Attacks on Implementations of Secure Systems

Edited by Yossi Oren, yos@bgu.ac.il,
with a lot of help from his students.

June 14, 2020

Contents

Contents ii

1 Introduction 1
- 1.1 Motivation . . . 1
- 1.2 System Implementation . . . 7
- 1.3 Security of a System . . . 8
- 1.4 Constructing and Using a Threat Model . . . 11

2 Temporal Side Channels I 15
- 2.1 History . . . 15
- 2.2 The Threat Model . . . 15
- 2.3 Timing attack . . . 17
- 2.4 Timing Attack Defenses . . . 18
- 2.5 The Algebra Behind RSA . . . 21

3 Temporal Side Channels Part 2 24
- 3.1 Efficiently Implementing Modular Exponentiations . . . 26
- 3.2 Temporal Side Channel on RSA . . . 33

4 Power/EM I 48
- 4.1 Electronic Circuits . . . 48
- 4.2 Measuring Power Consumption . . . 54

5 Low Data Complexity Power/EM 2 70
- 5.1 The New York Times, Take 2 . . . 70
- 5.2 Power Analysis Attacks . . . 71
- 5.3 Simple Power Analysis . . . 73

5.4 Other Types of Power Attacks 75
5.5 Tuning the power model 75
5.6 8-bit microcontroller 77
5.7 Power Model for Microcontrollers 78
5.8 Simple Power Analysis of RSA 79
5.9 Simple Power Analysis of AES 81
5.10 Advanced Encryption Standard 82
5.11 The Advanced Encryption Standard 83
5.12 AES Internals 85
5.13 AES Power Analysis 88
5.14 Template Attacks 100

6 Introduction to micro-architectural attacks 111
6.1 Background & Primitives 111
6.2 Cache Attacks Techniques 121
6.3 Step by Step Attack Demo 131

7 High Data Complexity Power/EM 1 137
7.1 High Data Complexity Attacks 141
7.2 DPA Lab 147

8 Correlation Power Analysis 152
8.1 Previous lectures recap 152
8.2 Correlation Power Analysis (CPA) 156

9 Fault Attacks 173
9.1 Active Attacks 174
9.2 Fault Attack Taxonomy 176
9.3 Fault attack on RSA-CRT 182
9.4 Rowhammer 184

10 Ethics and Responsible Disclosure 189
10.1 Computer Laws 189
10.2 Cyber Ethics 193
10.3 Responsible Disclosure 198

Bibliography 201

A Writing LaTeX 209

A.1 Basic Formatting 209
A.2 Lists 211
A.3 Verbatim text 212
A.4 Chapters and Sections 212
A.5 Tables 213
A.6 Footnotes 215
A.7 Inserting Images 216
A.8 TikZ Graphics 217

Foreword

This document contains the collected scribe notes created by the students of my course titled "Attacks on Secure Implementations", given in Ben-Gurion University during Spring 2019.

The following students contributed to the writing process:

- **Chapter 1: Introduction.** Transcription by Michal Hershkoviz, graphics by Shir Frumerman, editing by Yoni Tsionov.
- **Chapter 2: Temporal Side-Channels 1.** Transcription by Sagi Nakash, graphics by Ziv Tomarov, editing by Shaked Delarea.
- **Chapter 3: Temporal Side-Channels 2.** Transcription by Itay Rosh, graphics by Noga Agmon, editing by Boris Kazarski.
- **Chapter 4: Low Data Complexity Power/EM 1.** Transcription by Ronen Haber and Rotem Yoeli, graphics by Idan Mosseri, editing by Ben Hasson.
- **Chapter 5: Low Data Complexity Power/EM 2.** Transcription by Barak Davidovich, graphics by Dan Dvorin, editing by Omri Fichman. Section about Template Attacks written by Adnan Jaber.
- **Chapter 6: High Data Complexity Power/EM 1.** Transcription by Vitaly Dyadyuk, graphics by Alon Freund, editing by Omer Nizri.

- **Chapter 7: Cache Attacks (Guest Lecture by Prof. Clémentine Maurice).** Transcription by Ben Amos, graphics by Arbel Levy, editing by Yaniv Agman.
- **Chapter 8: High Data Complexity Power/EM 2.** Transcription by Rom Ogen, graphics by Shai Cohen and Tomer Gluck, editing by Adi Farshteindiker.
- **Chapter 9: Fault Attacks.** Transcription by Dan Arad, graphics by Dorel Yaffe, editing by Iliya Fayans.
- **Chapter 10: Ethics and Responsible Disclosure.** Transcription by Ron Korine, graphics by Daniel Portnoy, editing by Roy Radian..

The text for chapters 1 to 5 is based on lecture notes in Hebrew originally created by Yael Mathov. Tom Mahler was responsible for creating the chapter templates and masterminding the entire editing process. I am grateful to them and to everybody who contributed to this document.

Chapter 1

Introduction

1.1 Motivation

The following story is a true story about the NSA – the U.S intelligence agency - which has an internal and classified, cryptology journal. In the U.S there is a law called "The Freedom of Information Act" (FOIA), which means that, in general, any person can ask access to government documents. So, on 2007, one person went ahead and asked for the table of contents of all the articles in the NSA journal, and the law is the law, so the NSA gave him what he asked, with censorship on some classified titles. After he got that, of course, he asked for the unclassified articles!

The journal, obviously, was used to be secret, but it was approved for release by NSA because of the FOIA law. One article that published in the journal, tells us about something that happened in the years of World War II. In those years, there were some significant developments, and one of them was digital wireless communication. People were able to send themselves messages from one side of the world to the other, using digital signals, but the military forces that used this communication were afraid to be wired by the enemy and to protect those transmissions they had to encrypt them.

At the end of the encryption process, we get a ciphertext, which can be transmitted without any concern. The obstacle

which prevents attackers from decrypting the ciphertext and discover the key is the algorithm which has been written very well. Let's assume we have a weak computer to perform the encryption, because we are in the years of World War II - no resistors, no iterative circuits – what can the adversary do to find the key?

The adversaries can build a machine which does the decryption – insert the ciphertext into the machine and try all the keys (brute force). But... how do you know which key is the correct one? There is logic to the plaintext – maybe it's an English, a weather broadcast, an executable – and in general, it's something you can check the syntax, so with a very high probability, if you get an output that starts with "Heil Hitler" – the key you used to decrypt the ciphertext is the correct key. We need to remember that the encryption machine was not very complex in those years, and to keep the secure messages safe, they had to find a way to protect their ciphertexts. The solution to that issue is a one-time pad.

One-time pad, also called Vernam Cipher, is a simple and powerful encryption system. The idea behind this technique is that the length of the key is the same length as the plain text, and to encrypt you just add them together – if we use digital bits we use XOR, and if we use letters we define an addition operation. It's called a one-time pad because you can use your pre-shared key only once.

Why does this one-time pad protect from brute force attacks? Because $plaintext \oplus key = ciphertext$ and $ciphertext \oplus key = plaintext$, meaning we can take the ciphertext and any plaintext we want, XOR them together, and get the corresponding random key. In a very secure encryption technique, like AES-256, there are 2^{256} possible keys and with a ciphertext which is 1 MB, we have $2^{8000000}$ possible keys, but if we don't know the key there are 2^{256} random plaintext, and maybe just one of them is the real one. The one-time pad has been proven to be completely secure by Claude Shannon, and there is no way to break it.

Back to the years of World War II, to use one-time pad those days, they used machine which called **AN/FGQ-1 mixer** [1] as can be seen in Figure 1.1.

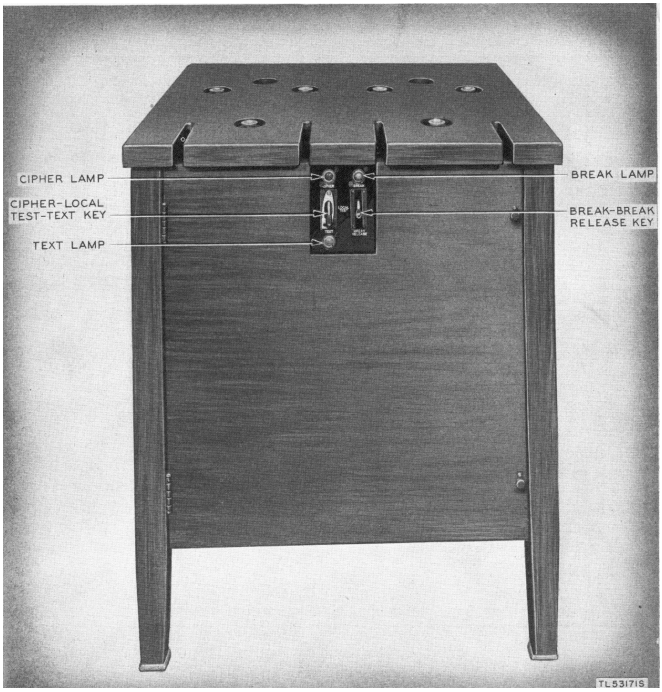


Figure 1.1: **AN/FGQ-1 Mixer**. Two-way teletypewriter repeater and mixer equipment enclosed in a wooden Table-type cabinet.

This machine is a kind of a box, and next to the box was seat a wireless operator with a typewriter, and the tape that came out from the typewriter was with holes that spooled into the box together with another piece of tape, which was the key. Inside the box were little lights that shine through the holes, and little punches which would punch holes in the third piece of tape – the XOR of the plain text and the key, and this is the output of one operation of the machine. Afterward, the wireless operator feeds the ciphertext into the digital radio to transmit.

Is the machine classified? If the system is designed right -

you can reveal the adversary whatever you want except for the secret key, and the system will remain secure.

So, these machines were used during the war, until they break down, and at the time that happened, they have been sent to the Bell Labs (which produced the machines). The engineers who tried to repair one of the machines sat in a room, and on the other side of the room, there was an **Oscilloscope**.

An Oscilloscope is like babies monitor - just for signals. You connect the Oscilloscope to an electric circuit, and there is a line that rising every time there is a difference in voltage. Figure 1.2 demonstrates such a setup. This Oscilloscope was not connected to the mixer, it was just laid at the other side of the room, connected to some other test equipment. The engineers discovered that every time this mixer enter the digit into the tape, they would get a pulse at the Oscilloscope. When you send a current into an electric monitor, the current is moving through the conductor and electromagnetic current is being generated. The Oscilloscope has little wires, and the electromagnetic waves traveling through these wires. As a result, we have a transmit antenna and a receive antenna, so the Oscilloscope measures the holes (the ciphertext) in the tape. Another possible explanation - when the punches punch a hole in the tape, it consumes so much current that the voltage in the room drops a little, and then the lines in the Oscilloscope – without being connected to anything - just jumps.

The engineers discovered [2] that the top-secret information inside this machine was being transmitted over the air. The process the engineers were supposed to do is called responsible disclosure, meaning to report a bug. Like good engineers, they told the Secret Service people about this bug, but they didn't take their diagnosis seriously. So what did they do? The Bell labs were located next to the U.S Secret Service office, so they set up an antenna and they listened for an hour for radio transmissions that the Secret Service got in their office, and they gave the Secret Service their "top-secrets" analyzed messages.

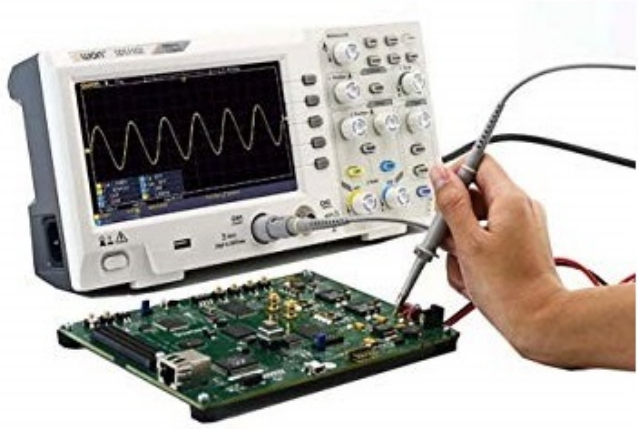


Figure 1.2: **Oscilloscope**. An electronic test instrument which graphically displays varying signal voltages as a function of time.

Obviously, the U.S Secret Service realized that the reported attack was not esoteric, and they asked for a solution from the engineers. The solution of Bell labs was to modify the mixers – to surround them in a wire cage which will swallow the radiation that coming out from the machine, to put a shield on the machine (to make sure the power consumption is not affecting the outside), and to make sure that people are not getting close to the machine more than 30 meters. The Secret Service people decided to accept just the distance solution and not the filtering/shielding/isolation solutions, because of the high expenses and time that will cost to modify all the machine during wartime.

In 1954, the Russians published a tender to build military phones. They were very strict about protecting the phones by shielding them and making sure they don't generate too much radiation. In parallel, they were publishing tenders for things like engines, turbines, etc. and they didn't were so strict for that tenders. This is an evidence to that in those years, the Russians know about the "bug". In conclusion, a one-time pad is the most secure cipher known, but from the

story above, we can see it was broken. So, what was broken? **The implementation**.

Here are a couple of other machines which will break using attacks on their implementation:

- **Xbox 360**: Xbox is a PC that plays nothing but games, it is very cheap and you pay for the games. Attackers, obviously, want to crack the Xbox – to play games for free, to watch movies on the device or to run Linux. In Xbox there are some integrity checks, and one of the checks was done by a command called "memory compare" [3], so you would calculate the integrity check over whatever software it supposes to run and you would have it stored in the secure memory, and then you would try to compare using these 2 values. This command leaks the length of the number of correct bytes before the first incorrect byte, so if you compare 2 blocks and the first bit is different – the response will be fast, and if the blocks identical until the very last bit – it would take a longer. This is one of the things that was enough to break the machine.
- **Oyster Card**: it's a computer without power supply and Inside this computer that are stored values. Attackers could attack this card to take the train for free. There are a lot of ways to attack the implantation of that card [4, 5].
- **Car Keys** [6]
- **FPGA** [7]: a piece of hardware which is a very versatile, meaning we can find it many kinds of hardware – routers, audio equipment, spaceships, etc. the FPGA has a firmware installed inside, and if you want to copy some designs you need to find the firmware. The firmware is encrypted, but a bunch of Germans researches discovered [8] that if you measure the power consumption of the FPGA while it is encrypting the firmware – you can find out what is the key.

When we implement an algorithm without being careful, we can be exposed to implementation attacks. To protects against these attacks, we must protect the implementation and do countermeasures, but as we saw at World War II, those countermeasures have a price, thus making the system more expensive, and they make the system heavier.

1.2 System Implementation

The simplest form of a computational system is a device which gets input, makes a computation and finally produces an output as can be seen in Figure 1.3. We assume that our system contains a secret, which cannot be revealed to anyone before, during and after the computation process.

Figure 1.3: **Simplest Model**. The device make a computation using the secret and the input, and outputs the result.

But, if that all what the "system" has, it is not a system, it is just an Algorithm. What turns an algorithm to be a system? It's implementation.

Think about ATM – very simple device without cryptography. The input is our credit card and a 4-digits PIN code, the output is money. If we don't know the PIN code we can go over the all possible combinations of 4-digits code (brute force), and finally find the correct PIN. Unfortunately, we have a limit of 3 trials until the card is being shredded. What an attacker can do?

As an output, and besides the money, we have also some additional outputs which have been produced by the implementation of the physical system. Those additional outputs

are called "Side Channels" and they are outputs which the system designer did not intend to produce. Those outputs are demonstrated in Figure 1.4

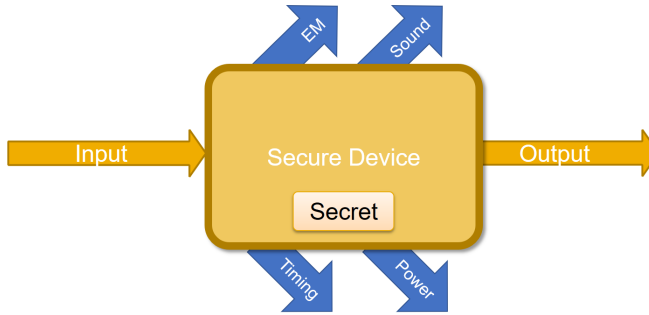


Figure 1.4: **Side Channels of the System**. Caused by the implementation of the system

For example, we can measure the time it takes to fulfill the operation, electromagnetic radiation, the sound of the device while the operation is being fulfilled, power measurements, etc. These are **Passive Attacks** – meaning we are letting the device to do his stuff while we just listening.

There are also **Active Attacks** – called also fault attacks - try to break the device in a way that it will be "just a little broken" – ruin the device's activity in some way. It can be by turn it off in the middle of calculation, change his clock, etc. As a result, we will maybe not get the actual secret, but we will get some kind of errors that can teach us a lot about the secret.

1.3 Security of a System

When we can say that a system is secured? Usually, we define a system as a "secured system" if the system maintains three aspects of information security, known as the CIA [9] triad:

- **Confidentiality** - when you are interacting with a system, you only get what you wanted to get. For instance,

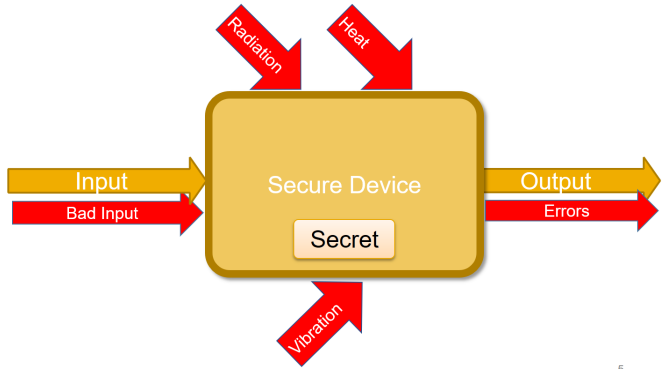


Figure 1.5: **Fault Attacks**. Manipulating the device through side channels

when I check my test grade, I'll get my grade and not my friend's grade. A possible attack could be data leakage.

- **Integrity** - all the data in the system is correct. A possible attack could be that one side of the communication will accept a message they are not supposed to accept.
- **Availability** - the system must work (in a reasonable time). A possible attack could be a Denial of Service (DOS).

In Figure 1.6 the relation between those aspects is demonstrated as the Triangle of Information Security.

We don't have to use cryptography to secure our system. For example, if someone goes to some event without invitation, there is a security to prevent him from getting in.

An **Algorithm** is a process or set of rules to be followed in calculations or other problem-solving operations. An example of an algorithm is GCD/Extended GCD. An algorithm is secure when it is implementing the CIA triad mentioned above. A **Protocol** is when you need to get something done,

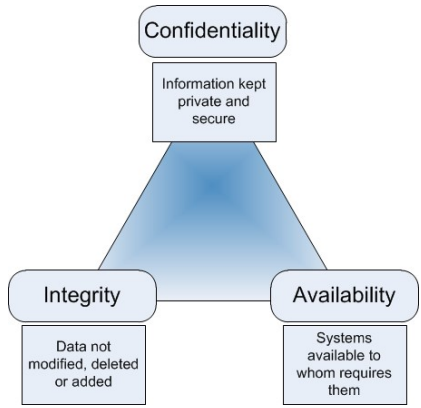


Figure 1.6: **CIA Triangle**. The classic model for information security. Defines three objectives of security: maintaining confidentiality, integrity, and availability. Each objective addresses a different aspect of providing protection for information.

for example, AES - the input is a 128-bit key and a 16-bytes plaintext (in case of more than 16-bytes we should use padding), and the output is a ciphertext.

The millionaire problem [10] is a classic problem by Yao and it is an example of a protocol – this problem asks whether two millionaires can learn who is richer, without revealing to one another how much money they each have. One possible solution, they could invite a poor man, tell him in secret how much money each has, and the poor man will announce who is richer.

Cryptographically Secure Algorithms and Protocols:

- **Encryption and Decryption** - Public key-RSA, Symmetric key-AES
- **Signing and verification** - must be an asymmetric key. There are 2 parties – signing party and verifying party (who gets the public key). The difference between signing and decryption is when one side sends a signed message it comes with a signature, and when

one side decrypts a message he is sending only the ciphertext.

- **Key Exchange** - Diffie Hellman algorithm.
- **Hashing and HMACs** - a hash is a function that gets a long input (the length doesn't matter) and outputs a fixed size output. A hash function is secure when it's hard to find collisions, e.g. 2 messages with the same hash. HMAC is a hash with a key – when you change the key, the hash is changed.
- **Multiparty Computation**
- **Cryptocurrency**

Secure Architectures without cryptography:

- **Secure Policies** - like access control to a military base, for instance.
- **User Separation and Sandboxing** - a program is divided into parts which are limited to the specific privileges they require to perform a specific task.
- **Virtual Memory** - application has a view of the memory, and we can take to pointer and point to some parts of the memory. We will get our old memory (which I'm allowed) or the app will crash due to access to invalid memory space. In theory, we can't get another user memory.

1.4 Constructing and Using a Threat Model

What is the main advantage we have as attackers which allowed us to break implementations that are secure in theory? The main advantage is that we have more inputs and outputs – side channels, leakage – and together it means that we

can break a completely secure algorithm. But when is an algorithm's implementation secure? CIA triad still holds, but the thing that missing here is the **story**, i.e. what are we allowing the adversary to do with my system – the more power we give the adversary the attack becomes less impressive.

Let's have a look at a little system where the assumptions were broken:

Figure 1.7: **ATM theft.** The thieves simply ripped the machine out of the wall.

Figure 1.7 presents a wall of a bank in Ireland, which an ATM was constructed on it. As we know, the ATMs are very secure systems – they have encryption, they check our ID very carefully and if we make any mistake they shred the card. What happened is that the ATM was stolen from the wall of a bank by thieves which took a digger and smashed through the wall, put the ATM on their track, and laid the digger across the road to prevent the police from chasing them. We can learn from this story, that the ATM was secure, but their threat model was wrong.

The most important thing about the threat model is the story. If the story is a thief which comes to the bank in the middle of the night with a digger it's one story, and if a thief comes in the middle of the day by foot, it's a different story.

Once we have the story, we can detail what are the properties of the story. We divide them as follows:

- **Victim Assets** - what the victim have that I want to steel?
  - **Cryptographic secretes (keys)** - the keys are short, and when one steals the key – the attack has succeeded. There are two kinds of keys, long term, and short term.
    - **Long term key** - the private key that identifies a server. If one steals this key, he will be able to sign malware as a software update.
    - **Short term key** - a key that generated during a session start from the long-term key. If one steals this key, he can decode all the messages in the current session and modify/inject them.

    Notice that if one steals the long-term key it doesn't mean he can steal the short-term key.
  - **State secrets** - for example, ASLR or configuration of a system. If one can find out the addresses of functions in the memory of a victim, he can write exploits.
  - **Human secrets** - things which the users do not want to expose like passwords, medical condition, etc.
- **Attacker Capabilities** - what can the attacker do?
  - **Off-path** - the attacker sitting somewhere in the world – can't observe or communicate with you, but he can attack you somehow.
  - **Passive Man in the Middle** - attacker who can see the victim communication with the server, for example, but he cannot communicate with the victim directly.

  - **Active Man in the Middle** - an attacker who can interact with the server and can do replay attacks.
  - **Physical Access** - an attacker who has physical access to the victim. For example, removing or unplugging or melting stuff in the system.

  An important thing we need to consider when we are talking about attacker capabilities is the other defenses we must protect our system – guard or camera, for example. Another thing is the scale of the attack – how many systems can we attack at once. If the attack is physical, probably just one system. If the attacker attacks from an android app, maybe he can attack all the phones in the world.

- **Attacker Objectives** - who are the attackers? What do they want?
  - **Stealing stuff** - the attacker might want to steal your secrets.
  - **Duplicating stuff** - for example, the attacker can buy one smart TV card, and generate a thousand duplicates from this card to sell them.
  - **Forging stuff** - the attacker creates something new, driver licenses for instance.
  - **Corrupting stuff** - the attacker breaks something and decommissioned the system.

Of course, the more limits we put on the attacker, the attack gets more and more impressive.

Chapter 2

Temporal Side Channels I

2.1 History

In 1995, When he was 22, Paul Kaucher released a paper called Timing attacks on implementations of Diffie-Helman, RSA, DSS and other systems [11].

Before it was published, he wrote about it in a mailing list of Cypherpunks[1] which included all sorts of people like mathematicians, photographers, artists anarchists and more. The attack was first demonstrated in 1997 in a cryptography conference. And later, in 1998 an academic paper was published describing how to perform the attack.

2.2 The Threat Model

Figure 2.1 descirbes the Threat Model on an implementation of some secure system. It is important to mention that we're

[1] A cypherpunk is any activist advocating widespread use of strong cryptography and privacy-enhancing technologies as a route to social and political change. Originally communicating through the Cypherpunks electronic mailing list, informal groups aimed to achieve privacy and security through proactive use of cryptography

talking about attacking an implementation, and not the algorithm itself as we consider the algorithm or protocol being examined as completely secure.

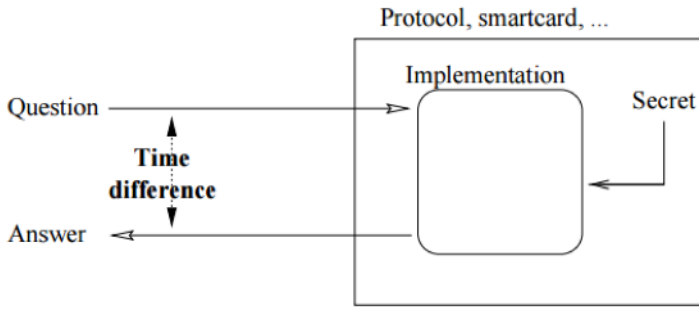


Figure 2.1: The Threat Model. Image from JF Dhem1998. The attacker can send a message to the implementation and get an answer from it. Also - the attacker is able to measure the time it takes for the implementation to compute the output

When is a timing attack even possible?

1. Physical access to the device. (for example: Smart card, Crypto wallet, Electronic voting machines)
2. Sharing a virtual machine with the service (for example: Swiping a credit card)
3. Remote access to a device (access to a device over the network, which may be very noisy and may be rather difficult to implement)

The objective of the attacker to discover the password. And what could the attacker do? The attacker can send unlimited queries and measure their time. Unlimited queries are not so trivial, If you think about it. Some devices can disable themselves after 3 or 4 or 5 periods. This can be a defense [12]

2.3 Timing attack

```
1  var secretPassword = "HASODSHELI";
2
3  function verifyPassword(inPassword) {
4      if (inPassword.length != secretPassword.length) {
5          return false;
6      }
7
8      for (i = 0; i < secretPassword.length; i++) {
9          if inPassword[i] != secretPassword[i] {
10                 return false;
11             } // if
12     } // for
13
14     return true;
15 }
```

Figure 2.2: A simple and efficient password checking algorithm. In line 4 there's a check if there length of the password is the same as the length of the input. In line 8 there's an iteration over all of the characters in the

Consider the algorithm for password checking as described in figure 2.2

In a scenario where timing attack is not possible, breaking the password requires the attacker to bruteforce the password. That is, checking every possible string for one successful attempt.

Let's assume the password is of length 16, if the password only contains english upper-case characters we have 26 possible values for each of the 16 characters in the password. The first character has 26 possiblites, the second has 26 possiblites and so on. So bruteforcing a password like this requries 26^{16} different attempts, which cannot be completed by any computer in a decent amount of time. In general, for a password of length n and character range of size k, breaking the password will take $O(k^n)$ attempts.

Now let's consider the scenrio where **timing attack is possible**. To perform a timing attack, the attacker takes ad-

vantage of the fact that when the program checks for strings equality, the comparison will finish as soon as it finds one character that doesn't match. Consider the following example where the attacker tries the 3 following attempts and measures the time it takes for the machine to respond. The attacker tries "AAAA" and measures 0.2ms, then tries "BAAA" and measures 0.5ms lastly tries "CAAA" and measures 0.2ms again. The attack can be pretty sure that the first character is "B". Now, checking the whole password does not require iterating of all possible combinations of strings of size n. All it takes is to iterate over all possible values of each character, say k, and repeat that n times. This results in a total runtime of $k + k + k + \cdots = nk = O(n)$

We can now break the password in a reasonable amount of time, even without knowing the length of the password we're trying to crack.

2.4 Timing Attack Defenses

After discussing the potential of such attack, we now consider the possible mitigations we can implement on our system to prevent such attacks.

Mitigation

We can consider adding a random wait time after checking each one of the characters. The problem of course is that the whole process of checking a valid process becomes slower. And of course, if the attacker can perform a lot of measurements on our system, since the noise is random, the attacker would be able to ignore it.

Prevention

The second type of countermeasure is prevention, making sure that our system is completely resistant to timing attacks.

Prevention Method 1: Padding

```
1  var secretPassword = "HASODSHELI";
2
3  function verifyPassword(inPassword) {
4      var result = true;
5
6      // pad secret password and input password to same length
7      var longPadding = '                          ';
8      var paddedInPassword =
9          (longPadding + inPassword).slice(-longPadding.length);
10     var paddedSecretPassword =
11         (longPadding + secretPassword).slice(-longPadding.length);
12
13     for (i = 0; i < paddedSecretPassword.length; i++) {
14         if paddedInPassword[i] != paddedSecretPassword[i] {
15                 result = false;
16             } // if
17     } // for
18
19     return result;
20 }
```

Figure 2.3: Prevention method 2, padding the user guess and the secret password to the same length and check all characters even if there's a mismatch in the first character.

The first method we examine is to pad the secret password and the user's guess to the same length. Also - we do not return as soon as we see a character mismatch. The code is described in figure 2.3

The problem with this implementation is that every time there's a character mismatch we execute additional code. Which is loading the variable `result` and writing the value `false` to it. This may seem insignificant in the beginning but actually, this is makes our code **much more vulnerable than it was before**. As now the time of it takes for the entire function to complete is linearly dependant on the amount of characters mismatches we have, this allows an attacker to perform the same attack from before with no significant changes to his original timing attack.

One might think of a fix which is adding another branch to the `if` statement that will do some garbage operation like `foo = false` just so that the attacker might not be able to tell the difference between a match and a mismatch. The problem with this fix might be that the access time to one

variable may be different than the access time to another variable, and the attacker will be able to tell the difference between them.

Prevention Method 2: Hashing

```
var secretPassword = "HASODSHELI";

function verifyPassword(inPassword) {
    var result = true;

    var hashedInPassword = CryptoJS.SHA3(inPassword);
    var hashedSecretPassword = CryptoJS.SHA3(secretPassword);

    for (i = 0; i < hashedSecretPassword.length; i++) {
        if hashedInPassword[i] != hashedSecretPassword[i] {
                result = false;
            } // if
    } // for

    return result;
}
```

Figure 2.4: Secure password checking using a secure hash function

The right way to store passwords is with hashing (storing the hash of the password rather than the password in plain text). A hash is a cryptographic function which has the property called "The Avalanche Property" which means if even 1 bit is fliped in the input, at least half of the bits of the output are flipped as a result. This means that even if my guess is really close to the password (1 bit away from the real password) I cannot really know which bits are correct and which are not due to the Avalanche property.

Using hashes to perform a secure password check is described in the algorithm in figure 2.4 A guess is being hashed before it is compared against the hash of the true password. An attacker might use the same methods from before to try to leak the hash of the true password, but that would be very difficult as the attacker does not input the hash, the attacker only controls a string that is later being hashed by the algorithm. So the trying the methods from before to leak the password would not work if the hash used is propely implemented.

A possible leak that can occur from this method is the length of the true password. In line 7 of the algorithm in figure 2.4 the hash of the true password is computed. Line 7 makes the total run time of the program dependant of the length of the true password. While this might be risky, this is easy to fix - we can just precompute the hash of the true password and use it whenever the algorithm runs. This is done for example in Linux where the hashes of user's passwords are stored in a file in `/etc/shadow`.

2.5 The Algebra Behind RSA

The next thing we're going to perform a timing attack on is the RSA crypto system. But before we delve into how we break RSA (next chapter) we're going to discuss the algebraic foundation of RSA [13].

The RSA cryptosystem lives in something called a multiplicative group. The group that is Z_n^*.

We're taking 2 random prime numbers; p, q and assign $n = pq$. The group Z_n^* contains all the numbers from 1 to $n-1$ which do not divide n (that is, do not divide p or q). For example: for $p = 3$ and $q = 5$, $n = 3 * 5 = 15$ so $Z_n^* = \{1, 2, 4, 7, 8, 11, 13, 14\}$.

Like all groups, this group has the associative operation which in our case is the modular multiplication, that is because we mentioned before that this group is a multiplicative group. The group is also closed under that operation.

The group also has an identity element, 1. Which when multiplied by another element from the group - does not change it.

The last property this group has is that every element r in the group has an inverse element $r^{-1} \in Z_n^*$.

Since the exponentiation is just repeated multiplication - we consider exponentiation to also be a closed operation under that group. Each element has an order, the order always

divides $(p-1)(q-1)$ (Fermat's little theorem). When the element is raised to the power of the order, the result of the exponentation (under the modulu of course) is 1, the identity element in the group.

It also important to note that we cannot compute $(p-1)(q-1)$ from knowing n

Elementary Operations of RSA

To use the RSA crypto system to encrypt and decrypt messages you first have to generate the infrastructure. That is deciding on p and q both large prime numbers and computing $n = pq$ and $(p-1)(q-1)$ denoted as $\phi(n)$

- **Choosing a public and private key pairs** Choose $e \in Z^*_{\phi(n)}$, $d \in Z^*_{\phi(n)}$ such that $ed = 1 \bmod \phi(n)$. The public key is the pair $\langle n, e\rangle$ and the private key pair is $\langle n, d\rangle$.
- **To encrypt a message** Choose $m \in Z^*_n$ as your message. The cipher is $m^e \bmod n = c$
- **To decrypt a message** Since $c = m^e \bmod n$, perform $c^d = m^{ed} = m^1 = m \bmod n$

Example: Consider $p = 3$, $q = 5$. So we can compute $n = 3 * 5 = 15$ and $\phi(15) = 2 * 4 = 8$ Let's assume we choose $e = 3$ and $d = 11$. We consider the message 2.

To encrypt, we compute $2^3 = 8 \bmod 11$, Which is the cipher. And to decrypt, we're using the private key 11 as follows: $8^{11} = 2 \bmod 15$. And now we have our message back, 2.

Since this is not a crypto course, we will not go into any much more details about the algebra behind RSA. However it is important for us to get familiarize with the basics of RSA in order to understand later how it was optimized and how can we break it.

See also

1. John Wiley and Sons Chichester , "Overview about Attacks on Smart Cards by Wolfgang Rankl, Munich", 3rd edition at John Wiley and Sons in September 2003.

2. Thomas Popp, "An Introduction to Implementation Attacks and Countermeasures", Graz University of Technology, Institute for Applied Information Processing and Communications (IAIK) Graz, Austria.

Chapter 3

Temporal Side Channels Part 2

recap

In the previous chapter we discussed about RSA cryptosystem [14]. In this chapter, we presented the most expensive operation in the RSA algorithm which is when we need to take the message (m) and raise it to the power of the key.

$$C = m^e (mod n)$$

The simplest algorithm for raising a number to a power uses modular multiplication which is an expensive operation.

Modular multiplication is pretty straightforward. It works just like modular addition. You just multiply the two numbers and then calculate the standard name. Examples for Modulo 7 and 15 can be found in Figure 3.1. Why modular multiplication is so expensive? Because we have to take the modulo many times which is as expensive as division[1]. So, one way to do this is by keep multiplying and doing the modular reduction only at the end. The problem with that approach is that the runtime of multiplication grows exponentially with the number of the multiplicands. Another way

[1] Division is the most expensive integer operation exists, and therefore we don't want to do many divisions

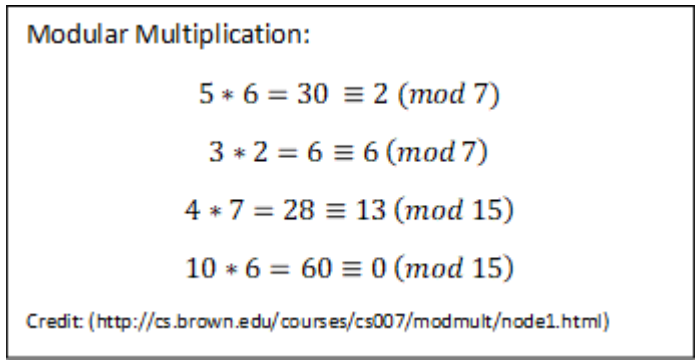


Figure 3.1: Modular multiplication

is to do a *modn* with each multiplication, but the numbers keep growing and the runtime keep raising, so we can't do that either.

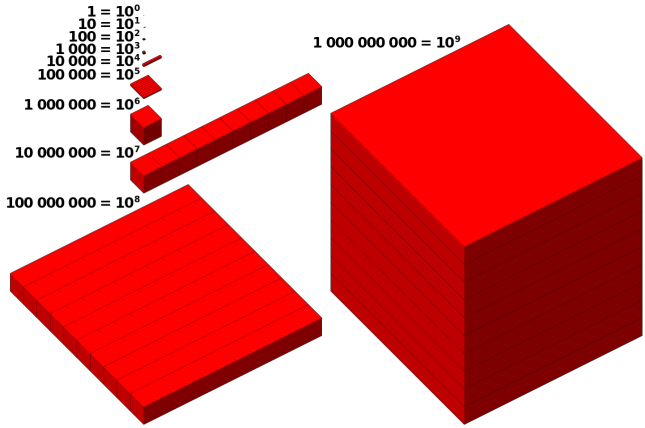


Figure 3.2: Visualization of powers of 10 from one to 1 billion.

As we discussed in the last chapter, the slowest and most trivial way of implementing modular multiplication $g^e modn$ is to take g and multiply it by itself e times, and every once in a while do modulo n, (either every multiplication or just in the end). The problem with this is that if e is a number that consists of 1024 bits, then in the worst case we might need to multiply g in itself 2^{1024} which is a huge number.

3.1 Efficiently Implementing Modular Exponentiations

There are two ways for efficiently implementing modular exponentiations:

1. performing fewer modular multiplications (instead of 2^{1000} we would like to do 1000).
2. Make each modular multiplication to be less expensive.

The best case would be same as regular multiplication over integer. These two ways will reduce the cost of modular multiplication and modular exponentiations, and this is something we really want to do in order to make RSA work on a device.

So, assumeing we are engineers, we want to implement modular exponentiations very cheaply. The right thing to do is obviously use some crypto library, but let's assume that's not possible, and we are inventing a new CPU. There is a very famous book online called the handbook of applied cryptography (Figure 3.3) [15]. The book is like a recipe book, and is filled with algorithms, proofs and equations you need for cryptography. Chapter 14 deals with efficient algorithms for multiplicative proofs. The book contains several ways to do modular exponentiation.

The following approach is called the left to right binary exponentiation, also known as square multiply.

The inputs are g (we want to raise g to the power of e, g^e) and e which is a bit string of t (t bits), (see Figure 3.4). The most significant bit is always 1, because there is no logic behind raising something to the power of zero. In the end of this algorithm A will be the result of g raised to the power of e.

In the beginning we set A to be equal to 1, and go over the bits from left to right (from the most significant to the least). Each time we square A ($A = A * A$), but we only multiply

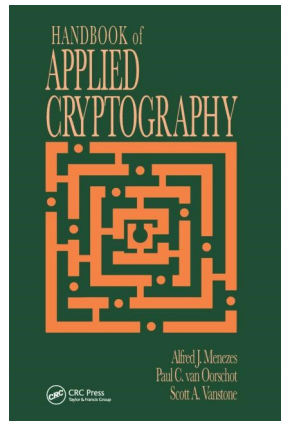


Figure 3.3: Cover of the handbook of applied cryptography

Algorithm Left-to-right binary exponentiation

INPUT: $g \in G$ and a positive integer $e = (e_t e_{t-1} \cdots e_1 e_0)_2$.
OUTPUT: g^e.

1. $A \leftarrow 1$.
2. For i from t down to 0 do the following:
   2.1 $A \leftarrow A \cdot A$.
   2.2 If $e_i = 1$, then $A \leftarrow A \cdot g$.
3. Return(A).

Figure 3.4: The pseudo-code was taken from the handbook of applied cryptography, page 615.

by g ($A = A * g$) if the current bit is 1 (the i^{th} bit). Now, what happens when we do a squaring operation downstairs? What happens to the exponent upstairs? Its multiplied by 2. So $g^{e2} = g^{2e}$ and $g * g^e = g^{e+1}$. Now, we can think of a binary string, you can write down the bits using shifting (multiplying by 2 is shifting) and adding 1 is just putting one in the place.

So how many modular multiplications will be performed in order to raise g to the power of e? The answer is $O(t)$. In the best case - what is the lowest amount of multiplications that will be performed? the answer is $t + 1$ as the first bit is 1 (most significant) so we do step 2.2 one time and all the

rest of the bits are zeros, and so we will only do step 2.1 in these cases. In the worst cast - what is the highest amount of multiplications that will be performed? Answer: $2t$, if all the bits equal to 1 we will do 2 multiplications for each bit (step 2.1 and step 2.2). In any case, this is equal to $O(t)$, so instead of 2^t as in the old algorithm we perform at most $2t$ multiplications.

Lets review an example. Lets calculate $7^6 = 7^{(110)}$ in the group $\mathbb{Z}_{15} = 1, 2, 4, 7, 8, 11, 13, 14$.

1. $A = 1 = 7^(0)$
2. $A = A * A = 1 = 7^{(0<<1)} = 7^{(0)}$
3. $A = A * 7 = 7 = 7^{(0+1)} = 7^{(1)}$
4. $A = A * A = 4 = 7^{(1<<1)} = 7^{(10)}$
5. $A = A * 7 = 13 = 7^{(10+1)} = 7^{(11)}$
6. $A = A * A = 4 = 7^{(11<<1)} = 7^{(110)}$

Are there any ways doing this even faster? The answer is yes as we can see in the handbook of applied cryptography. The general idea of these algorithms is that we do some pre-computation. The idea of RSA algorithm is that there is a public key g that is a known number. So if we know what g is we can prepare all sort of lookup tables. One method for doing that is called the window method, where we take three bits at a time, and instead of doing $A * g$ we do $A * g * g$ or $A * g * g * g$ .. (8 values we can use), and instead of $A = A * A$ we do $A = A * A * A$ and so on, this is the sliding window. Another method is called the binary method, which is well described in the handbook of applied cryptography (page 616 algorithm 14.83). We can assume that any reasonable crypto implementation is not doing the naive method. It will use the left to right or the right to left binary exponentiation which is basically the same idea as one of the window methods.

But what if we want the modular multiplication to be cheaper? A way to achive this is called the Chinese remainder theorem (CRT) [16], named after Sun Tzu Suan that was a teacher from the 5th century and wrote a book that contained all sorts of riddles and questions.

The Chinese remainder theorem addresses the following type of problem. One is asked to find a number that leaves a remainder of 0 when divided by 5, remainder 6 when divided by 7, and remainder 10 when divided by 12. The simplest solution is 370. Note that this solution is not unique, since any multiple of $5 \times 7 \times 12$ ($= 420$) can be added to it and the result will still solve the problem. The theorem can be expressed in modern general terms using congruence notation. Let $n1, n2, \ldots, nk$ be integers that are greater than one and pairwise relatively prime (that is, the only common factor between any two of them is 1), and let $a1, a2, \ldots, ak$ be any integers. Then there exists an integer solution a such that $a \equiv ai(modni)$ for each $i = 1, 2, \ldots, k$. Furthermore, for any other integer b that satisfies all the congruences, $b \equiv a(modN)$ where $N = n1n2 \cdots nk$. The theorem also gives a formula for finding a solution. Note that in the example above, 5, 7, and 12 ($n1$, $n2$, and $n3$ in congruence notation) are relatively prime. There is not necessarily any solution to such a system of equations when the moduli are not pairwise relatively prime.

Why does this help us? Allegedly, we have to do 2 operations instead of one. The way to do exponentiations in CRT is that we take our big number and do modulo p and then modulo q and then there is a CRT step. So what is the size of the operand used by these two modular exponentiations? Assuming that p and q are of the same size? Let's say p and q are 1000 bits so the size of n is 2000 bits (we add the number of bits of each number in the multiplication). So each number in the multiplicative group is about 2000 bitw because it's mod N, so multiplying two numbers is going to be multiplying two numbers which are 2000 bits. If we reduce it to modulo p and modulo q the numbers are going to be

half the size (1000 bits). So if we take a multiplication and now we will multiply two numbers that are half the bit length what will be the speed improvement? It's times 4. Instead of one big modular exponentiation we have one small modular exponentiation which costs quarter of the time and another one which cost quarter of the time followed by a CRT step which is a modular multiplication of the two. How much we spent in total? Answer: a bit more than half the time, twice speedup. Why can't we do that further? Divide p and do it again? Answer: because p and q are prime numbers and that's the whole point.

On top of that, there is a very nice trick, in order to make modular multiplications very cheap and this actually makes modular multiplications to be as cheap as regular multiplication. Multiply two numbers is pretty easy but the problem is reducing after multiplying. So what if there is a way of doing modular multiplications without the reduction step? So in 1985 a genius mathematician called peter Montgomery published a paper called "modular multiplication without a reduction step" [17][2]. The idea behind the paper is that we enter into a magical world called the Montgomery representation. When you step into the Montgomery world, modular multiplications do not require a reducing step and when you finish you just step out of this world and you are back with your result. It's still cost you like a multiplication but it doesn't cost you the extra reduction step. So, what is the idea of the Montgomery reduction? We want to calculate $g^e mod n$ and to do that we need to pay a lot for modular reduction. So, the first thing we do is enter the Montgomery representation and do the Mont(g^e) and each one of this multiplication steps is going to be about as difficult as regular multiplication (Figure Figure 3.5). After we finished with that we exit the Montgomery representation and then we have our result. Entering and leaving the Montgomery representation costs as much as modular multiplication but in the middle it's as cheap as regular representation.

[2]https://www.hackersdelight.org/MontgomeryMultiplication.pdf

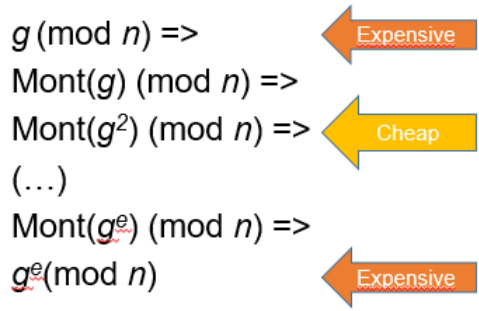


Figure 3.5: General sketch of Montgomery Exponentiation

Now lets review how the Montgomery Exponentiation works inside

1. Choose a value R, $R > n$, which is easy to use (usually a large power of 2)
2. $Mont(a) = a * R(modn) =^{def} a(modn)$
3. $Mont(ab) = a * b * R(modn) = \underline{a} * \underline{b} * R^{-1}(modn)$
4. $= (\underline{a} * \underline{b} + (\underline{a} * \underline{b} * n'(modR) * n))/R(modn)$
   // if this is more than n, subtract n
5. $A = A * A = 4 = 7^{(11<<1)} = 7^{(110)}$
6. Result: Instead of modular reduction, we only (sometimes) subtract

The first thing to do is to choose a very large value R which is larger than n and should be easy to use (a large power of 2) for instance 1 and 1000 bits of zero. What does it mean easy to use? To multiply and divide by a power of 2 you just shift left and right. To calculate the modulo of a very large number that is a power of 2 you do bitwise and with this large power of 2. if $R = 100000$ and $x = 10101010101110$ so to do a modular reduction we just take the lower bits which are 01110 in this case. So we see it's very cheap operation

with R, but R is not useful outside the situation. So to enter the Montgomery representation we are going to multiply by R. This multiplication is modulo n, and this is a bit expensive. But how do we know that R is inside the multiplicative group? How do we know that multiplying by R doesn't throw me outside of $mod n$? The answer is: how do we know that 2 is inside the group? Because what is this group? This group is a multiple of two prime numbers, and they are odd. Thus 2 doesn't divide either one of them so 2 is in the group and $2*2*2...2$ is in the group. We call the new number $\underline{a}$. The idea is that the numbers in the Montgomery representation is cheap so how is it cheap with $\underline{a}$ If we multiply a and b in the Montgomery's representation It will be: $a*b*Rmodn$. but if I want to do that using **a** and $\underline{b}$ we get: $\underline{a}*\underline{b}R^{-1}$mod n because of the extra R. So every time we multiply two numbers we need to take out the extra R. Inverting R is simple using GCD and can be done before we start the computation. There are much more derivations made to get to $(\underline{a}*\underline{b}+(\underline{a}*\underline{b}*n'(modR)*n))/R(modn)$. $\underline{a}$*$\underline{b}$ is just a single multiplication. $\underline{a}$*$\underline{b}$*n'(mod R) - Notice that $modR$ is a cheap operation because R is 1 with lot of zeroes. (n prime (n') is just precomputed number that doesn't really matter to us). Then we multiply it by n (another multiplication) and then we divide it by R which is also a simple operation. So, we have 4 multiplications and we need a modular reduction which is the main problem. Montgomery proofed that this sum is no more than 2^n. So how do you do a reduction if the number is between 0 to 2^n? You put an if statement. If x is less than n you do nothing, if x is larger than n you subtract it from the number n. So now, instead of division we do a couple of multiplications over integers which is not that expensive and then we sometimes subtract. This is a lot cheaper than the basic option and it is widely used. Notice that the if statement in the algorithm, which is influencing the execution time and might enable a timing attack.

3.2 Temporal Side Channel on RSA

Let's review how to do a temporal side channel attack on RSA. Kocher described this attack on his paper in 1995 among other things. If the RSA runs slower than there are more subtractions and by timing the execution we can recover the key. So how this can be done? How to recover the key by timing the execution? There was a fantastic paper "A Practical Implementation of the Timing Attack" [18] which the remainder of this section is based on. So what is the game here? Assume you are an attacker who wants to commit a timing attack on the secure implementation of RSA. If we have some device (e.g. a smart card) that we can send him requests, for instance "please allow to pay 1000 dollars to Alon" (see Figure 3.6). The smart card check that I have 1000 dollars in the bank account and then it replies "I approve this transaction and sign it". So this card has stored a value inside which means he has money inside. On the other hand, we want that if we request "pay Alon 1 million dollars" the smart card will say "I don't have enough funds! I am going to refuse". As an attacker we would like to be able to sign any message we want, mainly messages like "please send Yosi 1 billion dollars". The smart card will not allow it but if we got the private key (signing key) we can sign whatever message we want. So we send the smart card a message that is unsigned and he in return sends back a signature which is the response signed with the private key: $m^s mod n$ where s is the secret key. We assume that as an attacker we can send as many queries as we want and we and recover the responses (the signatures). The goal is to extract the secret key.

So, let's look a bit closer on the attack model - what messages we can send to the smart card? Can we send any message we want? The answer is that we can only send valid messages - if the message is not valid then the smart card will just throw an exception. This is somewhere in the scale between the weakest model and the most powerful model. In this

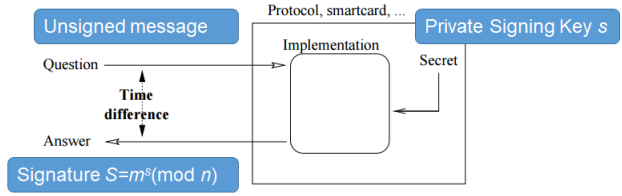


Figure 3.6: The timing attack principle

scenario the most permissive attack model is a known plain text. Known plain text means that we can see the messages as they are go in to the smart card but cannot change them. So the next thing is choosing the plain text. We can't just chose any plain text, it has to follow a certain rule. The next thing is completely chosen plain text which doesn't enforce any rules, and the last thing is adaptive plain text which means we can look at the response and we can choose the next query that we will send. So, what is the attack model, we send requests, the smart card is signing them, and we get the responses and also assuming that the smart card is using Montgomery RSA . So how can we use it to extract the key? We are going to use a method calles "Vaizata" method which can be found in the DPA handbook. This is a general way of performing a side channel attack using statistics.

The "Vaizata" Method

- Make a simple assumption about the implementation
- Guess a little part of the key
- Make hypothesis about the effect of the guess on the execution
- Classify the measurements according to the hypothesis
- If we guessed right, the classification will be statistically meaningful

How does it work? First, we make a simple assumption on the implementation, an assumption could be: We assume that the implementation run on software (the other possibility is hardware) what does it gives us? Software is executed serially and in the hardware it's not the case. We can assume for example that the key is stored in a flash memory on the device, and every time we need to use the key then we need to read the flash memory. How can we find that this is the case? How can we find first of all that a device is on using the hardware or the software? One way is to look at it - we can open the screws and look with a microscope, or we can go to this wonderful website called "I fixed it". They disassemble all sorts of devices and share this information. What is the sign that the device is using software? If there is an update on the firmware, because when it wakes up it needs to find out what software to run. We can say that this device uses memory, and we can also say things about when the device is doing the encryption. Let's assume the device under test is a remote controller. Inside the controller there is a secret key stored and also a counter. Whenever the button is pressed the remote controller constructs a package containing the serial number of the controller, the counter and a boolean state of the button (which ever button was pressed). Then, it sends it to a car and the car decrypts it. Why do we need a serial number? Each ECU in the car has a program to accept remote controls. So why do we need the counter? Without the counter, an attacker can repeat a message sent from the remote controller to the car only by reading the messages and sending them again. What happens when you press the button and the car is in the train station? The counter in the remote controller is out of sync with the counter in the car ans there is a window of counter values the car will accept. If its closed, the car will open without complaining. If it's a little closed we will need to press the remote control twice and when the car will see consecutive values and it will open, but if it is too far it won't open. Let's assume we can find out when the controller is transmitting (it is a very intensive operation that takes battery life and also radiates). So we

know the moment in time when it is transmitting, did the encryption happened before or after the transmition? The ansewr is before. Let's assume that the counter stored in the memory and let's say we found the moment in time when the chip is reading from memory. Did it happen before or after the encryption? The answer is after, since we need the counter to be included in the message that will be sent. We can also make more assumptions such as "the AES uses an 8 bit data pack or 16 bit or 32 bit". Some of these assumptions might be wrong but the "vizata" method will help us to find out if they are wrong.

So first of all we make a simple assumption, that the smart card is using left to right binary exponentiation using Montgomery and this is a very reasonable assumption because most implementations are using this method. The next thing to do is recursively (or inductively) guess a little part of the key. if we guessed the whole password it would take us exponential time but if we could guess a small part of the key each time it would take a linear time. So, we will try to guess small parts of the key first, and maybe the simplest thing to explain is guessing one bit or a single character, but lets say we are guessing a small part of the key. So now we know the beginning of the key and there is a little part we don't know. The next step is that we need to make a guess about what kind of effect our guess is going to have on the computation. We can say, for example, that if we will guess the bit correctly then something will happen to the computation, and if we will guess this key bit correctly it will take more power/take longer time/connect to the network more often or some kind of other phenomena we can measure. So, in our case what is the only thing we can measure? Answer: The anser is time. We assume that if there is a Montgomery reduction in the calculation of this bit then the entire computation is going to take a little more time.

SO what is going on here? There is a very large computation here and we were able to guess the beginning of it but not the end of it. Now we are going to classify the measurements

according to our hypothesis, in this case two groups (with or without Montgomery computation). If we guessed correctly then it will take longer to the group we said it will take longer. If not, it might take less or more, we can't detrmine. If we guessed correctly, the groups will have meaning, means will be able to statistically tell apart the set of the measurements that will take a longer time and the set of measurements that will take less time. We are going to guess the left bit of the key, and there is only one option. The next bit can be 1 or 0. Now, we can simulate the running of this algorithm with our guess, not for the whole key but only to the part we know, and if there is going to be Montgomery reduction. So assumeing we have g, e and N. The device under test is calculating $g^e mod n$, but where g came from? It is supplied by the attacker. What about e? Secret we want to discover $(e_t, e_{t-1}, .., e_0)$ What about N? public variable (known). The first thing it does, it enters the Montgomery representation. This information of how to enter the Montgomery representation is known to the attacker. It starts with g, and then g becomes $Mont(g)$, but the attacker can also do it. First $A = 1$, then $A = A * A$ the next thing is $A = A * g$ because we know that the most significant bit is 1. The next step is $A = A * A$, now what is next? It depends, if $e_{t-1} = 0$ then $A = A * A$ (skipping to next bit). Else if $e_{t-1} = 1$ then $A = A * g$. What happens next now we can't know. We can run both calculations, in particular we can find out if there was a reduction step in two optional operations. There are 4 options: Only one of then containing a reduction step, two of them containing a reduction steo or nither of them. We can know exactly, assuming we make a guess on e_{t-1}, if there is going to be an extra reduction step. We know enough to guess - if we guess correctly, we can know if there will be a reduction step because we can calculate all the alternative options completely. So, let's do this now, we have many different g's, and we have repeated this step many times, and for each of these g's we know if there is going to be an extra reduction step. So, now let's see how we can do an attack using this information. So, there is private key s, a public

key v and a signing operation $m^s mod n$.

We begin the attack with a bag of messages, all of them are valid, and we send them to the device under test (DUT). The DUT signs these messages (k messages), and for each one of the messages we get a trace, which is the data we collected using the side channel attack in this case it is only the time. Now we have a vector of size k and each element in the vector is the time it took to sign the message. Now, we are going to try and guess s_t, s_{t-1} ($s = s_t, s_{t-1}, s_{t-2}, \ldots, s_0$) and try to discover s_{t-2}.

So for each of the messages and each key guess we are going to simulate the computation as far as we already know and in addition for the parts of the key that we don't know we are going to simulate twice - one with a 0 and one with a 1. We are going to find out where the extra reduction step happens. If the next bit is 0, then some of the messages have extra reduction and we classify the message into two bins, those who got extra reduction and those who didn't. But maybe the key is not 0? maybe its 1. So we can simulate the same thing with the next bit as 1, and find out different set of messages with an extra reduction step. We can't find what the bits are but we can make a guess and simulate on both 0 and 1. So, now we divide our traces into two groups in 2 different ways, if the next key bit is 0 then m_1, m_2, m_4, m_5, m_7 got extra reduction and m_3, m_6, m_8, m_9 didn't. If the next key bit is 1 then m_2, m_3, m_5, m_8 got extra reduction and m_1, m_4, m_6, m_7, m_9 didn't (see Figure 3.7).

If the next key bit is 0	**If the next key bit is 1**
These messages get an extra reduction: m_1, m_2, m_4, m_5, m_7 These messages don't get an extra reduction: m_3, m_6, m_8, m_9	These messages get an extra reduction: m_2, m_3, m_5, m_8 These messages don't get an extra reduction: m_1, m_4, m_6, m_7, m_9

Figure 3.7: Key bit guess simulation

If we guessed correctly what can we tell about the messages that got an extra reduction? Their runtime will be a little longer if we guessed the key bit correctly and If we guessed in-

correctly it means we divided into two random groups which means the runtime will be similar in both groups. So if we guessed correctly the difference in runtime between the groups will be measurably different. So now we are going to change the discussion about the messages into the traces.

If the next key bit is 0	**If the next key bit is 1**
The statistics of the set of running times t_1, t_2, t_4, t_5, t_7 Should be measurably different than the statistics of the set t_3, t_6, t_8, t_9	The statistics of the set of running times t_2, t_3, t_5, t_8 Should be measurably different than the statistics of the set t_1, t_4, t_6, t_7, t_9

Figure 3.8: Guessing correctly the key bit makes the statics measurably different

if the next key bit is 0 then the statistics of the runtimes of 0 bit with extra reduction are going to be different from the runtimes of 0 bit without extra reduction. If we were wrong then the runtimes of 1 bit with extra reduction will be different from 1 bit without the extra reduction. Notice that we are not saying average or mean anywhere, because they are not required, it could be the variance changes or something else. The point is that there is some kind of difference that we can measure. So, how can we find out which of these two divisions is the correct one? Answer: we have two divisions of k traces and we measure the distance of the means. We are going to calculate the mean runtime of each part (0 bit with or without the extra reduction and 1 bit with or without the extra reduction) and subtract between with-/without extra reduction in each bit guess. If there is a large distance of means of the 0 bit guess as oppose to the 1 bit guess then probably the splitting of traces in the 0 bit guess is more meaningful than the 1 bit guess. If we are able to split meaningfully then we guessed the key correctly. Now, let's go back into statistics and talk about the T-test [19]. The T-test was invented by William Sealy Gosset [20] who was a chemist who worked for a very famous brewery in Ireland (Guinness). So, what is the idea? We have two populations that are different in some way. There are two kinds of t-tests, pair T-test and unpaired T-test. what is the paired T-test?

lets assume we are going to a tree and we taking the leaves that fall off the tree. We notice that the leaves that fall on the south side have less mold that the leaves that fall on the north side. Why is that? Because there is more sun on the south side that is drying the leaves.

How do we prove our theory? We send our undergrad research assistant to collect a bag of leaves form the north side and a bag of leaves from the south side and tell the undergrad to count the percentage of mold in the southern and northern leaves. When the undergrad comes back, very exhausted, he provides us with an excel file that have 1000 leaves from the north side and 1500 from the south side. Now we want to prove our theory, each one of the leaves has a mold, We want to prove that there is a statistical difference between the two groups - this is the unpaired T-test. What is the paired T-test? It is when we are talking about something which we can identify as pairs. For example, we are developing a cancer medicine and we want to test it. Now, we don't take any undergrads but only very sick mice with cancer. We measure the weight of the tumor in order to have a list of 100 mice and tumors. Then we divide them into two groups, one we treat with a medicine and the other we don't. In the end of the experiment, we measure the tumors again, but now each measurement is a pair, one before the treatment and one is after. We want to say that the size of the tumor is smaller after the treatment. Now let's review the student's unpaired T-test demo in Matlab. The mat in Matlab is for matrix, we can define matrix like this:

```
>> x=1:10

x =

     1     2     3     4     5     6     7     8     9    10
```

Figure 3.9: Defining a matrix of size 1x10 with increasing numbers from 1 [1,2,3,. . . ,10]

The function randn(1) which creates normally distributed

```
>> x+2

ans =

     3     4     5     6     7     8     9    10    11    12
```

Figure 3.10: Adding the matrix with 2 increases all the numbers in the matrix by 2 (Same with multiplication and log)

```
>> x>5

ans =

  1×10 logical array

   0   0   0   0   0   1   1   1   1   1
```

Figure 3.11: $x > 5$ will result with logical array [0, 0, 0, 0. . . , 1, 1, 1, 1]

random variable which follows a Gaussian distribution which means the variance is 1 and the mean of 0. What is the minimum value this function will output? Answer: None, but statistically the value is closer to 0. How can we create a random variable with mean different from 0? Answer: add a constant to the mean, if we want mean N. we will do N + randn (1). We can even create a vector of randomly chosen numbers using randn (1, 5) + 5 : [4.9369, 5.7147, 4.7950, 4.8759, 6.4897] and if we will do randn (1, 5)*.1 + 5 the numbers will be even closer to 5. So, now we are ready to try the student T-test.

Looking only on the graphs - can we say they are from the same distribution? Answer: we can't be sure. We want to run the T-test to find out if they are statistically significant. There is a hypothesis and we need to either reject or accept the hypothesis. If H_0 then they are from the same distribution and if H_1 than they are from different distributions. If we run the code, the result will be that they are from different distributions with 0.959 certainty. Each time we run we can get different results, if the certainty is smaller than 0.95,

```
mu_1 = 10;
mu_2 = 20;

sigma2_1 = 4;
sigma2_2 = 4;

vector_length = 400;

x = randn(1, vector_length) * sigma2_1 + mu_1;
y = randn(1, vector_length) * sigma2_2 + mu_2;

plot(x,zeros(1,400),'*', y, ones(1,400),'*');ylim([-2,3]);figure(gcf);
```

Figure 3.12: In the code, 2 vectors are created - x and y in the same size (which is not obligatory), x will be randomly distributed with a mean of mu_1 and variance of $sigma2_1$ and vector x is going to be random distributed with a mean of mu_2 and variance of $sigma2_2$. Then there is a code for plotting both vectors (in different colors).

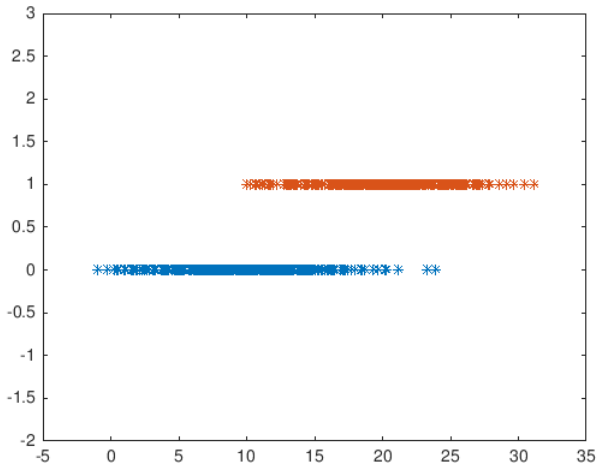


Figure 3.13: Plotting of the vectors for $mu_1 = 10, mu_2 = 20$

the hypothesis is rejected. How can we make it more difficult for the ttest2? Answer: if we change mu_2 to 11, then they will be very close distributions.

How we as attackers can handle this situation? Answer: run many times. So we change the vector length (for example to 50000). There is something called power analysis, which means given these parameters (mu and $sigma$) how many

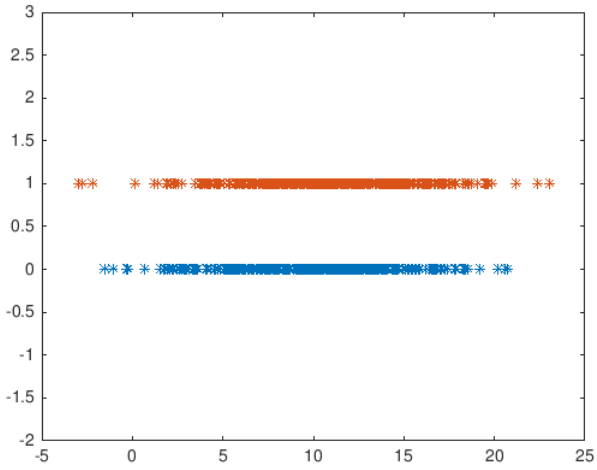


Figure 3.14: Plotting of the vectors for $mu_1 = 10, mu_2 = 11$

measurements you need to run to be sure with 95 percent. As engineers we don't really care about T-tests, we have a budget of measurements and we just take the one with the larger mean distance, but what if we are wrong (guessed 1 instead of 0)? After we guessed one bit wrong all the bits afterwards are wrong because they are simulated wrongly. So, how can we simulate a full attack? If we guessed 1 bit using the differences of means then we continue to the next bit in linear time. Let's review a figure from " A Practical Implementation of the Timing Attack" which you are encouraged to read.

Now let's talk about counter measurements. When Kocher announced the attack to the cipherpunks mailing list there was kind of discussion about it. Here is a message (see Figure 3.16) from there that was sent by Ron Riverst. He was replying to William Simpson, who was the author of Photuris which is related to the IPsec protocol and was used for kits change in the IP protocol.

When he read that this attack can attack Photuris, Bill replied: don't worry this will be fixed in Photuris. How to do this? By dithering the return time of identification message a

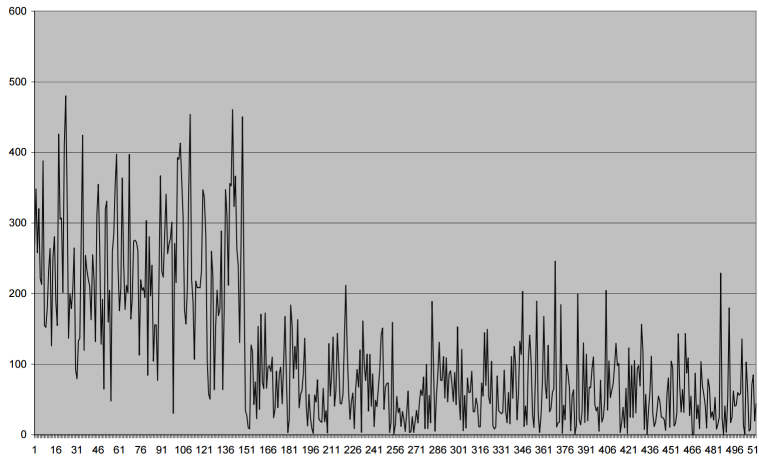


Figure 3.15: x axis – is the bit index from right to left (the left most bit is known to be 1) and y axis – is the distance of means we chose (some times its 500 or 100 but after 151 the distance of means is a lot smaller which means we guessed a bit wrong). How to solve it? Answer: go backwards and backtrack.

Paul Kocher's timing attack

- *To*: ipsec@ans.net
- *Subject*: Paul Kocher's timing attack
- *From*: rivest@theory.lcs.mit.edu (Ron Rivest)
- *Date*: Mon, 11 Dec 95 13:29:32 EST

```
Bill Simpson says, regarding Kocher's timing attack:

        This will be fixed in Photuris by dithering the return time of the
        Identification_Message.  A few extra milliseconds on top of a second
        won't be a problem.  ...

This helps to reduce the leakage of key information, but does not
eliminate it.  The best thing to do is to ensure that the public-key
computation time is CONSTANT, independent of the message being encrypted
(or signed).

[...] (As a side note, I suspect that this sort of attack is probably extremely
difficult to mount in an Internet environment, due to packet-routing
timing variabilities.  However, it's wise to be careful...)

        Ron Rivest
```

Figure 3.16: The message.

few extra milliseconds. Which means he is doing mitigation, as he is not adding a random delay because random is very expensive, he is going to finish the calculations and is going to look at the clock and exactly when the millisecond changes (or second) send the package. What does it mean? It means that since the beginning is random then he is going to add a random delay. So, what Ron Rivest replied? It will reduce the data leakage but will not eliminate it. Why so? How the attacker will overcome this? He will need to measure more, this is network-based protocol, the attacker can measure as many times as he wants. He says, in addition, the public key computation time should be constant and independent from the message being sent. Ron Rivest suggest prevention as a countermeasure. So, what time it is going to be? Answer: the worst-case time. he adds a side note that this kind of attack is very difficult to mount in an internet environment due to packet-routing timing variabilities, however it is wise to be careful. Few years later a demonstration was presented over the internet, there were more measurements to counter the routing problem. Adding noise is sometimes the only thing that works, if you add enough noise to delay the attack time up to a year the message might not be relevant by the time the attack succeeds.

So, let's talk about two more counter measurements, which are preventions. The first one is called RSA blinding. RSA blinding is a prevention counter measure which actually works on average time and not on worst-case time. if we have a secret key S and want to calculate $m^s mod n$, and the attacker gives us m and we don't want to leak s. We showed that with enough attempts from the attacker he will eventually retrieve s.

So, how do we do blinding? First of all, we can do this even before the attacker arrives, we generate random r and calculate $r^v mod n$ and $r^{-1} mod n$. These calculations are not reviling any secrets because v is the public key. The attacker gives us m, so we calculate:

$$X = (r^v * m) mod n$$
$$Y = X^s = (r^v * m)^s = r^{vs} * m^s = r * m^s mod n$$
$$//v * s mod n = 1 mod n)$$

Now, Y is leaking information because we raise a number to the power of the secret key, but r is a random number which the attacker doesn't know so he can't simulate the execution. How we remove r? We just calculate

$$S = Y * r^{-1} = r * m^s * r^{-1} = m^s mod n$$

Why doesn't everybody use it? Answer: because it's expensive, 2 modular exponentiations instead of 1. Another problem is the random number generation, its hard to find random number generator. You can see RSA blinding in openPGP.

Now let's review another countermeasure. It called square and always multiply (see Figure 3.17). It is very similar to the square and multiply, just instead of only when d_i is 1 we will always calculate the multiply but the assignment will be only when d_i is 1. The problem with this is not the extra computation that came from turning sometimes to always. Speculative execution is always looking for instruction to execute, how does it decide if it will execute an instruction? If all it's dependencies are met. If I say $a = b * c$ and $e = b * c$ they can run both in the same time. What happens is when the instruction brought to the CPU there is actually nobody is waiting for t, so as soon as it finishes to run the $s = s * s mod n$ and d_i is equal to 0 it will just return s. Moreover, the compilers are also capable of detecting such cases and optimize them by dismissing the else statement. So if we have a very simple CPU with no speculative execution no compiler and we wrote it in assembly we won't be able to attack it, but we could use power analysis.

If we run this algorithm as is, we have two places in memory for s and for t, every action we load s then multiply and then

Algorithm 2 Multiply-always binary exponentiation algorithm

```
s := 1 // set signature to initial value
for i from |d|-1 down to 0 do: // left-to-right
    s := s * s mod n // square
    if (d_i = 1), then
        s := s * m mod n // multiply
    else
        t := s * m mod n // multiply, discard result
return s
```

Figure 3.17: Square and always multiple algorithm

edit the memory of s. Same goes in the multiply section. We load s and m and then multiply and store it in s. It's always loading and storing s, but what happens when it goes to the else statment: it loads s and m and then store this into something other then s. So the power consumption is different between the store and the load. You can read more in the paper ("defeating RSA multiply-always and message binding" by Marc F. Witteman et al. [21])

Side channel attacks can get not only computer secrets but human secrets too. What exactly is a human secret? Browsing history for example. How does the website figure out our browsing history? Theoretically, we can delete the history in the browser. But what happens when we click on a hyper link (blue link)? It turns purple after the click. So, there was a nice trick that websites used to do, they attached the link to an HTML element and added JavaScript code that at the change of the color can now update that you have visited the site. The world wide web consortium decided that it was a privacy leak and you are not allowed to read the color of an HTML element anymore. Now we can set the color but can't read it. One of the speakers in the black hat 2013 used timing attacks to find out if a web site was visited or not[3]. He also demonstrated how he can also use timing attack to read the user's stream.

[3] https://www.youtube.com/watch?v=KcOQfYlyIqw

Chapter 4

Power/EM I

4.1 Electronic Circuits

A basic electronic circuit

The most basic electronic circuit consist of a power supply (i.e. a battery) that generates an electric potential that aim to move to the ground. And an electrical load (any component consuming electric power) connected to the power supply (Vdd) on one side and to the "ground" (the reference point from which voltages are measured) on the other side. As the electric potential goes through the load, it (the load) does some kind of a work. We can consider electric current to behave like water, in this example the water wants to go from the mountains (Vdd) to the sea level (Ground) and there are rivers and obstacles that tries to prevent it to do so. When the load has small resistance then more of the current will "flow" through it, and when the load has bigger resistance then the "flow" is smaller.

There are two different ways in which we can wire things together in an electric circuit, called series and parallel. When things are wired in series, things are wired one after another, such that electricity has to pass through one load, then the next load, then the next, and so on. When things are wired in parallel, they are wired side by side, such that electricity

passes through all of them at the same time, from one common point to another common point. The difference in the electric potential between the power supply and the ground creates an electric current which flows through the load toward the ground. The difference in electric potential between two points is measured in Volts (usually denoted by $\boldsymbol{V}$). The amount of current flowing through the circuit at a given time is measured in Amperes (denoted by $\boldsymbol{A}$). The electrical resistance of the load is a measure of its opposition to the flow of electric current through it. It is measured in Ohms (and denoted by $\boldsymbol{R}$).

Resistors

As the name implies, a resistor resists the flow of electrical current. The amount of resistance is measured in Ohms. A resistor is considered a passive component that consumes power that is dissipated as heat. The power rating of a resistor determines how much power it can consume without overheating.

Figure 4.1: Resistor Symbol in Circuit Diagrams.

Ohm's law

Ohm's law defines the relationship between the Voltage, Current and Resistance in a circuit: The voltage is equal to the current multiplied by the resistance of the load.

$$V = I * R$$

Since in most of the circuits we are using, the voltage is fixed (defined by the characteristics of the power supply), a change in the resistance of the circuit will cause a change in the

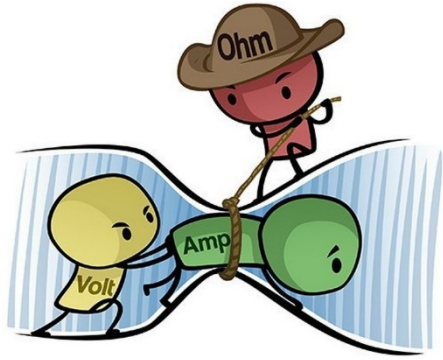


Figure 4.2: Ohm's Law.

current in the opposite direction. This means we can measure the current over time in order to calculate the resistance. A useful analogy for the relations between V, I and R is to imagine a fountain on a high mountain, where the water flow down through a river to the sea. The difference in height between the fountain and the sea is the Voltage, the width of the river can be thought of as the resistance, and the flow of the water is the current.

Power

Power is the rate at which work is done by the circuit and is measured in Watts. Electricity bills are measured by K Watts hour, i.e. 1 KWH is 1 kilo watts used over an hour, and this is the energy that we used and we need to pay for. $Power = Work/Time$

$$\text{Power consumption:} P = I * V$$

For example we can take a look on a phone; the battery can be measure in Milliamper hour, i.e. if the battery is 3000mil Amper Hour so if the current is 1 Amper then we can use it for 3 hours. If we want the battery to run out quickly, we can use services like streaming, flashlight and more. That means we have greater current that caused by leveraging the load

of the phone and the battery will run our much faster than before. The phone also, will get hot. If a device is getting hot then it sometimes uses its fans (noise), and so we can detect it for cyber usage.

Electricity can be used to do various kinds of work:

- Electromagnetic work (light a bulb, transmit a Wi-Fi signal)
- Thermal work (heating)
- Mechanical work (spin a motor, vibrate a speaker)
- Chemical work (charging a battery)
- Computational work (store or load from memory, compute a value)

Power Consumption

When the current leaves the circuit to the ground then we consume it as power, but sometimes we need to be careful as there are cases where the current is no leaving the circuit, like battery charging. In order to measure the power we will connect our measuring device between the load and the ground. The power consumption of a device is the work it does divided by time. It is measured in Watts ($\boldsymbol{W}$). The power consumption can be calculated as current ($\boldsymbol{I}$) multiplied by Voltage ($\boldsymbol{V}$).

Current and Voltage dividers

Before we take a look at two simple electronic circuits, we need to introduce two additional terms: A **short (closed) circuit** is a piece of wire with almost no resistance at all. The circuit is in a closed state and there is current in the circuit. In other words, it works as normal. An **open circuit** is a circuit which doesn't allow any current to pass through it. The circuit is in an open state and there is no current in the circuit; that's to say. It doesn't work.

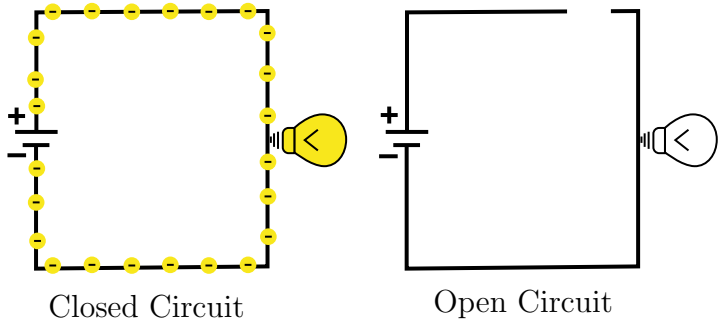


Figure 4.3: Open and closed circuits.

Connecting in serial

If we connect a short circuit between the load and the ground (See Figure 4.4), it will have no influence on it: the current will not change as from the power supply point of view – nothing has been change, we just cut a cable and put another one instead. The voltage drop will be very very low.

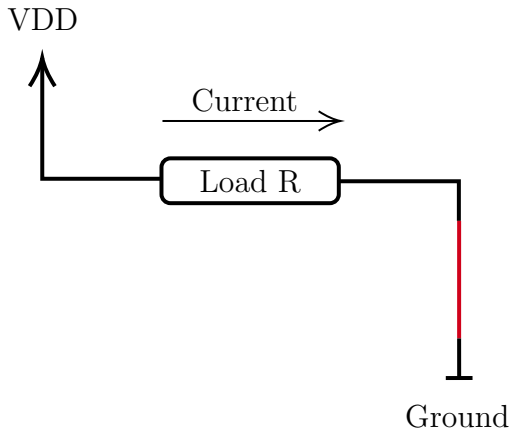


Figure 4.4: short(low resistance) circuit between the load and the ground.

If we connect an open circuit after the load (See Figure 4.5), it will increase the resistance to a very high value, causing

the current to become zero effectively. If the current is zero then the voltage is also zero (Ohm's Law).

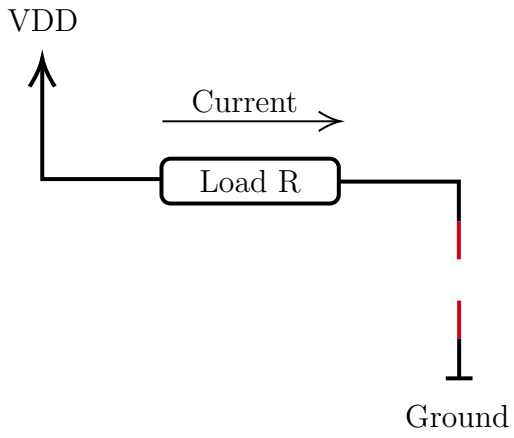


Figure 4.5: open circuit after the load.

Connecting in parallel

If we connect an open circuit in parallel to the load (See Figure 4.6), the current will flow only through the load path, so the current on the open circuit will be 0. However, the voltage drop between both points of the open circuit will be the same as the drop between the load sides.

If we connect a short circuit in parallel to the load (See Figure 4.7), the current will "prefer" flowing through it rather than through the load, so the current through the load will be equal to zero, while the current through the short circuit will be very high - by Ohm's law.

Since the cable is not a perfect conductor, some of the energy will be consumes in the form of thermal work, so the cable will heat, and possibly melt and start a fire.

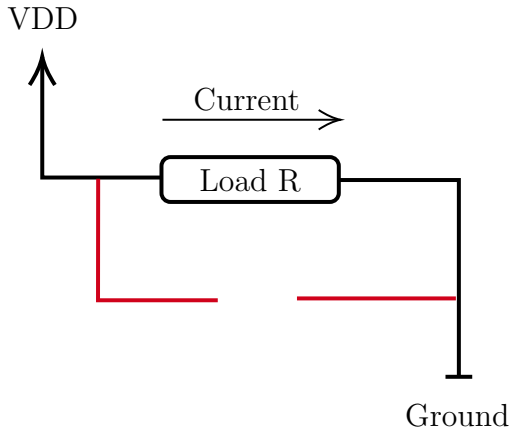


Figure 4.6: A close circuit in parallel to the load.

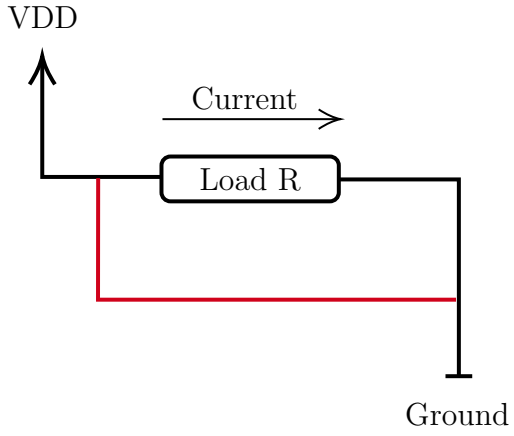


Figure 4.7: A short circuit in parallel to the load.

4.2 Measuring Power Consumption

Next, as an attackers, we want to measure the power consumption of this load, and to do so, we are going to use Ampermeter device.

Ampermeter

An Ampermeter (from **Am**pere **Meter**) is a device capable of measuring the amount of electric current going through it. It has very low resistance, so it doesn't interrupt the system connected to it.

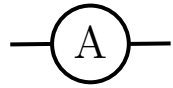


Figure 4.8: Ampermeter Symbol in Circuit Diagrams.

Using Ampermeter to measure power consumption

We need to "cut" the wire connected to the load and connect both sides to the Ampermeter.(See Figure 4.9)

Doing so will cause all current flowing through the load to pass through the Ampermeter as well, so we will be able to read the current at any given time. The resistance of the Ampermeter is very low so it will not affect the voltage that going through the load that will have the same voltage drop as before.

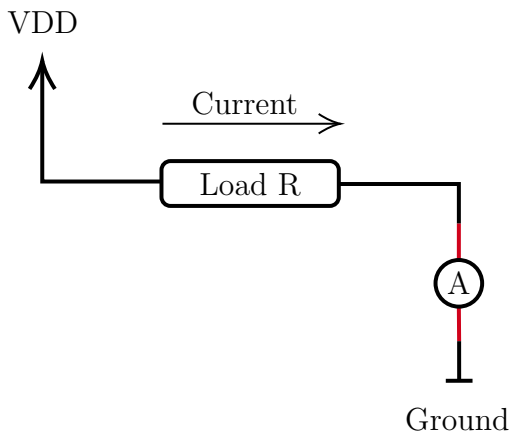


Figure 4.9: an Ampermeter connected in serial.

Now we will measure the current going through the ampermeter and next we will convert the current in order to find the power consumption of the load.

In case we know the voltage (i.e. a 5V battery, a 220V power socket), we can compute the power consumption: $P = I * V$.

The problem: sometimes, we don't want (or simply can't) cut the circuit after the load in order to connect an Ampermeter, and this is a way that the architect of the device are trying to protect it. So, let's see if we can use the ampermeter without cutting anything, and let's connect it in parallel to the load. (See Figure 4.10)

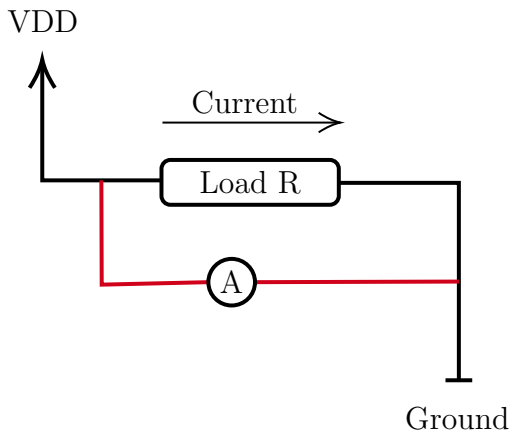


Figure 4.10: An ampermeter connected in parallel to the load

Connecting the ampermeter this way will burn the ampermeter as it has no resistance and so, all of the current will flow through it. So, instead of ampermeter we can use a voltmeter as follows (See Figure 4.11):

Voltmeter

The voltmeter resistance is very very high so the current will not go through it. The voltmeter is measuring the voltage drop between one side of the load and the other side of it. If we want to measure the current using the voltmeter we are

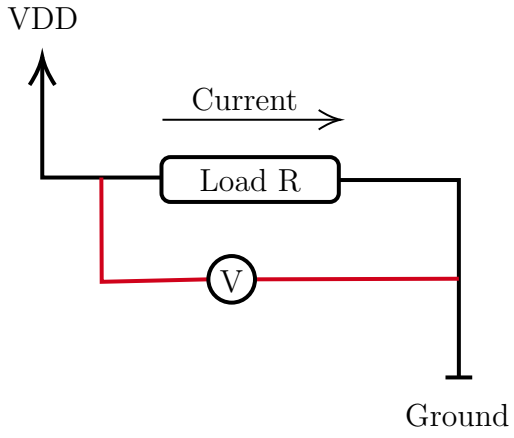


Figure 4.11: a voltmeter connected in parallel to the load

taking a load with a very small resistance and connect it as follows (See Figure 4.12):

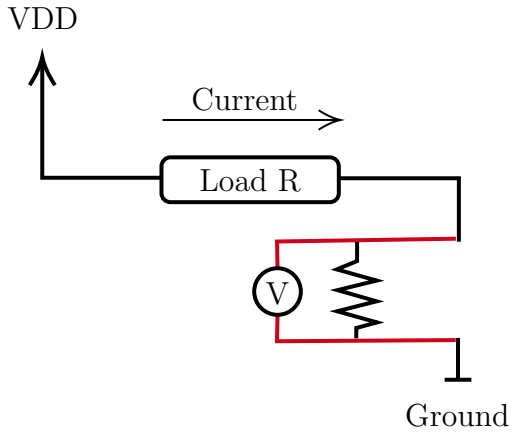


Figure 4.12: Circuit 8.

This way, because of the current divider, by connecting a Voltmeter in parallel with a very small and accurate resistor, we can measure the electric current using Ohm's law: $I = V/R$

Summary: we learned what is Power Consumption and how

we can measure it. A very important fact is that **Power Consumption varies with time!**. If we can find a relationship between the secret information we want to extract and the power consumption, we can recover this information by measuring the power consumption over time.

Types of electronic components

In general there are two types of elements in the circuit, the first one are Passive devices like a resistors (a passive two-terminal electrical component that implements electrical resistance as a circuit element), inductors (is a passive two-terminal electrical component that stores energy in a magnetic field when electric current flows through it), capacitors and diodes. And there are Active devices like transistors (a semiconductor device used to amplify or switch electronic signals and electrical power), amplifiers (an electronic device that can increase the power of a signal (a time-varying voltage or current)) and ICs. From our perspective, the Active devices are much more interesting for us (attackers) as they are using electricity in order to control electricity. One example can be an amplifier that has audio signal and power supply as inputs and it generates a greater audio signal as an output using the power supply.

Another interesting active element for this course is a transistor. In an integrated circuits, there are a lot of active devices such as transistors that if we can analyze their behavior we can learn about the data that they are processing. So when we look at the final consumption of these active elements, we can figure out some kind of secrets.

There are many kind of transistors and we will concentrate on understanding a certain type called Field-Effect Transistor.

Field-Effect Transistor

The field-effect transistor (FET) is an electronic device which uses an electric field to control the flow of current. FETs are

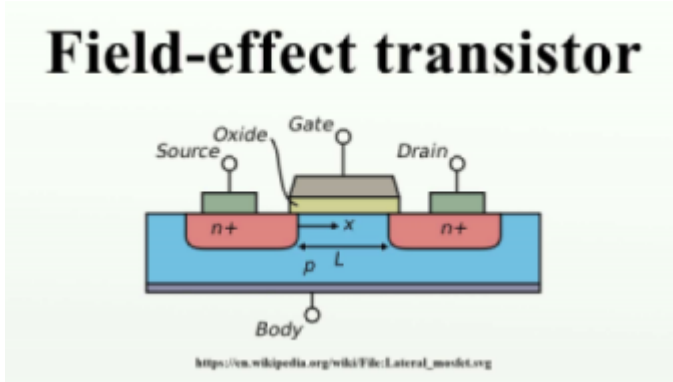


Figure 4.13: Field-Effect Transistor.

3-terminalled devices, having a source, gate, and drain terminal. FETs control the flow of current by the application of a voltage to the gate terminal, which in turn alters the conductivity between the drain and source terminals. In order to understand FET first we need to dive into the basics of semiconductors.

Semiconductors

A semiconductor is a substance, usually a solid chemical element or compound, that can conduct electricity under some conditions but not others, making it a good medium for the control of electrical current. Its conductance varies depending on the current or voltage applied to a control electrode, or on the intensity of irradiation by infrared (IR), visible light, ultraviolet (UV), or X rays. In general, transistors are made of semiconducting materials such as silicon. There are a conductor like copper or gold and there are insulators like plastic or glass.

Silicon atom has three parts: neutrons (not relevant for our use), protons with the positive charge (heavy) and electrons that have the negative charge and they are small and dynamic. Silicon atom has four electrons in its outer orbital and when we have a crystal silicon those 4 electrons are set

in place very nicely. It means that pure silicon is a very bad conductor as conducting means that the electrons can move around and in this case they are very comfortable where they are.

Metals can be good conductors of electricity as they have "free electrons" that can move easily from atom to atom, the electricity involves the flow of electrons. As all of the outer electrons in Silicon crystal are involved in perfect covalent bonds, they cannot move around. So, silicon crystal is nearly insulator and very little electricity will flow through it. We can change the behavior of the silicon and turn it into a conductor by doping it. In doping, we mix a small amount of an impurity into the silicon crystal.

There are two types of impurities:

- N-type – where phosphorus or arsenic is added to the silicon in small quantities. They both have five outer electrons, so one of them is out of place when they get into the silicon lattice. While having nothing to bond to, the fifth electron is free to move around. As electrons have a negative charge, this kind of impurity called N-type.
- P-type - where boron or gallium is added to the silicon. They both have only three outer electrons. So, when we mixed them into the silicon lattice, there will be "holes" in the lattice where a silicon electron has nothing to bond to. The hole is looking for an electron from a neighbor atom and when that's happens the hole is "moving". As the absence of an electron creates the effect of a positive charge, this kind of impurity called P-type.

How does Field-Effect Transistor work?

In the Field-Effect Transistor there are (as shown in Figure 4.13) n+ areas (N-type) and P area (P-type). The n+ area contains a lot of free electrons and the P area contains

a lot of 'holes'. When no electricity is connected to the gate, the free electrons from the N+ are moving to the holes so there are no free electrons within the semiconductor itself. That means electrons can't move from the source to the drain i.e. open circuit. When electricity is connected to the gate it charge a lot of free electrons to it. The free electrons in the gate can't move to the silicon itself as there is an oxide layer between them, but is pushes the electrons in the silicon down to the body in a way there are holes between the source and the drain. That way electrons can move from the source to the drain freely and we have close circuit.

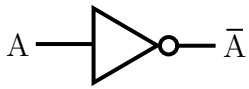


Figure 4.14: NOT Gate.

NOT Gate CMOS

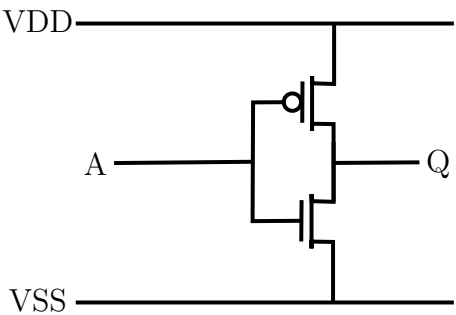


Figure 4.15: Circuit 9.

How does CMOS work? When the voltage of input A is low A = 0, the upper transistor's channel is closed and we have a connection between Vdd(1) and Q, so Q = 1, this is a Pull Up Network. And when the voltage of input A is high, then the lower transistor's channel is closed and we have a connection between Vss(0) and Q, so Q = 0, this is a

Table 4.1: NOT gate truth table.

input	A	0	1
output	Not A	1	0

Pull Down Network. A question is raised – when does this circuit consume power? There is almost no power consume as there is no connection between Vdd and Vss at any time. Although when CMOS is switching between states, there is a minor power usage and we will see how we can use it for our purpose.

Following is the NOT table:

AND Gate CMOS

Next we will see how to build a bit more complicated gate using 4 transistors. Following is the AND gate diagram:

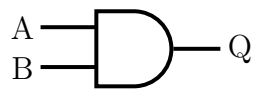


Figure 4.16: AND Gate.

We will implement the gate using 4 transistors to support the following AND table:

Table 4.2: AND gate truth table.

input a	0	0	1	1
input b	0	1	0	1
output	0	0	0	1

First we want to build the Pull Up Network, so for input A and B, for A=B=1 then the output Q is 1. Then we will build the Pull Down Network to support the other combination from the table to deliver 0 as the output Q. Sometimes

we don't want to have combination using the circuits, but we want to store information in it, next we will talk about another type of circuits called storage circuit or Sequential Circuit.

Storage/Sequential Circuit

A latch or flip-flop is a circuit that has two stable states and can be used to store state information.

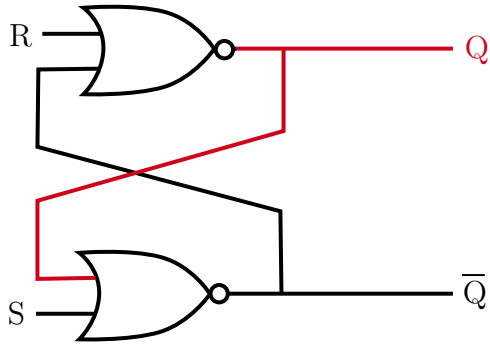


Figure 4.17: Flipflop.

The circuit can be made to change state by signals applied to one or more control inputs and will have one or two outputs. It is the basic storage element in sequential logic. Flip-flops and latches are fundamental building blocks of digital electronics systems used in computers, communications, and many other types of systems.

A flip-flop is a device which stores a single bit (binary digit) of data; one of its two states represents a "one" and the other represents a "zero". Such data storage can be used for storage of state, and such a circuit is described as sequential logic in electronics. When used in a finite-state machine, the output and next state depend not only on its current input, but also on its current state (and hence, previous inputs). It can also be used for counting of pulses, and for synchronizing variably-timed input signals to some reference timing signal.

Flip-flops can be either level-triggered (asynchronous, transparent or opaque) or edge-triggered (synchronous, or clocked). The term flip-flop has historically referred generically to both level-triggered and edge-triggered circuits that store a single bit of data using gates. We will refer Flip-Flop as edge-triggered i.e. clocked synchronized.

Flip-flop has to legs – data and clock, for each storage element in the circuit the data changes at the clock signal and so we have an amplified signal that we can monitor as attackers and try to learn the secret behind it.

Core I7 chip

Following is the core I7 chip image:

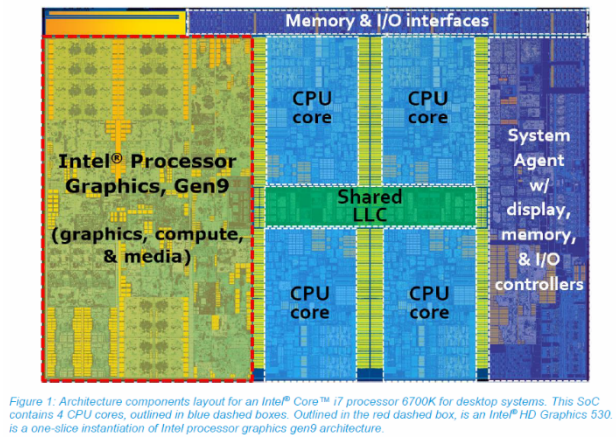


Figure 4.18: Intel i7.

We can see ordered items that are the storage elements, there are 24 items that are the GPUs and they have a little memory next to the (upper left). The CPU core has also a cash memory in it.

Power Consumption is Variable

Next we will see how to get secrets from the power consumption.

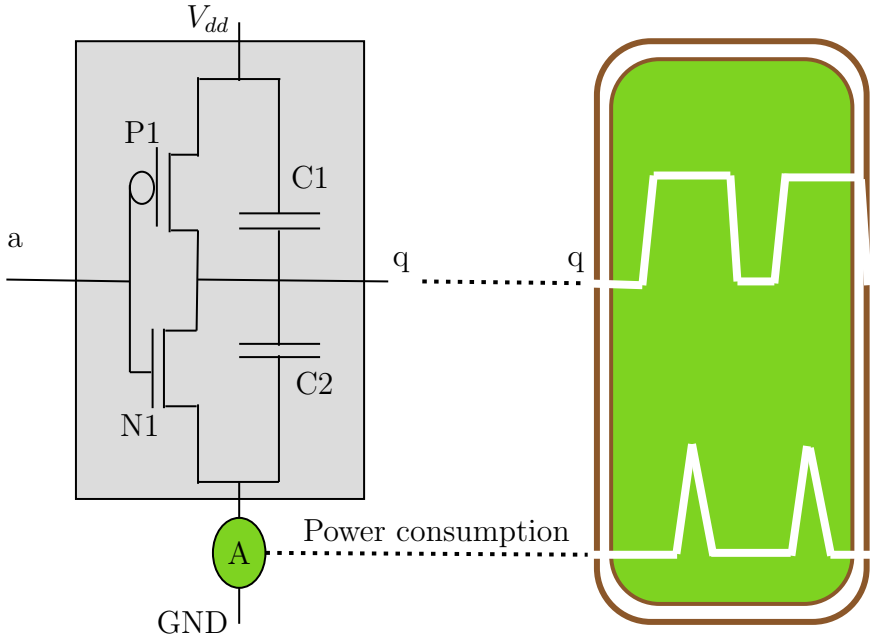


Figure 4.19: Not gate connected to an oscilloscope.

This is the NOT gate as before connected with an ampere meter so we can measure the power consumption, also there are two capacitors (C1 & C2). The capacitors are connected because the characteristics of the circuit which make it to act like capacitors. They are not transferring current but they will have an effect on the math we are going to do next.

Next we are taking a square wave, which looks like this:

And we have to feed it into the NOT gates, the output is going to be the opposite of the square wave, and we want to measure the power consumption using an oscilloscope. The x axis of the oscilloscope is time and the y axis in this case it's a voltage of the power. It's a current moving across the ampere meter. What we can see here is that when the circuit is stable, there was not much power consumption, but when the system is switching, when there is a high power consumption, because there is a glitch when one transistor

is opening and the other one is closing. Also, we can see that the lines in the oscilloscope are not having the same size as the capacitors are actually charging and discharging without we want it. We can learn from this if there was one or zero and what is the rate of switching, and we can store this information.

The power consumption is the sum of the statistic and the dynamic power consumption:

$$P(t) = P_{stat}(t) + P_{dyn}(t)$$

We don't care about the statistic power consumption as it is just defined by the kind of silicone that we are using, the size of the features decided to be in the transistors, the temperature, the technology, etc. But really interesting for us is the dynamic power consumption that depends on the o'clock rate and the circuit activity input data.

Because the power consumption is a function of the circuit activity, we can try to measure the power consumption and send the results to the equations and to get the circuit activity, in a way we can find out all the secrets.

In order to calculate the power consumption this way we need a lots of data about the circuit and about the environment and about the temperature etc. And this is too much work for us as an engineers. There is a much simplified way of measuring the power consumption of a device, assuming it's a CMOS device we can measure how many bits changes every clock.

So, our assumptions are the followings: 1. This is a CMOS device. 2. When it's static the static power is very low, and when it switches, there is a very high power consumption. 3. This is synchronous circuit, which means it has a lot of flip flops that all change at the same time. 4. The power consumption is proportional to the amount of changes and the outputs of these flip flops.

hamming distance model

For doing this we are going to use hamming distance model. The Hamming distance between two vectors is the number of bits we must change to change one into the other. Example, what is the distance between the vectors 01101010 and 11011011?

Answer: They differ in four places, so the Hamming distance d(01101010,11011011) = 4.

When talking about CMOS devices we are estimating our power consumption as amount of bits (transitions) that change from one to zero or from zero to one, this is our hamming distance. By monitoring the changes as explained we can find the power consumption.

But, this is not enough. By connecting the oscilloscope to our device we are trying to measure a physical device with another physical equipment, and what they are measuring is subject to measurement errors like switching noise, thermal noise and measurement noises.

Switching noise means that there are other kinds of activity which are going on in the circuit in addition. Measurement noise means that our desk setup is not always accurate, or we might be connected not properly, or we might not be measuring the precise correct moment of time etc. Thermal noise - we are measuring electrical activity, as electrons are very crazy particles and they like to disappear and reappear in a nearby location. So the higher is the temperature, the more likely they are to vanish and disappear. And when the electrons are moving, they generate radiation and they affect our measurements.

So now when we are measuring the power, we get the static power, the dynamic power and we get the noise:

$$P_{meas}(t) = P_{stat}(t) + P_{dyn}(t) + N(t)$$

In order to avoid the noise we can measure it again and again

and then the noise might be canceling itself. What happens sometimes is that the noise, especially the switching noise, is sometimes correlated with the activity of the circuits. Another thing we can do is controlling the noise somehow like running our experiment inside a freezer, or an isolated environment where there's no electric noise around. We can also, open the dives and try to kill the sources of noise.

That's one very nice thing we can do is instead of measuring power consumption, we can measure electromagnetic radiation. And this has two advantages, first of all, it's less invasive and second of all, it can be focused and, and reduce the switching noise.

From Power to Electromagnetic

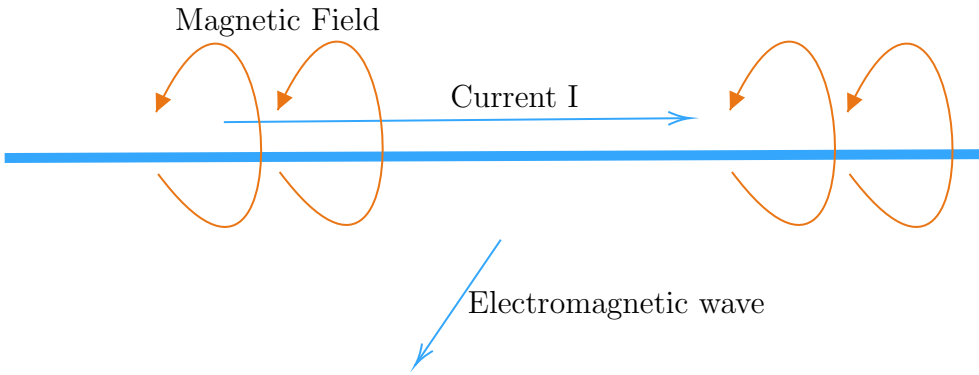


Figure 4.20: electromagnetic emission.

Electric fields are created by differences in voltage: the higher the voltage, the stronger will be the resultant field. We can use the right hand rule in order to remember the direction of the electromagnetic field direction when we know the current direction. We can measure the magnetic field, but one dis-

advantage is that it is very localize, so we need to get access to the device. We can connect an antenna to the device and then we can measure the electromagnetic wave using a receiver. There is a very nice paper called screaming channels where you can read more about this kind of attacks.

Chapter 5

Low Data Complexity Power/EM 2

We want to see what is power analysis all about and see a very simple power analysis attack that we can actually perform. Then we will dive into AES and its implementation.

All the things that we will see today are very optimistic and they assume that we have really good measurements, very good understanding and very good luck, but it's actually quite a little bit harder in practice.

5.1 The New York Times, Take 2

The hero for this article is Mister Kocher. He discovered something that can attack smart cards with what's called power analysis. Probably it was discovered before, but he has discovered it academically.

What did he do? He went to a conference and took people's smart cards and found out their secret private keys. We saw last week that if we have to sign something and we have the private key you can sign a thing that's not supposed to be signed and steal money, this was a big deal.

The 1995 timing attack went to the front page of the New York Times and there was a lot of discussion about the dis-

covery. The 1998 result about power analysis only goes to the bottom of page two in one of the supplements and actually nobody has discussed about it a lot.

Why didn't he get a lot of attention for this power analysis attack? Could be because timing is one thing and power analysis is a little more complicated but in the bottom line we can see all the secrets. To do a timing attack you need a stopwatch, while to power analysis attack you need more sophisticated equipment.

5.2 Power Analysis Attacks

What we need to make timing attack? We just need to be able to send request and gets responses to measured how much it took, while the target can be at the other side of the internet (can Amazon Cloud very far away).

What we need to make Power analysis attack? We need to be physically. How do we connect to the device? We need to cut the power supply and connect to it diractly to measure the power consumption. This is a very invasive attack, we need to be very very close. in 1995 let's assume that it's true. so if we go to the system architect:" listen there is an a power analysis attack that's can completely compromised the device. Attacker needs to come to the device end cuts the power supply. . . ." by the time you already lost the system architect because this is not practical. The threat model has to make sense.

nevertheless, the threat model does make sense in a lot of scenarios and we discovered that thing that we was not supposed existing these power analysis and side Channel attack. today we can attack this device and tomorrow we can attack another device.

To learn about power analysis attack you can go and search in the library the "Power Analysis Attacks" book."Cryptanalytic attacks that allow extraction of secret information from cryptographic devices by exploding their power consumption char-

acteristics"let's see what can we discover from this definition. First, What we are attacking here? Cryptographic devices. what we are not attacking? cryptographic algorithms. If I told you that I was able to break RSA on my phone by doing power analysis did I break RSA? No we didn't break RSA, we break the implementation of RSA. What else we are not attacking? The user is not under attack, we are not doing any key logging or human engineering attack or social engineering we are only attacking the device. What's the cryptographic device? Some kind of crypto, signing or encrypting. What a secret information we wants to extract? Probably we extract the key. And what's nice about the key? That he has lots of bits. At the start you don't know anything about the key, If you get half of the key you halfway, if you get the whole key You win. If I want to make the device more secure what do I need to do to key? we make a longer key.

This is very easy to say, but if I'm talking instead of secret information we cracked from the device, like medical information, it's a little harder to understand what we can do with half of it. How do we make the medical information more secure? What it's actually means?

Let's take credit cards, the companies say that you are more secure with credit card. Why do we think that we have a privacy with credit cards? We are giving the credit card number and our name... Looks like we're giving everything, the saying that we are actually have privacy, a lot of people have in the US have the same name, but you also get a zip code. Every time you pay with the credit card it generate a new credit card number automatically and doesn't have any digits on him. It is very hard to talk about privacy if it's not a cryptographic. What's left from the definition? we are exploiting the power conceptions characteristic.

What are we not exploiting? We're not doing math and we are not doing something that you can do by algorithms. It's important to know that exploiting the power at conception characteristics does not have to be actually measuring the power. We saw last week that we can actually measure

the power consumption with other methods. When Kotcher wrote his paper he announced three types of power analysis. One of them called Simple power analysis, the other one called differential power analysis. Let's see the simple power analysis today.

5.3 Simple Power Analysis

The simple power analysis means that I'm going to take the measurement of the device, making one measurement or two. With that we are going to get the key from those measurement. What's nice about this attack that it's very reasonable attack model. We need to get the power consumption trace (this is a vector overtime of the power consumption of the device), and we need only one or two traces. This is actually durable in a lot of scenarios even if I have the device for a little time or even if someone is looking at me. We will see the setup that can be used for simple analysis.

When you go to a store in Europe, you can't give the credit card for the cashier, they give you this terminale and then you need take your credit card insert it and put the pin number. And what is going over here, there is something like a cryptographic computer between your card and terminal. Let's assume that we want to attack the card, and it is in my possession. This model is very permissive to me and I can do whatever I want, I can do a lot of transactions I can do radiat, I can twist it and even melt it. so attacking the card is very easy.

But if I don't wants to attack the card? I want to attack the terminal using power analysis. Maybe the terminal has an SSL private key which is used to connect to the Central Center, and this can make a lot of damage and we can be very rich.

We want to do a power analysis on the terminal. This is a nice setup, this is something that looks like a credit card, but it has are wired connected to this fpga and connect to a

computer. I will be in a very cold country and I will take this card out from the jacket to my palm. Insert this chip into the terminal, while doing a power analysis on a this terminal. The idea is that I can do it about 1 or two times, but not making it for a thousand times or melt it. I can attack maybe this terminal or a gas station. The fact that we don't need a lot of traces is actually good.

What are the disadvantages? The problem here is that when I look at this power trace I need to be very very well prepared, and understand what's really going on in the power trace. Because we get a vector with a hundred thousand points and we need to understand where is the encryption starting? Where it is ending? What it means that there is a lack of power here a little power there? We need to have a very good understanding of the device processes.

Not only that we need also to have a very good measurements, because I only have one or two measurements. They're a lot of external Influence on the device, there can be noise, may be related to the device may be related to the environment, and if I have only one measurement I am going to get all of this noise at once and cannot do anything to reduce it. Statistically I'am not going to be able to get clean measurements to perform the attack.

Another problem is that I will need to work very hard to find the key. Let see an example: Yossi did a power analysis measurements and he got a Trace and there was a noise, he did his analysis and got the key. Now he want to check if this is the right key. How can we check the key? Try to decrypt and encrypt the private key. Is this the right key? no it's not :(.. Why? We have only one Trace. Maybe there was noise? Maybe one measurement was wrong? How do we recover from this? We can try a similar key and not there exact key, maybe to change you on bit here or there. This search might be so intensive that we getting basically the same effect like a Brute Force. This makes simple power analysis are not very effective because of all of those reasons.

5.4 Other Types of Power Attacks

So in general in power analysis there are two classes:

Low data complexity attacks where I get line trace or two traces (a very small amount of traces). Then I do a lot of post-processing and think really really hard and maybe do a reverse engineering before.

High data complexity attacks I will talk to you about it next week. These attacks require a lot of traces, maybe thousands or Millions traces, a Terabyte of data.

5.5 Tuning the power model

The first thing that you need to do for a simple power analysis attack is to understand what device you are attacking. We attack two general types of devices with simple power analysis, first is a microcontroller or a CPU and the other thing is ASIC.

Microcontroller is basically a regular computer, it gets commands and runs it one after another, if you want to write a new software for this computer you can write it in C or Java, they're cheap and commonly used.

Figure 5.1: Bitcoin Wallet

This is a Bitcoin wallet. The Bitcoin is basically numbers and if someone steals these numbers he steals your money, so you

don't want these numbers to be on your computers. These do all the calculation when you connect to the network, we want it to be very secure because if someone steal the secret key he can take all of your money.

Inside this device it has an orange Square and this is the microcontroller. Her some storage and you can write some code for it.

The other kind of a device is called ASIC, this device is manufactured only for a specific purpose. Let's say I want to have a sprinkler that starting the morning and ends in the evening, I will use a programming language that called HDL, once I compiled these software I am not getting an executable program that I can run on a device. These chips can only do one thing, I can't programming them and I can't upgrade them, if there is a bug I am in a big problem, but they have only one purpose. The power consumption of this devices are going to be smaller and sometimes they even be faster. Microcontrollers have programs that runs one line after another, an ASIC can do a parallel operation. ASIC is very cheap to manufacture, but there is a big wrap up before you produced this.

In this course we are going to talk only about microcontrollers, because we are going to see them more often than ASICs. But you will know enough to open the book attacking ASIC.

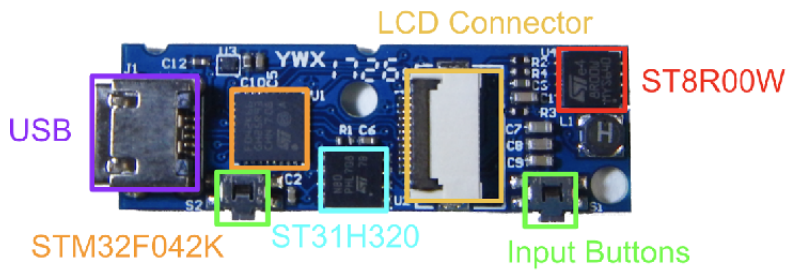


Figure 5.2: Arduino

So what is the line between microcontroller and an ASIC? On one of the student table there is a chip that he will show us, this is a FPGA field programmable this chip. If I want to identify a particular face I can program this ASIC to do patterning for this face and doing it very very cheaply. I can do also audio compression maybe I don't know what exactly I want to do but I know that I need to do some kind of compression while I don't know exactly the algorithm by using this ASIC. So you will find an ASIC if you have a piece of equipment 10000 pieces, maybe a router ,oscilloscope, submarine and things like that. Best to make sure what is an Arduino? Microcontroller.

5.6 8-bit microcontroller

This is an 8-bit microcontroller from the 80s, you see the center of this microcontroller , this pair of trousers the red that named ALU, this is where the magic happens, this piece of silicone can actually do logic like multiply, add, shift or compare.

The entire process of life is to get a line of code which represent an instruction, he has to understand what this instruction is trying to do. It can be multiply, read or write. And then you get the operator you need to do from the memory and fed it to the ALU. Then we'll get the next instruction and we'll do it again and again. what's important for me to show you that there is two long in lines from the top and the bottom they are called the buses. the top called the data bus and the bottom call address bus. What is data bus mean? Any sort of data you need to computation has to fetch from this bus. If something come from the memory it going from this bus. If I want to write to external input output, it also going on this bus. The width of this bus is 8 Bits and it means all of the operation that this microcontroller do our eight bits.

On the bottom there is another big bus which is the address

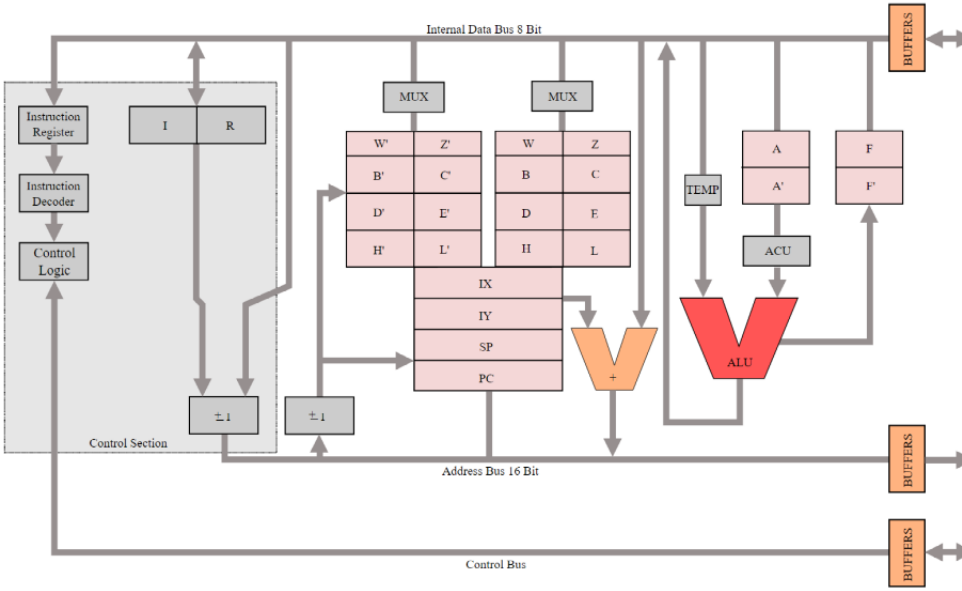


Figure 5.3: 8-bit microcontroller

bus. If I want to write to memory, I'm going to put their address in the address bus and then I'm going to put the data in the data bus. I will set the control bus to write, and then the memory which is somewhere else is it going to rise to this address. If I want to do a read, I will put the address I want to read in the address bus, and read in the control. What I will do in the data bus? I don't want to put something in there because I want to read, this is actually important and I will elaborate about it more later.

5.7 Power Model for Microcontrollers

What's interesting about microcontroller that there are in many cases the power model don't have Hamming distance and actually have Hamming Weight. What does that mean? That if I think that there are going to be a value going over

the bus I don't need to know what was the previous value on the bus, because it's going to be exact humming weight for this value.

When you have the best that is shared with several components the idea is that all of the components that are not using the bus are going to set how to put them, so all are ones, So if I want for example to read advice from memory the CPU is going to set everything to one and then he is going to extract the memory from the address, then the memory is going to lower all the relevant bits until what sitting on the memory bus and the data bus what I wanted.

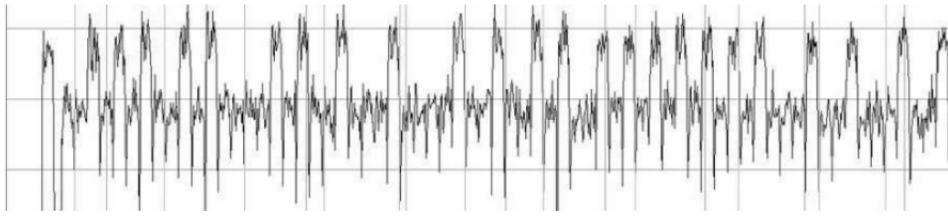
Let's say I want the memory 4, at first I'm going to see on the data bus 0xFF, then I will see 4 and then going to see agaoxin FF because the memory is finished. what was the power-consumption here to go from FF to 4, how many beats has to change? 7, and again it goes back to FF.

5.8 Simple Power Analysis of RSA

Let's see the power module this device under test, which does RSA decryption or signing. While it's doing it we are getting the power measures. Why it is doing an RSA decryption? Because it needs it. We can do it by sending encrypted emails from the phone, so you really have to decrypt the message.

The axis are vector of power measurements, x is time, y are the power consumption,(how did we measure the power consumption? i put a probe, measer the voltege so on.. What is the private key? What is missing? You need to do some reverse engineering first. If this is the only chance to get the key, I need to tell you a little bit more about the device so you will understand what is going on here. This device is microcontroller, it's doing RSA decryption using right to left binary exponentiation using square and multiply. Is it helping you finding the key? Yes. Let me show you the source code.

Binary exponentiation, it starts from the top most bit, off



Algorithm 1 Binary exponentiation algorithm

```
s := 1 // set signature to initial value
for i from |d|-1 down to 0 do: // left-to-right
    s := s * s mod n // square
    if (d_i = 1), then s := s * m mod n // multiply
return s
```

Figure 5.4: RSA Power Analysis

the secret and privates decryption key, and then for each bit we do Square and multiply. If the bit is zero we do Square, if this bit is one we do square and multiply. This help you now finding the key, let's look at that Power Trace.

We see two types of things here, this little thing and the big thing. We have some little thing, big thing, little thing, big thing, and then little little little, and then big thing.

All we need to do is to be able tell about were is the square and were is multiply, and then I can read the key, from top to bottom. So telling about square and the multiply to get to the key is the general method. We see square is take a little power and multiply more power.

What is square? square is multiply. So why is the power consumption of square using multiply is different? this is a microcontroller and his bus width is 8 bit. He's doing a convolution between numbers, so it's multiply thousands of time in frame. so why this is different? So what is consuming power, the ALU is consumer power, the data bus and the ad-

dress bus consuming power. In this case the power consumption module of the address bus is hemming distance, because it's not setting to 1 between accesses it's always containing what CPU is writing. So did you and multiplication, you are going to see a lot of difference values written into memory to the address bus because this microcontroller have very small component ALU always fetching addresses from memory so ding s*s it's fetching thousands of thousands of memory, but the address is fetching is very similar to each other, because they are all s. les say s is 1k bits in memory all the above are the same but the bathroom are different.But here we are doing s time m, so it's fetching s and m, and you doing convolution. so the address bus when it's doing S and M it's changing a lot, because he needs to do not only the bottom bits but also the top bits.

5.9 Simple Power Analysis of AES

I am going to attack an 8 bit microcontroller. Let's look on the setup. This microcontroller has a secret key, and it uses an AES encryption. We can ask him to encrypt and decrypt, we send the command in the serial line.

While he's doing this operation it is going to consume different amount of power and we would like to measure that. How do we going to measure them? We going to connect to the microcontroller power supply. But were do we need to cut and connect? The power is not going straight to the device it is going from this white box (small resistor). How much power is going to go through the resistor? Because he's small it's going to take very small power consumption. The voltage drops across this device is going to be measured by this Probe, and I'm going to measure the voltage over time. We know the voltage of the power supplies is 5 volt, so from this we can find out what is the current going through the microcontroller and from the current we can find out what is the power consumption. Do we need the code of the microcontroller? Yes. The only thing I don't know is what? The

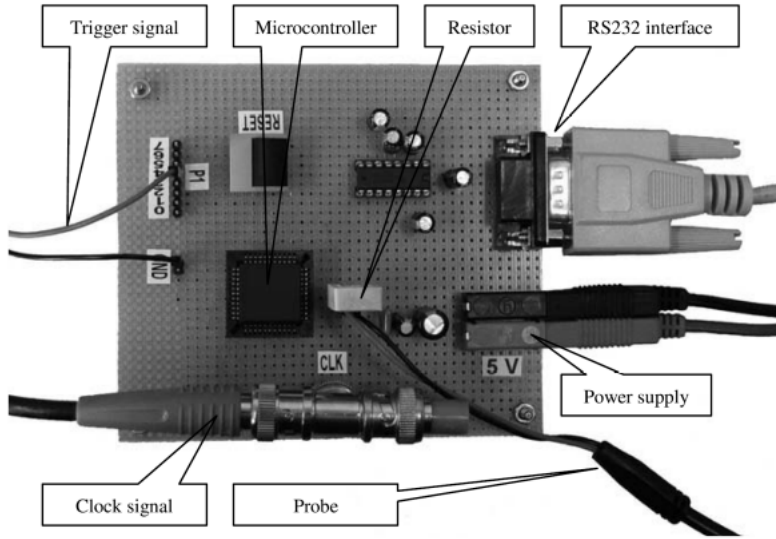


Figure 5.5: AES Setup

secret, but I know everything else.

5.10 Advanced Encryption Standard

What is the story about AES the encryption standard.

In Death Valley Days encryption was in military standards it was considered like a weapon you weren't supposed to have encryption, not by buy, export or sale. but in the seventies the US government what is it might be a good idea to have civilian encryption to protect civilian identities or health. they went to IBM and ask them to write civilian encryption standard. IBM gave them an algorithm called "Lucifer", which was based on civilian Knowledge from the World War II, the NSA I analyzed it and they say they don't actually like what you wrote and change it. they changed the key change one of the internal tables and say this was the standard.

and the DES actually announced. the changes that NSA made was making it difficult to implement it in software, and to make it Brute Force about using the computing power that has the NSA but to protect it against some kind of attack which was known to the NSA but not known in the Differential cryptography (not going to teach you).

The time passed and the computer got more and more powerful, there was something called Deep cracking the electronic something, which was able to crack DES at least. I'm not sure if they built it. This was too weak, so introduce something called triple DES, it was twice as secure. Triple DES only used two keys, but we're not going to study in this course. This wasn't so very efficient and it was very slow. The US National standard unit, we want to create a new Cipher which was at least as secure as Triple DES but much more efficient. Efficient on software and hardware and slow computers. Anybody could submit candidates, the winner of this competition was AES, which was a PhD work Raymond and Diamond.

It has some very good properties that we are going to talk about them.

5.11 The Advanced Encryption Standard

How would you say AES is?, it is an operation which take 2 input, a kye and plain text. And his output is ciphertext. How big is the plain text? 128 bit, The input of the key is not always 16 bits, 16, 64,128 bits. Can go to AES 256 key. No one can break the 256 AES key. What if I wants to decrypt? AES can be a reverse you can put the ciphertext the key and you get the plain text?. Can we get the plain text and ciphertext so we can get the key? In theory we can do it but the idea is that you cannot get the key. When you feed the key it has to work a little bit, has to extend the key, create around key, this is done in very secure facility. What

if my input is more than 16 bytes? What if I need to encrypt a file? I need to use a protocol and a mode, I can't just use AES, only works on 16 bytes. If I have a larger data I need to use AES with a particular way. Something called AES mode, the famous one called ECB, and CBC, the fashionable called GCM. Not in this course and we don't really care, don't use ECB. What if we want to encrypt just one byte? we need to do padding, there is a trick and we need to do something. Let's talk about the design of a AES, it was designed to resist all of source of the attacks, the attacks which were known in the days of DES. And all sorts of attacks which were discovered using the competition. AES was billed to be resistant for those attacks, script analysis attack but no power analysis attack. They are basically expose the cipher if you have enough plaintext and ciphertext.

AES won the competition because it was very fast or efficient. You have three optimization goals when you're building a crypto implementation. You wanted to be fast and to be able to encrypt as many bits. Maybe you want to encrypt all the data in the router? You need to be fast and you want it to be power efficient. You don't want to change your battery, and to be cheap so using as little transistors as possible. Take at least area in the Silicon using a smaller cheep. These three goals are conflict with each other, but if you want go hardcore which AES is very very fast, if you want AES be larger. Particular the AES submission the real-time paper head implementation of a s 8 bit 16 and 32 beats microcontroller which be being used till this day. One thing about a s which is very nice is that if you remember the things that people were very very suspicious about this AES that IBM present and a bunch of tables which values that IBM didn't explain and then the NSA came back sage no so use these different values and they also didn't explain why. I know that the NSA did something good and what's nice about AES that he use a single lookup table for all Xbox and this Lucas table is actually derived from mathematical relationship some kind of a multiplication are over algebra

field so it's not so hard there. The design is so simple that you not be able to crypto even if you try.

5.12 AES Internals

AES is an iterated cipher Which means eats has a very bASIC algorithm call drafts and he does them all over and over again, AES has 10 Rounds, if you wants to do it more complicated you as more and more rounds. How does as operated, you begin with 16 bytes and put them in a cyber that's called stage register and then you're on the round operations on these state bytes, every time you operate operations the plaintext gets mixed with the key and every time you do it it gets more and more. When you finish these 10 Rounds senior stage register you have the ciphertext.

If you want to do it in a reverse you put the ciphertext and the stage register and run the operation run after another and you get the plain text.

Each round consists of 4 basic operations, sub bytes, shift rows, mix columns, and add around key.

Every operation was chosen by the creator Rijndael to achieve a different objective. One of the objectives was to confusion, to make the aisle to put not linear a dependent on the input, and I don't think they wanted to do what is diffusion so all the output will depend on the input. We are mixing the plaintext with the key with each one of the operation.

SubBytes

First one of the AES is sub bytes. Let's talk about this while thinking about attacks.

How do you implement on 8-bit microcontroller? The state is stored in memory so I have 16 bytes of state, you read the first byte of state and the Sbox. Is a table that stored in memory and the size of the table? 2**8.

So I have this stage registered which is 16 bytes, and the sub box table 256 bytes. A full loop and I took the first byte, first you read it and then I needed to read from the table who is the address that I just read and then I get the value the Sbox, (we know inside the microcontroller) who does right component units the value of the states and the value of the Xbox table, no XOR them, no store that value in the stage register. Bridge from the state go to the table, reading from the sub byte table and xor, and write.This operation is very very leaky.

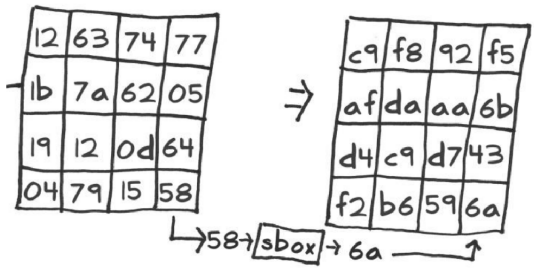


Figure 5.6: SubBytes

ShiftRows

Then I need to do shift rows. The first row you don't need to do anything, the other rows you need to shift them, how do you shift with 8-bit microcontroller? using a temporary value, I read the state into the temporal value and I'm right it into here. this is also a leaky operation, because I'm leaking all the bytes in the state, well digging the Hemming Weights of the byte. another way to implement it, just by imagination, it doesn't change the data so there are actually operations that don't do shift row.

MixColumns

Next operation mix columns, is the Matrix a complication, perform over algebra algorithm, it's takes as input 32 bits

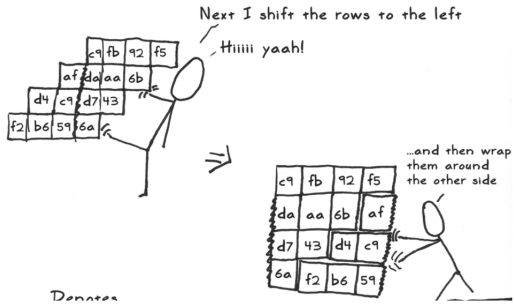


Figure 5.7: ShiftRows

columns and his output is 32 bytes value what are all of the bytes in the output depend of input. How do you do it on a microcontroller? One of the reasons that rhino run the competition that it was very efficient way was doing mix column 8-bit microcontroller, using 13 operations which are shifts and exor. This is the most leaky part of AES this mixed columns.

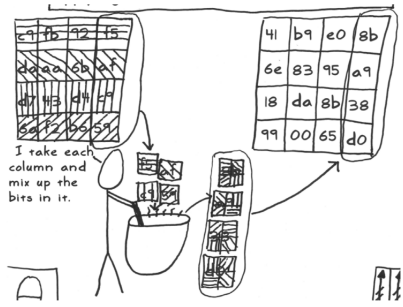


Figure 5.8: MixColumns

AddRoundKey

Then do a add round key, just xor, read xor write. This doesn't leak so much because it is inside the ALU, but the reading and writing is the leaking here. Each round of AES is 84 leaking actions. You take the key and you use the same

you used to do in creation but you don't have the plane text yet so you use sub bytes or shift, then you end with the round, and this key expansion is very sensitive for power analysis, you can really attack as very efficiently. We can assume that this expansion is very secure.

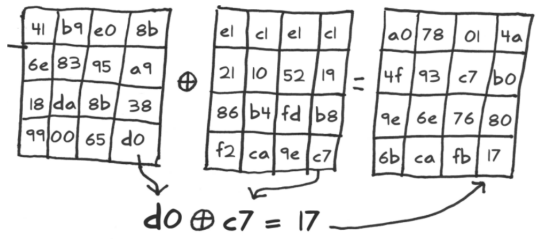


Figure 5.9: AddKey

5.13 AES Power Analysis

I told you about AES and what is our motivation and what is the structure, and I want to spend the time that has left to show you a little about internal of AES and very very very briefly talk about the reaction of doing simple power analysis of AES.

```
% Make sure the matlab AES scripts are in the path
addpath('matlab_aes_scripts');
%%
% Create two 128-bit plaintexts (exactly 16 byte)
plaintext_1 = uint8('Attack at 12:56!');
plaintext_2 = uint8('Attack at 12:57!');

% how many bits are different between the two?
disp(hamming_weight(bitxor(plaintext_1, plaintext_2)));
%%
% Create a key
key = uint8('1234512345123456');
```

```
ENCRYPT = 1;
DECRYPT = 0;
%%
% Encrypt the two plaintexts
ciphertext_1 = aes_crypt_8bit(plaintext_1, key, ENCRYPT);
ciphertext_2 = aes_crypt_8bit(plaintext_2, key, ENCRYPT);
% even though the plaintexts were very similar...
disp([plaintext_1;plaintext_2]);
% ... the ciphertexts are very different
disp([ciphertext_1;ciphertext_2]);
%%
% how many bits are different between the two?
disp(hamming_weight(bitxor(ciphertext_1, ciphertext_2)));

%%
% Decrypt the two ciphertexts
decrypted_1 =  aes_crypt_8bit(ciphertext_1, key, DECRYPT);
decrypted_2 =  aes_crypt_8bit(ciphertext_2, key, DECRYPT);

% Did we get the plaintext again?
disp([plaintext_1;plaintext_2]);
disp([decrypted_1;decrypted_2]);
%%
% Look at the internals of AES now
[ciphertext_1, leak_1] = aes_crypt_8bit_and_leak(plaintext_1
[ciphertext_2, leak_2] = aes_crypt_8bit_and_leak(plaintext_2

% Show an image showing the two leaks side by size
subplot(1,3,1)
image(squeeze(leak_1));colormap(hsv(256));
subplot(1,3,3)
image(squeeze(leak_2));colormap(hsv(256));
figure(gcf)
%%
% Show the difference in the middle
subplot(1,3,2)
image(squeeze(bitxor(leak_1,leak_2)));colormap(hsv(256));
figure(gcf)
```

```
%%
% plot the HW of the difference
subplot(1,1,1)
bar(hamming_weight(bitxor(leak_1,leak_2)));
```

What you see here is Matlab, I wrote this lab environment to do AES implementations, There is code that dose AES, and there i code that simulates the power leakages of AES as it was implemented on 8-bit microcontroller, all of the operations are 8 bits operations and each time operation happens I'm going to leak it's Hamming Weight .Let's see what's going on.

The first thing I'm going to do is to load some libraries and you can find them in the middle, now going to create 216 bytes plain text. I am going to take these string, to make it binary string Unit8, and I'm going to measure the Hamming distance between these two strings. What is the time distance between these two strings? One, the difference is 6 become 7. 110 - 111.

How do we do it, I have function called bitxor, and this is a vector of size 16,and then waiting Hamming weight. How many possibles Hamming weight are they for 8 bits value? What can the Hamming weight to be? zero,1,2 ... 8.So that Hamming weight here is going to be one. now I'm going to set up a AES I am going to choose a key.And now I'm going to encrypt AES, and then I'm going to use my to plain text and the key. The Hamming distance between the two was one. What will be the Hamming distance with the cipher text? Will it be one?no! what's possible values it can be, between 0 to 128, because the output. Would you like me to be 128? NO, I would like it to be 64. Why do I don't need it to be 100 or 2 or 3. Because that means that I don't have a very important property crypto assistance in the Avalanche property means that each bit in each bit affect all of the input. So if I change one bit in the input and I get only one change in the output it means that I don't have the ability point. But if I change one bit in the input and I get 100 bit

change in the output what it is mean? It's means that I don't have the average property and just to make things fun I'm flipping all the bits. What I want the Hamming distance to be is around 64, It's that exactly half of the bits changing. So you can see I'm doing the Hamming and measuring and show the ciphertext, at the end I'm doing the Hamming weight.

These two strings are very very similar until the end, the 6 and 7 change. But if you see the ciphertext you can see a lot of difference. And this humming weight is 52. sometimes he's more sometimes is less than 64 but this is okay. Now let's decrypt, How do I decrypt? I take my AES 8-bit function and I give it decrypt. The ciphertext and the key. and see if the plain text into ciphertext are the same. We can see that there are the same and so far so good. Now I want to show you the internal structure of AES. to do that I have a function that's called 8-bit and leak. Let's open the function. Here is the function. Let's go over the structure

```
function [result state rkeys mixcolumn_leak]=
aes_crypt_8bit_and_leak(input_data, secret_key, encrypt)

% performs AES-128 encryptions or decryptions like an 8-bit
% and leaks internal state
%
% DESCRIPTION:
%
% [result state rkeys mixcolumn_leak] = aes_crypt(input_data
%
% This function performs an AES-128 encryption or decryption
% data with the given secret key.
%
% PARAMETERS:
%
% - input_data:
%   A matrix of bytes, where each line consists of a 16 byte
%   data input value of the AES-128 en/decryption.
% - secret_key:
```

```
%   A vector of 16 bytes that represents the secret key.
% - encrypt:
%   Paramter indicating whether an encryption or a decryptio
%   (1=encryption, 0=decryption).
%
% RETURNVALUES:
%
% - result:
%   A matrix of bytes of the same size as the byte matrix 'i
%   Each line of this matrix consists of 16 bytes that repre
%   128-bit output of an AES-128 en/decryption of the corres
%   'input_data'.
% - state:
%   A matrix of byte of size |'input_data'| x 41, containins
%   progression of the encryption process.
%   Legend of the state progression:
%   (P= plaintext, C=Ciphertext, K=after AddKey, B=after Sub
%   ShiftRows, M=after MixColumns)
%   P K BRMK BRMK BRMK BRMK BRMK BRMK BRMK BRMK BRMK BRK(=C)
% - mixcolumn_leak:
%   A matrix of size |'input_data'| x 9 x 4 x 9 (for encrypt
%                    |'input_data'| x 9 x 4 x 18 (for decryp
%       where mixcolumn_leak(line, subround, col, :) is the
%       intermediate valutes generated by the 8-bit MC opera
%       [col] columns of line [line] in the input data durin
%       subroun [subround]
% EXAMPLE:
%
% result = aes_crypt([1:16; 17:32], 1:16, 1)


% AUTHORS: Stefan Mangard, Mario Kirschbaum, Yossi Oren
%
% CREATION_DATE: 31 July 2001
% LAST_REVISION: 28 October 2008

state = zeros([41 size(input_data)], 'uint8');
rkeys = zeros([10 16], 'uint8');
```

```
if (encrypt == 0) % decryption
    mixcolumn_leak = zeros([9 4 size(input_data,1) 18]);
else % encryption
    mixcolumn_leak = zeros([9 4 size(input_data,1) 9]);
end

for round = 1:10
    rkeys(round,:) = aes_round_key(secret_key,round);
end

% expand the keys

if encrypt == 0 %decryption
    state(41,:) = input_data;

    for i=10:-1:1
        if i ~= 10
            input_data = aes_add_round_key( aes_round_key(se
            state(3 + (i-1)*4 + 2,:) = input_data;

            [input_data leak] = aes_mix_columns_8bit_and_lea
            mixcolumn_leak(i, :, :, :) = leak;
            state(3 + (i-1)*4 + 1,:) = input_data;
        else
            input_data = aes_add_round_key( aes_round_key(se
            state(3 + (i-1)*4 + 1,:) = input_data;
        end

        input_data = aes_shift_rows(input_data,0);
        state(3 + (i-1)*4,:) = input_data;

        input_data = uint8(aes_sbox(input_data,0));
        state(3 + (i-1)*4 - 1,:) = input_data;
    end

    input_data = aes_add_round_key(secret_key, input_data);
    state(1,:) = input_data;
```

```
else % encryption

    state(1,:) = input_data;
    input_data = aes_add_round_key(secret_key, input_data);
    state(2,:) = input_data;

    for i=1:10
        input_data = uint8(aes_sbox(input_data,1));
        state(3 + (i-1)*4,:) = input_data;

        input_data = aes_shift_rows(input_data,1);
        state(3 + (i-1)*4 + 1,:) = input_data;

        if i ~= 10
            [input_data leak] = aes_mix_columns_8bit_and_lea
            mixcolumn_leak(i, :, :, :) = leak;
            state(3 + (i-1)*4 + 2,:) = input_data;

            input_data = aes_add_round_key( aes_round_key(se
            state(3 + (i-1)*4 + 3,:) = input_data;
        else
            input_data = aes_add_round_key( aes_round_key(se
            state(3 + (i-1)*4 + 2,:) = input_data;
        end
    end

end

result = input_data;
```

First of all I expand the key, so I do the AES key expansion and I don't leak the key expansion. This is the assumption. Let's disregarded that description for a moment and this is the encryption. First of all I took the state and the input data in the state, and then I do a add round key, then do SUB bytes, shift rows, mix column. In the last round I done did you mixed columns, and every time I do this I don't

replace the state and I saved the state. at the end I output the cipher text, but also the output to the state, so you can see the state is always changing. Now let's see how does it looks, I'm going to run AES twice, and then do some little figure.

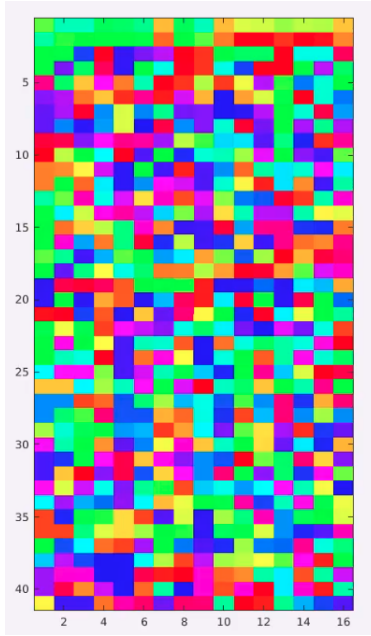


Figure 5.10: Diagram

The x-axis is the index of the byte in the state. I read the bytes not as a matrix I read it as a row. So there are 16 bites in the state. and the y-axis is the index of the rounds and round 40 is the final round ciphertext. So in between there is the stages. You can see that in the beginning it is very very similar.

If you go down it's become much different and in the bottom it's completely different. This is very easy to analyze. And what we are going to do now show the difference between state. What do you see in the middle is the Hemming distance between the right and the left where red is is 0 and blue is 256.

In stage 0 what is the difference between two plain texts? 1 bit, you can see one beat is changed to O. The first round is a add key, we xor the 15 bytes of the key who is 16 bytes of the state, the key is the same in both sides, if the Hamming distance between the two sides was one before I sold the key what is it going to be after? 1. The differential doesn't change at all, I think a distance of 1 and Ikes or a constant of both sides and The Outpost is still the same. so in the first row I can't see differences. look wild sheep will do, one by the interstate here, one byte is in the same place and one byte There and one byte here. Now there are 4 bytes different between the states, but this bytes are all of the column.What is the next operation? mixed columns. See what's mix column did, mixed this change and now all of the bytes of the states are different.We can't stop AES after 2 rounds, it still can break with crypto analysis, but you can see that this is very nice.

I want to plug the Hamming weight of the distance, The x-axis here he's there around, and the y-axis is the Hamming distance between the two stay. you can see state with the one then one stays one, after sub byte becomes 4, Shift columns doesn't change the 4, then mix columns make it grow a little bit. And then you see the distance stays around 64 until the end of the encryption. I want to show you how do we attack AES using simple power analysis.

There is a lot to talk about in very little time. in very very briefly I will give you the ideal way did you it and then I'll show you how it's actually done. I have connected a probe to my device, I have measured the power consumption over-time. so now I have a vector of size that lets say a hundred thousand, and I know that AES is there. Now I want to find the key out of my measurements from my trace.

I have a trace, which was recorded in the same device we saw here. I have a vector called 200 of traces of AES encryption, lets plot one of the traces. this is actually only one of the rounds of AES. Here is a trace. This is very very clean trace and I would like it to be in my lab.You can see some of them

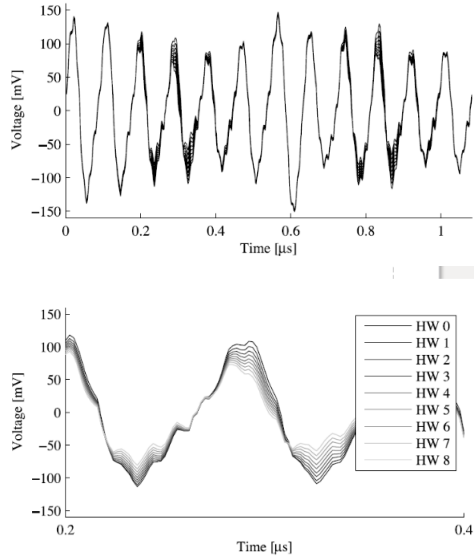


Figure 5.11: Traces

are high and some of them are low, how do I find the key out of this? let's start from the beginning then jump to the middle and the end I don't have time to teach you. the best thing I can hope for is not to get the bytes of the key because the bytes are not function of the bytes, what is the function of? The Hamming weight. So the best hope is the get the Hamming weight, so are you home we will get a vector like the state vector. is this enough to get the key? so you have to believe me that's yes. how do you do it? You will use algebraic the solver and you will have to read the paper. if you have the vector off all the time in Hamming weight you can get the key. But how do I get the vector Hamming weight from distance? Obviously somewhere in this trace, let's look at very very leaky operation like sub bytes, or add around key; it read the states you leave the key and xor them together and then you write the state again. Somewhere in this vector there is a peak where we know that exactly in this moment there was a read of the key. The key was read from memory and was travel in the data bus in this exact

peak. And how do I find this particular moment in time I will show you in the next lecture.

So I need to write a function that gets an input about 20 points and what is its out puts? the Hamming weight. maybe it will output of Hamming weight. how do I do it? we need to write a decoder that's as single processing.

Here is a figure showing the move operation they did the same move operation in the microcontroller over and over again this is the average of thousands of measurements where are they moving 0 or 1, until moving 255 ff. Can anybody, you see how Hamming Weight zero have big change, why Hamming weight 0 is more power consumption of Hamming weight 8? Between moves it settings old ones, so moving to zero take more effort from changing.

Here is like a longer trace, let's assume I hate this data how do I build the decoder. I want the function that can take an array of values and output Hamming weight. Let's start simple, what if I had just one point? I have an input and function that gets one point and give me the Hamming weight of this point. I can calculate the mean each one, the mean for 0 the mean for 1, for 2. if I get it right what we will do, so do you like a nearest neighbor. if I know that the mean of 4 Hamming weight for is 7 and the mean of Hamming weight five is 8 and I got 7.9, what am I going to chose 4 or 5? 5.nearest neighbor.

This is a nice idea, but I have to give them a little extra dangerous. the problem is here that's the Hamming weight are not identically distributed. how many possible values of bytes I get? 256. how many bytes are there with Hamming weight zero? 1. how many bytes Hamming weight 8? 1. how many Hamming weight 1? 8. Hamming weight of 1 is 8 times more likely than Hamming weight of 0. So why do we need to do?we need to do Bayes. The idea is are you going to favor the output more likely. I will found out what are the odds that its five, 6?, 7? then look on my decoder and then I'm going to look and give a bonus for more common

Hamming weight. The idea is I need to learn How likely bytes based on the trace decoding and then I'm going to go and look at the distribution of this and then multiply The probability I got. this is for one point. I calculated the odds and then I multiply the probability. what if I have more than one point? watch we actually wants to do here he's actually called multivariate normal distribution. each one of these points, let's say four points, the dimension, and I have a like a cloud In this distribution space, how do you do this? that is the very very fundamental paper called "template power analysis attack" which explain this.

So if you want to do a simple power analysis on AES the steps you do first of all you profiled the device, you understand where all this stuff is happening, you need to build Template that will help you to recognize different Hamming Weight, then you're play the same place in the trace and you get guesses for Hamming weight for each States and you hope you get the right guesses, and then you take this Hamming weight and you take a few equation solver we'll take the Hamming weight and output the key, or something that we don't know? I don't know because noise. If we can correlate the Hamming weight to wrong then the equation solver will fail. what can we do in this case? We try again.

5.14 Template Attacks

Introduction

Devices performing cryptographic operations can be analyzed by various means. Traditional cryptanalysis looks at the relations between input and output data and the used keys. However, even if the implemented algorithms are secure from a cryptanalysis point of view, side-channel attacks pose a serious threat. Sidechannel attacks are a subgroup of implementation attacks. Examples thereof are timing attacks, power attacks like DPA or SPA. Traditional DPA style attacks assume the following threat model: The secret key stored in the device is used to perform some cryptographic operations. The attacker monitors these operations using captured side-channel information like power consumption or electromagnetic emanation. The attack is successful if the used secret key can be reconstructed after a certain number of operations.**If the number of operations is limited by the protocol used to initiate** these operations, the attacker has an upper bound on the number of operations he can observe. If the operation, which leaks usable side-channel information, is executed just once, the threat model is different: The attacker has to reconstruct the secret key using a single trace of side-channel information. Besides protocol limitations, ephemeral keys can be the reason for such a constraint. Techniques like SPA are a general way to tackle this problem. These techniques useeasily distinguishable features of operations like double and add, or add and multiply, to infer key-bits. The majority of the available literature deals with these two types of scenarios. **If the observed signal-to-noise ratio is not high enough, or the implementation is done in a way that ensures the used operations being independent of the key(i. e. no key-dependent jumps), SPA style attacks are not possible anymore.** The attacker has to think of other ways to get hold of the secret key: One way to do this is to use a similar device and build a model of it. Using this model, an attacker

might now be able to recover the secret key.

Algorithm

Template attacks are a powerful type of side-channel attack. These attacks are a subset of profiling attacks, where an attacker creates a "profile" of a sensitive device and applies this profile to quickly find a victim's secret key.

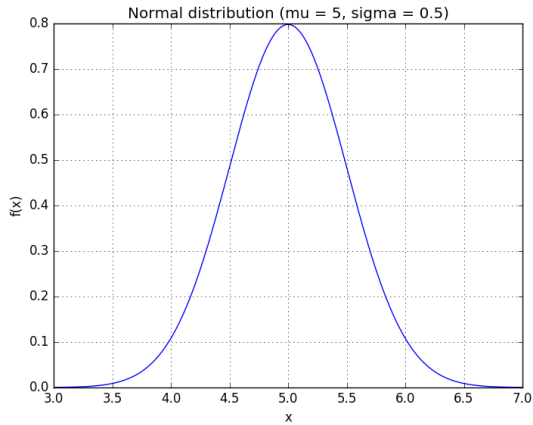
Template attacks require more setup than CPA attacks. To perform a template attack, the attacker must have access to another copy of the protected device that they can fully control. Then, they must perform a great deal of pre-processing to create the template - in practice, this may take dozens of thousands of power traces. However, the advantages are that template attacks require a very small number of traces from the victim to complete the attack. With enough preprocessing, **the key may be able to be recovered from just a single trace** .

There are four steps to a template attack:

- Using a copy of the protected device, record a large number of power traces using many different inputs (plaintexts and keys). Ensure that enough traces are recorded to give us information about each subkey value.
- Create a template of the device's operation. This template notes a few "points of interest" in the power traces and a multivariate distribution of the power traces at each point.
- the victim device, record a small number of power traces. Use multiple plaintexts. (We have no control over the secret key, which is fixed.)
- Apply the template to the attack traces. For each subkey, track which value is most likely to be the correct subkey. Continue until the key has been recovered.

Signals, Noise, and Statistics

Before looking at the details of the template attack, it is important to understand the statistics concepts that are involved. A template is effectively a multivariate distribution that describes several key samples in the power traces. This section will describe what a multivariate distribution is and how it can be used in this context. Noise Distributions
Electrical signals are inherently noisy. Any time we take a voltage measurement, we don't expect to see a perfect, constant level. For example, if we attached a multi-meter to a 5 V source and took 4 measurements, we might expect to see a data set like (4.95, 5.01, 5.06, 4.98). One way of modelling this voltage source is: $\mathbf{X} = \mathbf{X} + \mathbf{N}$ where $\mathbf{X}$ is the noise-free level and $\mathbf{N}$ is the additional noise. In our example, $\mathbf{X}$ would be exactly 5 V. Then, N is a random variable: every time we take a measurement, we can expect to see a different value. Note that $\mathbf{X}$ and $\mathbf{N}$ are bolded to show that they are random variables. A simple model for these random variables uses a Gaussian distribution (read: a bell curve). The probability density function (PDF) of a Gaussian distribution is $f(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{-(x-\mu)^2/2\sigma^2}$ where μ is the mean and σ is the standard deviation. For instance, our voltage source might have a mean of 5 and a standard deviation of 0.5, making the PDF look like:



We can use the PDF to calculate how likely a certain measurement is. Using this distribution, $f(5.1) \approx 0.7821$ $f(7.0) \approx$

0.0003 so we're very unlikely to see a reading of 7 V. We'll use this to our advantage in this attack: if f(x) is very small for one of our subkey guesses, it's probably a wrong guess.
Multivariate Statistics The 1-variable Gaussian distribution works well for one measurement. What if we're working with more than one random variable? Suppose we're measuring two voltages that have some amount of noise on them. We'll call them $\mathbf{X}$ and $\mathbf{Y}$. As a first attempt, we could write down a model for $\mathbf{X}$ using a normal distribution and a separate model for $\mathbf{Y}$ using a different distribution. However, this might not always make sense. If we write two separate distributions, what we're saying is that the two variables are independent: when $\mathbf{X}$ goes up, there's no guarantee that $\mathbf{Y}$ will follow it. Multivariate distributions let us model multiple random variables that may or may not be correlated. In a multivariate distribution, instead of writing down a single variance σ, we keep track of a whole matrix of covariances. For example, to model three random variables $(\mathbf{X}, \mathbf{Y}, \mathbf{Z})$, this matrix would be

$$\mathbf{\Sigma} = \begin{bmatrix} Var(\mathbf{X}) & Cov(\mathbf{X},\mathbf{Y}) & Cov(\mathbf{X},\mathbf{Z}) \\ Cov(\mathbf{Y},\mathbf{X}) & Var(\mathbf{Y}) & Cov(\mathbf{Y},\mathbf{Z}) \\ Cov(\mathbf{Z},\mathbf{X}) & Cov(\mathbf{Z},\mathbf{Y}) & Var(\mathbf{Z}) \end{bmatrix}$$

Also, note that this distribution needs to have a mean for each random variable:

$$\mu = \begin{bmatrix} \mu_X \\ \mu_Y \\ \mu_Z \end{bmatrix}$$

The PDF of this distribution is more complicated: The equation for k random variables is:

$$f(\mathbf{x}) = \frac{1}{\sqrt{(2\pi)^k |\mathbf{\Sigma}|}} e^{-((\mathbf{x}-\mu)^T \mathbf{\Sigma}^{-1} (\mathbf{x}-\mu)/2}$$

Creating the Template

A template is a set of probability distributions that describe what the power traces look like for many different keys. Effectively, a template says: "If you're going to use key k, your

power trace will look like the distribution $f_k(\mathbf{x})$"
. We can use this information to find subtle differences between power traces and to make very good key guesses for a single power trace.

Number of Traces
One of the downsides of template attacks is that they require a great number of traces to be preprocessed before the attack can begin. This is mainly for statistical reasons. In order to come up with a good distribution to model the power traces for every key, we need a large number of traces for every key. For example, if we're going to attack a single subkey of AES-128, then we need to create 256 power consumption models (one for every number from 0 to 255). In order to get enough data to make good models, we need tens of thousands of traces.

Note that we don't have to model every single key. One good alternative is to model a sensitive part of the algorithm, like the substitution box in AES. We can get away with a much smaller number of traces here; if we make a model for every possible Hamming weight, then we would end up with 9 models, which is an order of magnitude smaller. However, then we can't recover the key from a single attack trace - we need more information to recover the secret key.

Points of Interest
Our goal is to create a multivariate probability describing the power traces for every possible key. If we modeled the entire power trace this way (with, say, 3000 samples), then we would need a 3000-dimension distribution. This is insane, so we'll find an alternative.

Thankfully, not every point on the power trace is important to us. There are two main reasons for this:

- We might be taking more than one sample per clock cycle. There's no real reason to use all of these samples - we can get just as much information from a single

sample at the right time.

- Our choice of key doesn't affect the entire power trace. It's likely that the subkeys only influence the power consumption at a few critical times. If we can pick these important times, then we can ignore most of the samples.

These two points mean that we can usually live with a handful (3-5) of points of interest. If we can pick out good points and write down a model using these samples, then we can use a 3D or 5D distribution - a great improvement over the original 3000D model.

Analyzing the Data

Suppose that we've picked I points of interest, which are at samples $s_i (0 \leq i < I)$. Then, our goal is to find a mean and covariance matrix for every operation (every choice of subkey or intermediate Hamming weight). Let's say that there are K of these operations (maybe 256 subkeys or 9 possible Hamming weights).

For now, we'll look at a single operation k ($0 \leq k < K$). The steps are:

- Find every power trace t that falls under the category of "operation k". (ex: find every power trace where we used a subkey of 0x01.) We'll say that there are T_k of these, so t_{j,s_i} means the value at trace j and POI i.
- Find the average power μ_i at every point of interest. This calculation will look like:

$$\mu_i = \frac{1}{T_k} \sum_{j=1}^{T_k} t_{j,s_i}$$

- Find the variance v_i of the power at each point of interest. One way of calculating this is:

$$v_i = \frac{1}{T_k} \sum_{j=1}^{T_k} (t_{j,s_i} - \mu_i)^2$$

- Find the covariance c_{i,i^*} between the power at every pair of POIs ($iandi^*$). One way of calculating this is:

$$c_{i,i^*} = \frac{1}{T_k} \sum_{j=1}^{T_k} (t_{j,s_i} - \mu_i)(t_{j,s_{i^*}} - \mu_{i^*})$$

- Put together the mean and covariance matrices as:

$$\mathbf{\Sigma} = \begin{bmatrix} v_1 & c_{1,2} & c_{1,3} & \dots \\ c_{2,1} & v_2 & c_{2,3} & \dots \\ c_{3,1} & c_{3,2} & v_3 & \dots \\ \vdots & \vdots & \vdots & \ddots \end{bmatrix}$$

$$\mu = \begin{bmatrix} \mu_1 \\ \mu_2 \\ \mu_3 \\ \vdots \end{bmatrix}$$

These steps must be done for every operation k. At the end of this preprocessing, we'll have K mean and covariance matrices, modelling each of the K different operations that the target can do.

Attack Time

With a template in hand, we can finish our attack. For the attack, we need a smaller number of traces - we'll say that we have A traces. The sample values will be labeled $a_{j,s_i} (1 \leq j \leq A)$. First, let's apply the template to a single trace. Our job is to decide how likely all of our key guesses are. We need to do the following:

- Put our trace values at the POIs into a vector. This vector will be $\mathbf{a_j} = \begin{bmatrix} a_{j,1} \\ a_{j,2} \\ a_{j,3} \\ \vdots \end{bmatrix}$
- Calculate the PDF for every key guess and save these for later. This might look like: $p_{k,j} = f_k(\mathbf{a_j})$
- Repeat these two steps for all of the attack traces

This process gives us an array of $p_{k,j}$, which says: "Looking at trace j, how likely is it that key k is the correct one?"

Combining the Results

The very last step is to combine our $p_{k,j}$ values to decide which key is the best fit. The easiest way to do this is to combine them as:

$$P_k = \prod_{j=1}^{A} p_{k,j}$$

practical template attacks

In this section we will show a differant ways to select the most important point of a power trace, that will lead us to improved computation time and make template attack more practical.

sum of difference

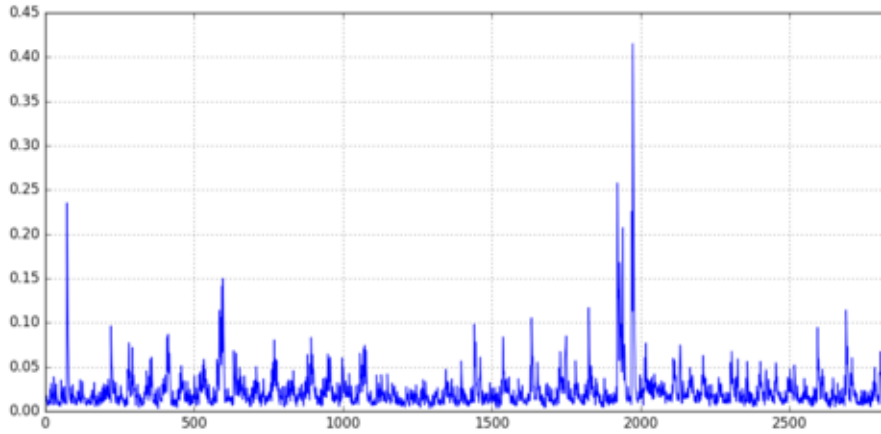
- For every operation k and every sample i, find the average power $M_{k,i}$. For instance, if there are T_k traces where we performed operation k, then this average is

$$M_{k,i} = \frac{1}{T_k} \sum_{j=1}^{T_k} t_{j,i}$$

- After finding all of the means, calculate all of their absolute pairwise differences. Add these up. This will give one "trace" which has peaks where the samples are usually different. The calculation looks like

$$D_i = \sum_{k_1,k_2} |M_{k_1,i} - M_{k_2,i}|$$

An example of this sum of differences is:



Preprocessing

In practical side-channel analysis, the raw input data is often preprocessed. Sometimes this is just due to simplicity or efficiency reasons, e. g. summarizing sampled points. There are however cases where the preprocessing step heavily affects the results. Even if no thinkable transformation can add additional information to a signal, information extraction procedures do improve. The template attack under consideration is such a case and a lucrative preprocessing transformation is described subsequently. It turns out that the transformation of the input traces from the time domain into the frequency domain is such a lucrative transformation. In our practical work, an FFT algorithm was used to accomplish this transformation (a fast algorithm to calculate the discrete Fourier transform, for background information

refer to [BP85]). In order to show the impact of this preprocessing step a number of experiments were carried out. First some characteristic differences between time domain analysis and frequency domain analysis are illustrated. Afterwards, to highlight the influence of the number of selected points on the classification performance in the frequency domain, a number of experiments were carried out. After preprocessing, the resulting traces can be used to perform a template attack in exact the same way as without preprocessing. There is however a difference in the number of points to consider. Figure 6 shows the classifications results as a function of the number of selected points after preprocessing. The considered numbers of points are ranging between 1 and 40. Additionally three different minimum distances where chosen. Results show that much less points are sufficient in comparison to a template attack without the preprocessing step. At the price of performing an FFT on every input trace (those used to build up the templates as well as those to classify) we get a major advantage

Amplified Template Attack

Even if the aim of a template attack is to recover the secret key using a single trace, in many real world settings implementations allow for several iterations of the same operation with the same secret key. The application of template attacks is not restricted to stream ciphers like RC4 and can be applied to block ciphers as well. Since every symmetric cipher contains some sort of key scheduling mechanism which processes the secret key, this generalization is possible. Smartcards often use block ciphers for encryption or authentication, hence let us consider the following example: A malicious petrol station tenant, named Eve, is using a modified smartcard based payment terminal. Everytime a customer uses this terminal, Eve captures one trace of side-channel information. This single trace could already be used by Eve to carry out a template attack. However, some customers are coming again and Eve gets hold of another trace.

The template attack can easily be extended to take advantage of such situations, e.g. by adding up noise-probabilities p(Ni) of every captured trace and applying the maximum-likelihood approach on these sums. As a consequence, the power of the attacker is amplified. Using this approach, if n is the number of iterations, the error probability of template classification is reduced by the factor $\sqrt{n}$.

Chapter 6

Introduction to micro-architectural attacks

Clementine Maurice, CNRS, IRISA
April 30, 2019 — Ben Gurion University, Israel

6.1 Background & Primitives

Micro-architectural side-channel attacks refers to a side-channel attack that exploit information leakage from the hardware infrastructure itself. The attacks can be found in a large scope on devices - servers, workstations, laptops, smart-phones, etc.

Normally, we assume a safe software infrastructure. Meaning that no software bugs, such as buffer overflow, is present. Nevertheless, such assumption does not imply safe execution, due to the fact that the information leaks because of the implementation, which is often driven by complex optimizations and design. Such leakages are not considered as 'mistakes' or 'bugs', but rather a trade-off decision between optimizing some aspects of the execution and potential information leakage.

Potential outcomes of such attacks as described can be crypto primitives, which is common with other side-channel attacks, but also other sensitive information such as keystrokes and mouse movements.

In terms of sources of leakage, we saw in previous chapters sources such as power consumption or electromagnetic leaks. However such sources are harder to come across by an attacker because they require physical proximity and access to the device, which is more typically to embedded devices. In our case we are only require that the attacker have somewhat remote access to the device, meaning that the attacker can run code on the hardware infrastructure of the machine. Such scenarios can be found in cloud providers renting computational resources to a costumer, Java-Script code running in a browser and others.

Example: Cache attack on RSA Square-and-Multiply Exponentiation

Input : base b, exponent e, modulus n
Output: b^e mod n
$X \leftarrow 1$ **for** $i \leftarrow bitlen(e)$ ***downto*** 0 **do**
  $X \leftarrow multiply(X, X)$ **if** $e_i = 1$ **then**
    $X \leftarrow multiply(X, b)$
  end
end
return X

Algorithm 1: Square-and-multiply exponentiation

Consider the binary exponentiation algorithm presented in Algorithm 1. Notice that the execution flow of the algorithm relies heavily on the value of the bit e_i of the private key. Now consider a scenario in which an attacker has information on the changes regarding the buffer holding the multiplier b. Such information can be considered as a query to the buffer and receiving as a result the latency of the query. If the

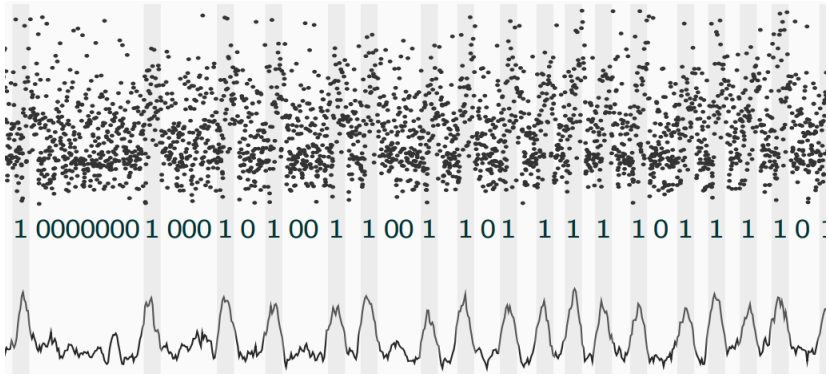


Figure 6.1: Querying the buffer holding the multiplier b. The y axis is a latency scale, and the x axis represent the query index. The dotted plot is a scatter plot while the solid line is the normalized moving average.

buffer is in use, the query will be longer then if the buffer is unused.

In Figure 6.1 the resulting graph of the querying can be seen, in which the actual bits of the exponent can be clearly detected.

Attack Model

We assume that the attacker has no physical access to the device. Moreover, the attacker can only execute unprivileged code on the same machine as victim. Such scenarios in which this happens can be found, for example, when the victim install some malicious program on his machine/smartphone. Another examples can be found in could computing where an attacker has a victual machine on the same physical machine as the victim, or in a web environment in which the attacker runs unprivileged JavaScript code.

In the scope of this chapter we will focus on shared hardware in the form of data/instruction cache. While other shared hardware components can also include the DRAM and the memory bus (memory shared hardware) or the branch pre-

diction unit and the arithmetic logic unit (CPU shared hardware), they are not in the scope of this chapter.

Today's CPU Complexity

From 2012 onwards Intel has released a new family of microprocessors every year. Each comes with its own new microarchitectural schemes with new Characteristics. To gain performance upgrade in each new installment, that has on average 5% increase in performance depending on the feature, very small optimizations, such as caches and branch prediction units are added. Such optimizations are the main reason for a side-channel information leakage to exist. Unfortunately, said optimization often come with no documentation since this is Intel intellectual property. Such lack in documentation makes it harder for an attacker to perform said side-channel attacks.

To emphasize today's CPU complexity, it is said that "Intel x86 documentation has more pages than the 6502 has transistors" [22], with 6502 being a reference to a 8-bit microprocessor, used in Apple II, Commodore 64, Atari 800 and more. It had, in the year 1975, roughly 3510 transistors, while the Intel Software Developer's Manuals is 4844 pages (may. 2018). In a more advance microprocessor, such as the 22-core Intel Xeon Broadwell-E5, more than 7 billion transistors can be found.

Background on Caches

In designing a cache side-channel attack, a proper understanding of how caches work, in great detail, is required. First, we need to acknowledge that the requirements for an 'ideal memory' oppose each other. Such requirements are zero latency, infinite capacity, zero cost and infinite bandwidth. The trade-offs are interconnect: **Bigger is Slower** - Bigger memory often comes with higher latency, due to the fact the it would take longer to determine the location to retrieve. **Faster is Expensive** - A reduction in latency often

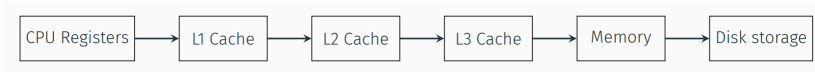


Figure 6.2: Memory Hierarchy of a common CPU.

means that a more expensive technology a needed. For example, SRAM is faster than DRAM, which in turn is faster than Disk memory. However, their cost ratio is the other way around. **Bandwidth is Expensive** - A wider bandwidth needs to come with additional banks, ports, higher frequency or faster technology.

In our need to understand the use of caches, we need to examine the different memory technologies and why they are used the way they do. **DRAM** - Dynamic Random Access Memory is the technology that is often used as the main memory (or RAM), while **SRAM** - Static Random Access Memory is the technology used in caches. The DRAM latency is higher than the SRAM latency, but is cheaper to make. The main difference in cost arises from the fact that DRAM consist only of one transistor and one capacitor per cell, while the SRAM consist of six transistors per cell. Said difference also mean that DRAM is more dense. Finally, the DRAM cell layout and technology requires to periodically refresh each cell.

Since having Both a large and fast, single level of memory is unlikely, CPU's today are made with multiple levels of storage. The main idea is that progressively bigger and slower storage units are located in higher levels of memory, which are farther from the processor. The motivation for such design is to ensure that most of the data the processor needs is kept in the closer and faster levels.

The memory hierarchy can be seen in Figure 6.2, where data can reside in, at a given point in time, in the following storage units - in CPU registers, the different levels of the CPU cache, main memory or disk memory.

Caching Basics

The two main idea of caching is to exploit temporal and spatial locality. **Temporal Locality** is the notion that a program tends to reference the same memory location many times within a small window of time (for example, loops). The anticipation of such thinking is that recently accessed data will be accessed again soon. Therefor, a good strategy will be to store recently accessed data in automatically managed fast memory. **Spatial Locality** is the notion that a program tends to reference a cluster of memory locations at a time (for example, sequential instruction access or array traversal). The anticipation will be that surrounding of some accessed data will be accessed soon. A good strategy will consider storing addresses adjacent to the recently accessed one in an automatically managed fast memory. In detail, we want to logically divide memory into equal size blocks (lines) and fetch to cache the accessed block in its entirety.

Moreover, there are two approaches to the management scheme, manual and automatic. **Manual Management** means that the programmer manages data movement across levels, which is not scalable for substantial programs. However, it is sometimes used in some embedded systems. **Automatic management** refer to a scheme in which the hardware itself manages data movement across the different levels in a way that is transparent to the programmer. Meaning that the average programmer does not need to know anything related to the memory hierarchy levels to write a program. On the down side, which begs the question on how to write a fast program if the management is automatic, in addition to what kind of side-channels could arise from such scenario.

The basic unit of storage in the cache is called a block or a line. The main memory is logically divided into cache blocks that map to locations in the cache. And when data is referenced, two outcomes can come of such action - a cache hit or a cache miss. A **Cache Hit** occurs when a line is referenced and is in the cache. The cached data will be retrieved

instead of accessing main memory. A **Cache Miss** occurs when a line is referenced and is not in the cache. The data will be fetched from main memory, passing though the cache, possibly eviction other data to ensure enough storage.

When designing a cache, we are faced with a number of design decisions to handle. **Placement** - Essentially the place in which we will place or find a block in the cache. **Replacement** - We need often to remove data from the cache to make room for newer data. Which begs the question of what data to evict. **Granularity of Management** - The basic units of storage in different levels. Do we store the same amount of blocks across different levels or do we uniformly store the same size of blocks. **Write Policy** - When a write is being made, we need to decide whether to preform the write operation in all levels or passing the write data to a lower level only upon eviction from that level. **Instruction/Data** - When a program is executing, the instructions that are in need to be executed are required to be fetched from memory as well. The designer needs to decide whether to treat instruction and data memory as two separate types or as one of the same.

Set-Associative Caches

We will be focusing on a cache design that is often used in today's CPUs, which is set-associative caches. Which means that the cache will be logically divided into **Cache Sets**, which will be divided into **Cache Ways**. Each way will store a cache line worth of data. The division of the memory into different cache sets will be a direct mapping from the memory address. Namely, each address will be associated with a cache set according to a group of bits in the address which will represent the cache set index. The specific way in which the line will is to be fetch will be determine according to the cache replacement policy, since the cache set is expected to be full. An abstraction can be seen in Figure 6.3

Consider the following example: The cache has 8B cache

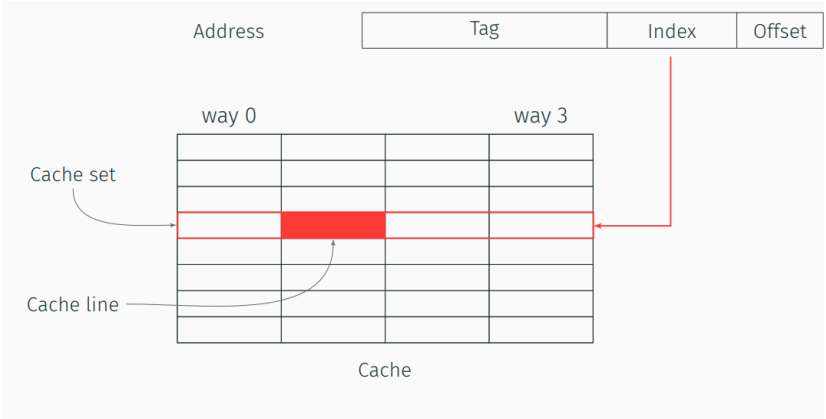


Figure 6.3: Set-Associative Cache - The set index of a line derives from the set index bits of the address. The cache way is determine according to the cache replacement policy.

lines, 16 cache sets each consisting of 2 ways. We can compute the set index of the address $(1011111110)_b$ by only looking at bits b_3, b_4, b_5, b_6, since we have 16 cache sets (4 bits) and 8 different cache line offset (bits b_0, b_1, b_2). Hence the cache set will be $(1111)_b$. We can also compute the size of the whole cache by multiplying the cache dimensions $8 \times 16 \times 2 = 256$B.

Virtual Addresses or Physical Addresses

Since we need the bits of address to determine its cache set, we are face with a problem due to the fact that programs running are only aware to the virtual addresses while the hardware side is aware to the physical addresses. The translation between virtual addresses and physical addresses is the responsibility of the MMU (Memory Management Unit).

There are four major ways of implementing such cache mapping according to the addresses. **VIVT** - Virtually-Indexed, virtually-Tagged. Meaning that there is no need to translate the addresses in order to know the mapping of the addresses into the cache, which is faster. On the other hand, such implementation causes aliasing issues as same virtual address

maps to several different physical addresses. This aliasing is due to the tag not being unique, and will force a flush action to the entire cache on context swiching. **VIPT** - Virtually-Indexed, Physically-Tagged. Meaning that there will be a need for TLB translation for the tag, but can be looked up in parallel. Such mapping can be quiet fast. Moreover, if the set index bits are derived from the page offset, there is no aliasing, but the cache size is limited to the page size multiply by the number of cache sets. VIPT is used in Intel's L1 caches, with the default page size being 4K and each cache line is 64B, resulting in no larger than $2^6 = 64$ sets. **PIPT** - Physically-Indexed, Physically-Tagged. Meaning that the translation between virtual and physical addresses will have to be preformed, taking up time. On the other hand, no aliasing issues occurs and no required limitation on the number of sets is present. Typically used in the larger Intel caches - L2 and L3. **PIVT** - Physically-Indexed, Virtually-Tagged. Meaning that all the limitations of previous ways apply, and is rarely used in practice.

Replacement Policy

We need to decide on a policy according to which a cache line will be evicted in order to make room for incoming data. Many replacement policies exist, namely FIFO (first in, first out), LRU (least recently used), LFU (least frequently used), random, a hybrid and more.

Intel commonly applies a LRU policy or a pseudo-LRU policy (since pure LRU is complex to implement). Which determines that the oldest cache line to be referenced will be evicted, As if each cache line is attached with a time-stamp that is updated on access. An example can be thought of by having n way cache set and $n + 1$ different lines of memory that are mapped into said cache set and are accessed in order. The first $1, \ldots, n$ lines will fill the cache ways, while the $n+1$ access will evict the first memory line from the set, since all the set ways are full.

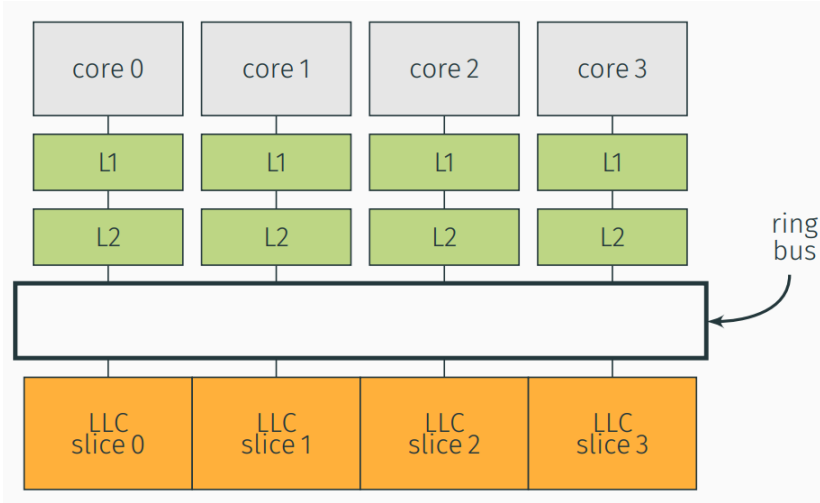


Figure 6.4: Basic Intel CPU - each core has its own L1 and L2 caches, while all connected to a larger sliced L3 cache which can be accessed from all cores.

A potential issue arises with LRU policy when a program need to access cyclically $n+1$ memory lines ("program working set") that are mapped to the same n way cache set. This is referenced as a 'set thrasing' and will result in 0% hit rate. If we compare LRU policy to a random policy, depending on the workload, some studies have shown that LRU and random replacement policies have the same hit rate on average.

When implementing a cache side-channel attack, we will normally will want to evict some special memory location from the cache. Considering a non-LRU replacement policy will mean that evicting a memory line from the cache will not necessarily be practical, since there is no guaranty that if the attacker access memory locations in a specific pattern we will evict the whole set from the cache.

Caches on Intel CPUs

In Figure 6.4 we can see an abstraction of an Intel CPU architecture, on which every core has its own dedicated L1

instruction and data cache, in addition to a slightly bigger L2. All cores are connected via an interconnected bus to the last level cache (LLC/L3). The LLC often is sliced intro the number of cores, having a dedicated slice to each core, while having all cores being able to access all slices. The common practice is that the different levels of caches are inclusive, meaning that a lower level cache is a super-set of the higher ones.

In order to preform a cache side-channel attack, an attacker can optimize its cache usage, among other things, using the following command: `prefetch` - A suggestion to the CPU that some memory line will be accessed soon, can trigger fetching that line into the cache by the CPU. `clflush` - Cache line flush, instructs the CPU to flush a memory line from all levels of the cache. The two instructions are based on virtual addressed.

The different latencies of the different cache levels, as well as the timing difference between a cache miss and a cache hit, as can be seen in Figure 6.5, are the main primitives of most cache side-channel attacks that will be discussed by the end of this chapter.

6.2 Cache Attacks Techniques

Microarchitectural attacks exploit hardware properties that allow inferring information on other processes running on the same system. In particular, cache attacks exploit the measurable timing difference between the CPU cache and the main memory. They have been the most studied microarchitectural attacks for the past 20 years, and were found to be powerful to derive cryptographic secrets [23]. In such attacks, the attacker monitors which memory lines are accessed, not the content of a certain memory line.

Cache attacks are being used in one of the two following common scenarios:

- **Covert channel**: two processes communicating with each other when they are not allowed to do so, e.g., across VMs.
- **Side channel attack**: one malicious process spies on benign processes, e.g., steals crypto keys, spies on keystrokes etc.

We will focus on side-channel attacks.

Cache Side-Channel Timing Attacks

Every timing attack works by the following steps:

1. Learn timing of different *corner cases*.
2. Recognizing these corner cases by timing only.

Here, our corner cases are *hits* and *misses*.

Building the Histogram

The first step towards the attack is to build the histogram of the corner cases, cache hits and cache misses. We measure the time for each case many times in order to get rid of noise. Thus, we have a histogram and we can find a threshold to distinguish the two cases.

Building the histogram for cache hits is done by the following loop:

1. Measure time.
2. Access variable (always cache hit).
3. Measure time again.
4. Update histogram with delta.

Building the histogram for cache misses is done by the following loop:

1. Flush variable (`clflush` instruction).
2. Measure time.
3. Access variable (always cache miss).
4. Measure time again.
5. Update histogram with delta.

Having the two histograms, as in Figure 6.5, we determine the threshold to be as high as possible such that there will be no cache miss below.

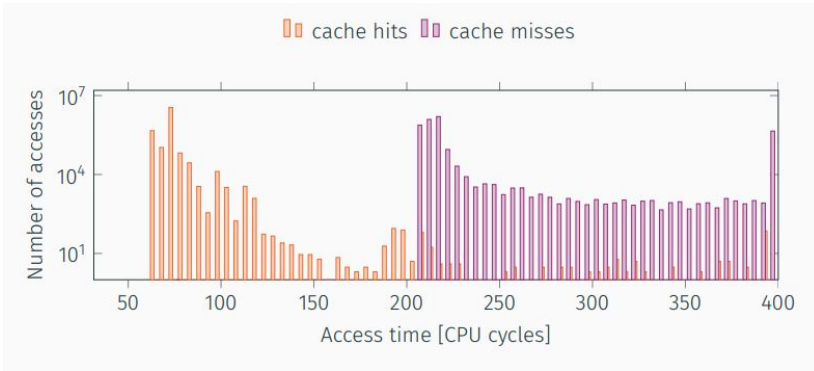


Figure 6.5: Timing differences histogram of cache hits and cache misses.

How to Measure Time Accurately

Consider the fact that the time intervals of our cases is (relatively) very short timings, we ask how to measure time accurately. For such short timings, it is common to use the Time Stamp Counter (TSC) via the unprivileged `rdtsc` instruction as following:

$$[...] \rightarrow \texttt{rstsc} \rightarrow \text{function}() \rightarrow \texttt{rstsc} \rightarrow [...]$$

By using this instruction, due to out-of-execution, the actual execution order could be different, resulting with wrong

measurements. The solution is to use the pseudo-serializing instruction `rdtscp` or to insert a serializing instruction like `cpuid` or use to fence instructions like `mfence` [24].

We will now introduce two cache attack (main) techniques: Flush+Reload [25, 26, 27] and Prime+Probe [23, 26, 28]. Both of them are exploitable on x86 and ARM and can be used for both covert channels and side-channel attacks.

Flush+Reload

The attack is made of four basic steps and it goes as following:

1. **Map.** The attacker maps a shared library (Illustration in Figure 6.6), by doing that he will have a shared cache line with the victim.

2. **Flush.** The attacker *flushes* the shared cache line, it can be done via unprivileged instruction like `clflush`.

3. **Victim.** The attacker lets the victim load (or not load) the shared line, depending on the victim's behaviour.

4. **Reload.** The attacker reloads the shared cache line. If the cache line has a *hit* the attacker infers that the victim loaded the data on the previous step, if the cache line has a *miss* the attacker infers that the victim did not load the data on the previous step.

The first step (Mapping a shared library) can be done once, while steps 2-4 are repeatable as much as the attacker wants. The main advantage of this attack technique is the fine granularity, which is 1 memory line (usually 64B). On the other hand, this technique is somewhat restrictive, it needs `clflush` instruction, which is not always available e.g., on ARM-v7 and it needs a shared memory. Illustration of the attack flow in Figure 6.7.

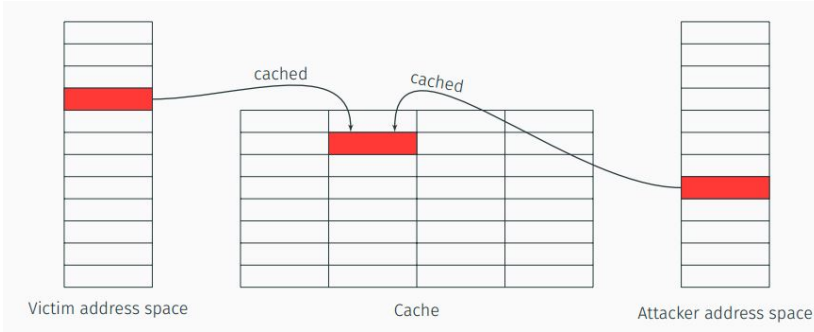


Figure 6.6: Attacker maps shared library (shared memory, in cache), shared cache line marked in red

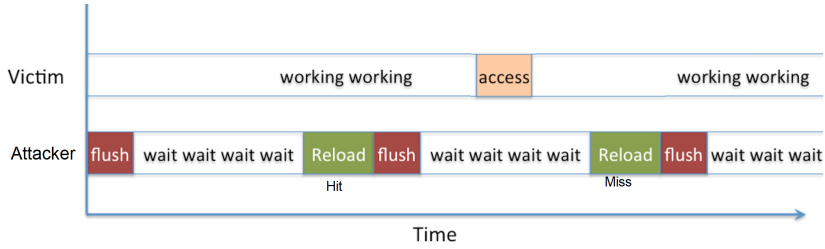


Figure 6.7: Flush+Reload attack flow. Between the first flush and the first reload the victim did not access the shared cache line, so the first reload resulted with a cache hit. Then, after the second flush the victim accessed the shared cache line and hence, the second reload resulted with a miss.

Shared Memory

Common method to achieve a shared memory is by a feature called *page-deduplication*. When two different independent processes are loading the same system library, some of their memory pages will be identical, when the operating system (in a cloud scenario - the hypervisor) identify separate identical physical memory pages, if page-deduplication is enabled, it will merge the physical pages in order to save physical memory, and two different virtual memory pages, one of each process, will be mapped into the same physical memory page. As of today, no cloud service company that takes itself seri-

ously is using memory deduplication, e.g., Amazon EC2.

Evicts+Reload

A Flush+Reload variant which not require using the clflush instruction. Instead of using the clflush instruction, Evicts+Reload [29], is accessing physically congruent addresses in a large array which is placed in large pages by the operating system. In order to compute physically congruent addresses we need to determine the lowest 18 bits of the physical address to attack, which can then be used to evict specific cache sets.

Prime+Probe

First, for the Prime+Probe attack to work, the Last-Level-Cache (LLC) should be inclusive, i.e., the LLC is a superset of the L1 cache and the L2 cache. Thus, data evicted from the LLC is also evicted from L1 and L2. In inclusive caches, a core can evict lines in the private L1 of another core. The attack is made of the three following steps:

1. **Prime.** The attacker primes the cache by reading memory lines from its own exclusive memory (no shared memory).

2. **Victim.** The attacker lets the victim evict (or not evict) lines while running, depending on the victim's behaviour.

3. **Probe.** The attacker probes data in a similar way of the prime step but with measurement of how much time it takes to load the lines (Hit / Miss), for each cache set, the attacker determine if the cache set has been accessed.

In compare to Flush+Reload, this attack technique is less restrictive: it does not require `clflush`, does not assume shared memory and possible from JavaScript. On the other

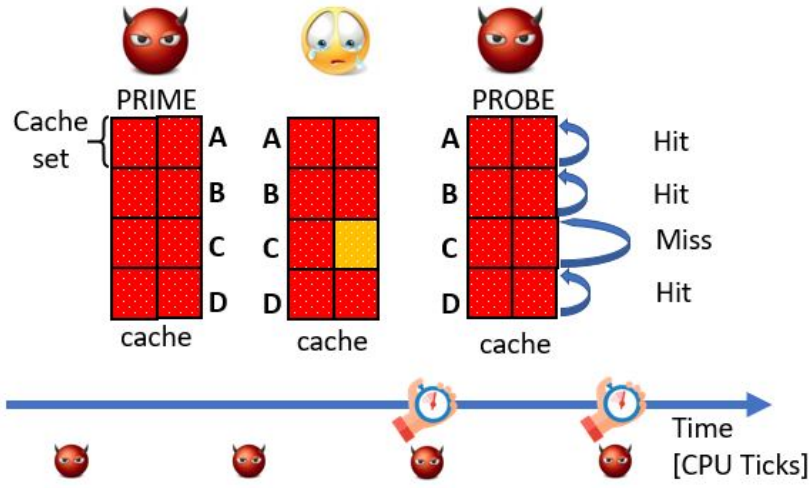


Figure 6.8: Prime+Probe flow.

hand, the granularity is coarser: 1 set. Illustration of the attack flow in Figure 6.8.

In practice, we need to evict caches lines without `clflush` or shared memory, so the following questions arise:

- Which addresses do we access to have congruent cache lines?
- How do we do that without any privilege?
- In which order do we need to access them?

For doing that, we need an *eviction set*: addresses in the same set, in the same slice and an *eviction strategy*.

Eviction Set

[1] We want to target the L3 for cross-core attacks and we need addresses that have the same set index. Consider the

[1] From now on, most of the details are correct for a common Intel CPU architecture.

following cache settings, L3 for a 2-core CPU: 4096 sets, 64B-lines, 12 or 16 ways. Since each memory address indicates one memory byte, the 6 least significant bits of the physical address indicate the line offset. For the sake of simplicity, we assume that the cache is not sliced, since there are $4096 = 2^{12}$ cache sets, the next 12 bits indicate the cache set. The L3 is physically indexed, so we need to choose addresses with fixed physical address bits. Unfortunately, address translation from virtual to physical is privileged. One of the properties of a virtual address is that a page offset stays the same from virtual to physical address, thus, some of the least significant bits of the virtual address can be used as a sneak peak to the physical address. A typical page size is 4KB, which means just 12 bits of page offset, i.e, 6 bits of line offset and 6 least significant bits of the cache set out of 12. To overcome this limitation, we can use a special type of enlarged pages called *Huge Pages* which are 2MB size each, that is - 21 bits of page offset. This way the set index bits are included in the 21 LSB of the address.

We now have another issue, in practice, the L3 is divided in slices, as many slices as cores. We usually have 2048 sets per slice, that is, actually 11 bits for the set index. We cannot infer the slice number directly from the address, neither from the virtual or the physical. The slice number of each memory line is determined by a hash function which takes all the address bits as input, including physical page number bits (outside the known bits from page offset). Illustration of which bits of the address indicates what for both typical pages and huge pages is shown in Figure 6.9.

In addition, the mentioned hash function is undocumented, it designed for performance. But, it does not mean that it is impossible to target the same set in the same slice. Previous work [30] showed that the hash function can be reverse engineered, for example in Figure 6.10.

If the function is unknown, the process will be somewhat slower. But an eviction set can still be achieved via the following algorithm:

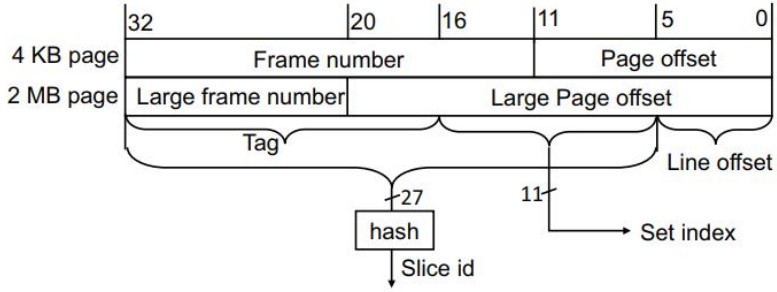


Figure 6.9: Sliced cache.

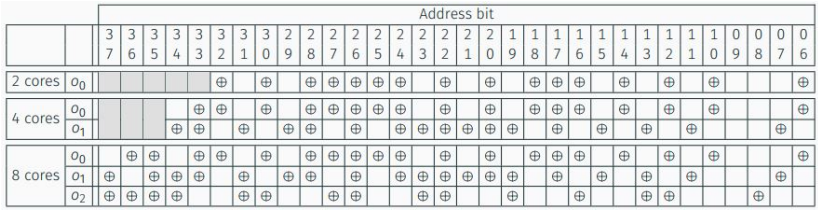


Figure 6.10: Three reversed engineered hash functions, depending on the number of cores. Function valid for Sandy Bridge, Ivy Bridge, Haswell, Broadwell

1. Construct S, set of addresses with the same set index.
2. Access reference address $x \in S$ (to load it in cache).
3. Iteratively access all elements of S.
4. Measure t_1, the time it takes to access x. it should be evicted.
5. Select a random address s from S and remove it.
6. Iteratively access all elements of $S \backslash s$.
7. Measure t_2, the time it takes to access x - is it evicted?
    - If not, s is part of the same set as x, place it back into S.

- If it was evicted, s is not part of the same set as x, discard s.

Note that for a CPU with c cores: $16/c$ addresses in the same set and slice per 2MB page, we can apply the same algorithm with groups of addresses instead of single addresses and speed up the eviction set building process by up to three orders of magnitude.

Eviction Strategy

In the Prime or the Probe step, the attacker evicts a cache set by filling it with n addresses for a n-way cache. If the replacement policy is LRU, it access addresses from eviction set 1 by 1. If the replacement policy is not LRU, the eviction rate is lesser than 100%, e.g. 75% on Haswell. For non-LRU caches, we can use some heuristics, as in Figure 6.11, that will result with a higher eviction rate.

Figure 6.11: $a_1 \dots a_9$ are in the same cache set. Fast and effective on Haswell: eviction rate $> 99.97\%$

Conclusion

To sum it up, in practice, for Prime+Probe on recent processors we need:

- An eviction set, i.e., addresses in the same slice and with the same set index. Depends on the addressing.
- An eviction strategy, i.e., the order with which we access the eviction set. Depends on the replacement policy

Hardware vs. implementations

To perform a cache side-channel attack on some software you need two things: First, shared and vulnerable hardware. Note that there will be no side channel if every memory access takes the same time or if you cannot share the hardware component. Second, a vulnerable implementation. Note that a vulnerable implementation doesn't mean that the algorithm is vulnerable. For example, we can take specific implementation of AES and RSA, this doesn't mean that AES and RSA are broken. Not all implementations are created equal.

To sum up, hardware will most likely stay vulnerable, so patch implementations when you can. And remember, constant time is not enough - because an attacker can modify the internal state of the micro-architecture.

6.3 Step by Step Attack Demo

The target of the attack in this demonstration, is to get the timestamp of the keystrokes pressed by a user in a gedit program. We only target the timestamps and not the keystrokes themselves, as we are not able to fully recover the pressed keys.

The demonstration is performed on a non-virtualized Linux environment. A requirement to perform the attack is having an Intel CPU, as we need the inclusive property of L3 cache. The code for performing the attack can be cloned from the git repository [31]. It is based on the Flush+Reload cache attack that we mentioned in section 6.2 presented in [27] and [29].

The attack is performed in 3 steps: calibration, profiling and exploit. For each of these steps, a folder exists in the repository.

Step 1: Calibration

In this step we want to create the histogram which depicts the cache misses and hits as a function of the number of CPU cycles. Then we will be able to extract the threshold that will match the CPU on which we perform the attack. In order to perform the calibration, we run the following commands:

```
1    $ cd calibration
2    $ make
3    $ ./calibration
```

The calibration works by generating multiple cache misses and cache hits, and measuring the number of CPU cycles it takes to access a variable. Each of these cases is being performed multiple times in order to get rid of the noise.

We build a histogram of cache hits and cache misses as described in previous section 6.2. The output of the calibration program is a histogram of the cache misses and hits as shown in Figure 6.5.

We can then find the threshold so it satisfies the following requirements:

1. As high as possible
2. Most cache hits are below it
3. No cache miss below (we may see one exception due to the way the calibration is coded)

For example, in Figure 6.5, we can see that there is a clear line in approximately 220 CPU cycles.

Step 2: Profiling

After finding the threshold, we can now profile in order to find the cache lines that are actually useful to get information about the target program. Choosing gedit as the target program, we first need to find the shared library that it uses,

so we can give it as an input to the profiler. We then need to find this shared library file location and size. In order to do that, we can use the following one-liner:

```
$ cat /proc/‘ps -A | grep gedit | grep -oE "^[0-9]+"‘/maps | grep r-x | grep libgedit
```

This is equivalent to first finding the pid with

```
$ ps -A | grep gedit
```

and then using this pid in

```
$ cat /proc/<pid>/maps | grep libgedit
```

then copying the line with r-xp permissions (x stands for executable).

Doing that gives the following line (memory range, access rights, offset, –, –, file name):

```
7f2d83197000-7f2d8326d000 r-xp 00000000 08:02 1080575 /usr/lib/gedit/libgedit.so
```

Before we can feed the above line to the profiler, we need to update the threshold to the value we found in the calibration step. We can do this by editing the profiler source file (under the profiling directory) and updating the line with the constant

```
#define MIN_CACHE_MISS_CYCLES
```

to the threshold we have.

After updating the threshold, we can run `make` to compile the profiler. We then use `sleep 3` so we can have time to trigger the event before the profiler starts, and run the profiler with the line we found above:

```
1  sleep 3; ./profiling 200 7f2d83197000-7↩
       f2d8326d000 r-xp 00000000 08:02 ↩
       1080575                      /usr/lib/↩
       gedit/libgedit.so
```

The profiler does its job by loading the shared library to its address space, and then doing flushes and reloads for each address in the address range given by the offset argument (0 in the above case), and the library size (1080575 bytes) for some given time (200 μsec for every address in the above command). The output is the number of hits that happened in every address.

The idea is that if a 0 is shown for a given address while triggering the event, then it means that the line in this address was never accessed for this event. Eventually, we will get lines with a number of cache hits, and we want the ones that have at least a non zero value when profiling.

```
/usr/lib/x86_64-linux-gnu/gedit/libgedit.so, 0x20e40,  15
/usr/lib/x86_64-linux-gnu/gedit/libgedit.so, 0x20e80,  27
/usr/lib/x86_64-linux-gnu/gedit/libgedit.so, 0x20ec0,   7
/usr/lib/x86_64-linux-gnu/gedit/libgedit.so, 0x20f00,  10
/usr/lib/x86_64-linux-gnu/gedit/libgedit.so, 0x20f40,  16
/usr/lib/x86_64-linux-gnu/gedit/libgedit.so, 0x20f80,  13
/usr/lib/x86_64-linux-gnu/gedit/libgedit.so, 0x20fc0,  10
/usr/lib/x86_64-linux-gnu/gedit/libgedit.so, 0x21000,  18
/usr/lib/x86_64-linux-gnu/gedit/libgedit.so, 0x21040,  15
/usr/lib/x86_64-linux-gnu/gedit/libgedit.so, 0x21080,   3
/usr/lib/x86_64-linux-gnu/gedit/libgedit.so, 0x210c0,   1
```

Figure 6.12: Addresses with cache hits.

Running the profiler with offset 0 while jamming a key, doesn't seem to generate cache hits, and we may only see 0. The reason for that, is that the code that handles keystrokes in the library is probably just not in the beginning of the library. Instead (and this is a cheat), we change the starting offset to 20000, and this is approximately where the code that handles keystrokes in the library is found. In real life we will have to

wait until we get to something by running the template attack on the whole library. Running the profiler with this new offset, we can see in Figure 6.12 some addresses that have a non-zero value quite fast. The addresses with the high numbers are the ones we should further investigate in the exploit phase.

Step 3: Exploitation

Equipped with the addresses from the previous step, we can now go to the exploitation dir, change the threshold constant (MIN_CACHE_MISS_CYCLES) as we did for the profiler, and run `make`. Then we can run the program by

```
1  ./spy <library-file> <offset>
```

giving one of the addresses we found as the offset argument. Starting from the address with the most hits, we can see that even if we are not pressing any key, we are still getting events. The reason for that is that this address is related to the blinking cursor. This is, of course, not really interesting information, and we can understand that it's not always the address that has the most hits that is the most valuable. Trying the next interesting address indeed gives us information about the pressed keys, which is what we wanted to achieve. However, moving the mouse also gives us a lot of hits, which is not perfect. What we will need to do is to inspect the addresses one by one until we find an address that will only work for keystrokes.

We can even further improve the attack by finding the complete matrix of the keystrokes for each of the keys as shown in Figure 6.13. We can see that for different keys, there is a different "signature" of the accessed addresses. Although we may not be able to identify each keystroke precisely, we can try to group the keystrokes to different addresses and eliminate some guesses if we are able to spy on more than one address at a time.

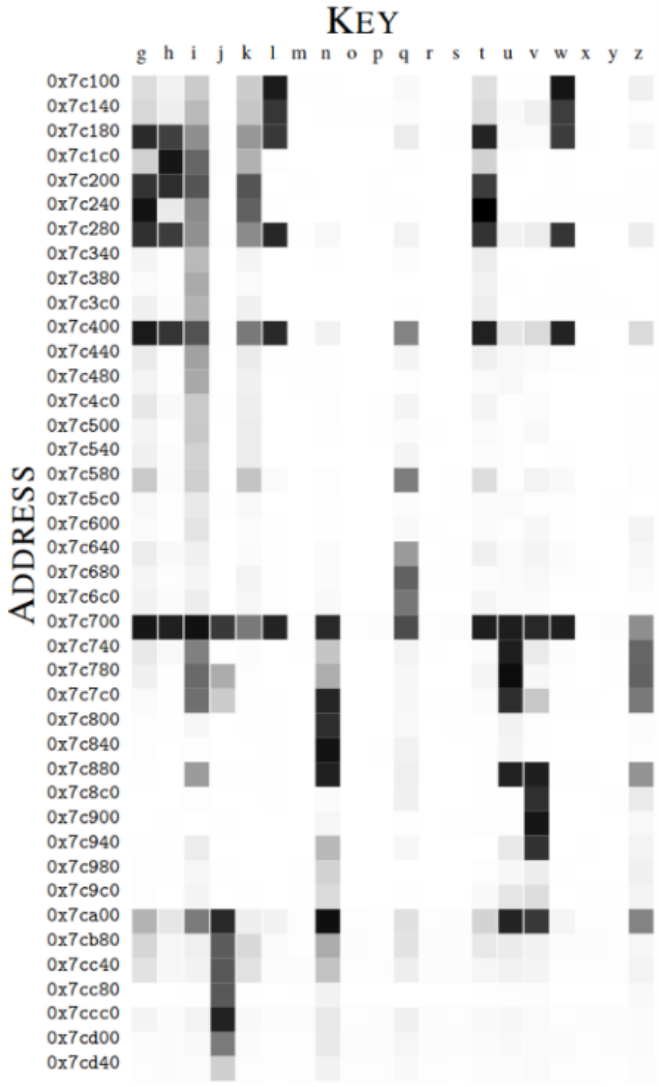


Figure 6.13: Complete matrix for each keystroke.

Finally, as it may be annoying to perform the above process manually, we can automate the event triggering and other stuff as shown in [32].

Chapter 7

High Data Complexity Power/EM 1

So far, we've attacked two cryptographic systems in the course:

- RSA using Montgomery reduction with time as a side-channel
- AES using simple power analysis

We attacked AES using simple power analysis, we had as an input a power trace consist of hundreds of thousands of points and then we actually wrote a classifier or let's say dozen of classifiers and the classifier takes as input the power traces and outputs what is the hamming weight of the first byte of the state, and what was the hamming weight of the second byte of the state and so on...

For AES you have 84 of this classifiers and we didn't actually talked about how these classifier are constructed but you can assume they take as input that power trace and output the hamming weight.

A question arise: does the classifier look at the entire power trace? let's say I have a classifier that wants to find out the hamming weight of the first sub-bytes operation of AES so you take the first byte of the plain text start with the first

Table 7.1: Hamming weights of different bit strings [33]

String	Hamming Weight
11101	4
11101000	4
00000000	0
789012340567	10

byte of the key and then you do sub bytes on this byte: you go to that sub bytes table and you replace the state with sub bytes of the state.

Do you think that this classifier needs to look at the entire 1000 or 100,000 points of the trace? Or maybe it needs to look at the certain limited amount of points in the trace? The answer is that our given power trace contains a lot of redundant information. We start our measurement a while before the first sub-byte operation happens, and therefore there are a lot of points in the data which is irrelevant for us (just an arbitrary operations that happened before the encryption started).

So lets assume we have an army of classifiers, each one of these classifiers is going to be look at the certain points of the trace maybe 10 or 15 or hundred points in this very very large power traces. How can we know to find at which point to look at? We haven't taught that yet and we will go into that a little bit during this lecture.

Now, we want to combine the two attack methods: timing and simple power analysis and take the best part of each one of these methods combining them somehow and the results is going to be called differential power analysis.

When we did a timing attack what was our threat model? What was the adversary allowed to do? We was allowed to send inputs to the device and get the outputs from the device and measure how long it took.

We weren't necessary in control of the inputs we just had to know when the input was provided we didn't have to know what the input was. We just needed to know when was the input when was the output and find the time difference. We had to measure a lot of times and have many inputs, and then we did some statistics to analyze it. The various scenarios when we can justify a timing attack – when the device is in your position, or it's accessible across the web and so on.

What about simple power analysis? What was the threat model there? What we allowed to do with the device? First of all we had to open the device and cut it power supply and connect a power probe which is the meter that measures the power consumption, this is an invasive attack so maybe it's considered less probable that we actually allowed to do that as attackers. If you are doing an electromagnetic probe it would be less invasive but you still will have to get very close to the device, but it would be less detectable.

But what else we had to do before we perform simple power analysis? And now I'm reminding you what I just said couple of minutes ago, how do we actually do a simple power analysis, we have an army of classifiers, where this classifier came from? We need to train them. who do we train classifiers? we give them a lot of labeled data, we need to do what is called: *profiling*, which means we have to get really good understanding of the DUT. How do we get this understanding? We have an offline phase. We have a device to which we can do reverse engineering and get really into it's internals. Maybe look at it with a microscope or we can understand completely how it works and then this device this similar to the DUT. The only things we're not allowed to know as adversary is the secret key.

What is the difference between two devices? according to the Kirchhoff's principles the difference between our DUT to what we use to learn is the *current*.

Now, one might ask: how do we get this copy of the device in the real world? well, we can buy it, we can steal it, maybe

buy it online. you have to have a very involve offline phase before you can do a simple power analysis you have to get close to the device and do a reverse engineering if an justify it by buying stuff on eBay you can go to the trash you can look at the data sheets and so on.

Next thing in our attack is the *online phase*. In simple power analysis you can perform one or two simple measurements and what is the problem with these measurements is that it's our only chance to measure and we have one trace that have to be very clean with very little noise. And of course, we can't really control this noise. For example if we're doing this outside the lab there's going to be some kind of noise, but if we are using measurement equipment the measurement equipment is going to introduce noise to and the DUT is introducing noise by it self. Unfortunately, we can't do so much about the noise in the simple power analysis scenario because we only have the online phase.

What can we do if we had a little noise and we don't have the exact key?

That's exactly our problem with simple power analysis

Maybe only one byte is incorrect and we might try to search for the key around the guess? Let's do a little bit of combinatorics: let's say I got AES key which is 16 bytes long and it's 8 bit implementation and I tried the key and it's wrong. Now, I'm trying to see if I change one byte in the key maybe that will fix the key. So, how many attempts well I have to perform? Lets try to change the byte with index 0 to all possible values, then change at index 1, then 2 and so it goes... How many decryptions should we do? We have 256 per position times the 16 bytes of the key. And if that didn't work I didn't find a key, maybe I'll try 2 bytes combinations, so how many attempts I'll have to do now? It's 16 choose 2 which is more or less 16 squared so it's two to the power of 4×2. So I chose to byte and then what I do is that I go over all the possible combination of this bytes which is two

to the power of 16 so it is two to the power of $8 + 8 = 16$. It grows up really fast and it's going to be as bad as brute force pretty quickly. And what if I have to change tree bytes? it's going up so I have to work very very hard if I don't get the key right it blows up exponentially. One might ask why we are talking about bytes and not bits? if we are looking at a 8 bit micro-controller a lot of the operations are going to be byte oriented so it's going to be more likely to get the entire byte wrong. So if I don't have the key right I am in trouble and I have to search and search very very hard.

The solution to this problem is to have a more complex offline phase which prepare us to the online phase.

Simple power-analysis will work just fine if it's reasonable to conduct thousands of measurement on the same device in order to eliminate noise with statistics, however, it's not always practical.

7.1 High Data Complexity Attacks

(AKA: *DPA* and *CPA*)

The main idea is to capture many traces and use statistics to recover the key. The advantages of this methods are:

- Doesn't require detailed information about DUT (save us the cost of doing reverse engineering to the device)
- Succeeds even under extreme noise

On the other hand, the drawback is that the attack model is different now, we need to own the device. That may cause some people to think that this attack model is irrelevant because it's not practical. In truth, it might be practical in a quite wide range of situations.

Why would we want to get the secret of somebody else?

A good example for it is how TV suppliers calculate your bill. You've a secret and you use it to identify yourself to the supplier. Based on that, the supplier should decide which TV shows you have access to, and how to charge you in case you watch paid TV shows, use VOD and etc... If someone get his hand on your password, he can use it to watch whatever TV shows he wants at whatever cost, because you're the one to pay the bill.

How complex power analysis actually works

Until now, we knew about 2 ways of attacking with side channel:

1. *Timing Attack*
2. *Simple power analysis*

When we used timing attack, we tried many input, and the measurement for each input contained 1 output: for 1 input we had the time it took. On the other hand, when we did Simple Power Analysis, for each input we had a huge power trace (vector of points which indicates the power consumption over time for a given input). In other words, the data complexity was simple in both of these methods - we had a $1d$ input. Now we'll try to take it a step further and combine these methods and create a new method with $2d$ input, which mean now we have high data complexity. Instead of giving just the power trace, we'll give as an input the traces for various data at a fixated points in time. We have a set of data $D_1, D_2, ..., D_n$ and we have the power trace of the system for each one of these inputs, at the the time, i.e. the first point in the power trace for all these inputs, was taken at the same relative time. So now our output isn't a vector but rather a Matrix M_{DxT}.

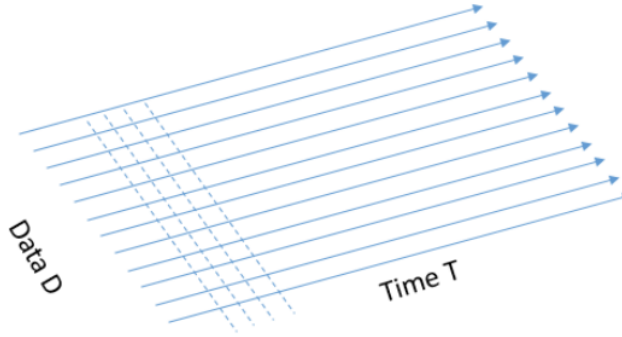


Figure 7.1: many measurements over time

The suspicious reader should ask: how it's possible that we measure the power-traces of all the different data inputs all at the same time? That is indeed a challenge to align all the measurement along the time-line and some of the countermeasures to the attack we present here, will insert randomized operations or will change the clock speed so that we'll have trouble to align the power-traces along the time-line. If you're curious about that, you might find it interesting to look on the DPA book (the course book) and read about the countermeasures and the ways to overcome them.

Using the Vaizata method

The next thing we're going to do is to use the Vaizata similarly to how we did it with timing attacks. When we did timing attack, we made a guess about a bit of the key and then we measured how much time it took. Now, we're going to make an assumption that at sometime, the algorithm depends on the key, therefore the power consumption will be depended on the key. Finally we're going to check it with the measurements using some statistics.

Unlike the timing attack where we knew that we measure the relevant data, when we used power analysis, we have a very long power trace where only few points relevant to us and

all the other points represent data which isn't related to the key.

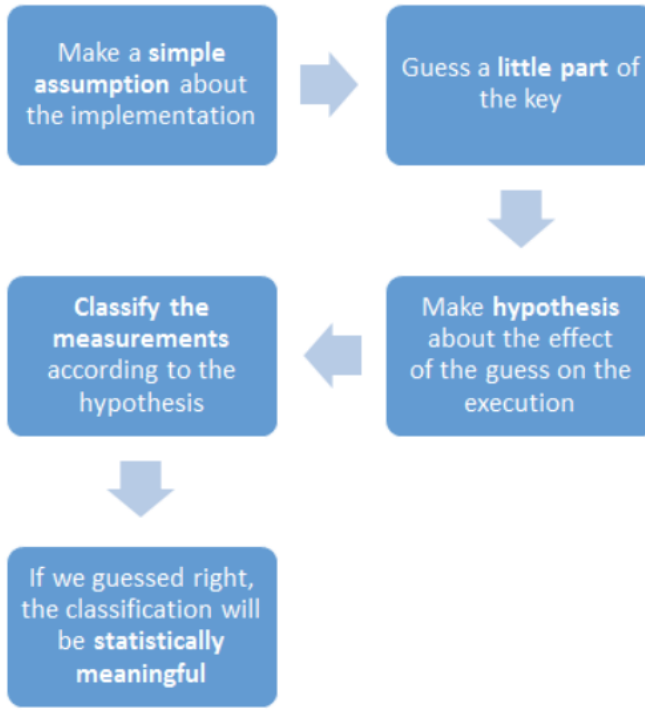


Figure 7.2: Vaizata method stages

Now, we're looking for a stage during the AES algorithm, where a small amount of bits of the key, affect large amount of the bits in the state. The intuition here is that we want to find a stage in the algorithm, where the power consumption depends on the key value.

We might consider measuring the power consumption just before the start of the algorithm, but in this case the key has no affect on the state. Next, we have the add round key. After this operation, each bit of the key affect exactly one bit of the state. Why is that? because we essentially "XOR" the key with the state, which so the the bit $k[0]$ affects the bit $s[0]$.

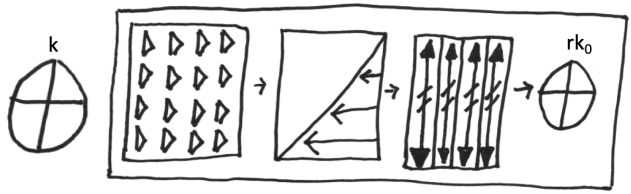


Figure 7.3: AES stages

Our second operation will be the `subbytes`. The `subbytes` operation is a value based substitution, where each possible state value is mapped to a different value as defined by the substitution matrix. In this stage, every bit of the key, affects 8 bits of the state! Why is that? if only one bit in the key was different, our add round key will cause another bit of the state to change, and that would lead the `subbytes` look-up to bring an entirely different byte to the result state.

Next we have `shiftRows`, where still every bit of the key affects 8 bits of the state. Why is that? because we just moved the data, we didn't diffused it.

And now we have the `MixColumns` operation. How many bits of the state now is affected by every bit of the key? In the `MixColumns` operation, each byte depends upon 4 bytes, and each one of these 4 bytes are 8 bits that depends on the key. So we have now 32 bits of the state which is depends upon a single bit of the key.

As an attackers, we're going to focus on the point immediately after the `subbytes` and right before the `shiftRows`.

So based on this information, what should be our Vaizata assumption? The assumption is that's the power consumption of the device is the function of the data so if we change the data, the power consumption changes. (A counter-measure for that will be to build a machine that consume the maximum power all the time).

Similarly to our attack on the RSA algorithm where we guessed a bit of the key, we're going now to guess a byte

of the key (since AES is byte oriented). When we attacked RSA, we've splitted our data to 2 groups and then conducted $t-test$, but since we guess a byte, now we have a 256 groups.

We know the value of the plaintext (it's given) and we're making a guess about the key, but how will we know the value of the `subbytes`? by Kerckhoffs's principle (which means that everything regarding the algorithm but the key is shared publicly).

So for each data point, and for each byte guesses, we have 256 options. 255 out of them will be meaningless and only one is the correct guess which should have statistic significance.

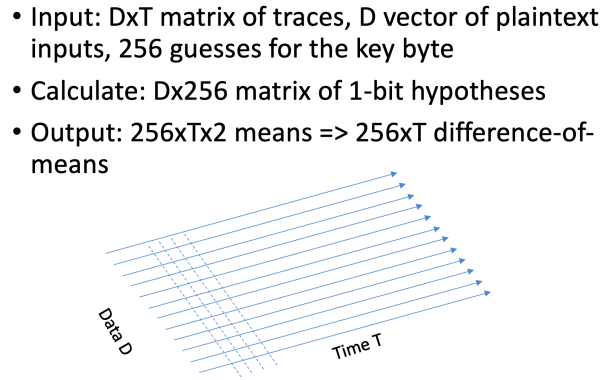


Figure 7.4: The way we separate the data to groups based on the guess

The problem is that we have many different points on time, and we shall see a difference only at a certain point. Therefore what we do is as follows: for each guess, we split the measurements to 2 sets and we measure the difference between the means of their power traces on every point of time. At most of the points, the difference won't be significant, except to the point where the guess was done correctly. We illustrate it at Figure 7.5.

On the $X-axis$ we have the time and on the $Y-axis$ there's the difference of means. In Gray we can see the noise and even if we guessed the correct key we have a lot of similar

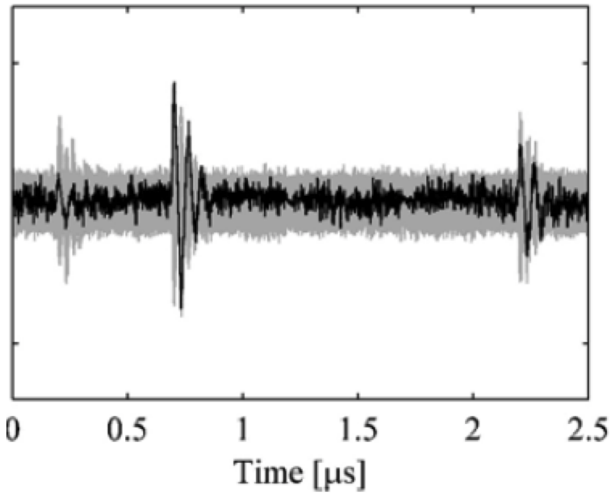


Figure 7.5: the difference of means as a function of time

points and that is because not all the times depends on the key. But we can see in the figure the moments of peak in the means difference, these moments are also called "the right times". These essentially should be the moments of the sub-bytes. If we incorrectly picked a peak which is a ghost noise, like in a point when we xor the plain the with the key, then we won't be able to find meaningful separation later on.

7.2 DPA Lab

In the class examp le, we attacked AES algorithm with 200 trace with size $30,000$ using an example data from the *Power Analysis Attacks* book and we visualized the process. First, we load the data and plot it in Figure 7.6 with the following axes: $X-Axis$ will be the time and $Y-Axis$ will be the trace number. In addition, we had a dimension of color intensity which will be depend on the power consumption.

What we can see in plot 7.6 is that there are very aligned columns which indicate that the traces are aligned properly. Our goal is to separate the power traces of the correct key

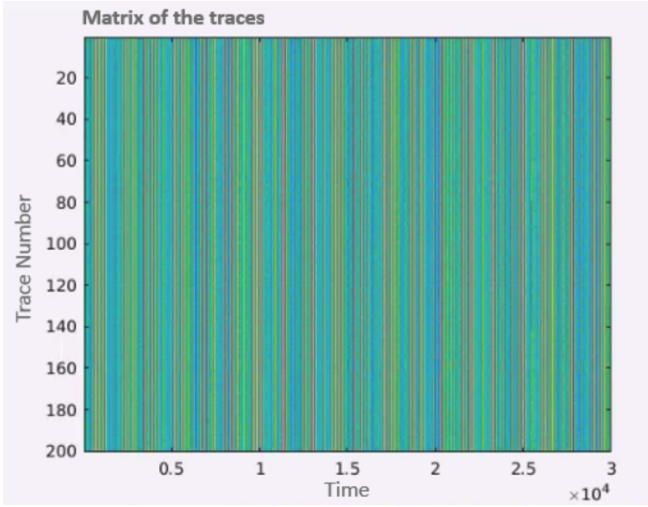


Figure 7.6: the traces' power consumption by time

from all the other traces. To do that, we start by showing in Figure 7.7 by comparing the power consumption of 2 traces based on time.

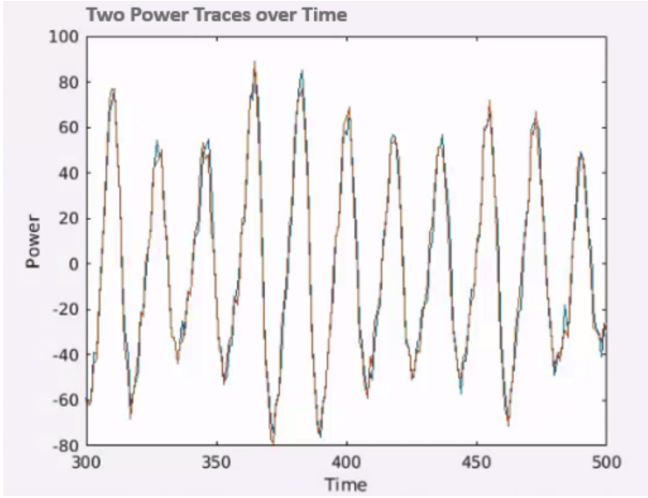


Figure 7.7: 2 groups of traces by times

We can see that both of them are overlapping, but if we had a difference between them we would be able to find a

separations on this graph.

Therefore, in Figure 7.8 we test for each key guess and input, what was the power consumption, whereas yellow represent less power and blue more power.

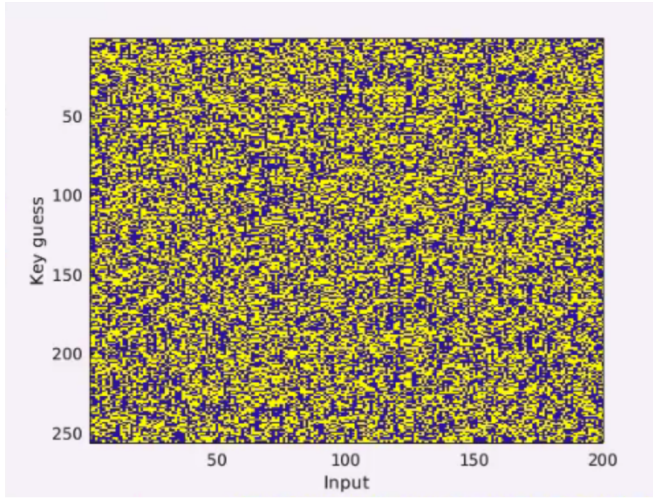


Figure 7.8: power consumption by guess and input

Now we calculate the mean of power consumption for each guess and we plot if over time

If we made the split correctly, this power trace average will be different from all the other 255 graphs. The problem with this chart is that we need to check many graphs, so we need something that shows the distance of means for all the guesses. A better graph will be to plot the distance of means of each key guess by time and then to search for the moment with the maximum difference ("the right time") like in Figure 7.10.

While it might be hard to spot it on this chart, there's a certain point of time where the difference is 8.91, and at all the other points it's somewhere near 0 as we can see on the charts.

What if we focus on the correct time across the different key guesses? we should be able to spot the correct guess as shown

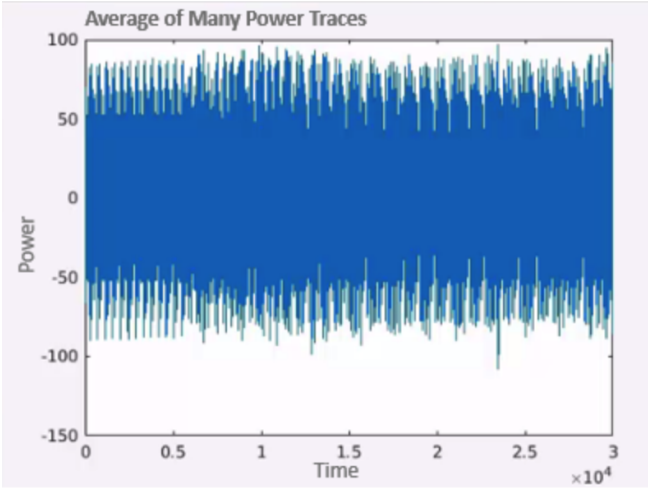


Figure 7.9: mean of each guess across the different inputs by time

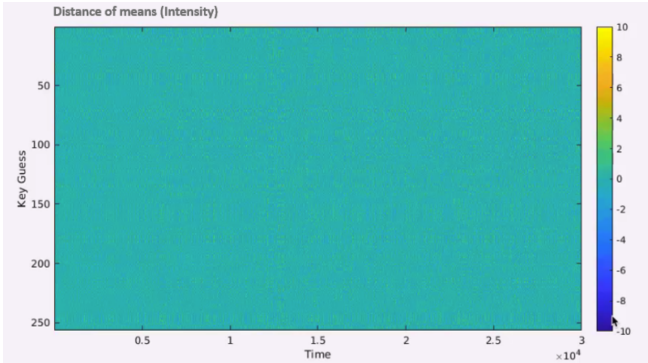


Figure 7.10: distance of means by time and key guess, color intensity represents the distance

in Figure 7.11.

We finish by plotting the distance of means across time for all the keys, we can see in green the distance of means for the correct key and we can see that in the right time it's separated from the other keys as shown in Figure 7.12.

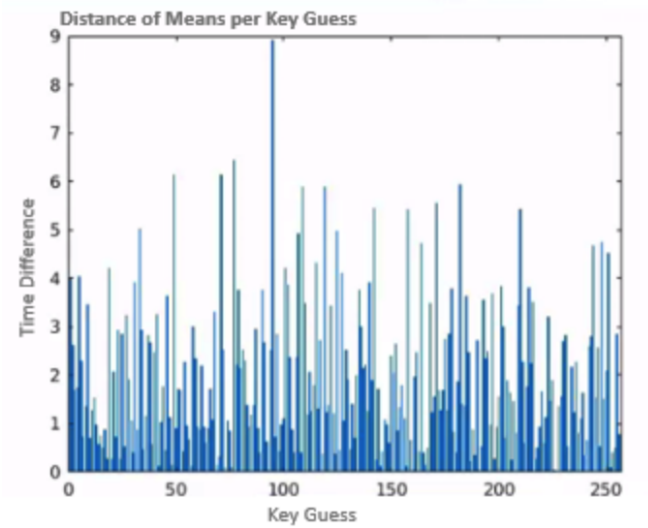


Figure 7.11: means distance at the correct time

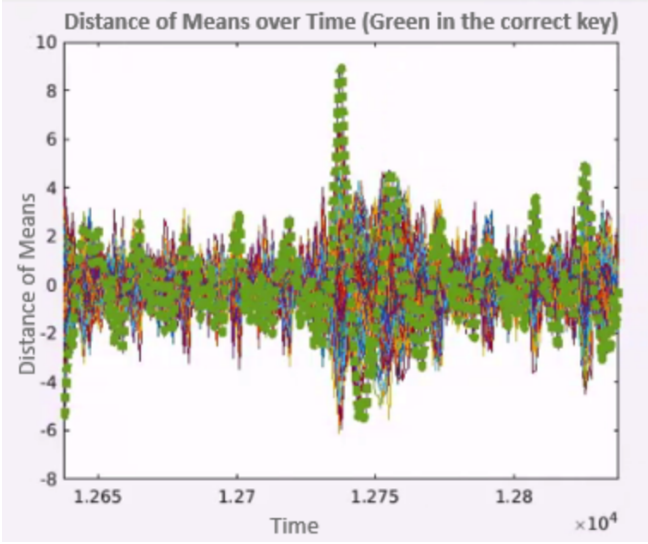


Figure 7.12: distance of means for all key guesses across time

Chapter 8

Correlation Power Analysis

This lecture is the last technical lecture of the course about High Data Complexity Power Analysis, CPA.

8.1 Previous lectures recap

We are interested in laying a side channel attack. We got a device preforming secret operation, AES operation. The device works as follows: we give input to the device (plain text) and then it works on the input to produce an output (cipher text) using a secret key. We want to recover the key. So, if we collect many pairs of plain text and cipher text, what can we do with that? Lots of things. In addition to plain text and cipher text, we have a side channel [34]. What is a side-channel? A side channel is an artifact of the computation and what's nice about the channel is that it depends on the intermediate values of the computation and not the output of it. We talked about number of side channels throughout the course

Timing side channel

First, we used the time it took the computer to check for correct password in order to find the password. Later, we used a timing attack [35] to attack RSA [36]. Actually, we didn't attack RSA, we attacked an implementation of RSA which was left to right multiplication exponentiation that uses a special representation which is called Montgomery (the Montgomery basis). The Montgomery basis [37] representation which is very easy to do multiplications in this implementation, almost as easy as doing regular multiplications, but there is a step called Montgomery reduction which the computer does sometimes. Not a very expensive operation but the computer only do it sometimes and because it does that that sometime, in each encryption operation each exponentiation takes different time. The time is a function of the plain text and the secret (in the case of RSA is the private key). So how can we perform timing attack on RSA? We had a device, the device got an input or message to sign and output the signed message and it also outputted the time it took for the calculation. Our data structure had plain text, signature(cipher text) and time. The first step of the attack is to take all the signatures and to throw them away. In the next step we tried to guess one bit of the key. This guess helped us simulate a tiny little bit of the secret operation, just one step and this step which we simulated. Since we guessed the key, we could also guess this tiny step included a Montgomery reduction. Our hypothesis is that if there is an extra Montgomery reduction, the run time will be longer and if there is no Montgomery reduction the run time will be shorter. Now we have a data structure: message and a bit, if the bit is 1 it means we think this was take longer and if it zero it means shorter. So, we divided our set of traces into two groups, each one we gave a bit, 1 or 0.

Now, how do we know that we guessed this bit correctly? We have in each row a bit which says if we think it is longer or shorter and the time it took. The hypothesis that we are trying to prove is that the messages that we marked with 1

are going to take a longer time to compute than the ones that we marked with 0. We have two sets of traces: the one which are marked as 0 and the one marked as 1. Our hypothesis is that they take different time to execute. We test this hypothesis using statistics. We can use student's T-test or we can just compare the average. So we guessed a key bit, how many ways are there of guessing this key bit? Two ways. So we have two hypothesizes, two proposed ways of splitting our traces into two sets. In such a way, we can crack the key.

Differential Power Analysis (DPA)

In DPA [38, 39] the output of our device under test (DUT) is the cipher text. AES is a very structured cipher, it has a fixed structure and it's going to be very rare that we will have a different run time. So, usually there is a program running on a CPU which has a clock and in each tick execute operation. AES is very simple: it shifts, xor and so on. Luckily, we have other side channels, such as current. The attacker took the device and measured the instantaneous power consumption of the device. There are two ways for the attacker to do this. The first way is to do this directly: take the current that is exiting the circuit to the ground and instead of letting it go directly to the ground you send it through a little probe, that will measure the power consumption on the circuit. The advantage is that we get is a clean signal and the disadvantage is that we will have to quite aggressively manipulate the device and maybe it's not a reasonable assumption to make. The second way is electromagnetic (EM). What does EM means?

Current (I) flowing through a wire and the change in the current generates EM wave, so if we will put an antenna next to a wire we will be able to pick up the current on this wire next to the antenna and to do this we only need to get close (don't need to cut anything open), which makes it more reasonable attack model. The disadvantages is that you have

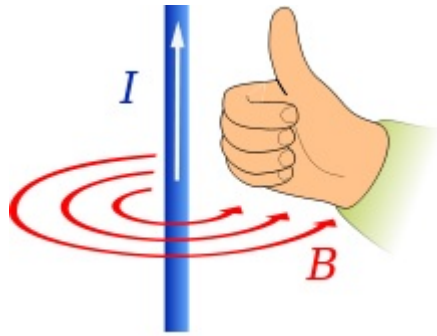


Figure 8.1: Right hand rule

a lot of noise going on and you will have a lot of RF signal processing to do so it's not trivial to get from there to the key. Let's assume we are doing PA without EM. So, your output now is a data structure. This data structure has plain text, cipher text and power consumption, but power consumption is not a fixed value, it's a trace vector. Now, if you think about device under test (DUT), most of the time when we are doing the measurement of the power consumption it's not actually handling the secret, it's doing something else. So if we look at the first byte of the key of AES, its processed in the beginning of the encryption where there is adding the key operation. The idea of AES is that every bit of the key will get mixed into the whole cipher text, so it's not very easy to track it. In the beginning of the encryption the key is not being processed at all and at the end its affected so dilute into the cipher text. So there is going to be very specific time when this trace is interesting. From a very long trace with millions of points there are only few interesting points for an attacker. In DPA we had one number, the time, and we knew that the signal was decoded in this number. Now we have a big vector and we know the signal is somewhere in there but it's not in all of the points. But we did the same thing here, we guessed a little part of key and then we gave each one of the rows in the data structure a bit, one or zero. So, we looked at some kind of intermediate value, we have the AES

state and we took a little bit of the state and we gave each one of the rows in the data structure again, a one bit value and the one bit value was exactly the value of this bit of the state. Actually we have a way of splitting our group of traces into two, same thing as the timing attack. We claim that if we guessed the bit correctly we just made a meaningful split of the traces. Now we divide our traces into two sets and we want to test if there are different distributions. We hope that at some point (the correct time) there will be a good distribution. An important thing to understand is how many ways we have to split the traces into two in the timing attack. If we guess one bit of the key, there are two ways of splitting. Each time we decided we guessed a key bit we simulate it a little bit and then we gave each entry in the data structure a value of 1 or 0. However, AES is a byte oriented algorithm, so there is 256 ways of splitting, i.e. We have 256 different competing ways of splitting the traces into two. But again, we split the traces into two. So, once we decide on a split we go over the data structure and each row we assign a value of 1 or 0. Then we perform difference of means, i.e. we take the mean of the right, the mean of the left and we have 256 different ways of splitting into two and look at the difference of mean where we get the largest difference between the two means.

8.2 Correlation Power Analysis (CPA)

Hamming weight

The hamming weight [33] of a number is the amount of bits which are 1 in it's binary representation. Hamming weight can refer to a hamming weight of state, or to hamming distance, which means how many bits change between from one state to the next. Lets assume we are looking at the hamming weight of a byte. This hamming weight have values between 0 and 8. A number with hamming weight 0 will be

0 and a number with hamming weight one will be 8, 2, 4. For m independent and uniformly distibuted bits, the whole word has an avergae Hamming weight $\mu_h = m/2$.

CPA background

We need to find a way to know when the traces are different. In DPA [38, 39] we guessed one byte of the key so we have 256 ways of splitting, but still, based on this guess we only guess one bit of side-channel. So now we have 256 different ways of splitting the cipher text into two groups and if we guessed right the two groups will be different, but they will be different only at the correct time. At previous lessons we learned that most computers these days are based on technology called CMOS. The idea of CMOS is that if nothing much is happening, if it's in a static state, the power consumption is very low. But, if it's dynamic, if it switches between 1 and 0, there are bursts of power consumption. So, in a CMOS device the power instantaneous power consumption is correlated with the amount of bits which are flipping in this time period. This fact was completely ignored when we did DPA. We just said the power consumption was different. As opposed to DPA, CPA doesn't ignore that fact.

In CPA [40, 41, 42, 43], we are going to guess one byte of the key, but we are not going to look at one bit of the state but we are going to look at the hamming weight of the state. Assume that if we guessed a byte correctly, the CPU is going to have to handle a byte with it's hamming weight. For example, if we guessed that the state was 3, then the CPU will have to handle a byte with hamming weight of 2. Our data structure is plain text and instead of a division to 0 or 1 we will have a number between 0 and 8. This number is the hamming weight of the byte we expecting the device to handle. Instead of splitting the trace into two groups, we are going to use correlation. Due to CMOS, we can add a very reasonable claim that when more bits are changing the power consumption raises.

CPA on AES motivation

AES [44] is byte oriented so we have 256 different guesses. Now, we are trying to give label to each one of the inputs. We start with a guess and then we give it a label. The first step of AES is to take the bytes of the plain text and XOR them with the secret (bit wise operation). The next step is to apply the subByte transformation, which is basically a look-up table. After applying subByte we have a byte which is a mixture of the plain text that we know and the key which is secret. So if we guess a key we can completely simulate the value. After subByte we apply shiftRows. But shiftRows is just reading and writing the same value from memory. It actually reading this value (HW(S[P xor K])) and make it even stronger. Next there is Mix columns operation. The mixColumns operation uses 32 bits so we will have 232 different guesses to do mixColumns. Since 232 is a low we will need a lot of traces and we will need to measure the DUT a lot of times. Then we classify the measurements. There is going to be a statistical meaningful classification for correct guesses. We refer readers to [45] for a further detailed explanation of AES algorithm.

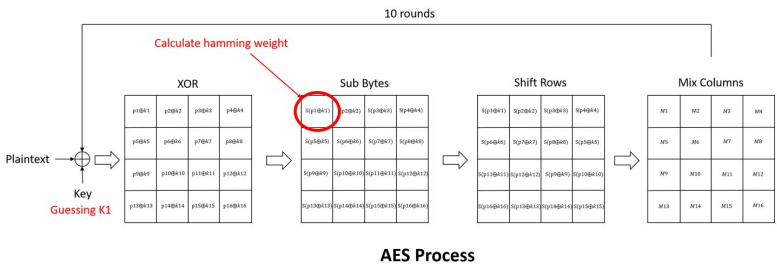


Figure 8.2: AES illustration

In the timing attack days the meaningful was that we can run the students t-test and get a division into 2 sets. In DPA we did difference of means. To find meaningful classification in CPA we will need the statistical Pearson Correlation Coefficient [46]. The Correlation Coefficient is simply the covariance normalized to be between -1 and 1. The intuition is

that if the correlation coefficient is zero than it means we are not guessing the key correctly and if we are guessing the key correctly we are getting a correlation coefficient which is high, notably better than what we get on the other key guesses.

CPA's data structures

When we were looking at a timing attack we had a data structure of size D, we had D inputs, each input was a message we are trying to sign. To each of these inputs we had a trace, our side-channel trace. The trace was a single number. Our data structure was D times one. When we do High Data Complexity power analysis we are going to do D different inputs, and each one of these lines represent a single trace, i.e. long vector , and not a single point like the timing attack. The vector with size T represents power consumption of the device running a input in T different points of time. So we have a matrix data structure of size D times T. D different inputs and T different times. Each line of the matrix can be thought as if we fix the plain text and ask what the power consumption over time of this plain text is.

Each column of the matrix can be thought as if we fix time and ask what is the power consumption of the device in that particular time. One of these times called the correct time, and we will get statistical meaning in that time. In other incorrect times we won't get any statistical meaning because the DUT is not processing the data we are looking at this particular time.

Note that in order that the CPA attack succeed, we have to align the traces to the same time. Together with this matrix we have another data structure which is the inputs, so each one of these lines says "this is the encryption of plain text A, this is the encryption of plain text B" and so on. There are all sort of ways of getting it, depending on the attack model. One way is that the attacker has a signing oracle and the attacker chooses the plain texts and send them to

the device. This is the easiest model of the attack. Another way which is a little more difficult is that the attacker has to commit ahead of time all the plain texts and then the device encrypts them all at once, so he can't adapt to whatever input he gives.

CPA on AES step by step

We want to do a CPA of the AES encryption. We want to guess one byte of the key and we want to get this byte mixed with the plain text which we know and we don't want to many bits of the key to get mixed in because than we will have a lot of guesses. So, the plain text comes in here and we know the plain text. We mix it with the unknown key. The key is a guess. How many guesses are there for the key? 2^{128} ways of guessing the key. However in order to not guess the entire key at once, we can guess only one byte at a time, so actually there is 256 guesses.

For a key guess, We can simulate the key bit XOR plain text. Then, we can simulate subByte of (key XOR plain text), and then we can also simulate shiftRows on that result. This is where our simulation stops, because to simulate mixColumns we need to guess 32 bits of the key and we don't want to guess 32 bits. The trick is that when the DUT is manipulating this value (the value we got after the simulation stopped), it will have some power consumption, and that power consumption will be correlated to the hamming weight of that value (according to the assumption we have on the CPA attack).

Example: Let's assume we guessed the key byte as 5. So, if we know the guess of the key byte is 5 we can look at the plain text and we know the first byte of the plain text, so we can do key XOR plain text. We can do subByte of key XOR plain text and we can calculate the hamming weight of it. So, we have a vector of size D which is the hamming weight that we guessed for each one of the inputs. We have a function which receives as input the plain text and our guess and outputs the hamming weight. We have D different

inputs, so we will call this function D times, and will get an output of vector of size D.

If we guessed the key correctly, at some point in time, which is called the correct time, the power consumption will indeed be correlated (close to +-1) with this vector that we guessed and all of the other times will not be correlated (since the DUT is not only processing this byte of the key, but it is processing other bytes of the key, other things and more). But in the correct time there is going to be a correlation. If we didn't guessed the right key, there will be no time which will have a correlation, meaning the correlation is close to zero. If I say that there is a correlation, meaning the correlation is close to +-1.

CPA warm up example

We have a DUT, we have a matrix of traces, we know the plain text, we know the cipher text, we know the power traces and we know the key. So we have the power trace and you want to find where inside the power trace is this bit being processed. Our objective now is to find out when is the DUT processing the first byte of the key. It could be interesting for us if you want to do fault attack [47] or template attack [48] or if you just want to make sure that you know how to do CPA. As we saw earlier, the hamming weight of the subByte output is a good hypothesis.

Now, let's look at our data structure. We have a matrix of traces, D time T. Assume we have 1000 traces. Each one of these traces covers a long period of time, several millisecond so it has 1 million points. Each one of these values is not Boolean, usually it's one byte if you are using a regular source code. We also have the inputs, we know for each one of these traces what was the plain text being processed. The size of the plain text in AES is 16 bytes (128 bits). So we have another data structure which is 128 bit by 1000 rows, so each input is 128 bit has 1 Megabyte of a trace attached to it. We also know the key, so we don't have to guess it. So now,

we are trying to find out where is the correct time. We want to find out for example, when is the DUT processing the first byte of the key. So we go over each one of our plain text. We can do plain text XOR key, we can also do subByte of that and we can do hamming weight of the output of subByte. To each of the input plain text we associate the output hamming weight. Our hypothesis is that the hamming weight the device will have to manipulate is the output hamming weight we calculated. We created 1000 such numbers. Each of the rows will now have plain text, trace, and hypothesis.

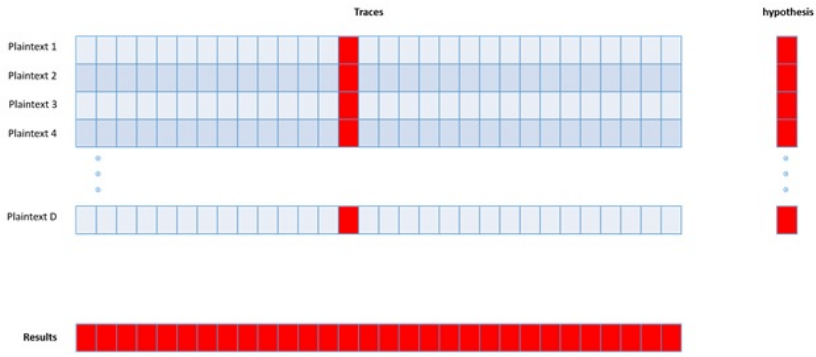


Figure 8.3: Graphic way of CPA warm up example

The next step is to take the hypothesis and to sweep it across all moments of time. Assume we have a fixed certain moment in time, i.e. a vector of size 1 time 1000. This vector is the instantaneous power consumption of the device at the fixed particular time. Recall that the hypothesis is also a vector of size 1 time 1000. For 2 vectors of size 1000 we calculate the Pearson Correlation Coefficient. The Pearson Correlation Coefficient function receives as an input 2 vectors of 1 by 1000 and outputs a single value between -1 and 1. If this value is very high or low it means that there is a correlation between the input vectors, but if it is close to zero it means there is no correlation between the vectors. We will execute this process for each fixed moment in time, in our example we will run in 1 million times. So the output of this step is a vector of size 1 million by 1. So 1 million different times,

for each one of these times we have a single value between -1 and 1. Now we can look at this and see where the maximum absolute valued point is and it is probably the correct time.

CPA example

However, in the warm-up example we had one assumption that not always correct - we assumed we know the key. In the following example we want to do CPA in case we don't know the key. We still have a matrix of traces. To each trace we have the associated input plain text. In order to solve the key problem, we will run the warm-up CPA 256 times. Each time we guess the key. If we are guessing the key as 0 we will have a hypothesis vector and we can sweep it and get a vector of million times one. We can also try key 1 and sweep it again and get another vector. Also for key 2, key 3 all going to key 0xff, i.e. 256 different key guesses, which gives us 256 times a million possible correlations. In this case we have 256 guesses and not single hypothesis vector, but a hypothesis matrix. 256 different hypothesizes, each column in the hypothesizes matrix is like the warm-up. Then we take this matrix and sweep it across this matrix. At the end we get million different time by 256 different key guesses and each one of the points in this matrix represents the correlation between our hypothesis and the instantaneous power consumption of all of the devices at one particular moment in time. So this is a two dimensional matrix and each row corresponds to a key and each column corresponds to a time. So, most of the times, 99.9%, this matrix is going to be more or less 0. But there is going to be some points where there is a good correlation, and this point are at the correct time and the correct key!

Let's do this in pictures. So, when we had a single hypothesis we would take this red row and we sweep it across this red line, this was the warm-up. But now we don't know the key if it is red or blue or green, so we take this blue vector and sweep it across and we will take this green vector and sweep

it across. So this is the output here, it has the same number of rows as the number of key guesses and the same number of columns as we have times. This is full CPA and hopefully in one of these rows we will have a peak and we will be happy.

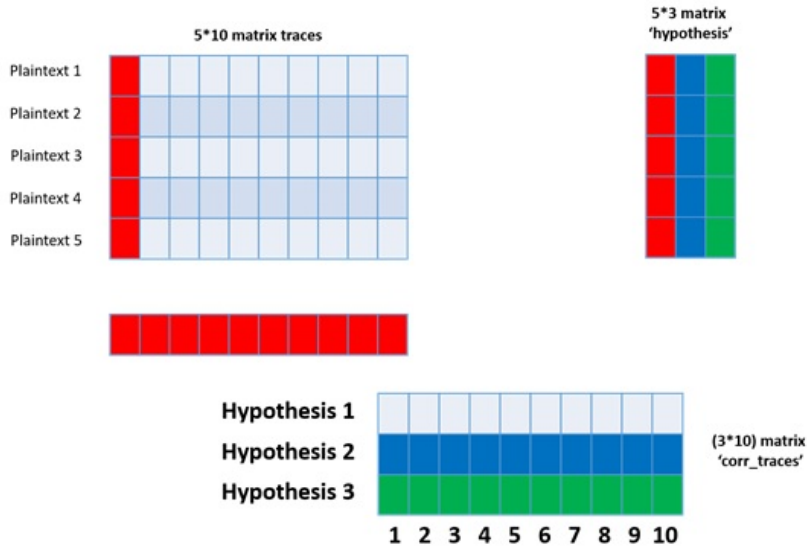


Figure 8.4: Graphic way of full CPA example

CPA in Matlab examples

We will show Matlab example which includes CPA, warm-up CPA, and the full CPA as seen in this section. We are working on a data set called WS2 [49] which is part of the DPA toolkit. It has 200 traces, each one of size 30,000. These are power traces of the first round of AES on an 8-bit microcontroller which is very low clock speed and it's very vulnerable to DPA. In addition, that test setup is a lab setup so this is the best we can hope for in terms of DPA. For each one of the traces the plain text is known. Figure 8.5 describes the 200 traces power consumption as a function of the sample number.

The density of the colors indicates the instantaneous power consumption at a particular moment at the DUT. The traces

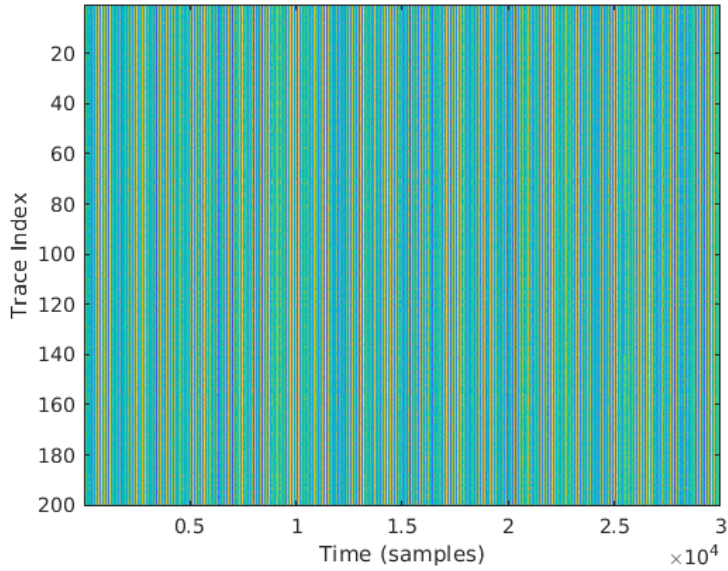


Figure 8.5: Power consumption as a function of sample number

are very well aligned. Aligning is not a simple task but this device was actually aligned. It was a lab setup so it is easy to align. We can see that the device is doing the same thing at the same time for all of these devices. If we will zoom in, we will see variation between the traces. If we will zoom in to a two different traces, we will get figure 8.6. The X axis represent the time (in samples) and the Y axis represent the power. The power consumption is between +-80. These two traces are very similar but there are some differences. The differences could be measurement noise or some other signal processing.

For explanation let's do the warm-up example. At this example we know that the first byte of the key is 120. Now that we know the key and the plain text we could say that when we are at the correct time the DUT is going to be processing a hamming weight of the correct key. In a DPA, first of all we find the plain text XOR the key, at first we are looking

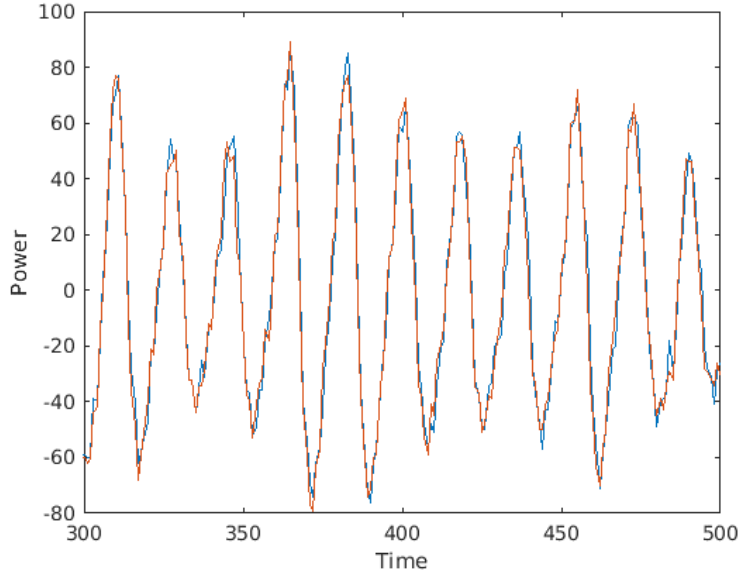


Figure 8.6: Zoom-in on two different traces

only at the first byte, the result is a number in the range of 0-255. Then we apply the AES subByte operation. Then we are calculating the hamming weight of the outcome the values of the results are between 0 to 8. By doing this we get the hypothesis_vector for each particular input.

Figure 8.7 presents the hypothesis_vector values (the X axis is not important). For each trace (Y axis) we calculate the hamming distance (presented by different colors). Now that we have figure 8.5 and figure 8.7 we are going to sweep the later figure over the former and for each time we are going to calculate the Pearson Correlation Coefficient. Thus, for each position we are going to get a number, and this number will be between -1 and 1. Figure 8.8 describes the results. The X axis it the time in samples and the Y axis is the correlation Coefficient between the hypothesis and the actual power consumption. Most of the time it's around +-1 but there is a peak described by the red circle. This peak describes the actual time that the AES making the sub byte operation on

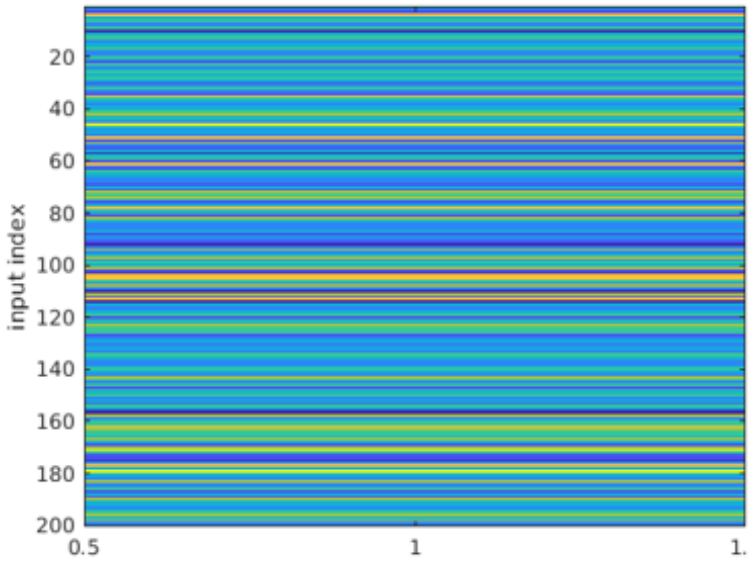


Figure 8.7: Hypothesis vector values

the first byte of the key.

However, this was the warm-up example. In a real attack scenario we don't know the key. A real attack is very similar to the warm-up example, expect that we are padding the key guessing by a loop and at this loop we are checking all the 256 available for the key. A real attack is described at figure 8.9.

The X axis is time, and the Y axis are the key guesses. The color is the correlation. Figure 8.10 is the trace classification matrix. The X axis is time, and the Y axis is the key guess. The color is the correlation. From a first glance, the correlation is more or less 0. However, there is on this matrix one high value. If we will print out the maximum of this matrix we will get 0.9168, which is very far from zero.

Now, we are going to find the correct time. The correct time is where there is a maximum correlation and the correct key is the row number of this value. In figure 8.11 we are taking

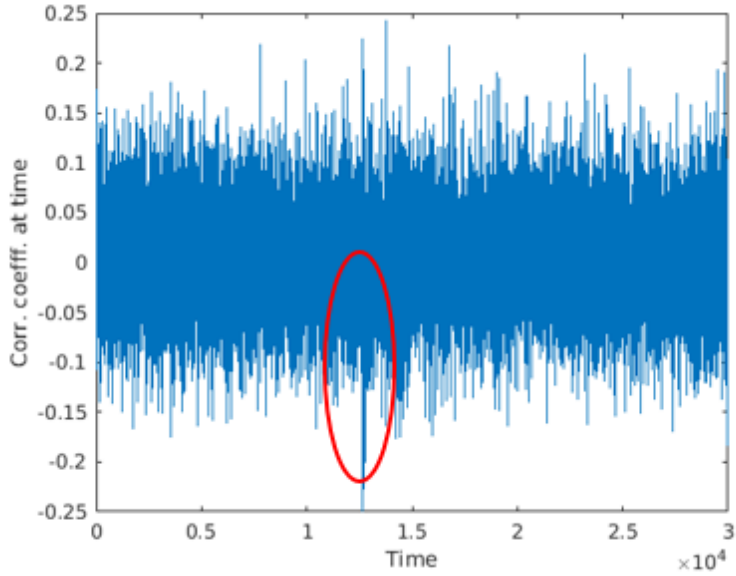


Figure 8.8: Pearson correlation graph

the classification matrix shown in 8.10 and we are looking at the correct time. It has 256 different key guesses. The X axis is the key guess and the Y axis is the correlation at the correct time. Most of the time the correlation is around 0.3 but at the correct key guess the correlation is 0.9.

In figure 8.12 we can see the correlation around the correct time. The green line is the correlation for the correct key guess, while the other lines are the correlation for the wrong key guesses in the correct time. In addition, we can see in figure 8.13 the real power consumption of the DUT vs. the hypothesis. The blue line is the power consumption of the DUT at the correct time. The Y axis is traces. The red line is the hypothesis that we tested. This two lines look correlated even to naked eye, and it is a negative correlation.

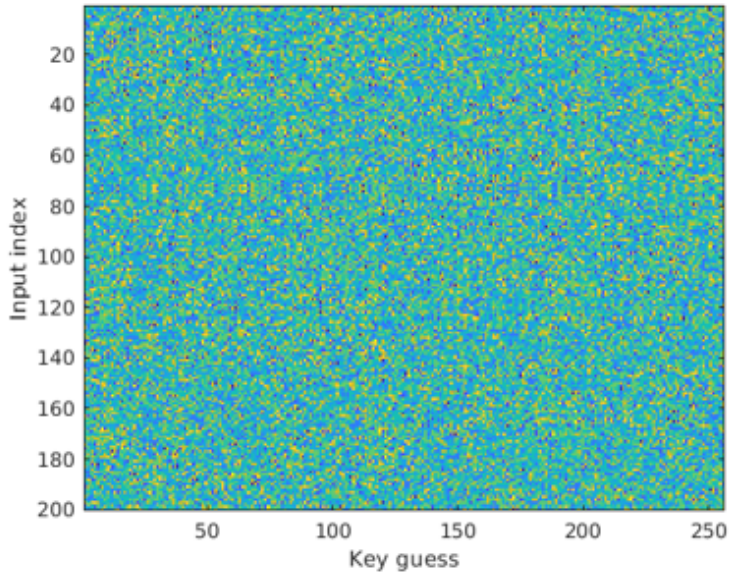


Figure 8.9: Real attack scenario

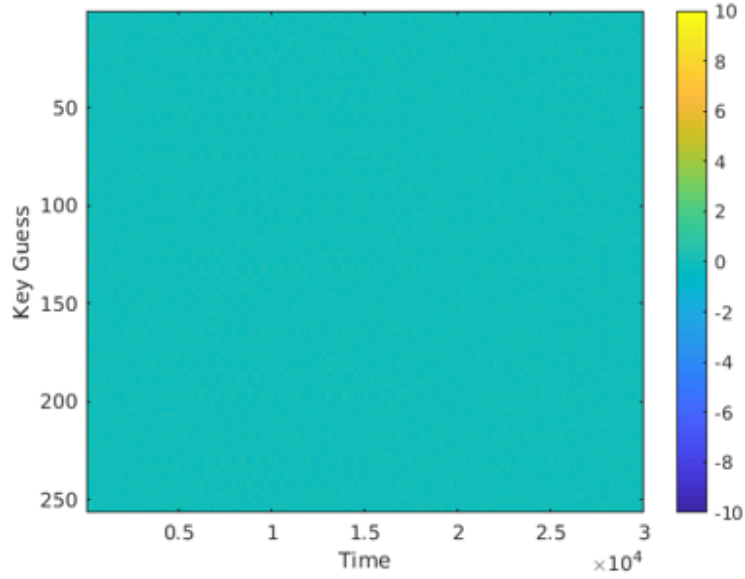


Figure 8.10: Real attack scenario classification matrix

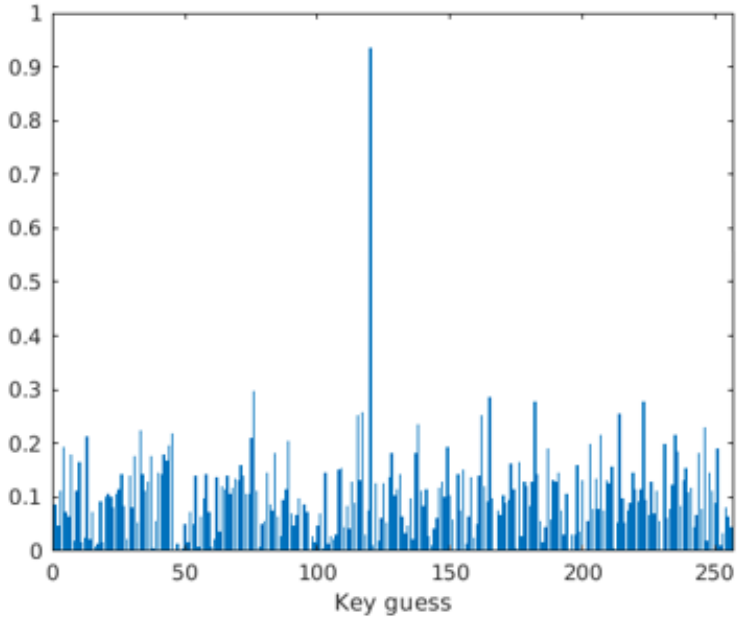


Figure 8.11: Correct time graph

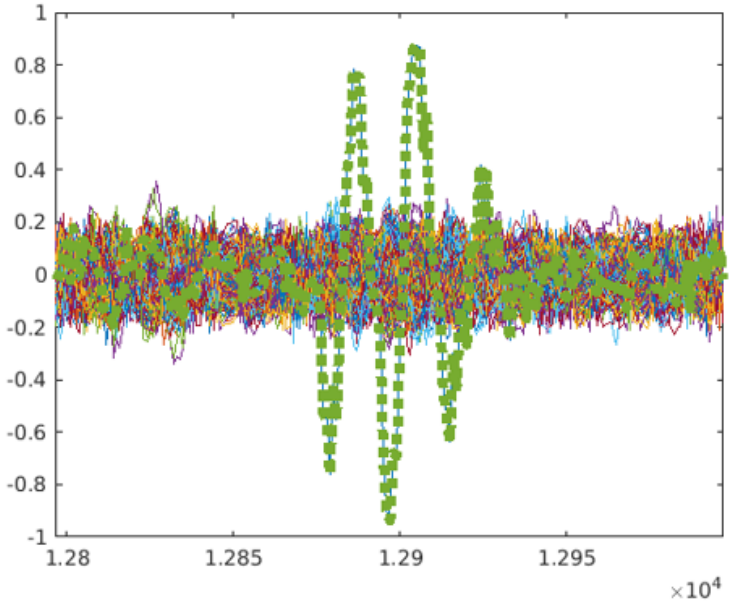


Figure 8.12: Correlation around the correct time

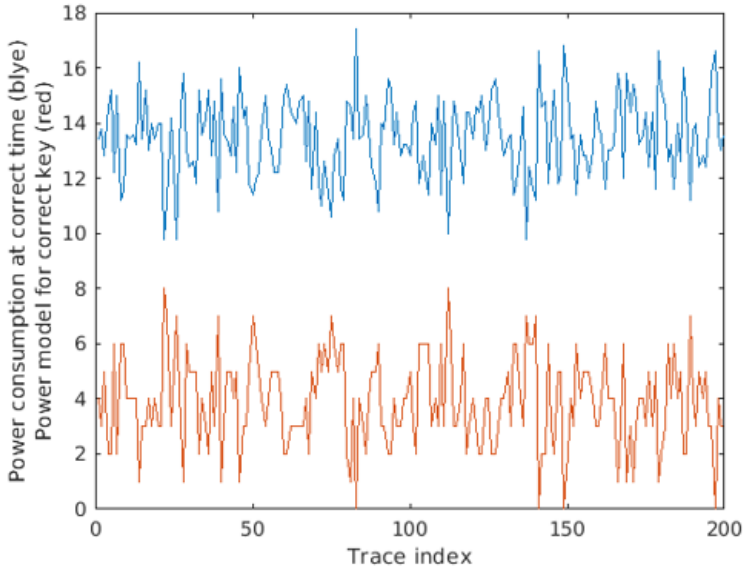


Figure 8.13: Power consumption correlation around the correct time

Chapter 9

Fault Attacks

Topics

1. Introduction to Fault Attacks
2. Fault Attack on RSA-CRT
3. DRAM and Rowhammer
4. Flip-Feng-Shui: Rowhammer attack on RSA

Introduction to Fault Attacks

Up until now in the course we learned about *passive* attacks – i.e. attacks which measure *leakage* such as timing information and power traces. The advantage of these attacks is that they allow an attacker to acquire information in the process of an ongoing computation e.g. an AES key *before* it was fully mixed with the input – this fact can help the attacker extract secrets.

In fault attacks we will become *active* in the sense that we will give the device-under-test (DUT) additional inputs such as heat or radiation.

One problem with Fault attacks: they use the strongest attack model, meaning – we assume most power on the attacker's part.

9.1 Active Attacks

Definition:

> *"A faul attack is an active attack that allows extraction of secret information from cryptographic devices by breaking those devices."*

In fault attacks we are being *active* – we give additional inputs beside the main input such as:

1. Fuzzing (garbage or bad input)
2. Radiation
3. Heat
4. Vibration
5. Power spikes etc.

This way, we receive other (usually erroneous) outputs which might give us additional information about the computation and/or the secret. This process is described in Figure 9.1.

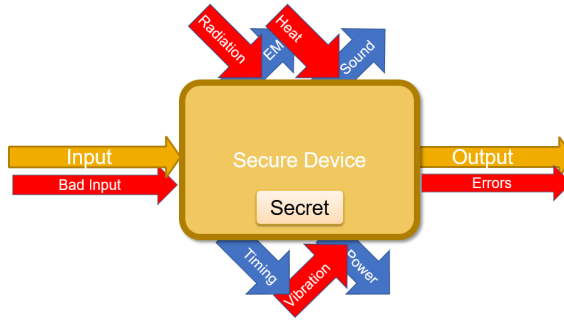


Figure 9.1: A schematic diagram of fault attacks and leakage types.

Like with passive attacks, some of those outputs can be acquired at different stages of the computation process.

Many fault attacks are inspired by studies in the field of *reliability*: a study in reliability will research a device's physical boundaries e.g. the maximum or minimum temperature under which it performs reliably. Other examples of reliability studies are aimed at improving device performance under extreme conditions such as:

- Space and X-Rays
- Dense CPU Layouts
- Data Center Recovery (ECC-RAM)

A security researcher implementing fault attacks will, on the other hand, purposefully subject the DUT to extreme conditions in order to inject errors in the device's functionality to achieve their goal. In that sense, a reliability study of a given device can lay the ground-work for the fault attacks to come.

> *"In the reliability community things happen by mistake. In the security community – things happen on purpose."*

Breaking a device-under-test

How can *breaking* a device help an attacker?

1. BORE – *"Break Once, Run Everywhere"*: Some device families share a single secret among all instances.
2. Repairable Devices: Temporary breakage is fine. Sometimes restarting the device is enough to "repair" the damage.
3. Partial breakage: Sometimes it's convenient to break *part* of a device, for example – destroy the subsystem responsible for DRM verification.

9.2 Fault Attack Taxonomy

The tree ways we can examine a Fault Attack in order to understand it are:

1. Method - *"How to inject?"*
2. Properties - *"What fault to create?"*
3. Target - *"Which part to break?"*

Fault Methods

Power Supply Attacks

What happens if the device is underpowered? As we have previously seen, power in electronic devices is used to drive the CMOS transistors. If the device is slightly underpowered it might fail to switch some of the transistors and thus produce false calculations, and with even less power it might

struggle with getting into operational state (boot loop) or even transition to an entirely faulty state. Another attack method involving the power supply is injecting power spikes (to a similar effect).

Some parts of a device are typically more sensitive to the power supply than others, and thus under- or over-powering the device will de-stabilize it and inject faults.

The obvious scenario for such an attack is when the DUT belongs to or is being controlled by the attacker – for example if they're examining their own set-top box etc. In that case, the attacker can supply the device with as much/little power as they wish.

Another example of such attack scenarios is in the field of RFID readers – the device powering an RFID chip is the reader, so a *malicious* RFID reader can over/under-power the chip to achieve various faulty results.

Clock/Timing Attacks

The clock is typically a bus shared by many of the system's components which synchronizes the propagation of calculations through the system – i.e. at the beginning all inputs are ready, and when there is a rising edge on the clock bus they start propagating throughout the various computational components. When all computations are finished they all wait for the next rising edge on the clock bus in order to proceed to the next stage. In a clock glitching attack the attacker would inject a rising edge on the clock bus at an arbitrary time. This way only *some* of the computations will have finished by that time while others are still being processed, and thus the device will be in a faulty (unstable) state.

A notable example is the attack on Mifare Classic RFID chips we talked about in the beginning of the course [50] – the RNG in the chip is only dependent on the time between powering up the RFID tag and challenging it. An RFID reader operated by the attacker can control both parameters, thus

making the generated challenges deterministic.

Temperature attacks

This attack method relies on the physical property of electrons (current). Electrons "jump", and the higher the temperature – they will jump more frequently and to longer distances. If a device gets *too hot* – enough electrons can "jump" e.g. over the insulation layer in a transistor to flip it from logical 1 to 0. This results in a fault.

Because of the ubiquity of devices failures due to temperature, nowadays temperature sensors are integrated into most devices, so when it overheats – the device will shut down.

An attacker can bypass the temperature sensor by disconnecting it. Another method would be to quickly alternate the temperature of the device from extremely high to extremely low, so that *on average* the temperature is reasonable, but it will still experience faults during the extreme phases of the cycles.

In an interesting research paper [51] a type-confusion attack on the Java virtual-machine was demonstrated: at first, the entire memory was filled with small arrays (say of size one). The Java VM is type-safe, so it is normally impossible to access one of the memory regions using a pointer to a different region. To inject a type-confusion fault the researchers used a 50W light bulb to heat the memory chip of the device enough to flip some of the bits (for illustration see Figure 9.2). As a result, a small portion of the data-structures describing the arrays in memory now held wrong values (e.g. changed their value from $size = 1$ to $size = 20$). At this point, some affected data-structure *contains* a header of a different data-structure, to which the attackers now have read and write access. Changing the header of the second data structure to an arbitrary value gave the attackers access to the entirety of the device's memory.

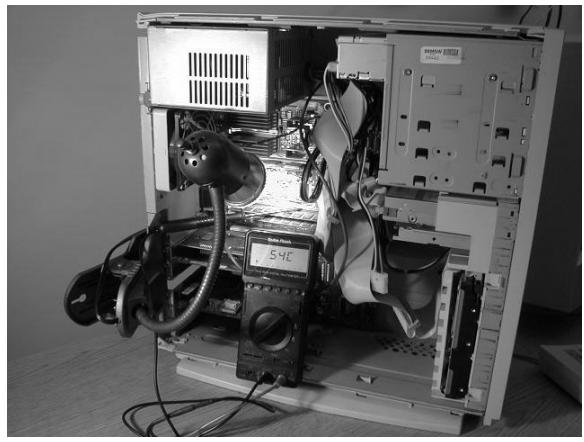


Figure 9.2: A light bulb flipping memory bits filled with safe Java structures.

Optical, Electromagnetic

When a laser hits a transistor it changes the energy level of the silicon inside, and sometimes it can change the transistor's state. Notably different wavelengths are absorbed by different materials, so in a typical silicon chip different lasers will hit different layers of the device etc. Magnetic/Electromagnetic radiation and pulses have similar effects.

The underlying principal of those attacks is that the attacker forcefully injects charge (energy) into the device. Once it's stored inside it will have to dissipate one way or another, so as a result it injects a random faulty state into the device.

Reading from RAM

All of the attacks described above require very high engagement with the DUT – in order for the attacker to control the power/clock sources, for example, they sometimes would need to drill, cut or otherwise tamper with the device. Shining a laser on a device requires at the very least having it at a visible distance.

The final attack method involves only *reading* from memory,

and thus is very practical and requires very little physical engagement. This attack method is called *Rowhammer* and is discussed later in the lecture.

Fault Properties

In this section we discuss:

1. How controllable is the fault's location/size? Precise? Loose? None?
2. How controllable is the fault timing?
3. What's the fault's duration? Transient? Permanent? Destructive?

Destructive fault attacks on cryptographic devices

What can be done with fault attacks to symmetric ciphers?

Easy example: Imagine that we have a pile of cipher devices with a 64bit key length, which work the following way: we can give the device a key and it tells us whether it's the right key.

What if we have a DESTRUCTIVE fault attack that resets the top 32bits of a device's key? We can brute force the key in $2 \cdot 2^{31}$ instead of 2^{63} (on average):

First we inject the fault into one of the devices and brute-force the lower 32 bits of the key ($O(2^{31})$), then we pick another device from the pile and brute-force only the higher 32 bits (another $O(2^{31})$).

A less trivial example: We have a public-key device which we can ZAP and one bit of the key flips to zero.

We can save all of the plaintexts-cipher pairs until we reach the one matching an all zero key – which we can pre-calculate. This gives us the Hamming weight of the key. Now we go back one plaintext-cipher pair – we know that pair's key's

Hamming weight to be exactly one – meaning we need to brute-force only N keys (N is the key bit-length). Now we have the position of a single bit of the key.

If we iterate all the way backwards to the original plaintext-cipher pair, we can acquire the key in $O(N^2)$ time!

Fault Target

What could be targeted by a fault attack?

1. Input parameters (fuzzing, clock glitching)
2. Storage (volatile/non-volatile)
    a) HDD – Destructive (persists after reset)
    b) RAM – Permanent
    c) Cache – Transient
3. Data processing: inject a fault mid-computation and the device gives a different answer.
4. Instruction Processing/Control Flow: inject a fault in the IP register and change the instruction flow.
    a) ARM32 instructions are very densely packed, thus there is a very high probability of hitting a valid instruction after flipping a single bit. `jnz` and `jmp` are only one bit apart.
    b) Change for loop condition to dump RAM contents including source code.

Two examples of Fault Attacks targeting Control Flow:

1. The CHDK hacking community, used to dump the firmwares of Canon cameras via blinking one of their LEDs [52, 53].

2. "The Unlooper": Back in the 90's pay-tv devices started cryptographically signing the content, and if the cryptographic checksum did not check out – the device would enter an endless loop. The hacking community invented "unloopers" – a gadget that would inject a power spike and fault the IP register, so that the pay-tv device would jump to some other section of the code, from which point it would function normally.

9.3 Fault attack on RSA-CRT

RSA decryption

$N = p \cdot q$ $M = C^d \pmod n = M^{ed} \pmod n = M^1 \pmod n$

RSA decryption is hard!

Let's speed it up using CRT (the Chinese Remainder Theorem): Multiplication operations are $O(|n|^2)$. If we can do operations $\pmod p$ and then $\pmod q$ instead of $\pmod n$, we will reduce computation time by half.

Explanation: The bit-lengths of p and q are each half that of n

$$|p| = |q| = \frac{1}{2}|n|$$

The computational complexity of multiplying by a number of length x is (roughly) $O(x^2)$. Thus:

$$O(|p|^2) = O(|q|^2) = \frac{1}{4}O(|n|^2) \Rightarrow (O(|p|^2)+O(|q|^2)) = \frac{1}{2}O(|n|^2)$$

So if we could multiply by p and q instead of by n, we would cut *each* multiplication operation's time complexity in half!

Chinese Remainder Theorem

The idea is that if we know both $x \pmod p$ and $x \pmod q$ then we can easily calculate $x \pmod n$.

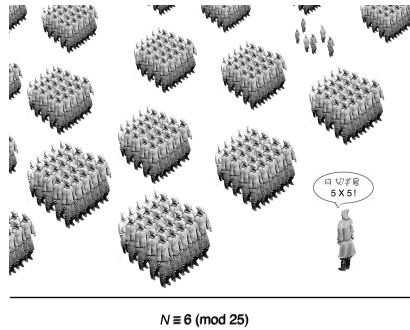


Figure 9.3: Chinese Remainder Theorem (Source: https://russinoff.com/papers/crt.html)

So, given a message M calculate M_p and M_q: $M_p = C^d \pmod{n} \pmod{p} = C^d \pmod{p}$ $M_q = C^d \pmod{n} \pmod{q} = C^d \pmod{q}$

To combine the values, we do:

$$M^* = CRT(M_p, M_q) =$$

$$M_p \cdot q \cdot (q^{-1} \pmod{p}) + M_q \cdot p \cdot (p^{-1} \pmod{q})$$

It is easily provable that $M^* \pmod{p} = M_q$ and $M^* \pmod{q} = M_p$, so by the Chinese Remainder Theorem, this value *must* be equal to M.

The Boneh, DeMillo & Lipton Fault Attack on RSA-CRT [54]

The attacker has a decryption box (known plaintext scenario) with public key n and would like to recover d (the private key). Additionally, the attacker knows that the decryption box is decrypting using CRT. Finally, let us assume that the attacker can inject a fault (any fault) in the decryption process.

The attacker first gets $M = M_p \cdot q \cdot (q^{-1} \pmod{p}) + M_q \cdot p \cdot (p^{-1} \pmod{q})$ through the regular decryption process.

Then, the attacker primes the device to re-calculate the message from the same cipher, this time injecting a *fault* during

the calculation of M_p, resulting in the device erroneously producing M'_p instead: $M'_p \neq C^d \pmod{p}$ The device will then proceed to combine M'_p with the correct result of M_q, resulting in:

$$M' = M'_p \cdot q \cdot (q^{-1} \pmod{p}) + M_q \cdot p \cdot (p^{-1} \pmod{q})$$

Now the attacker can calculate the value of $M - M'$:

$$[M_p \cdot q \cdot (q^{-1} \pmod{p}) + M_q \cdot p \cdot (p^{-1} \pmod{q})] -$$

$$[M'_p \cdot q \cdot (q^{-1} \pmod{p}) + M_q \cdot p \cdot (p^{-1} \pmod{q})] =$$

$$(M_p - M'_p) \cdot q \cdot (q^{-1} \pmod{p})$$

Finally, calculating the gcd of n and $M - M'$ yields:

$$\gcd(n, M - M') = \gcd(p \cdot q, (M_p - M'_p) \cdot q \cdot (q^{-1} \pmod{p})) = q$$

Why does this work? The greatest common divisor of n and anything can be only p, q, n or 1. On the other hand, M_p and M'_p can never be multiples of p, otherwise both would equal 0. So, by that reasoning, $\gcd(p \cdot q, (M_p - M'_p) \cdot q \cdot (q^{-1} \pmod{p}))$ must equal q, and thus we have cracked the cipher using a single fault attack.

A later paper co-written by Arjen Lenstra [55] further improved upon this attack to not require knowledge of M.

BML in practice: A paper [56] showed how ZAPPING a device with an electric spark from a lighter during computation can achieve the described effect.

9.4 Rowhammer

In the final section, we will describe the Rowhammer attack.

Rowhammer attack taxonomy

- Target: DRAM on modern computers
- Properties: Permanent, controlled location
- Method: Memory accesses

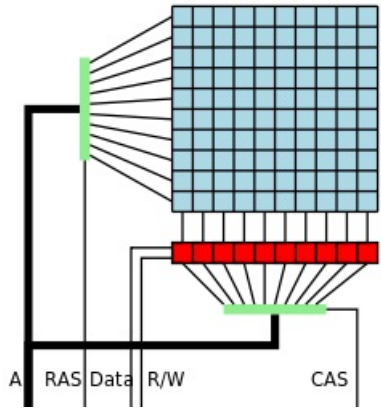


Figure 9.4: High Level Illustration of DRAM Organization (Source: Wikipedia:Row Hammer)

Double-sided Rowhammer

DRAM is the most common type of volatile memory. It is slow, dense and cheap relatively to SRAM. Every bit of RAM is stored in a single capacitor.

The attack [57] utilizes the physical structure of RAM chips in order to induce faults: Due to parasitic leakage in DRAM capacitors, if enough consecutive reads are performed on the immediately adjacent rows eventually a bit will flip in the target row.

The challenge of CPU caching

The CPU cache prevents the same memory address to be read consecutively from main memory, for performance rea-

sons. To circumvent this limitation, several techniques can be employed:

1. Intel CPUs provide non-temporal read/write opcodes – special instructions that read from memory and don't get cached.
2. The special `clflush` instruction can be used to explicitly flush the cache after each read operation (privileged operation).
3. Finally, cache-population algorithms had been extensively studied and reverse-engineered, so it is possible to arrange for *arbitrary* cache misses.

Countermeasures

Refresh-rate increase: In order to overcome parasitic leakage, DRAM chips already have a mechanism in place to read and then re-write the values stored in each row periodically. One method of overcoming Rowhammer could be to significantly increase the refresh-rate of the chip. This obviously results in both performance degradation and increased power consumption.

ECC-RAM: High-end DRAM chips (typically meant for data center environments) contain error-correction coding (ECC) logic which can typically *correct* one erroneous bit and *detect* two (at which point it will crash the program/system). Those chips are immune to the basic form of Rowhammer described above, but as discussed later, are not bullet-proof.

Rowhammer variations

Flip Feng-Shui

Page de-duplication: On modern systems, typically much memory is shared among many processes running on the system. This is even more true of virtualized environments

where the guest and the host, for example, could run the same OS. A common optimization is for the system to detect and de-duplicate pages containing the same data, thus freeing up memory.

Rowhammer + page de-duplication: In a paper [58] researchers from VUA demonstrated how they can utilize page-deduplication in a virtualized environment to weaken cryptographic keys, resulting in unauthorized access via OpenSSH, and breach of trust via forging GPG signatures. The attack relies on the fact that the the attacker can *read* a de-duplicated page as much as they want. The attacker has to wait (or arrange) for a page containing sensitive information to be de-duped, then hammer on it until a bit in the key flips, making it much easier to factor.

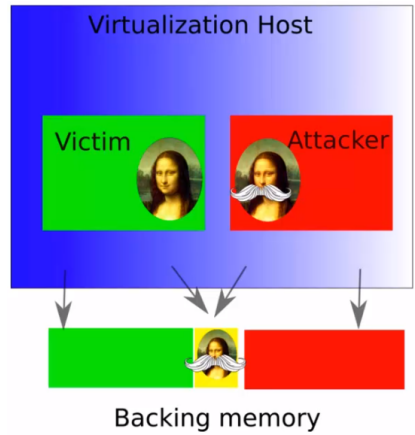


Figure 9.5: The attacker maps the same page as the victim, then utilizes rowhammer to change the victim's memory without causing page duplication.

ECCPloit

In another paper [59] researchers from VUA showed how they can use Rowhammer to quickly flip *enough* (typically three) bits on an ECC-RAM chip that error correction will not be

able to detect it, thus defeating the ECC mitigation. The attack relies on the fact that error-correction takes time to compute, and this gives the attacker a window of opportunity.

Chapter 10

Ethics and Responsible Disclosure

Layout

1. Computer Laws
2. Cyber-ethical dilemmas
3. Responsible disclosure

10.1 Computer Laws

Israel's computer laws[1] are *case-laws* – meaning there are a few written laws, and a judge would use those along with precedents and personal judgment to determine whether a person is guilty. This is opposed to *continental-law*, in which everything is written and proceedings are executed literally by the book.

[1] Published in *Sefer Hachukkim* 5755 No. 1534, 25 July 1995 p. 366 (P.L. 5754 No. 2278 p. 478).

Definitions

1. "computer material" - software or computer information
2. "computer" - A device that works with the software to perform arithmetic or logical processing of data and peripherals, with exception to auxiliary computer
3. "auxiliary computer" - A device that able only to perform arithmetic
4. "information" - or computer information, is signs,instructions,data or concepts, except for software. the information is expressed via computer language and stored in the computer or any other media or volume.
5. "output" - any information that being produced or derived via the computer
6. "computer language" - appropriate expressive form for interpretation, delivery or processing by computer
7. "software" - A group of instructions expressed in the computer language that can cause a computer to function or to perform an action, and is either embedded or marked with a device or by means of electronic, electromagnetic, electro-optics or any other means, or either united with the computer in some way or separated from it

The laws state which cyber-activities are considered illegal, as follows:

1. Unlawfully disrupting or interfering with a computer system
2. Unlawfully modify or delete any material or files from the computer system
3. Providing false information or false output

4. Unlawful access, invade or penetrate to a computer system

5. Unlawful eavesdrop, according to the Wiretap Act 1979 rule

6. As the above, but in order to commit another offence

7. Creating, distributing or offering to another person a computer virus (a virus can perform variety of malicious actions, such remove/change files, ransomware, backdoor etc.)

From this we learn that not any crime committed with the aid of a computer constitutes a cyber-crime; if a computer was used in the process of the crime, but none of the above computer laws were broken – for example while profiteering[2] off an online ticket sale – the action, while illegal, is not a cyber crime.

Unlawful Disruption or Interfering with a Computer System

Definition. Disrupting the normal operation of a computer, interfering with the use of a computer or deleting of materials saved on a computer.

Let's observe some examples:

☒ Performing a DDoS attack on a server

☒ Infecting a computer with a virus preventing access to information saved on it

☒ Recruiting a computer into a botnet, thus draining its resources

[2]The activity of taking unfair advantage of a situation to make a large profit, often by selling goods that are difficult to get at a very high price (Cambridge Business English Dictionary).

- ☐ Infecting a computer with a program that sends information from it to a remote server
- ☐ Cheating at a single player video game
- ☐ Installing an unlicensed program

All of the above actions are illegal. The first three are violations of the first section of Israel's computer laws, while the latter three are illegal for other reasons, since they do not directly prevent/interfere with the victim's system's functions.

False Information or Output

Definition. Misinformation; generating, or changing the code of a program such that it would generate, information that is forged or misleading. Let's observe some examples:

- ☒ Sending an email on behalf of someone else without their consent
- ☒ Change the code of a grade-calculating program in order to change the output grades
- ☒ Cheat in an online game to gain an unfair advantage
- ☒ Using a *key generator*: a program that forges fake software licenses
- ☐ Recruiting a computer into a botnet
- ☐ Accessing an email inbox of another user

All of the above actions are illegal, but only the first four violate the second section of Israel's computer laws.

Unlawful Access

Definition. Virtual trespassing; any action which results in the perpetrator accessing information to which they do not have access, such as images, text files, code, etc.

Let's observe some examples:

- ☒ Viewing personal information of coffeehouse patrons by using the same wi-fi
- ☒ Penetration testing (attempting to find weaknesses) on a company's server without its consent
- ☒ Using some else's password to access their inbox without their consent
- ☐ Using a program to forge credit card numbers

Forging credit card numbers, while illegal, does not fall under this section of the law; it is constitutes as false information. Note that penetration testing is a violation of this section of the law even if it is unsuccessful.

10.2 Cyber Ethics

We first present two case studies, and proceed to discuss them and cyber ethics in general.

Case Studies

Stresser. Two people from Israel ran a DDOS service, known as a Stresser, which performed DDOS attacks on websites for money. They made a large profit, but were eventually caught, and claimed they were ethical, and there are occasions where they refused to attack Israeli websites. What were their motives?

Eternal Blue. In 2013, the NSA found a vulnerability in Windows [60] and secretly built a tool called Eternal Blue to exploit it. At some point, a group of hackers known as the Shadow Brokers stole security tools from the NSA and offered to sell them on the Dark Net. When nobody paid, the Shadow Brokers leaked the tools. Microsoft released a patch

to fix the bug a short time later; the patch was prepared in advance after Microsoft was tipped off by the NSA.

In 2017, a piece of malware known as "WannaCry" first appeared; it utilized Eternal Blue to spread quickly through computers on which the security patch wasn't yet installed, and encrypt the victim's files, then asking for money to decrypt them[3].

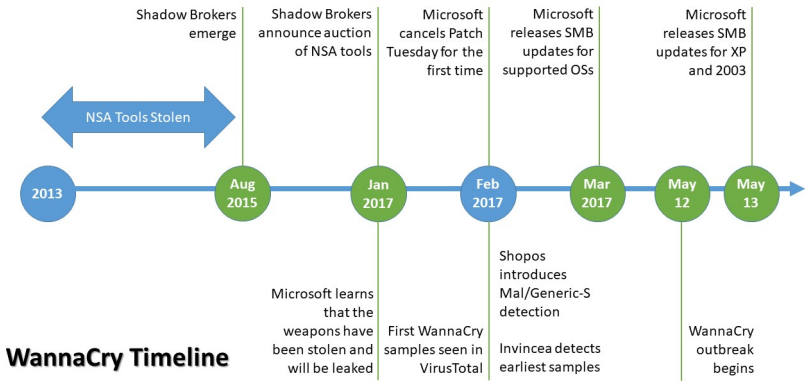


Figure 10.1: EthernalBlue-WannaCry Timeline

WannaCry was finally beaten when a malware researcher found it made requests to an unused control domain, and purchased the domain.[4] As it turned out, purchasing the domain triggered the destruction of WannaCry, whether by mistake or by design (the domain could have been designed as a kill-switch).

While WannaCry did not make a lot of money, it had a nasty side-effect on healthcare systems; medical equipment that got infected could not function normally, with dangerous, possibly lethal, repercussions.

[3]This is known as ransomware. Recall that this falls under the first section of the computer laws.

[4]Malware researchers often try to purchase control domains in order to gain information on the behavior of the malware.

Suppose a man was killed due to a failure induced by WannaCry; let John be such a casualty. There are many parties in this story, each holding part of the blame for John's death:

- WannaCry's writers, whose malware caused John's death
- Microsoft, whose software was used and trusted by the hospital
- The hospital, which relied solely on Microsoft's software without providing safety measures of their own
- The NSA, who knew about the bug well in advance, but did not perform responsible disclosure
- The Shadow Brokers, who leaked the NSA's tools to the public, which lead to the development of WannaCry

Discussion

Ethics. Ethical behavior is deciding how to act according to one's moral system. It is mostly instinctive, or composed of a personally devised code, rather than conforming to a set of rigid universal rules.

The Trolley Problem[61]. Imagine a trolley approaching a fork in the railway. On the first path lie five people, and on the second only a single person. The trolley, if its course remains unchanged, will head for the first path, resulting in the deaths of the five people there. However, you as a bystander have the option to pull a lever to change the direction of the trolley to the second path, resulting in the single death of the man there.

According to the numbers, you should pull the switch to save the five. But in that case, by sending the trolley toward the single person on the other path, you are *directly* responsible for his death! So should you avoid getting involved, thus allowing the deaths of five people instead?

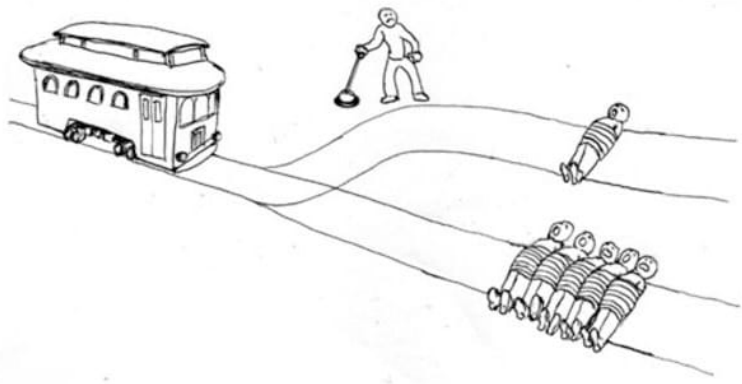


Figure 10.2: The Trolley Problem

What if one of the people is a close friend? Would that change your answer?

There is another version to this problem, in which there is only a single road with five people on it. You, along with a very large man, are standing on a bridge above the road, and you have the option of *pushing* the large man beside you off the bridge to block the approaching trolley. The stakes are the same, but in the scenario, in order to save the five you need to actually commit murder!

Our "closeness" to a situation affects how we judge it by our moral standards.

How to decide. There are two main theories by which the morality of an action is judged:

- Deontological ethics (Kant[5]) – what do my rules/morals

[5]Immanuel Kant was a German philosopher during the Enlightenment era. His contributions have had a profound impact on almost every philosophical movement that followed him

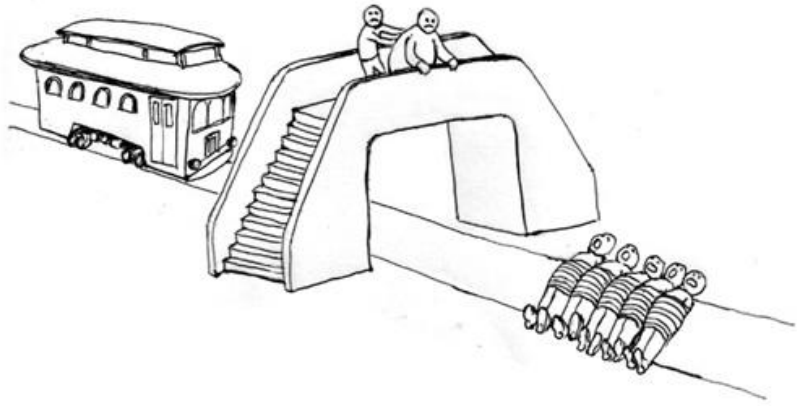


Figure 10.3: The Trolley Problem (Fat Man)

say about this action? (i.e. judge by motives)

- Utilitarian ethics (Bentham[6]) – what are the consequences of this action? (i.e. judge by consequences)

Consider the cyber setting. If we were to program ethics into an AI (say, an autonomous car), we would have to use Utilitarian Ethics – a score system by which the algorithm could judge what would be the least terrible outcome according to our moral system.

This presents complications; for example, let's consider a practical implementation of the Trolley Problem in autonomous cars: Assume a car is driving on a bridge containing five pedestrians in the middle of the road. If the car continues on its path, they will all die. However, the car can choose to veer off the bridge, saving five lives but killing the driver. By utilitarian considerations, this is the best approach – but who will purchase a car which is programmed to kill him on certain circumstances?

[6]Jeremy Bentham was an English philosopherduring the Enlightenment era and a social reformer regarded as the founder of modern utilitarianism.

Cyber crime. Our brain is "programmed" to feel distressed when confronted with an unethical scene; i.e. when we see a scene of a person committing an unethical act, we feel bad in sympathy with the victim. This is why it is easier to commit a cyber crime – we do not see the victim, and do not sympathize with them. Upholding cyber ethics would mean artificially causing users to use their judgment when their brains would not.

10.3 Responsible Disclosure

Disclosure is the act of alerting a product vendor to a vulnerability in their product so that they could patch it. Disclosure is performed by an honest party (i.e. not a hacker who wishes to exploit the vulnerability) with the intention of increasing the security of the product. There is a question of exactly when and how to perform this disclosure.

An exploit is considered a *zero day* when the vendor is unaware of the vulnerability it exploits (meaning it is known only to a few, and unknown to the general public - i.e. it has not been disclosed).

Figure 10.4 shows what happens from the moment a vulnerability is found to the moment its patch is deployed.

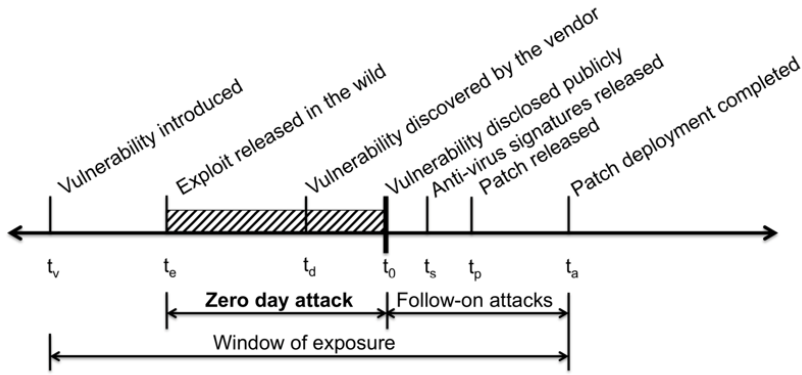


Figure 10.4: Timeline of a vulnerability

Symantec, the company founding Nortron[7], had a lot of data regarding vulnerabilities and exploits, and used it to research the matter. Figure 10.5 shows their results.

Table 1: Summary of findings.

Findings	Implications
Zero-day attacks are more frequent than previously thought: 11 out of 18 vulnerabilities identified were not known zero-day vulnerabilities.	Zero-day attacks are serious threats that may have a significant impact on the organizations affected.
Zero-day attacks last between 19 days and 30 months, with a median of 8 months and an average of approximately 10 months.	Zero-day attacks are not detected in a timely manner using current policies and technologies.
Most zero-day attacks affect few hosts, with the exception of a few high-profile attacks (e.g., Stuxnet).	Most zero-day vulnerabilities are employed in targeted attacks.
58% of the anti-virus signatures are still active at the time of writing.	Data covering 4 years is not sufficient for observing all the phases in the vulnerability lifecycle.
After zero-day vulnerabilities are disclosed, the number of malware variants exploiting them increases 183–85,000 times and the number of attacks increases 2–100,000 times.	The public disclosure of vulnerabilities is followed by an increase of up to five orders of magnitude in the volume of attacks.
Exploits for 42% of all vulnerabilities employed in host-based threats are detected in field data within 30 days after the disclosure date.	Cyber criminals watch closely the disclosure of new vulnerabilities, in order to start exploiting them.

Figure 10.5: Symantec's disclosure research findings

From Symantec's results, we conclude the following:

- Zero days exist
- Usually zero days are targeted, and not highly used so that they would remain secret.
- Disclosure is terrible for security; after a patch is released, attackers use it to find the vulnerabilities and exploit them. Attackers go through public responsible disclosure and attack even before a patch is released. Between the point where the disclosure was made publicly and the patch deployment was completed, the scale of attacks rose by 5 orders of magnitude, making public disclosure a bad idea.

Responsible disclosure. There is a government operated website called CVE – Common Vulnerabilities and Exposures. Disclosure could be performed by uploading a vulnerability to the site such that only the owner of the software can see it, and publish the details only when a patch is released (between points t_p and t_a in fig. 10.4), to avoid the

[7]Norton is an anti-virus.

window where the vulnerability is public and unpatched (between t_0 and t_p in fig. 10.4). Microsoft called this practice of disclosing privately Coordinated Vulnerability Disclosure, and claim it significantly reduces the amount of exploits on new vulnerabilities[62].

Google introduced a system[63] where the vulnerability remains private for 90 days, after which it goes public, with or without a patch. The strict deadline is meant to make sure the vulnerable vendor does not dismiss fixing the bug due to it being unknown, and thus security is improved.

Over the years using machine learning with CVEs was used for detecting and classifing vulnerabilities in code[64], as well as making automatic predictions for unseen vulnerabilities[65].

On the other hand, CVEs can be used to find unpatched exploits – for example, there was a bug in Linux which hasn't yet gone public, but someone needed to commit the changes, which contained the CVE-ID. Thus hackers could see the fix and exploit of the vulnerability before a patch was posted.

Bibliography

[1] Us navy crypto equipment - 1950's-60's. http://www.navy-radio.com/crypto.htm.

[2] Ryan Singel. Declassified nsa document reveals the secret history of tempest. https://www.wired.com/2008/04/nsa-releases-se/.

[3] memcmp(3) - linux man page. https://linux.die.net/man/3/memcmp.

[4] Flavio D Garcia, Gerhard de Koning Gans, Ruben Muijrers, Peter Van Rossum, Roel Verdult, Ronny Wichers Schreur, and Bart Jacobs. Dismantling mifare classic. In *European symposium on research in computer security*, pages 97–114. Springer, 2008.

[5] Nicolas Courtois, Karsten Nohl, and Sean O'Neil. Algebraic attacks on the crypto-1 stream cipher in mifare classic and oyster cards. *IACR Cryptology ePrint Archive*, 2008:166, 2008.

[6] Signal amplification relay attack (sara). https://hackernoon.com/signal-amplification-relay-attack-sara-609ce6c20d4f.

[7] What is an fpga? https://www.xilinx.com/products/silicon-devices/fpga/what-is-an-fpga.html.

[8] Amir Moradi, Alessandro Barenghi, Timo Kasper, and Christof Paar. On the vulnerability of fpga bitstream encryption against power analysis attacks: extracting

keys from xilinx virtex-ii fpgas. In *Proceedings of the 18th ACM conference on Computer and communications security*, pages 111–124. ACM, 2011.

[9] Confidentiality, integrity, and availability. https://developer.mozilla.org/en-US/docs/Web/Security/Information_Security_Basics/Confidentiality,_Integrity,_and_Availability.

[10] Hsiao-Ying Lin and Wen-Guey Tzeng. An efficient solution to the millionaires' problem based on homomorphic encryption. In *International Conference on Applied Cryptography and Network Security*, pages 456–466. Springer, 2005.

[11] Paul C Kocher. Timing attacks on implementations of diffie-hellman, rsa, dss, and other systems. In *Annual International Cryptology Conference*, pages 104–113. Springer, 1996.

[12] Atm card blocked after certain numbers of wrong pin.

[13] Burt Kaliski. The mathematics of the rsa public-key cryptosystem. *RSA Laboratories*, 2006.

[14] Wikipedia. Rsa (cryptosystem), 2019.

[15] Jonathan Katz, Alfred J Menezes, Paul C Van Oorschot, and Scott A Vanstone. *Handbook of applied cryptography*. CRC press, 1996.

[16] Pei Dingyi, Salomaa Arto, and Ding Cunsheng. *Chinese remainder theorem: applications in computing, coding, cryptography*. World Scientific, 1996.

[17] Henry S Warren. *Hacker's delight*. Pearson Education, 2013.

[18] Jean-Francois Dhem, Francois Koeune, Philippe-Alexandre Leroux, Patrick Mestré, Jean-Jacques

Quisquater, and Jean-Louis Willems. A practical implementation of the timing attack. In *International Conference on Smart Card Research and Advanced Applications*, pages 167–182. Springer, 1998.

[19] Wikipedia. Student's t-test, 2019.

[20] Wikipedia. William sealy gosset, 2019.

[21] Marc F Witteman, Jasper GJ van Woudenberg, and Federico Menarini. Defeating rsa multiply-always and message blinding countermeasures. In *Cryptographers' Track at the RSA Conference*, pages 77–88. Springer, 2011.

[22] K. Shirriff. Ken shirriff's blog. http://www.righto.com/2013/09/intel-x86-documentation-has-more-pages.html.

[23] Colin Percival. Cache missing for fun and profit. *BSDCan 2005*, 2005.

[24] Gabriele Paoloni. How to benchmark code execution times on intel® ia-32 and ia-64 instruction set architectures. *White Paper*, 2010.

[25] David Gullasch, Endre Bangerter, and Stephan Krenn. Cache games – bringing access-based cache attacks on aes to practice. In *Proceedings of the 2011 IEEE Symposium on Security and Privacy*, SP '11, pages 490–505, Washington, DC, USA, 2011. IEEE Computer Society.

[26] Dag Arne Osvik, Adi Shamir, and Eran Tromer. Cache attacks and countermeasures: The case of aes. In *Proceedings of the 2006 The Cryptographers' Track at the RSA Conference on Topics in Cryptology*, CT-RSA'06, pages 1–20, Berlin, Heidelberg, 2006. Springer-Verlag.

[27] Falkner Yarom. Flush+reload: a high resolution, low noise, l3 cache side-channel attack. *23rd USENIX Security Symposium*, 2014.

[28] Fangfei Liu, Yuval Yarom, Qian Ge, Gernot Heiser, and Ruby B. Lee. Last-level cache side-channel attacks are practical. In *Proceedings of the 2015 IEEE Symposium on Security and Privacy*, SP '15, pages 605–622, Washington, DC, USA, 2015. IEEE Computer Society.

[29] Mangard Gruss, Spreitzer. Cache template attacks: Automating attacks on inclusive last-level caches. *24th USENIX Security Symposium*, 2015.

[30] Clémentine Maurice, Nicolas Le Scouarnec, Christoph Neumann, Olivier Heen, and Aurélien Francillon. Reverse engineering intel last-level cache complex addressing using performance counters. In Herbert Bos, Fabian Monrose, and Gregory Blanc, editors, *Research in Attacks, Intrusions, and Defenses - 18th International Symposium, RAID 2015, Kyoto, Japan, November 2-4, 2015, Proceedings*, volume 9404 of *Lecture Notes in Computer Science*, pages 48–65. Springer, 2015.

[31] C. Maurice. Cache template attacks. https://github.com/clementine-m/cache_template_attacks.

[32] D. Gruss. Cache template attacks. https://github.com/IAIK/cache_template_attacks.

[33] Hamming weight. https://en.wikipedia.org/wiki/Hamming_weight.

[34] SideChannel sidechannelattack. https://en.wikipedia.org/wiki/Side-channel_attack.

[35] TimingAttack timingattack. https://en.wikipedia.org/wiki/Timing_attack.

[36] Ronald L Rivest, Adi Shamir, and Leonard Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Communications of the ACM*, 21(2):120–126, 1978.

[37] MontgomeryModularMultiplication montgomerymodularmultiplication. https://en.wikipedia.org/wiki/Montgomery_modular_multiplication.

[38] Paul Kocher, Joshua Jaffe, and Benjamin Jun. Differential power analysis. In *Annual International Cryptology Conference*, pages 388–397. Springer, 1999.

[39] Paul Kocher, Joshua Jaffe, Benjamin Jun, et al. Introduction to differential power analysis and related attacks, 1998.

[40] Eric Brier, Christophe Clavier, and Francis Olivier. Correlation power analysis with a leakage model. In *International workshop on cryptographic hardware and embedded systems*, pages 16–29. Springer, 2004.

[41] Jean-Sébastien Coron, Paul C Kocher, and David Naccache. Statistics and secret leakage, financial cryptography. *Lecture Notes in Computer Science*, 1962:20–24.

[42] Rita Mayer-Sommer. Smartly analyzing the simplicity and the power of simple power analysis on smartcards. In *International Workshop on Cryptographic Hardware and Embedded Systems*, pages 78–92. Springer, 2000.

[43] Elisabeth Oswald. *On side-channel attacks and the application of algorithmic countermeasures*. na, 2003.

[44] Ross Anderson, Eli Biham, and Lars Knudsen. Serpent: A proposal for the advanced encryption standard. *NIST AES Proposal*, 174:1–23, 1998.

[45] AES aes description. https://www.comparitech.com/blog/information-security/what-is-aes-encryption/.

[46] https://en.wikipedia.org/wiki/Pearson_correlation_coefficient.

[47] FaultAttack faultattack description. https://link.springer.com/referenceworkentry/10.1007%2F978-1-4419-5906-5_505.

[48] TemplateAttack templateattack description. https://wiki.newae.com/Template_Attacks.

[49] WS2Dataset ds2 dataset. https://catalog.data.gov/dataset/ws2.

[50] K. Nohl et al. Reverse-engineering a cryptographic rfid tag. In *17th USENIX Security Symposium*, volume 17. USENIX, 2008. Online; https://www.usenix.org/legacy/events/sec08/tech/full_papers/nohl/nohl.pdf.

[51] A. Govindavajhala, S. & Appel. Using memoryerrors to attack a virtual machine. Princeton University. Online; https://www.cs.princeton.edu/~appel/papers/memerr.pdf.

[52] CHDK Wiki. Obtaining a firmware dump. CHDK Wiki. Online; https://chdk.fandom.com/wiki/Obtaining_a_firmware_dump#Q._How_can_I_get_a_firmware_dump.3F.

[53] HackingHood. Canon sd300 camera analysis. HackingHood. Online; https://sites.google.com/site/hackinghood2/home.

[54] D. Boneh et al. On the importance of eliminating errors in cryptographic computations. Dept. of Computer Science, Stanford University. Online; https://link.springer.com/content/pdf/10.1007/s001450010016.pdf.

[55] M. Joye et al. Chinese remaindering based cryptosystems in the presence of faults. UCL Crypto Group, Dep. de Mathematique, Universite de Louvain. Online; http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.55.5491&rep=rep1&type=pdf.

[56] Schmidt J.M. & Hutter M. Optical and em fault-attacks on crt-based rsa: Concrete results. Institute for Applied Information Processing and Communciations, Graz University of Technology. Online; https://online.tugraz.at/tug_online/voe_main2.getvolltext?pCurrPk=32877.

[57] Y. Kim et al. Flipping bits in memory without accessing them: An experimental study of dram disturbance errors. Intel Labs, Carnegie Mellon University. Online; https://users.ece.cmu.edu/~yoonguk/papers/kim-isca14.pdf.

[58] K. Razavi et al. Flip feng shui: Hammering a needle in the software stack. Vrije Universiteit Amsterdam. Online; https://www.cs.vu.nl/~herbertb/download/papers/flip-feng-shui_bheu16.pdf.

[59] L. Cojocar et al. Exploiting correcting codes: On the effectiveness of ecc memory against rowhammer attacks. Vrije Universiteit Amsterdam. Online; https://cs.vu.nl/~lcr220/ecc/ecc-rh-paper-sp2019-cr.pdf.

[60] CVE-2017-0144. Available from MITRE, CVE-ID CVE-2017-0144. Online; https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2017-0144.

[61] Philippa Foot. The problem of abortion and the doctrine of double effect. 1967.

[62] Coordinated vulnerability disclosure. Online; https://query.prod.cms.rt.microsoft.com/cms/api/am/binary/RW5Alv.

[63] Google. How google handles security vulnerabilities. https://www.google.com/about/appsecurity/.

[64] Serguei A Mokhov. The use of machine learning with signal-and nlp processing of source code to fingerprint, detect, and classify vulnerabilities and weaknesses with marfcat. *arXiv preprint arXiv:1010.2511*, 2010.

[65] Michel Edkrantz, Staffan Truvé, and Alan Said. Predicting vulnerability exploits in the wild. In *2015 IEEE 2nd International Conference on Cyber Security and Cloud Computing*, pages 513–514. IEEE, 2015.

[66] Clemens Lode. *Philosophy for Heroes: Knowledge*. Clemens Lode Verlag e.K., 2016.

[67] Clemens Lode. *Philosophy for Heroes: Continuum*. Clemens Lode Verlag e.K., 2017.

Appendix A

Writing LaTeX

This document shows how you can get ePub-like formatting in LaTeX with the `memoir` document class. You can't yet export directly to ePub from writeLaTeX, but you can download the source and run it through a format conversion tool, such as `htlatex` to get HTML, and then go from HTML to ePub with a tool like Sigil or Calibre. See http://tex.stackexchange.com/questions/16569 for more advice. And they lived happily ever after.

A.1 Basic Formatting

Comments. If you want to just add a comment to a file without it being printed, add a `%` (percentage) sign in front of it. In the template files, you will find a number of such comments as well as deactivated commands.

Bold formatting. You can make your text bold by surrounding it with the command `\textbf{}`.

Italics formatting. You can make your text italic by surrounding it with the command `\textit{}`.

Small caps. You can change your text into small capitals by surrounding it with the command `\textsc{}`.

Text em dashes. Em dashes are used to connect two related sentences. There is no space before or after the em dash. Within the template, use the command `\textemdash{}` instead of using the dash you copied over from your text file. This will also take care of issues relating to line breaks.

Paragraphs. Paragraphs are handled automatically by leaving an empty line between each paragraph. Adding more than one empty line will not change anything—remember it is not a "what you see is what you get" editor.

Empty line. If you want to force an empty line (recommended only in special cases), you can use `~\\` (tilde followed by two backslashes).

New page. Pages are handled automatically by LaTeX. It tries to be smart in terms of positioning paragraphs and pictures. Sometimes it is necessary to add a page break, though (ideally, at the very end when polishing the final text). For that, simply add a `\newpage`.

Quotation marks. In the normal computer character set, there are more than one type of quotation marks. It is required to change all quotation marks into ``` ``\dots'' ``` (two back ticks at the beginning and two single ticks at the end) and refrain from using "..." (or "...") altogether. This is because Word's "..." uses special characters, and "..." do not mark the beginning and end of the quotation.

Horizontal line. For a horizontal line, simply write `\toprule`, `\midrule`, or `\bottomrule` from booktabs. You can also use the less recommend `\hline`.

Underlined text. It is generally not recommended to use underlined text.

URLs. For URLs you need a special monospaced font. Also, for URLs in e-books, you want to make them clickable. Both can be accomplished by putting the URL in the `\url{}` environment, for example `\url{https://www.lode.de}`.

Special characters. If you need special characters or mathematical formulas, there is a whole body of work on that subject. It is not in the scope of this book to provide you a comprehensive list.

A.2 Lists

Itemized list. To create a bullet point list (like the list in this section), use the following construct:

```
\begin{itemize}
    \item Your first item.
    \item Your second item.
    \item Your third item.
%   \item Your commented item.
\end{itemize}
```

The result will look like this:

- Your first item.
- Your second item.
- Your third item.

Numbered list. To create a numbered list, replace itemize with enumerate:

```
\begin{enumerate}
    \item Your first item.
    \item Your second item.
    \item Your third item.
\end{enumerate}
```

The result will look like this:

1. Your first item.
2. Your second item.
3. Your third item.

A.3 Verbatim text

Sometimes, you do want to simply use text in a verbatim way (including special characters and LaTeX commands). For this, simply use the `\lstlisting` environment: `\begin{lstlisting}...\end{lstlisting}`I put the itemize and enumerate listings above into a `\lstlisting` block. If I did not, LaTeX would have displayed the list as a list, instead of displaying the code.

A.4 Chapters and Sections

LaTeX uses a hierarchy of chapters, sections, and subsections. There are also sub-subsections, but for the sake of the reader, it is best to not go that deep. If you come across a situation where it looks like you need it anyway, I recommend thinking over the structure of your book rather than using sub-subsections.

In terms of their use in the code, they are all similar:

- `\chapter{Title of the Chapter}\label{cha:c1_chaptername}`
- `\section{Title of the Section}\label{sec:c1_sectionname}`
- `\subsection{Title of the Subsection}\label{subsec:c1_subsectionname}`
- `\paragraph{Title of the Paragraph}\label{par:c1_paragraph}`

When using these commands, obviously replace the title, but also the label. For the label, I recommend to have it start with either "cha:", "sec:", "subsec:", etc. to specify what kind of label it is, followed with c and the current chapter number, an underscore, and the chapter, section, or subsection in one word and lowercase. These labels can then be used for references like we used previously for the images. For example, if you have defined a section `\section{Chapters and Sections}\label{sec:c1_chaptersandsections}`, you could write "We will discuss chapters and sections in section `\cref{sec:c1_chaptersandsections}` " which results in the document in "We will discuss chapters and sections in appendix A.4".

A.5 Tables

In LaTeX, tables are like images and put into the figure environment. As such, they have a caption, label, and a positioning like we discussed above with the images. Drawing a table requires a bit of coding:

```
\begin{table}[!ht]
    \centering
    \begin{tabular}{p{2.5cm}|p{3.5cm}|p{3.5cm}}
    \hline
    & \textbf{Word} & \textbf{\LaTeX{}} \\
    \hline

    Editor & ``what you see is what you get'' & source file is compiled \\
    \hline

    Compatibility & dependent on editor & independent of editor \\
    \hline

```

```
14      Graphics & simple inbuilt editor & ↩
           powerful but complex editor \\
15      \hline
16
17      Typography & optimized for speed & ↩
           optimized for quality \\
18      \hline
19
20      Style & inbuilt style & separate ↩
           style document \\
21      \hline
22
23      Multi-platform & only via export & ↩
           possible with scripting \\
24      \hline
25
26      Refresh & some elements need, ↩
           manual refresh & everything is ↩
           refreshed with each compile \\
27      \hline
28
29      Formulas & basic support needs ↩
           external tools & complete ↩
           support \\
30      \hline
31
32      \end{tabular}
33      \caption{Comparison of Word and \↩
           LaTeX{}} \label{tab:c1_↩
           comparisonwordlatex}
34  \end{table}
```

This table from the beginning of the book has the familiar figure, label, caption, and centering commands. The actual table is configured with the `\tabular{}` environment. Following the tabular command, you configure the columns in curly braces. Each column is separated with a vertical line and the `p{...}` specifies the width of the column. With `{p{2.5cm↩ }|p{3.5cm}|p{3.5cm}}`, you would have three columns with 2.5cm width for the first column and 3.5cm width for the two

others. Alternatively, you can use `c` instead of `p` and leave out the curly braces with the width. Then, LaTeX simply calculates the required widths automatically. Then, for each line of the table, simply write: `content of the first cell & ↩ content of the second cell & content of the third cell↩ \\\midrule`.

	Word	**LaTeX**
Editor	"what you see is what you get"	source file is compiled
Compatibility	dependent on editor	independent of editor
Graphics	simple inbuilt editor	powerful but complex editor
Typography	optimized for speed	optimized for quality
Style	inbuilt style	separate style document
Multi-platform	only via export	possible with scripting
Refresh	some elements need, manual refresh	everything is refreshed with each compile
Formulas	basic support needs external tools	complete support

Table A.1: Comparison of Word and LaTeX

A.6 Footnotes

Finally, for footnotes, there is the command `\footnote↩ {}`. You can place it anywhere you like, LaTeX will then automatically add the number of the footnote at that place,

and put the footnote text into the footer area. It looks like this.[1] The challenge here relates to grammar: footnotes start with capital letters, parentheses with lower case, and the footnote comes after the period, the parentheses have to start before the period.

A.7 Inserting Images

As in Word, in LaTeX, images are separate from the text. Images are usually packaged together with a caption and a label to reference it from the text. These three entities are packaged together into a figure. The figure itself configures the size of the image as well as where it should be put. Let us look at a code sample:

```
1 \begin{figure}[!ht]
2     \centering
3     \includegraphics{images/ebookLatex_↩
          Cover.jpg}
4     \caption{The cover of this book.} \↩
          label{fig:c1_cover}
5 \end{figure}
```

Let us go through this line by line. At the core is the image, included with `\includegraphics{path to file}`. It inserts the image specified by the "path to file." With the `\adjustbox{}` command, we can adjust the image size according to the page width (`\columnwidth`) and page height (`\textheight`).

Below there is the caption and the label. LaTeX automatically numbers each figure, so in the text, we can later refer to it with `\ref{c1_cover:fig}` which prints out the number of the figure. Finally, all these commands are centered with the `\centering` command and surrounded with the figure environment. The `[ht]!` instructs LaTeX to try to place the image exactly where it is in the LaTeX code.

[1] This is a footnote.

Figure A.1: The cover of this book.

In Figure A.1, you can see the result of the command. Instead of graphics, you can also include other TEX files that contain graphics (or commands to draw graphics, see Appendix A.8).

A.8 TikZ Graphics

For graphics, you can use the inbuilt TikZ graphics generator. Due to its flexibility, I even recommend images you already have for a number of reasons:

- TikZ graphics can very easily changed (especially for for example translations or making corrections).

- TikZ graphics are small and flexible. They can be easily scaled to any size and are directly integrated into your project (no time-consuming editing in an external graphics program necessary).
- TikZ graphics look better. As vector graphics are sent directly to the printer, we need not to worry about readability.

If you want to create a TikZ graphic, simply create a new TEX file in the *tex-images* folder and include it with `\input{...}` `\includegraphics{}`) where you want to.

Then, do a "recompile from scratch" by clicking on the top right corner of the preview window (showing Warning or Error) to regenerate the TikZ file. If "up-to-date and saved" is shown, delete the *tikz-cache* directory and recreate it.

For the format of the file itself, it is a series of commands surrounded by the `\begin{tikzpicture}...\end{tikzpicture}`Discussing all the commands is beyond the scope of this book, so I recommend three options:

- Check out the PGF manual at https://www.ctan.org/pkg/pgf. It is more than 1100 pages full with documentation of each command and corresponding examples.
- Check out the few example TikZ pictures from my two books [66, 67] in the *tex-images* directory.

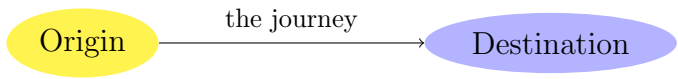


Figure A.2: TikZ drawings will be output as SVG, which should be rendered by most modern browsers.

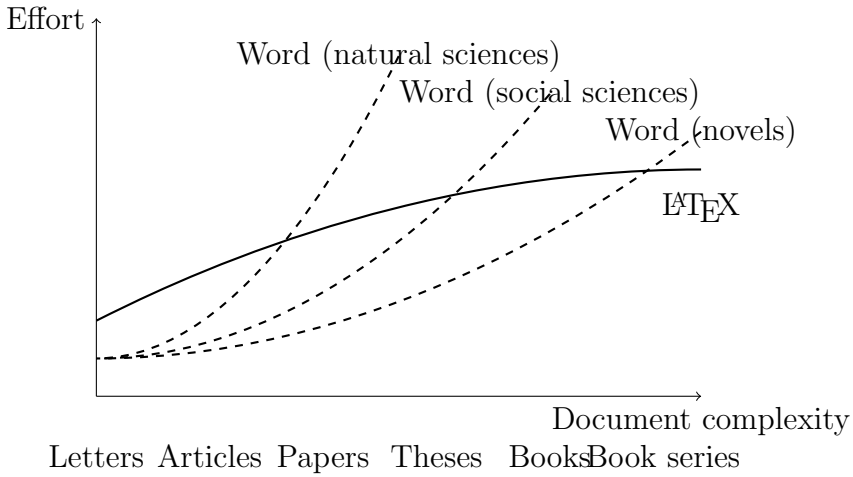


Figure A.3: Comparing complexity of *Word* and LaTeX depending on the application.

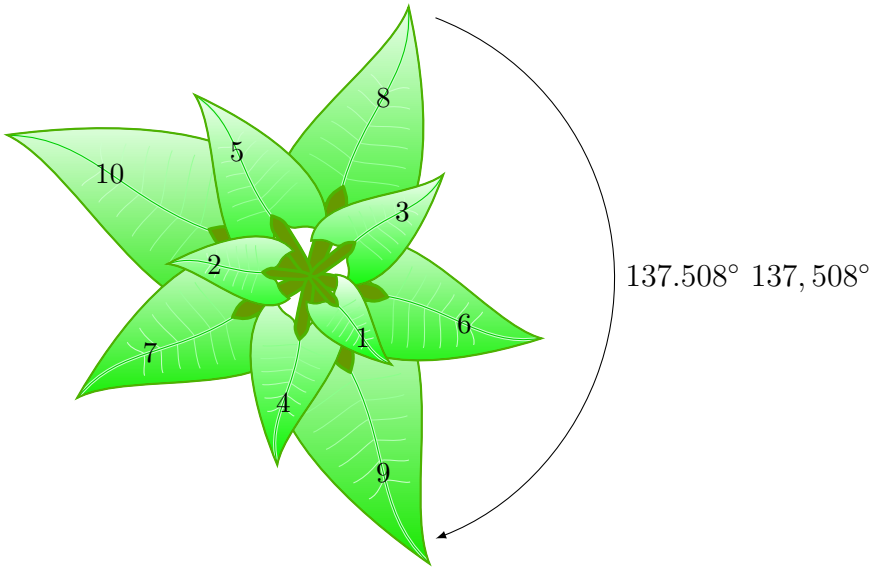


Figure A.4: Example of a drawing made in TikZ.

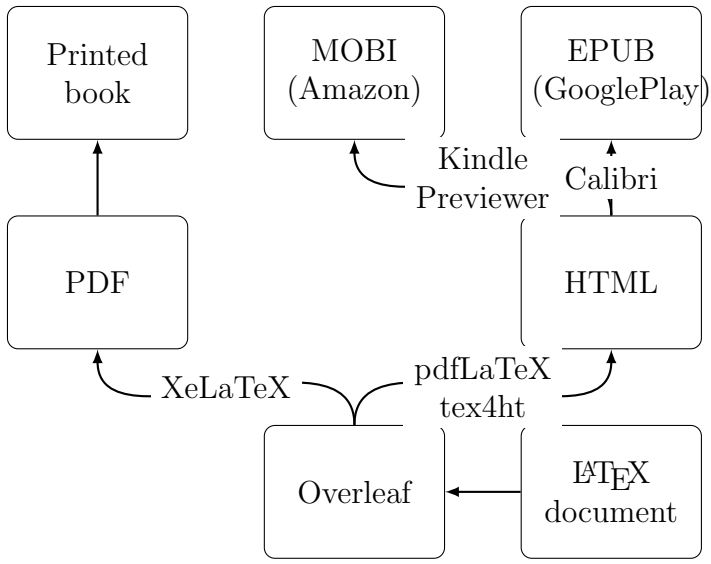


Figure A.5: Example 2 of a drawing made in TikZ.