts.

So far, all the cryptography we've seen has assumed that the machine storing the secret keys and performing the cryptographic calculations is a perfect "black-box:" input comes in (i.e., a message to be encrypted), and output comes out (i.e., the ciphertext). The attacker has no access to the internals of the machine and cannot observe it operating. Today we'll see just how far from this ideal "black-box" model the real world is — there are many "side channels" that one can monitor passively, and sometimes one can even perform active attacks on the machine. The moral of the story is that one must exercise great care in storing and computing with secrets. If possible, avoid trusting hardware to keep secrets for you, especially if it can fall into an attacker's hands.

1 Hard Disk Storage

There are a few levels at which data can be recovered from hard disks. There are dozens of firms (e.g., DriveSavers; Google "hard drive recovery" for many more) that offer varying levels of recovery, costing from hundred to tens of thousands of dollars, depending on damage to the data and physical media.

1.1 "Deleted" files that aren't

First, the information in a "deleted" file is almost never immediately destroyed. When a file is deleted, the operating system merely marks the file's location on disk as overwriteable — however, the actual data remains magnetically encoded on the physical medium. The data is not overwritten until the operating system writes a new file at the location. This can take an indefinite amount of time, depending on usage patterns. Simple software filesystem analysis tools can reconstruct "deleted" files, in part or in full, using the data that remains on the disk.

[If Simson Garfinkel is not scheduled to guest lecture, elaborate more on what tends to get found and how.]

Countermeasures: software utilities like Unix's `shred`, which not only deletes a file but overwrites its contents (many times). However, certain kinds of filesystems (e.g., journaling) don't provide access to the physical location where a file is stored, so `shred` is of limited value.

1.2 Swap files

Operating systems often temporarily move memory contents onto a special disk location called a "swap file," in order to increase the amount of working memory. The user typically has little control over what is written to swap and when it is overwritten — anything from

web pages, sensitive documents, passwords, or cryptographic keys could be written and stored indefinitely. Software tools can search the swap file for goodies.

Countermeasures: many operating systems allow programmers to designate memory areas as “non-swappable.” If you are writing cryptographic software, any memory location containing secret keys or passwords should *absolutely* be declared non-swappable. If this is not possible, the software should overwrite sensitive memory locations as soon as they are done being used by the program.

Another solution is to use an encrypted swap file. It works as follows: on startup, the operating system chooses a random key, and encrypts everything written to swap with that key (decrypting it when it is accessed). When the machine is shut down (or the swap file is disabled), the key is erased from memory, rendering the contents of the swap file useless. This approach works because the swap file does not need to store any persistent data — it only needs to serve as a temporary placeholder for data in memory.

1.3 “Spraypaint” recovery

Hard disks store bits by setting the magnetic polarity of tiny regions of a metallic plate in the drive. While bits are digital, the physical storage method is analog. When overwriting a region of the disk, the old magnetic polarity is not completely obliterated — it still resides around the “edges” of the region. Think of it as spray-painting a wall many times with different colors: the most recent color appears on top, but previous colors show through due to the “fuzziness” of the spraying. Expensive, specialized hardware is needed to recover the old data, but some adversaries may be able to read back the most recent 20 writes to a location.

Countermeasures: using `shred` properly can overwrite the data enough times to completely destroy the original data. The military has been known to take apart disks and sandblast the platters, wiping all the magnetic material off of them before disposal. Dipping the platters in strong acid can also strip the magnetic coating.

1.4 A silver bullet?

It is best simply not to let sensitive data be stored “in the clear” on the disk in the first place. Instead, set up an encrypted disk partition or filesystem. (See Figure 1.)

In such a setup, all data is encrypted with a secret key before it touches the physical disk. Of course, this raises several questions: e.g., “where is the secret key stored (in the long-term, and short-term)?” and “where are the encryption/decryption operations performed?”

The secret key could be stored for the long-term in several places: on a separate tamper-resistant device, or in the “user’s head” (the key would be derived from some memorable passphrase). In the short-term, the operating system would keep the secret key in non-swappable volatile memory, and perform the cryptographic operations on the main processor (alternatively, the OS could “outsource” the crypto to a smart card or specialized piece of expansion hardware). When the machine is powered down (or the OS is told to “forget” the secret), there is almost no physical trace of the secret key.

(The encrypted swap file solution described above is very similar to an encrypted filesystem, but with the swap file there is no need to remember the key for later use.)

Figure 1: Diagram of an encrypted filesystem setup

2 Tempest Eavesdropping

Computers and their components, *especially CRT and LCD monitors*, emit all kinds of radiation, both visible and invisible. “Tempest” is the general name given to any kind of spying on this radiation. By monitoring this radiation, a lot of information can be inferred about what is being displayed on the monitor. There are some government standards for testing and shielding to mitigate this kind of side channel (e.g., NACSIM 5100A and its NATO equivalent AMMSG 720B); however, these documents are classified. In Germany, even the *names* of the relevant government standards documents are classified. In any case, proper Tempest shielding is exceedingly expensive.

2.1 Radio-Frequency Emanations

The image being displayed on a monitor causes it to emit different frequencies and amplitudes, which can be measured by an AM receiver. Here are a few uses for this technique:

- *A passive attack:* Eavesdrop on the actual displayed text on someone’s monitor. Ross Anderson and Markus Kuhn have built a working apparatus that can read text on a screen up to a half-mile away, using cheap hardware. [More details?.. but they are very technical.]

The same authors introduced a variety of cheap software countermeasures: techniques to “embed” an alternate image (which the eavesdropper will see) in the intended one,

and special “Tempest fonts” made up of high-frequency components that are difficult to detect.

- *An active attack:* Suppose you wanted to infiltrate a computing facility that was not connected to any outside network. You could sneak some malicious code (via virus, trojan horse, insider attack, etc.) onto a computer in the facility. The code would collect any interesting secret information. Then, while the computer was out of use (say, during the night), the code would broadcast the information to you by displaying images on the monitor that induce different tones on the AM radio. Even when the monitor is turned off, the power line and monitor cable serve as strong enough antennas to broadcast the information. Transmission rates are estimated to be around 50 bits/sec.

[A good project for a more hardware-oriented team might be to build one of these Tempest detectors. It is hard to say how much effort this would take.]

2.2 Optical Emanations

It turns out that radio frequencies are not the only channel by which monitors leak information. There is another channel, and it’s the most obvious one: the visible light spectrum! (After all, that’s what monitors are made for.)

Kuhn showed how to eavesdrop on a monitor that is out of direct sight using a telescope and a moderately-priced photosensor (an instrument which takes a succession of very quick “snapshots” of the total amount of light coming into it at each moment). The image can be even be reconstructed from reflections off a wall, assuming there isn’t too much background light.

Figure 2: Diagram of the setup of Kuhn’s attack

Before we get to the attack, here is some background information: on a CRT, the image is updated as a sequence of “scan lines” traced by an electron beam, which moves back and forth over the screen with constant velocity. As the electron beam hits each pixel, the pixel’s luminosity “spikes,” then degrades over time. So at any given moment, the total luminosity of the CRT is a weighted average of the last few thousand pixels that the beam has activated (a convolution).

If an attacker can get estimates of the *total* luminosity in sync with the electron beam, this information is a “low-pass filtered” version of the actual video signal. However, individual pixels decay fast enough so that enough high-frequency information gets through the filter. When the raw intensity measurements are processed using a high-pass filter or deconvolution, one gets a very clear image of the screen (see diagrams in Kuhn’s paper).

Countermeasures: curtains, frosted glass, etc. *don’t* protect from this attack, because they don’t hide the total luminosity in a room. The two best protections are: (1) work in a room that is completely out of sight to outsiders, and (2) work with lots of ambient light (this is better for your eyes, anyway). Note that the ambient light should span many frequencies (incandescents are good), or fluorescents should output similar frequencies as the CRT.

Also, the attack doesn’t work well on LCD screens, because entire lines of pixels are activated at once, not pixel-by-pixel, and the pixels “spike” and decay much slower.

3 Smart Cards

Smart cards are used in a lot of places. They are essentially cheap, commodity cards with a small amount of memory, some basic computational power, and some input/output ports. Interestingly, they get their operating power and clock from an *external* source over their ports.

Figure 3: Diagram of a smart card usage scenario

Often, smart cards must store cryptographic keys and keep them secret. Ideally, the cards are tamper-proof and only provide input-output access to cryptographic operations, so the secrets in the card’s memory cannot be extracted. The attacker who may be someone who stole the smart card, or its legitimate owner, or someone interacting with the card over its ports.

Here are some scenarios for smart card usage, and what might go wrong if the card were compromised:

- *A bank authenticating a customer at an ATM, or a merchant verifying a customer's smart credit card.*

The smart card stores the customer's secret key, and engages in some authentication protocol using specific cryptographic operations. One threat is that a thief will steal the card and attempt to extract the secret key, in order to pose as the legitimate customer and raid his account. Another risk is that a malicious ATM will interact with the card and attempt to learn the key.

- *Pay TV set-top boxes using smart cards for decryption.* The raw TV signal is encrypted, and the smart card has a secret key as well as code for the decryption algorithm. This allows the keys and decryption algorithm to be changed easily, without having to deploy expensive new set-top boxes.

In this case the customer may be the attacker. She may want to learn the secret key inside the smart card. This would allow her to sell cloned cards on the black market. (In fact, this has happened to several iterations of DirecTV's encryption system, and cloned cards are widely available.)

In practice, there are many successful ways to attack smart cards.

3.1 Simple power analysis (passive)

The idea is simple: passively monitor the power usage of the card over time. This correlates very well with the computation happening inside the card.

For example, DES has 16 rounds, and a power trace will reveal 16 bumps in the power usage. Each round uses a different piece of the secret key. Loading a '1' bit into a register uses a different amount of power than loading a '0' bit. Therefore the amplitude of the bumps will give a lot of information about the secret key (sometimes the entire key itself).

For RSA and Diffie-Hellman-based cryptosystems, there is a very easy attack too: in these systems, we often take a supplied c and raise it to d modulo n , where d is the secret key. The exponentiation algorithm is typically "repeated squaring," which cycles through the bits of d and conditionally multiplies by a running accumulator, then squares the accumulator (see Figure 4). Each conditional multiplication is extra work that requires more power, so the secret key d can be read directly from the power trace.

Countermeasure: instead of branching on secret data, do both computations unconditionally, and just throw out the one you don't need. The "throw out" operation should take the same amount of power, regardless of which quantity is being discarded. Shamir also described a hardware solution that defends against all passive power attacks; the advantage with his solution is that one need not take any special care in the implementation of algorithms (to avoid passive power-monitoring attacks, anyway.)

```

// compute  $c^d \bmod p$ , for  $d \geq 0$ 
exponentiate(c, d, p):
total = 1
s = c
while(d > 0)
  if d is odd:
    total = total * s
    d = d - 1
  s =  $s^2$ 
  d = d / 2
return total

```

Figure 4: Pseudocode for repeated squaring

3.2 Differential power analysis (passive)

A much more advanced attack known as “differential power analysis” (DPA) was invented by Kocher, Jaffe, and Jun. It can reveal secrets when algorithm, through its use of power, exposes information about the secret in a much less obvious way.

The idea behind DPA is to feed many slightly different inputs (say, differing in 1 bit) to the smart card and observe the *differences* in power usage over time for the different inputs. Since the cryptographic algorithm is known to the attacker, he can trace how the 1-bit difference propagates through the algorithm and interacts with parts of the secret key. Using statistical analysis on enough measurements, the secret key can be recovered. Furthermore, the attack can be automated and generally only requires a few thousand samples.

While simple power analysis has some easy and generally successful countermeasures, it is more difficult to protect hardware from DPA. The problem must even be addressed at the transistor level, to build gates which leak less information about the values on which they are computing. Shamir’s solution should protect from DPA.

3.3 Power glitches/resets (active)

Glitches and resets are an *active* attack with the power source. Glitches exploit the following idea: if you control the voltage and clock rate, you can run the chip outside of its designed parameters. This can be used to skip over certain instructions. For example, this kind of code is very common:

```

1      b = answer address
2      a = length of answer
3 loop: if a = 0 goto end
4      transmit(*b) over port
5      b = b + 1
6      a = a - 1
7      goto loop

```

8 end:

If you twiddle the voltage to skip over line 6 or 3, you can make the loop run forever. This dumps the card's entire memory over the port, which isn't typically available to an external user.

Resets exploit the following facts: many protocols require the card to have some source of randomness. Typically, this randomness is pre-loaded onto the card at the factory (since it is very hard for the card to generate its own randomness). In most protocols, re-using random bits can leak the secret key. Therefore, if the card can be reset to its "fresh" state, it may be easily attacked. Disconnecting any internal battery will achieve this goal. The solution is to only use "resetably-secure" protocols, for which reset attacks are of no use. Designing such protocols is very tricky business. . .

3.4 Timing attacks

The basic idea is to measure how long some crypto operation takes. Any operation that takes a variable amount of time, depending on some secret data, potentially leaks that data. For example, RC5 rotates bytes by a number of positions given by the secret key. If the smart card doesn't have a "barrel shifter," several rotations-by-1 must be performed in sequence. Measuring the time reveals the number of rotations performed.

Here is another example that can reveal the factorization of an RSA modulus $n = pq$. For efficiency, when operating on a value y , an implementation will typically reduce $y \bmod p$ and $y \bmod q$ and compute with those two values separately, then combine them later using the Chinese remainder theorem. These initial modular reductions can be vulnerable to timing attacks: choose an input y that is near p , and use the timing measurements to determine whether y is greater than or less than p (a division/subtraction step will be required if y is greater). Using binary search, one can learn p and discover n 's factorization.

3.5 Tamper (non)resistance

So far, the techniques we've seen have been relatively unsophisticated, don't require much equipment or skill, and are non-invasive. However, a determined and skilled attacker with a large budget can do much more.

Using a variety of chemical techniques, it is sometimes possible to strip the epoxy from a chip, then burrow into it or strip the top layers, and cut or place probes directly on internal wires. It is even possible to have the entire innards of a chip exposed for viewing, while the chip runs normally. Such an attack is devastating: none of the data used by the processor can be kept secret.

Using similar techniques, it is possible to strip away a chip one layer at a time, scan the area with a high-powered microscope, and learn its entire internal design. There are less invasive ways, too: under certain wavelengths of light, silicon is transparent and reveals the metallic traces within the chip.

Protecting against these kinds of attacks is very hard. Basically, it is very prohibitive to hide the design of a chip from a motivated opponent with time and a good budget. There are a few strategies that can make attacks harder, though: (1) place chips in *pressurized*

containers, and program the circuitry to disable itself when loss of pressure is sensed; (2) place *wires in the epoxy setting* around the chip, which detect damage to the epoxy and disable the chip; (3) include self-destruct mechanisms that detect when the chip is being tampered with.

4 A Theory of Physical Observability

At this stage, most physical attacks (especially side-channel attacks like Tempest and power analysis) are relatively ad-hoc, as are their countermeasures. This is an unsatisfying state of affairs, because we can never be sure whether our hardware gives us any security.

A framework called “physically observable cryptography” by Silvio Micali and Leo Reyzin aims to prove the security of physical cryptographic systems, even under observation of all kinds of side-channels.

Their framework “is not about ‘shielding’ hardware (neither perfectly nor partially) but rather about how to *use partially shielded hardware in a provably secure manner.*” Their constructions build high-level cryptographic protocols out of primitive building blocks that are assumed to leak only a certain amount of information. Of course, one must still build these primitives and shield them properly, but having done so, one can be assured that the overall system suffers from no further physical exploits.

5 Conclusions

In order to do computations in the real world, the data on which you’re computing, and the computation itself, manifest themselves in the physical world. Securing these manifestations is a very difficult task. One should minimize one’s assumptions about what protections exist, and design systems to mitigate failures in case those assumptions are broken. (In future lectures we will look at ways to control the damage, such as “threshold cryptography.”)