Universidade Estadual de Campinas
Instituto de Computação

Rafael Junio da Cruz

Rowhammer-based fault injection and its effects on aged memories

Ataques de injeção baseados em Rowhammer e seus efeitos em memórias envelhecidas

CAMPINAS
2023

Rafael Junio da Cruz

Rowhammer-based fault injection and its effects on aged memories

Ataques de injeção baseados em Rowhammer e seus efeitos em memórias envelhecidas

Dissertação apresentada ao Instituto de Computação da Universidade Estadual de Campinas como parte dos requisitos para a obtenção do título de Mestre em Ciência da Computação.

Dissertation presented to the Institute of Computing of the University of Campinas in partial fulfillment of the requirements for the degree of Master in Computer Science.

Supervisor/Orientador: Prof. Dr. Diego de Freitas Aranha
Co-supervisor/Coorientador: Prof. Dr. Rodolfo Jardim de Azevedo

Este exemplar corresponde à versão final da Dissertação defendida por Rafael Junio da Cruz e orientada pelo Prof. Dr. Diego de Freitas Aranha.

CAMPINAS
2023

Ficha catalográfica
Universidade Estadual de Campinas
Biblioteca do Instituto de Matemática, Estatística e Computação Científica
Ana Regina Machado - CRB 8/5467

C889r

Cruz, Rafael Junio da, 1990-
Rowhammer-based fault injection and its effects on aged memories / Rafael Junio da Cruz. – Campinas, SP : [s.n.], 2023.

Orientador: Diego de Freitas Aranha.
Coorientador: Rodolfo Jardim de Azevedo.
Dissertação (mestrado) – Universidade Estadual de Campinas, Instituto de Computação.

1. Criptografia. 2. Criptografia de chaves públicas. 3. Curvas elípticas. 4. Bitcoin. I. Aranha, Diego de Freitas, 1982-. II. Azevedo, Rodolfo Jardim de, 1974-. III. Universidade Estadual de Campinas. Instituto de Computação. IV. Título.

Informações Complementares

Título em outro idioma: Ataques de injeção baseados em Rowhammer e seus efeitos em memórias envelhecidas
Palavras-chave em inglês:
Cryptography
Public key cryptography
Elliptic curves
Bitcoin
Área de concentração: Ciência da Computação
Titulação: Mestre em Ciência da Computação
Banca examinadora:
Diego de Freitas Aranha [Orientador]
Bruno de Carvalho Albertini
Julio César López Hernández
Data de defesa: 16-10-2023
Programa de Pós-Graduação: Ciência da Computação

Identificação e informações acadêmicas do(a) aluno(a)
- ORCID do autor: https://orcid.org/0009-0002-8789-7174
- Currículo Lattes do autor: http://lattes.cnpq.br/6803837483695062

Universidade Estadual de Campinas
Instituto de Computação

Rafael Junio da Cruz

Rowhammer-based fault injection and its effects on aged memories

Ataques de injeção baseados em Rowhammer e seus efeitos em memórias envelhecidas

Banca Examinadora:

- Prof. Dr. Diego de Freitas Aranha (Supervisor)
  Institute of Computing of the University of Campinas
- Prof. Dr. Julio César López Hernández
  Institute of Computing of the University of Campinas
- Prof. Dr. Bruno de Carvalho Albertini
  Polytechnic School of the University of Sao Paulo

A ata da defesa, assinada pelos membros da Comissão Examinadora, consta no SIGA/Sistema de Fluxo de Dissertação/Tese e na Secretaria do Programa da Unidade.

Campinas, 16 de outubro de 2023

Dedicatória

Dedico este trabalho a todas as pessoas que me ensinaram durante minha vida. Meu vô Valdecir que foi o pai que nunca tive, minha vó Maria que ensinou como ser bom, minha mãe Cidinha que ensinou resiliência. Professores que auxiliaram no meu crescimento intelectual durante toda minha vida acadêmica, como o Prof. Pedro Rezende que ensinou ser firme nas afirmações através de referências ou provas matemáticas, como Prof. Rodolfo Azevedo que ensinou a responder perguntas com outras perguntas e que a maioria das vezes a repostas a uma pergunta é depende, como Prof. Diego Aranha que foi meu grande mentor. Durante minha vida profissional com ensinamentos de gerentes como Nelson Uto que tornou um professional e um ser humano melhor.

There can be no triumph without loss.
No victory without suffering.
No freedom without sacrifice.
(J.R.R. Tolkien)

Acknowledgements

Thank you, Intel®, for financing and investing in almost everything of my master's research.

This study was financed in part by the *Coordenação de Aperfeiçoamento de Pessoal de Nível Superior* - Brasil (CAPES) - Finance Code 001.

Thank you, Dr. Lois Orosa, Prof. Dr. Lucas Wanner, and Prof. Dr. Newton Frateschi for supporting with knowledge in the DRAM aging research.

Thank you, Samsung Electronics®, for give me time to write this work.

Resumo

Bitcoin é a criptomoeda mais famosa mundialmente devido ao seu alto valor especulativo e uso descentralizado, dispensando a existência de agentes centralizados como, por exemplo, instituições financeiras. O protocolo da Bitcoin adota a curva elíptica `secp256k1`, haja vista o seu alto desempenho. Entretanto, a curva `secp256k1` é reconhecida como fraca por não conseguir prover computações simples, eficientes e seguras ao mesmo tempo, além de conter curvas irmãs, *twists*, consideradas inseguras, podendo tornar o protocolo suscetível a ataques de curva inválida. Apesar de difícil execução, ataques de injeção de falhas são muito poderosos pois, em sua maioria, podem explorar aspectos difíceis de serem controlados remotamente como, por exemplo, temperatura, voltagem e frequência. Dentre os ataques de injeção de falhas está o *Rowhammer*, um ataque via software que afeta diretamente a memória DRAM, invertendo bits devido a uma grande quantidade de leituras consecutivas. Neste trabalho, o Rowhammer foi utilizado para injetar falhas na implementação da curva adotada pela Bitcoin, visando a subversão do protocolo e a extração da chave privada. Além disso, observou-se que o ataque proposto é capaz de afetar outras curvas elípticas e demais protocolos que utilizam o Algoritmo de Assinatura Digital de Curvas Elípticas (ECDSA). Por fim, conjectura-se que a idade ou o tempo de uso possa induzir problemas em memórias DRAM que permitam ataques de Rowhammer. Nessa direção, iniciou-se um estudo de envelhecimento artificial de memórias DRAM através do aquecimento usando resistências e alterações de voltagem através da placa-mãe. As resistências possibilitaram experimentos com temperaturas acima de 200°C. Como resultado, percebeu-se a existência de uma faixa de operação segura para a memória DRAM. Falhas de leitura ou escrita puderam ser observadas a partir de uma determinada temperatura e voltagem. Além disso, encontrou-se um limiar máximo de temperatura e voltagem, a partir do qual a memória tornava-se inoperante.

Abstract

Bitcoin is the most famous worldwide cryptocurrency due to its high speculative value and decentralized use, avoiding the need for centralized agents as financial institutions. The Bitcoin protocol adopts the `secp256k1` elliptic curve because of its high performance. However, `secp256k1` is known to be a weak curve due to not provide a simple, efficient and secure computing at same time, in addition to containing unsafe twists, making the protocol susceptible to invalid curve attacks. Despite their complex execution, fault-injection attacks are powerful as most exploit aspects hard of being controlled remotely, such as temperature, voltage, frequency, and others. Among these attacks, *Rowhammer* is a software attack that directly affects the DRAM memory by inverting bits via many subsequent reads. In this work, we used Rowhammer to inject flaws in the implementation of curve within Bitcoin, aiming to subvert the protocol and extract the private key. However, we observed that the proposed attack could also affect other curves and protocols that use the Elliptic Curve Digital Signature Algorithm (ECDSA). Finally, we conjecture that age or time of use may introduce vulnerabilities in DRAM memories that allow Rowhammer attacks. In this direction, we started exploring the artificial aging of DRAM memories by heating, using resistors, and regulating the voltage in the motherboard. The resistors allowed experiments with temperatures above 200°C. As a result, we obtained a secure operating threshold for DRAM memories. Thus, read or write failures were observed above a specific temperature and voltage. Moreover, we found a maximum threshold for temperature and voltage, above which the memory became inoperative.

List of Figures

1.1 Security pyramid model of fault injection 18

2.1 Symmetric system 22
2.2 Asymmetric system 23
2.3 Elliptic curves example 25
2.4 Elliptic curve operations in $\mathbb{R}$ 26
2.5 Elliptic curve operations in $\mathbb{F}_{43}$ 27
2.6 Elliptic curves' composition of subgroups 29
2.7 NATO soldiers 32

3.1 Memory capacity growth through the years 38
3.2 Memory cells 40
3.3 Scaling of DRAM capacitors 41
3.4 Memory subsystem 41
3.5 High level overview of a memory bank 43
3.6 Memory DIMM 44
3.7 Memory controller write and precharge commands 48
3.8 Memory controller during a read command 48
3.9 Memory mapping from physical address to memory location 49

4.1 Rowhammer types 55
4.2 Step-by-step Rowhammer attack 56
4.3 Mapping physical memory using Prefetch Attack 57
4.4 Kernel Page-Table Isolation 58
4.5 Page deduplication 59

5.1 ECDSA support modules 64
5.2 x86-64 mov operation 70

6.1 Comparison between the Intel Kaby Lake i5-7400 and Skylake i7-6700k . . 74
6.2 DRAM aging setup 76
6.3 Thermostat diagram 76
6.4 DRAM operation: temperature and voltage relation 78

List of Tables

2.1 Uniqueness of elements modulo 15 32

3.1 Memory controller commands 45

5.1 `secp256k1` twists 66
5.2 x86-64 calling conventions 70

6.1 Stable bits under different operating conditions 73

List of Listings

1 Double-sided Rowhammer 52
2 Single-sided Rowhammer 54
3 One-location Rowhammer 54

List of Algorithms

2.1 ElGamal public and private key pair generation 30
2.2 ElGamal encryption algorithm 30
2.3 ElGamal decryption algorithm 30
2.4 RSA decryption using CRT 36
5.1 ECDSA key recovery attack 69
A.1 ECDSA generation 90
A.2 ECDSA verification 91

List of Examples and Theorems

2.1 Example (Prime field $\mathbb{F}_{11}$) 24
2.2 Example (Extension field $\mathbb{F}_{41^4}$) 24
2.1 Theorem ([94, Theorem 2.19] Short Weierstrass Isomorphism) 28
2.3 Example (Discrete logarithm group modulo 13) 31
2.2 Theorem ([94, Theorem 2.19] Chinese Remainder Theorem (CRT)) 31
2.4 Example (Uniqueness of all elements mod 15) 32
2.5 Example (NATO against Russia) 32

3.1 Example (Finding the physical address `0b000100001`) 40
3.2 Example (16GB DDR4 laptop) 44

Contents

1 Introduction **17**
- 1.1 Objectives . . . 18
- 1.2 Contributions . . . 19
- 1.3 Organization . . . 19

2 Preliminaries **21**
- 2.1 Basic Concepts of Cryptography . . . 21
  - 2.1.1 Symmetric System . . . 22
  - 2.1.2 Asymmetric System . . . 23
- 2.2 Elliptic-Curve Cryptography . . . 23
  - 2.2.1 Fields . . . 23
  - 2.2.2 Elliptic Curves . . . 25
  - 2.2.3 Short Weierstrass . . . 27
  - 2.2.4 Elliptic Curves for Cryptography . . . 28
  - 2.2.5 Discrete Logarithm Problem . . . 31
  - 2.2.6 Chinese Remainder Theorem . . . 31
- 2.3 Side-channel Attacks . . . 33
  - 2.3.1 Timing Attacks . . . 33
  - 2.3.2 Cache Attacks . . . 33
  - 2.3.3 Speculative Attacks . . . 34
  - 2.3.4 Simple Power Analysis . . . 34
  - 2.3.5 Differential Power Analysis . . . 34
  - 2.3.6 Differential Fault Analysis . . . 34
  - 2.3.7 Acoustic Cryptanalysis . . . 34
  - 2.3.8 Remaining Data . . . 35
- 2.4 Fault Injection Attacks . . . 35

3 DRAM Memory **38**
- 3.1 DRAM Architecture . . . 39
  - 3.1.1 DRAM Cells . . . 39
  - 3.1.2 DRAM Construction . . . 40
  - 3.1.3 DRAM Subsystem . . . 41
- 3.2 Memory Controller . . . 44
  - 3.2.1 Memory Controller Commands . . . 45
  - 3.2.2 Memory Timings Explained . . . 47
  - 3.2.3 Address Mapping . . . 48
  - 3.2.4 The Error-Correcting Code Memory . . . 49
  - 3.2.5 The Target Row Refresh Solution . . . 49

4 Rowhammer **50**
4.1 Types of Rowhammer . . . 52
4.1.1 Double-sided Rowhammer . . . 52
4.1.2 Single-sided Rowhammer . . . 53
4.1.3 One-location Rowhammer . . . 53
4.1.4 Many-sided Rowhammer . . . 54
4.2 Rowhammer Attack: Step-by-step . . . 55
4.2.1 Finding a Vulnerable Region . . . 56
4.2.2 Putting a Target on the Vulnerable Region . . . 58
4.2.3 Flipping Bits From the Target . . . 60
4.3 Countermeasures . . . 60
4.3.1 Software Countermeasures . . . 60
4.3.2 Hardware Countermeasures . . . 61

5 ECDSA, secp256k1, and Bitcoin **63**
5.1 ECDSA . . . 64
5.1.1 Invalid Curve Attacks . . . 65
5.1.2 The Elliptic Curve secp256k1 . . . 65
5.2 Bitcoin Core Wallet . . . 66
5.2.1 Countermeasures Against Fault Injections . . . 66
5.2.2 A Fault Injection in secp256k1 . . . 68
5.2.3 A Fault Injection in ECDSA . . . 70

6 DRAM Aging **72**
6.1 DRAM Aging Evaluation . . . 73
6.2 Methodology Setup . . . 74
6.2.1 Experimental Setup . . . 74
6.2.2 Heating Setup . . . 75
6.3 Preliminary Results . . . 77

7 Conclusions **79**
7.1 Future Work . . . 79

Bibliography **81**

A ECDSA Algorithms **90**

Chapter 1

Introduction

In cryptography, a side-channel attack is performed on a collection of information gathered during the execution of a cryptographic implementation. This information is obtained differently from a brute-force attack – that exhaust all possible inputs – or some inherent theoretical weakness of the algorithm, as in the case of the Data Encryption Standard (DES), which cryptanalysis reduces a 2^{56}-bit brute-force attack to 2^{41} [48]. Side-channel attacks can be performed on data obtained from running time, energy consumption, electromagnetic emanation, sound waves, and others. Paul Kocher was a pioneer in this field, showing how to perform side-channel attacks using the time channel on implementations of public-key cryptographic systems [58].

A class of side-channel attacks is Differential Fault Analysis (DFA). The principle is to induce failures during the execution of cryptographic implementations to partially or entirely reveal an internal state and then derive the cryptographic key or plaintext. DFA attacks are carried out by first injecting flaws in hardware structures, including main memory, CPU registers, logic and arithmetic units, transistors, among others. Then, the attacker observes the effect of the induced failure on higher layers, such as arithmetic, cryptographic primitives, and protocols. This process of injecting and observing is exemplified by The Fault Attack Jungle [92], which is given in Figure 1.1. This fault injection model is a pyramid, showing where faults are injected and where they manifest.

Faults can also occur in natural ways. For example, cosmic rays can induce bit flips in electronic equipment's [99]. In the 50s, anomalies were recorded by monitoring electronic equipment thanks to nuclear bomb tests [99]. In the 60s, minor flaws appeared in electronics sent into space by the space industry [99]. In the 70s, IBM started a study on how α particles interfered in semiconductors [99]. These examples show that the electronics industry has been concerned with the accidental inversion of bits for over half a century. However, the injection of intentional failures depends on the level of control over the software or hardware by the malicious agent. Projects admit a fault tolerance probability rate, which is usually estimated for natural causes. In the case of purposeful failures, this probability can be as high as the failure injection power.

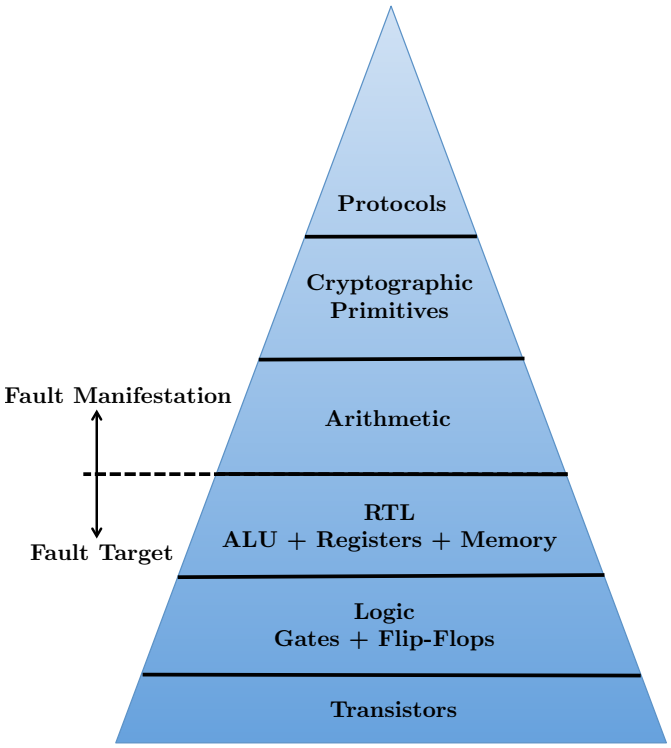


Figure 1.1: Security pyramid model of fault injection adapted from [92]. The base represents hardware at its lowest level, and the top represents software at its higher abstraction.

1.1 Objectives

The first step to being an excellent defender is to know how to perform an attack. During the process, the attacker needs to know how the system is defended, which may require much creativity. One way of attacking is to explore side channels, which allows observing design flaws and adding barriers to system protection. The study of fault-injection side-channel attacks is increasing in academy since the technologies using them are becoming accessible. For example, Karim M. et al. showed a low cost setup for Electromagnetic Fault-Injection (EMFI) attacks costing around $200 [1].

Some software fault-injection attacks are well known, one of which uses excessive writing on SSDs to burn cells and perform a Denial-of-Service attack [31]. Another attack could be the distribution of malicious binaries that perform flawed computations, such as commercial cryptographic binaries modified by the National Security Agency (NSA) [5]. The most relevant software fault-injection attack for this work is *Rowhammer* [54]. The Rowhammer attack exploits the memory design, which contains several banks with a two-dimensional array made of rows and columns. The attack tries to focus on keeping a set of row readings in the same memory bank using a set of physical memory address. Given a consecutive series of readings ("hammering") in some memory rows, it ends up flipping bits in a predefined memory cell. The initial research of Rowhammer in Dynamic Random-Access Memory (DRAM) memories with potential for building attacks was published by Kim et al. in 2014 [54].

In late 2015, the Joint Electron Device Engineering Council (JEDEC) [47] standards

association proposed the LPDDR4 memory model that includes support for hardware that prevents the effect of Rowhammer without loss of performance and increase of power consumption. However, these measures are not enough to mitigate Rowhammer forever. Even with protections like Error-Correcting Code (ECC), Cojocar et al. showed in 2019 that Rowhammer can circumvent these countermeasures and affect this category of memories [19]. In 2020, two more attacks involving LPDDR4 models were conceived. One was used to read memories after some bit flips, also bypassing ECC protection [59]. The other can bypass the Target Row Refresh (TRR) protection, known as the ultimate defense against Rowhammer [29].

These new attacks guarantee long research on how to design new memory models and how to build countermeasures against Rowhammer in legacy memory systems. The attack can also be used on personal computers and Android smartphones to gain root privilege [89, 84]. From a cryptographic point of view, the attack has already been used to recover RSA private keys by injecting flaws and combining cache attacks [10]. It is important to remember that even if the hardware industry mitigates the attack at some point, several vulnerable memory chips – the problem arose on DDR3 memories and part of DDR4 – were sold and are still in use. In this context, this work aims to perform fault-injection attacks in a software implementation of the elliptic curve `secp256k1` used by Bitcoin. Also, this work seeks to provide a brief study on how memory aging can lead to attacks similar to Rowhammer.

1.2 Contributions

This work claims four contributions. First, we survey the DRAM memory operation, including its architecture with design and construction and the memory controller responsible for managing all operations over DRAM. Secondly, we survey the Rowhammer attack, detailing the attack types, how to execute an attack step-by-step, and some countermeasures. The third contribution focuses on fault injection in Bitcoin-core wallet using Rowhammer, in the Elliptic Curve Digital Signature Algorithm (ECDSA) protocol, and in the implementation of the elliptic curve `secp256k1`. Finally, the fourth contribution focuses on DRAM aging, seeking answers on whether a used memory may present failures similar to Rowhammer over time.

1.3 Organization

This dissertation is organized as follows. Chapter 2 presents basic concepts of cryptography and elliptic curves, the Discrete Logarithm Problem, the Chinese Remainder Theorem, side-channel attacks, and fault injection. Then, Chapter 3 introduces the DRAM architecture, showing the cells' organization, construction details, and subsystem. It also presents a working memory controller, showing commands executed over the DRAM memory, memory timings used to control the memory, address mapping, Error-Correcting Code memory, and the Target Row Refresh solution.

Chapter 4 presents types of Rowhammer attacks, such as double-sided, single-sided,

one-location, and many-sided. This chapter continues addressing a step-by-step Rowhammer attack, showing how to find and put a target in a vulnerable region and flip a bit (or bits) from the target location. Lately, this chapter enumerates some software and hardware countermeasures. Chapter 5 approaches how to use Rowhammer to inject faults in the Bitcoin-core Wallet, the ECDSA protocol, and the curve `secp256k1`. Then, Chapter 6 addresses an initial study about DRAM aging and how it can lead to memories vulnerable to Rowhammer. Finally, Chapter 7 presents final remarks and directions for future works.

Chapter 2

Preliminaries

Cryptography is the practice and the study of techniques for secure communications in the presence of adversarial behavior [79]. These techniques aim to enforce the five pillars from Information Assurance [95]:

Confidentiality: a security measure that protects data against unauthorized access.

Integrity: a security measure that protects data against unauthorized modifications.

Authenticity: a security measure that provides confidence in data senders and validity of their message.

Availability: a security measure that ensures that data is retrieved or modified by authorized individuals.

Non-repudiation: a security measure that provides capability to determine if a particular action occurred such as creating information, sending a message, approving information, and receiving a message. In other words, perfect non-repudiation cannot argue falsely about the source or authenticity of data.

This chapter is organized as follows. Section 2.1 details basic concepts of cryptography, presenting symmetric and asymmetric cryptosystems. Then, Section 2.2 introduces Elliptic Curve Cryptography, including mathematical concepts, such as fields, elliptic curves and their operations, and elliptic curve for cryptography. Subsection 2.2.6 presents the Chinese Remainder Theorem (CRT), a fundamental theorem used to compose modular operations. Next, Section 2.3 details the categories of side-channel attacks: timing attack, cache attack, differential fault analysis, among others. Finally, Section 2.4 discusses fault-injection attacks.

2.1 Basic Concepts of Cryptography

The etymology of the word "cryptography" comes from "the art of writing secrets". A cryptographic system is defined by two bijective functions [68]: one for encryption of a

plaintext m into a ciphertext c, denoted $E_e(m)$, and another, $D_d(c)$, for decryption of a ciphertext c into a plaintext m. The encryption and decryption functions are parameterized by the keys e and d, respectively, each belonging to a finite set of keys $\mathcal{K}$. For every key $e \in \mathcal{K}$, there exists a key $d \in \mathcal{K}$ such that $D_d = E_e^{-1}$; that is, $D_d(E_e(m)) = m$.

A cryptographic system is weak if an attacker can systematically retrieve the plaintexts from the ciphertexts without knowledge of the keys (e, d) in a reasonable time [68]. How much is reasonable depends on the value of the information contained in the plaintext, that is, the attacker must not be able to easily enumerate all decryptions for each key $d \in \mathcal{K}$ (brute force attack) or retrieve the plaintext by using some weakness in the algorithm (cryptoanalysis) or implementation (side-channel). In the worst case, the attacker must spend more resources, such as time, energy, and money, than the plaintext value. These definitions make it possible to classify cryptographic systems as *symmetric* or *asymmetric*.

2.1.1 Symmetric System

A symmetric system contains a key d for decryption and a key e for encryption, and each can be calculated from the other in polynomial time [68, Definition 1.24]. Most symmetric cryptosystems choose the encryption key to be identical to the decryption key [68]. A symmetric system is represented in Figure 2.1, in which two entities, Alice and Bob, want to establish a secure communication. The keys are generated and distributed over a secure channel inaccessible to the attacker. In Alice's entity, a plaintext m is obtained and given as input to the encryption function, thus generating a ciphertext c. Then, the ciphertext is delivered to Bob through an insecure channel, where the attacker can interact with their communication to confuse Alice and Bob. When the ciphertext arrives at Bob, it goes through decryption, and the plaintext m can be retrieved if there was no tempering.

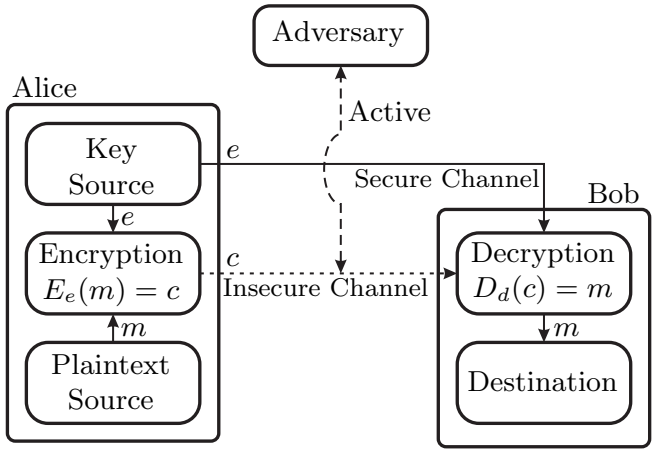


Figure 2.1: Representation of a symmetric cryptosystem [68].

Based on the model presented above, a modern symmetric cipher uses layers of confusion (substitution), diffusion (permutation), and key addition, also known as Substitution-Permutation Networks (SPN) [39]. During decryption and encryption, there exists an internal state that goes through each of these layers for repeated rounds. A natural example of SPN is the Advanced Encryption Standard (AES) [25].

2.1.2 Asymmetric System

An asymmetric system contains a public and a private key pair, so the private key cannot be derived from the public key in polynomial time [68, Definition 1.50]. Figure 2.2 represents an asymmetric system in which Alice is sending a ciphertext to Bob.

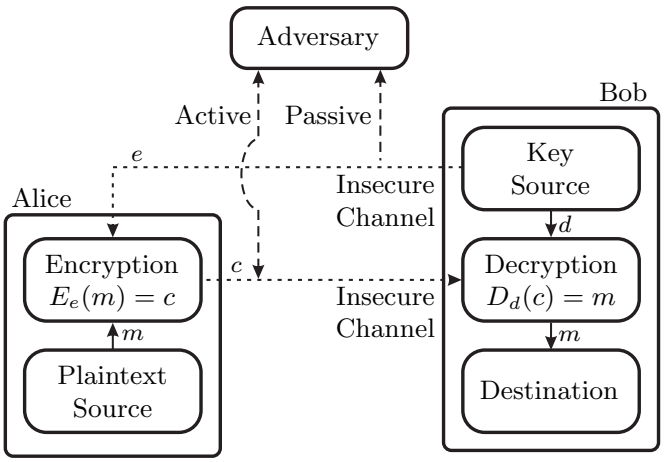


Figure 2.2: Representation of an asymmetric cryptosystem [68].

In this scenario, the attacker has access to two channels: one where he can actively interact with the insecure channel, and the other is passive, and he can observe without performing changes. Bob transmits his public key to Alice to encrypt the plaintext and send the ciphertext, which Bob then decrypts to obtain the plaintext.

2.2 Elliptic-Curve Cryptography

This section covers introductory concepts in elliptic-curve cryptography, such as the definition of fields, elliptic curves, equivalent curves, and isomorphisms, required for a greater understanding of the following chapters.

2.2.1 Fields

The triple $(\mathbb{K}, +, \star)$ represents a field $\mathbb{K}$, in which $\mathbb{K}$ is an non-empty set, $+$ is the addition in the field $\mathbb{K}$, and $\star$ is the multiplication in $\mathbb{K}$. The binary operations of addition and multiplication in the field $\mathbb{K}$ satisfy the following properties:

- **Addition** defined by the function $+ : \mathbb{K} \times \mathbb{K} \to \mathbb{K}$.
  1. **Commutative:** $\alpha + \beta = \beta + \alpha$, $\forall \alpha, \beta \in \mathbb{K}$.
  2. **Associative:** $(\alpha + \beta) + \gamma = \alpha + (\beta + \gamma)$, $\forall \alpha, \beta, \gamma \in \mathbb{K}$.
  3. **Neutral element:** $\exists \mathbb{0} \in \mathbb{K} \mid \alpha + \mathbb{0} = \alpha$, $\forall \alpha \in \mathbb{K}$.
  4. **Inverse:** $\exists -\alpha \in \mathbb{K} \mid \alpha + (-\alpha) = \mathbb{0}$, $\forall \alpha \in \mathbb{K}$.
- **Multiplication** defined by the function $\star : \mathbb{K} \times \mathbb{K} \to \mathbb{K}$.

1. **Commutative:** $\alpha \star \beta = \beta \star \alpha$, $\forall \alpha, \beta \in \mathbb{K}$.
2. **Associative:** $(\alpha \star \beta) \star \gamma = \alpha \star (\beta \star \gamma)$, $\forall \alpha, \beta, \gamma \in \mathbb{K}$.
3. **Distributive:** $\gamma \star (\alpha + \beta) = \alpha \star \gamma + \beta \star \gamma$, $\forall \alpha, \beta, \gamma \in \mathbb{K}$.
4. **Neutral element:** $\exists \mathbb{1} \in \mathbb{K} \mid \alpha \star \mathbb{1} = \alpha$, $\forall \alpha \in \mathbb{K}$.
5. **Inverse element:** $\exists \alpha^{-1} \in \mathbb{K} \setminus \{\mathbb{0}\} \mid \alpha \star \alpha^{-1} = \mathbb{1}$, $\forall \alpha \in \mathbb{K} \setminus \{\mathbb{0}\}$.

In the field $\mathbb{K}$, subtraction is done by adding the inverse additive, and division is performed by multiplying by the multiplicative inverse. The sets of rational $\mathbb{Q}$, real $\mathbb{R}$, and complex numbers $\mathbb{C}$ are examples of infinite fields.

The *order* of a field is the number of elements in $\mathbb{K}$. There exists a finite field $\mathbb{F}$ of order q if and only if q is a prime power, that is, $q = p^m$ for a prime number p called the *characteristic* of $\mathbb{F}$, and m a positive integer. If $m = 1$, $\mathbb{F}$ is said to be a *prime field*; otherwise, it is an *extension field*.

Example 2.1 (Prime field $\mathbb{F}_{11}$)**.** The elements of the prime field $\mathbb{F}_{11}$ are $\{0, 1, 2, \ldots, 10\}$. Examples of arithmetic operations in $\mathbb{F}_{11}$ are as follows.

i) Addition: $5 + 9 = 3$, since $14 \bmod 11 = 3$.

ii) Subtraction: $6 - 9 = 8$, since $-3 \bmod 11 = 8$.

iii) Multiplication: $7 \star 3 = 10$, since $21 \bmod 11 = 10$.

iv) Inversion: $7^{-1} = 8$, since $7 \star 8 \bmod 11 = 1$.

Let p be a prime and $m \geq 2$. $\mathbb{F}_p[x]$ denotes the set of all polynomials in the variable x with coefficients in $\mathbb{F}_p$. Let $f(x)$ be an irreducible polynomial of degree m in $\mathbb{F}_p[x]$. A polynomial $f(x)$ that cannot be factored as a product of polynomials in $\mathbb{F}_p[x]$, each of degree less than m, is said to be *irreducible*. The elements of $\mathbb{F}_{p^m}$ are the polynomials in $\mathbb{F}_p[x]$ of degree at most $m - 1$:

$$\mathbb{F}_{p^m} = \{a_{m-1}x^{m-1} + a_{m-2}x^{m-2} + \ldots + a_2x^2 + a_1x + a_0 : a_i \in \mathbb{F}_p\}$$

The addition in a field is the usual addition of polynomials with coefficient arithmetic done in $\mathbb{F}_p$. In turn, the multiplication of field elements is performed modulo an irreducible polynomial $f(x)$.

Example 2.2 (Extension field $\mathbb{F}_{41^4}$)**.** Let $p = 41$ and $m = 4$. It is possible to use the irreducible polynomial $f(x) = 2x^4 + 15x^2 + 29x + 21$ in $\mathbb{F}_{41^4}$. Examples of arithmetic operations in $\mathbb{F}_{41^4}$ are as follows. Let $a = 32x^3 + 39x + 5$ and $b = 15x^3 + 10x^2 + 21$.

i) Addition: $a + b = 6x^3 + 10x^2 + 39x + 26$.

ii) Subtraction: $a - b = 17x^3 + 31x^2 + 39x + 25$.

iii) Multiplication: $a \star b = 18x^3 + 6x^2 + 33x + 8$.

iv) Inversion: $a^{-1} = 21x^3 + 14x^2 + 2x + 24$.

The *algebraic closure* of a field $\mathbb{K}$ is denoted $\overline{\mathbb{K}}$, such that $\alpha \in \overline{\mathbb{K}}$ is root of a non-constant polynomial $P(x) = \sum_{i=0}^{n} a_i x^i, a_i \in \mathbb{K}$, that is, $P(\alpha) = 0$ [49]. For example, the algebraic closure of $\mathbb{R}$ is the set of complex numbers $\mathbb{C}$. Also, every non-constant polynomial $P(x)$ with coefficients in $\mathbb{R}$ has roots in the set of complex numbers. For example, the polynomial $P(x) = x^3 - x^2 + x - 1$ has one real root, which is 1, and two complex roots: i and $-i$. The algebraic closure of a field $\mathbb{F}_{p^m}$ is the union of all fields of extensions $\mathbb{F}_{p^m}$ for all $m \geq 1$.

2.2.2 Elliptic Curves

An *elliptic curve* E over a field $\mathbb{K}$, denoted $E(\mathbb{K})$, can be defined by the Weierstrass equation [38, 9] as

$$E : a_0 y^2 + a_1 xy + a_3 y = x^3 + a_2 x^2 + a_4 x + a_6, \tag{2.1}$$

for all $a_i \in \mathbb{K}$ and $a_0 \neq 0$. An elliptic curve requires the discriminant to be to ensure the non-existence of points $P = (x_P, y_P) \in E(\mathbb{K})$ for which the curve has two or more distinct tangent lines. This requirement forms a rule used to perform point additions. The discriminant is obtained using the j-invariant equation [94, 23], which was studied as a parameterization of elliptic curves over complex numbers, denoted $j(E)$ in the short Weierstrass (Equation 2.2) and Montgomery (Equation 2.3) equations. The j-invariant equation indicates if an elliptic curve equation can be transformed into another one in the algebraic closure $\overline{\mathbb{K}}$. In other words, the points satisfying the curve equations E and E' can be transformed into each other if $j(E) = j(E')$, showing that E is isomorphic to E' as described in Theorem 2.1. Moreover, elliptic curves with the same j-invariant are said to be a *twist* of each other.

Figure 2.3 exemplifies the elliptic curve $E : y^2 = x^3 - x + 1$ over the infinite field $\mathbb{R}$ and the finite prime field $\mathbb{F}_{43}$.

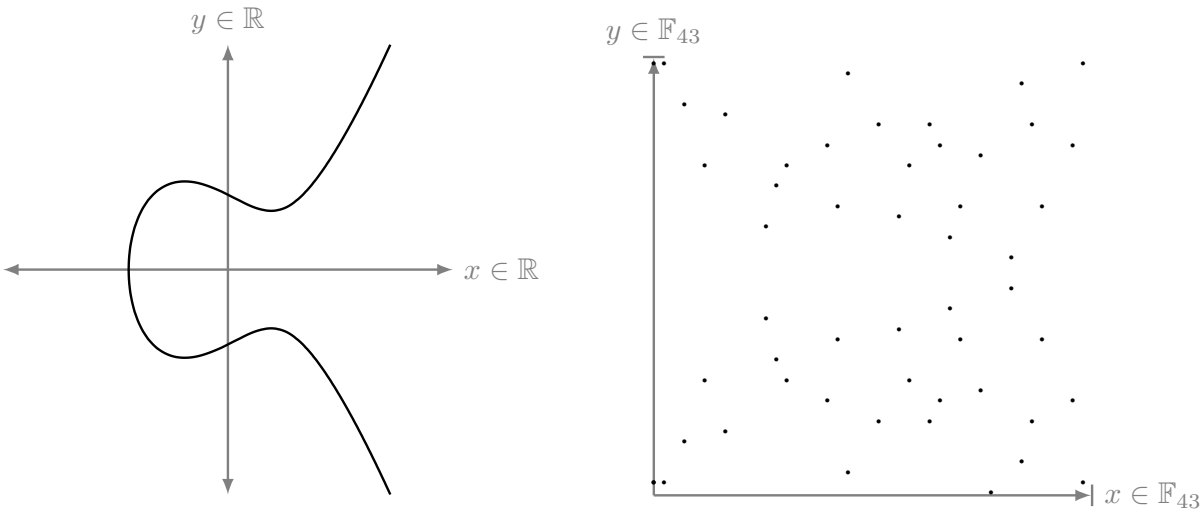


Figure 2.3: Example of elliptic curves over $\mathbb{R}$ and $\mathbb{F}_{43}$.

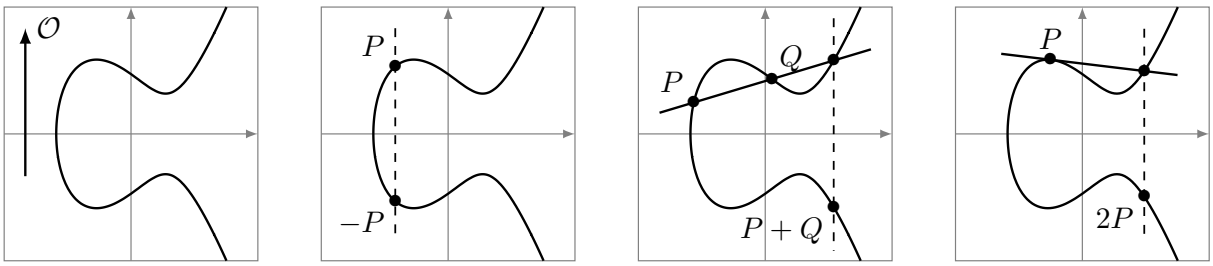
The Weierstrass equation, given in Equation 2.1, contains many simplifications, including the short Weierstrass and Montgomery curves equations, commonly used to represent elliptic curves defined over a finite prime field $\mathbb{F}_p$. The short Weierstrass equation has the form

$$\begin{aligned} &E : y^2 = x^3 + ax + b \\ &j(E) = 1728\frac{4a^3}{\Delta} = 1728\frac{4a^3}{4a^3 + 27b^2} \\ &4a^3 + 27b^2 \neq 0 \bmod p, \end{aligned} \tag{2.2}$$

whereas the Montgomery equation is given by form:

$$\begin{aligned} &E : by^2 = x^3 + ax^2 + x \\ &j(E) = 256\frac{(a^2-3)^3}{\Delta} = 256\frac{(a^2-3)^3}{a^2-4} \\ &b(a^2 - 4) \neq 0 \bmod p. \end{aligned} \tag{2.3}$$

The addition of two points in $E(\mathbb{K})$ is given by chord-and-tangent rules, for which the result is a third point in $E(\mathbb{K})$. The chord rule, also known as the secant rule, is used to add two distinct points $P + Q$. In contrast, the tangent rule is used to double a point P as $P + P = 2P$. Thus, elements of $E(\mathbb{K})$ form an additive group endowed with chord-and-tangent rules for addition, having $\mathcal{O}$ as the neutral element. Commonly, the neutral element is also represented as ∞ and referred to as the point "at infinity". Additive operations and the representation of the neutral and inverse elements are illustrated in Figure 2.4 for the elliptic curve $E(\mathbb{R})$. Note that the secant-tangent rules can be used interchangeably to obtain n doublings of the point P, that is, for computing nP. For example, the calculation of $5P$ can be made up of two doublings using the tangent rule – one for getting $2P$ and the other for $4P$ – and a sum of points using the chord rule for obtaining $4P + P$.



Neutral element $\mathcal{O}$ | Inverse element $-P$ | Addition $P + Q$ "Secant rule" | Doubling $P + P$ "Tangent rule"

Figure 2.4: Operations performed in an elliptic curve E in its additive group over $\mathbb{R}$.

We can also perform addition operations in an elliptic curve over a finite field. For instance, Figure 2.5 represents the chord-and-tangent rules for the curve $E : y^2 = x^3 - x + 1$ over $\mathbb{F}_{43}$. In the figure, consider that $P = (7, 6)$ and $Q = (12, 13)$. Thus, $P + Q = (38, 28)$ and $2P = (11, 17)$. Also, the angular coefficient α of the tangent line $y = \alpha x + \beta$

is $\alpha = (3x_R^2 - 1)2^{-1}y_R^{-1} \bmod 43$ for $R = (x_R, y_R) \in E(\mathbb{F}_{43})$. Thus, from the angular coefficient equation, we obtain that $\alpha = (3x_P^2 - 1)2^{-1}y_P^{-1} = (3 \cdot 7^2 - 1)2^{-1}6^{-1} = 5$ and, from the line equation, the first β is $\beta = y_P - \alpha x_P = 6 - 5 \cdot 7 = -29$. Observe that β needs to be changed for the line being kept inside $\mathbb{F}_{43}$.

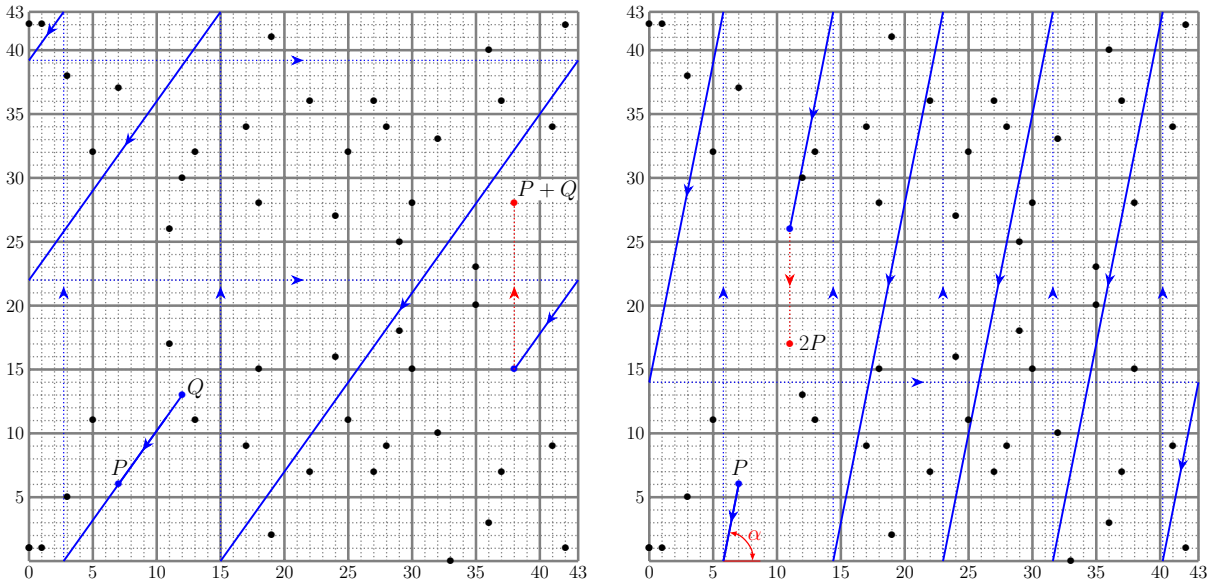


Figure 2.5: Representation of the addition and doubling operations for the elliptic curve $E : y^2 = x^3 - x + 1$ in its additive group over the field $\mathbb{F}_{43}$.

The order of an elliptic curve E defined over a prime finite field $\mathbb{F}_p$ is of the form $\#E(\mathbb{F}_p) = h \cdot \ell$, where $\ell \in \mathbb{Z}$ is the largest prime number that divides $\#E(\mathbb{F}_p)$ and $h \in \mathbb{Z}$ is the cofactor. In particular, the curve $E : y^2 = x^3 - x + 1$ over $\mathbb{F}_{43}$, depicted in Figure 2.3, has order $52 = (2^4) \cdot 13$, meaning that $h = 2^4 = 4$ and $\ell = 13$.

2.2.3 Short Weierstrass

This section introduces concepts of group law and isomorphism for elliptic curves, as required by Short Weierstrass curves that are elaborated in Chapter 5.

Group Law

A group law for the elliptic curve $E(\mathbb{K}) : y^2 = x^3 + ax + b$ with characteristic $\mathbb{K} \neq \{2, 3\}$ contains the following properties [38, Section 3.1.2].

i) Identity: $P + \infty = \infty + P = P$ for all $P \in E(\mathbb{K})$.

ii) Inverses: If $P = (x, y) \in E(\mathbb{K})$, then $(x, y) + (x, -y) = \infty$. The point $(x, -y)$ is denoted by $-P$ and is called the *additive inverse* of P. Note that $-P$ is a point in $E(\mathbb{K})$. Also, $-\infty = \infty$.

iii) Point addition: Consider $P = (x_1, y_1) \in E(\mathbb{K})$ and $Q = (x_2, y_2) \in E(\mathbb{K})$ with $P \neq \pm Q$. Then, the addition $P + Q = (x_3, y_3)$ is given by

$$x_3 = \left(\frac{y_2 - y_1}{x_2 - x_1}\right)^2 - x_1 - x_2 \quad \text{and} \quad y_3 = \left(\frac{y_2 - y_1}{x_2 - x_1}\right)(x_1 - x_3) - y_1.$$

iv) Point doubling: Consider $P = (x_1, y_1) \in E(\mathbb{K})$ with $P \neq -P$. Then, $2P = (x_3, y_3)$ is given by

$$x_3 = \left(\frac{3x_1^2 + a}{2y_1}\right)^2 - 2x_1 \quad \text{and} \quad y_3 = \left(\frac{3x_1^2 + a}{2y_1}\right)(x_1 - x_3) - y_1.$$

Isomorphisms

The isomorphism of two short Weierstrass elliptic curves is addressed by the Short Weierstrass Isomorphism, which is given in Theorem 2.1.

Theorem 2.1 ([94, Theorem 2.19] Short Weierstrass Isomorphism)**.** Let $y_1^2 = x_1^3 + a_1x_1 + b_1$ and $y_2^2 = x_2^3 + a_2x_2 + b_2$ be two elliptic curves with j-invariants j_1 and j_2, respectively. If $j_1 = j_2$, then there exists $0 \neq \mu \in \overline{\mathbb{K}}$ such that

$$a_2 = \mu^4 a_1 \quad \text{and} \quad b_2 = \mu^6 b_1.$$

Moreover, the transformation

$$x_2 = \mu^2 x_1, \qquad y_2 = \mu^3 y_1$$

takes one equation to the other.

Proof. First assume that $a_1 \neq 0$. Since this is equivalent to $j_1 \neq 0$, this imply also $a_2 \neq 0$. Choose μ such that $a_2 = \mu^4 a_1$. Then

$$\frac{4a_2^3}{4a_2^3 + 27b_2^2} = \frac{4a_1^3}{4a_1^3 + 27b_1^2} = \frac{4\mu^{-12}a_2^3}{4\mu^{-12}a_2^3 + 27b_1^2} = \frac{4a_2^3}{4a_2^3 + 27\mu^{12}b_1^2}$$

which implies that

$$b_2^2 = (\mu^6 b_1^2)^2.$$

Therefore $b^2 = \pm\mu^6 b_1$. If $b_2 = \mu^6 b_1$, ended. If $b_2 = -\mu^6 b_1$, then change μ to $i\mu$ (where $i^2 = -1$). This preserve the realation $a_2 = \mu^4 a_1$ and also yields to $b_2 = \mu^6 b_1$.

If $a_1 = 0$, then $a_2 = 0$. Since $4a_i^3 + 27b_i^2 \neq 0$, we have $b_1, b_2 \neq 0$. Choose μ such that $b_2 = \mu^6 b_1$. □

2.2.4 Elliptic Curves for Cryptography

An additive group $E(\mathbb{F}_p)$, for p prime, can be composed with linear combinations of its subgroups. In particular, it is possible to compose $E(\mathbb{F}_p)$ with the subgroups $E[h]$ and $E[\ell]$, which order subgroups are h and ℓ, respectively. When the integer h is composite,

the subgroup $E[h]$ can be decomposed into the subgroups corresponding to the factors of h. A point G is said to be a generator of the subgroup $E[x]$ if, for each point R in the subgroup x, there exists a scalar n such that $R = n \cdot G$.

For cryptography, the order $h \cdot \ell$ of an elliptic curve is appropriate when ℓ is a large prime number, and h is a small integer. Thus, this format of group order avoids a decomposition of $E(\mathbb{F}_p)$ into much smaller subgroups [8]. Additionally, only one generator is used for the subgroup $E[\ell]$, avoiding confinement in subgroups. In other words, for a generator $G \in E[\ell]$ with $k \in \mathbb{Z}$, we choose $k \cdot G \in E[\ell]$ instead of a random generator $G \in E(\mathbb{F}_p)$ that can lie in some subgroup in $E[h]$.

A curve E_1 with order $h \cdot \ell$ may contain an unsafe twist curve E_2 with order $h' \cdot \ell'$, for which the subgroup $E[h']$ is composed of several small subgroups and $h' \gg h$. Thus, an invalid curve attack can be performed by computing a point $Q \in E_2$ as if it was on E_1 [8]. Figure 2.6 exemplifies a curve E_1 with good properties for cryptography and an insecure twist E_2 containing several smaller subgroups.

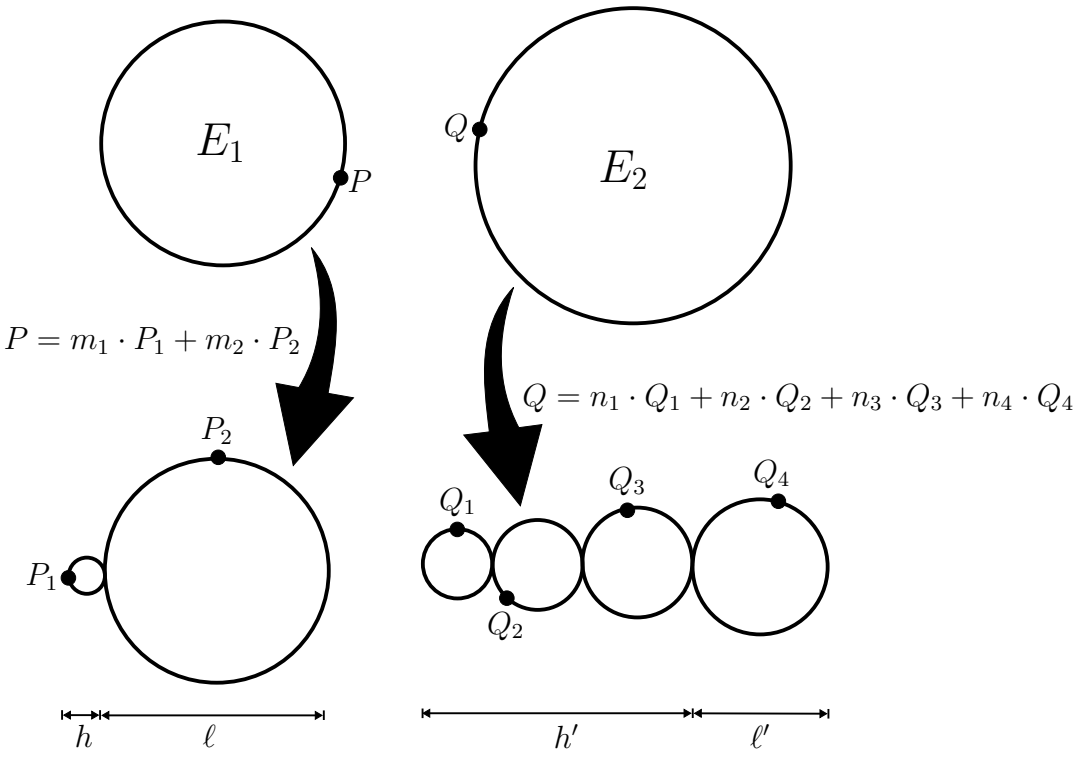


Figure 2.6: Two elliptic curves E_1 and E_2, where one is a twist of the other. If ℓ is a large prime, the curve E_1 contains a good safety margin since h is a small integer. The curve E_2 is considered unsafe because it contains a composite cofactor h', allowing the subgroup $E[h']$ to be factored into several smaller subgroups. Both curves contain illustrations of how a point behaves in $E(\mathbb{F}_p)$ and in its subgroups.

Compared to RSA[1] [78], asymmetric cryptosystems based on elliptic curves excel in two aspects. The first is performance, as curve implementations are more compact and efficient [38]. The second aspect is the key size to achieve a certain security level. When considering the bit security provided by a symmetric system using a 256-bit AES key [25],

[1]RSA, an asymmetric cryptographic system based on integer factorization.

RSA requires a 15360-bit key, whereas a curve-based cryptosystem needs only a 512-bit [6].

For illustration, the ElGamal cryptographic system using elliptic curves is presented in Algorithm 2.1, Algorithm 2.2, and Algorithm 2.3, which describe the tasks of key generation, encryption, and decryption, respectively.

During encryption, the function CHANGE_TO_POINT takes as input a plaintext m, given as a sequence of bytes, and encodes it into a point $M = (x, y)$. In the Elliptic Curve Cryptosystems [56, section 3], Koblitz describes some methods to encode plaintext messages as elliptic curve points. One of the methods is usually described as Koblitz's method and assumes that we are working over a prime field $\mathbb{F}_p$ and that plaintext is represented as a non-negative integer m less than $\frac{p}{1000} - 1$. To encode, let x coordinates be $x = 1000m, 1000m + 1, \cdots, 1000m + 999$ until find one where it is possible to compute the corresponding y-coordinate. Generally speaking, if the Koblitz method does not lead to a valid point on the curve, an alternative approach is to adopt a more complex encoding mechanism [28].

At the decryption's end, the inverse function CHANGE_TO_MESSAGE maps a point M back into a byte array m. In the above example, if $M = (x, y)$ and $x = 1000m, 1000m + 1, \cdots, 1000m + 999$, then CHANGE_TO_MESSAGE decoding is achieved by deleting the last 3 digits of the x-coordinate of an elliptic curve point.

Algorithm 2.1 Public and private key pair generation for ElGamal

Require: Prime p, curve E, generator point G, and order n
Ensure: Point S as the public key and scalar d as the private key

1: $d \xleftarrow{\$} \mathbb{Z}_n^*$ ▷ Choose a random number in $\mathbb{Z}_n^* = \{a \mid \gcd(a, n) = 1\}$
2: $S \leftarrow d \cdot G$

Algorithm 2.2 Encryption of a plaintext m using elliptic curves in ElGamal

Require: Prime p, curve E, generator point G, order n, public key S, and plaintext m
Ensure: Ciphertext (C_1, C_2)

1: $M \leftarrow$ CHANGE_TO_POINT(m) ▷ Transform the message m into a point (x, y)
2: $k \xleftarrow{\$} \mathbb{Z}_n^*$ ▷ Choose a random number in $\mathbb{Z}_n^* = \{a \mid \gcd(a, n) = 1\}$
3: $C_1 = k \cdot G$
4: $C_2 = M + k \cdot S$

Algorithm 2.3 Decryption of a ciphertext c using elliptic curves in ElGamal

Require: Prime p, curve E, order n, private key d, and ciphertext (C_1, C_2)
Ensure: Plaintext m

1: $X \leftarrow d \cdot C_1$ ▷ $d \cdot C_1 = d \cdot k \cdot G = k \cdot S$
2: $M \leftarrow C_2 - X$
3: $m \leftarrow$ CHANGE_TO_MESSAGE(M) ▷ Transform the point (x, y) into the message m

2.2.5 Discrete Logarithm Problem

The logarithm $\log_b a$ is a number k such that $b^k = a$. In mathematics, a group is a set equipped with an operation that associates any two elements with a third element belonging to the same group. A discrete logarithm is a group G, where $\log_b a$ is an integer k such that $b^k = a$. For *finite cyclic groups*, it follows that $\log_b a = k$ is $b^k \equiv a \bmod m$.

Example 2.3 (Discrete logarithm group modulo 13)**.** Next, we enumerate every integer number b in the group G that has solution for $\log_b 5$ in the cyclic group $G = \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12\}$, the set of integers modulo 13. By definition, $b^k = 5 \bmod 13$ with $b, k \in G$. Thus, the solutions are:

- $b = 2, k = 9$: $2^9 \equiv 5 \bmod 13$
- $b = 5, k = \{1, 5, 9\}$: $5^1 \equiv 5^5 \equiv 5^9 \equiv 5 \bmod 13$
- $b = 6, k = 9$: $6^9 \equiv 5 \bmod 13$
- $b = 7, k = 3$: $7^3 \equiv 5 \bmod 13$
- $b = 8, k = \{3, 7, 11\}$: $8^3 \equiv 8^7 \equiv 8^{11} \equiv 5 \bmod 13$
- $b = 11, k = 3$: $11^3 \equiv 5 \bmod 13$

Similar to integer factorization, the complexity of solving an instance of the Discrete Logarithm Problem increases according to the size of the numbers. In 1993, Menezes et al. showed how to convert Elliptic Curve Discrete Logarithm Problem (ECDLP) into the Discrete Logarithm Problem over a finite field [67]. This made it possible to finding the private key of an elliptic curve system instantiated with supersingular curves.

2.2.6 Chinese Remainder Theorem

Invalid curve attacks occurs when an attacker forces the victim to compute a common secret point $d \cdot G'$ using a point $G' \in E'[\ell']$ outside of the original defined curve E subverting the computation of $d \cdot G$ to $d \cdot G'$. Where G is the generator of $E[\ell]$, G' is the generator of $E'[\ell']$ with $\ell \gg \ell'$, and d is the private key. Because of that, an attacker can solve the Discrete Logarithm Problem over ℓ' and partially recover d. As a result, instead of obtaining $d \bmod \ell$, the attacker will have $d \bmod \ell'$. Then, using some computations in other subgroups, the result is combined using the Chinese Remainder Theorem (CRT):

Theorem 2.2 ([94, Theorem 2.19] Chinese Remainder Theorem (CRT))**.** If the integers n_1, n_2, ..., n_k are pairwise relatively prime, that is, $\gcd(n_i, n_j) = 1$ for $i \neq j$, then the system of simultaneous congruences

$$\begin{aligned} x &\equiv a_1 \bmod n_1 \\ x &\equiv a_2 \bmod n_2 \\ &\vdots \\ x &\equiv a_k \bmod n_k \end{aligned}$$

has an unique solution modulo $n = n_1 n_2 \cdots n_k$ [68, Definition 2.120]. By Gauss's algorithm [68, Definition 2.121] the solution x to the simultaneous congruences in the Chinese Remainder Theorem may be computed as $x = \sum_{i=1}^{k} a_i N_i M_i \bmod n$, where $N_i = n/n_i$ and $M_i \equiv N_i^{-1} \bmod n_i$.

The CRT can be used to recover an elliptic curve point from subgroups, as shown in Figure 2.6. The potential of CRT is illustrated in Example 2.4 and Example 2.5. Example 2.4 shows the uniqueness of group elements when represented in its subgroups. Also, Example 2.5 illustrates how the theorem can be used with large numbers.

Example 2.4 (Uniqueness of all elements mod 15)**.** Table 2.1 enumerates all numbers from 0 to 14, showing that each number is unique when written modulo 3 and modulo 5. For example, $11 \equiv 1 \bmod 5$ and $11 \equiv 2 \bmod 3$.

Table 2.1: Uniqueness of elements mod 15. Rows and columns show elements modulo 3 and 5, respectively.

mod3 \ **mod5**	**0**	**1**	**2**	**3**	**4**
0	0	6	12	3	9
1	10	1	7	13	4
2	5	11	2	8	14

Example 2.5 (NATO against Russia)**.** A North Atlantic Treaty Organization (NATO) captain is responsible for counting the number of soldiers about to invade Russian territory. When he arrived in front of the squad of soldiers, they were organized in groups of $n_1 = 11 \times 11$. One of the groups contained only four soldiers. The captain wanted to know the exact number of soldiers, and he knew the approximate number was 200,000. So, he asked the soldiers to reorganize themselves into groups of $n_2 = 5 \times 5$, $n_3 = 4 \times 4$, and $n_4 = 3 \times 3$. In all configurations, there was an incomplete group, as depicted in Figure 2.7. Particularly, for the 5×5 and 4×4 compositions, the number of remaining soldiers in the incomplete group was 24 and 13, respectively.

$a_1 \bmod n_1$	$a_2 \bmod n_2$	$a_3 \bmod n_3$	$a_4 \bmod n_4$
$4 \bmod 121$	$24 \bmod 25$	$13 \bmod 16$	$0 \bmod 9$

Figure 2.7: Columns and columns of soldiers organized in groups of $11 \times 11 = 121$, $5 \times 5 = 25$, $4 \times 4 = 16$, and $3 \times 3 = 9$.

Thus, the exact number of soldiers sent for the mission was obtained via CRT. First, consider that $n = n_1 n_2 n_3 n_4 = 121 \cdot 25 \cdot 16 \cdot 9 = 435{,}600$. Then, the computation proceeds

as follows.

$$N_1 = n/n_1 = n_2 n_3 n_4 = 25 \cdot 16 \cdot 9 = 3{,}600$$
$$N_2 = n/n_2 = n_1 n_3 n_4 = 121 \cdot 16 \cdot 9 = 17{,}424$$
$$N_3 = n/n_3 = n_1 n_2 n_4 = 121 \cdot 25 \cdot 9 = 27{,}225$$
$$N_4 = n/n_4 = n_1 n_2 n_3 = 121 \cdot 25 \cdot 16 = 48{,}400$$

$$M_1 \equiv N_1^{-1} \bmod n_1 \equiv 3{,}600^{-1} \bmod 121 \equiv 91^{-1} \bmod 121 \equiv 4 \bmod 121$$
$$M_2 \equiv N_2^{-1} \bmod n_2 \equiv 17{,}424^{-1} \bmod 25 \equiv 24^{-1} \bmod 25 \equiv 24 \bmod 25$$
$$M_3 \equiv N_3^{-1} \bmod n_3 \equiv 27{,}225^{-1} \bmod 16 \equiv 9^{-1} \bmod 16 \equiv 9 \bmod 16$$
$$M_4 \equiv N_4^{-1} \bmod n_4 \equiv 48{,}400^{-1} \bmod 9 \equiv 7^{-1} \bmod 9 \equiv 4 \bmod 9$$

Finally, the exact number of soldiers is the combination of the partial results

$$\begin{aligned} x &= (a_1 N_1 M_1 + a_2 N_2 M_2 + a_3 N_3 M_3 + a_4 N_4 M_4) \bmod n \\ &= (4 \cdot 3{,}600 \cdot 4 + 24 \cdot 17{,}424 \cdot 24 + 13 \cdot 27{,}225 \cdot 9 + 0 \cdot 48{,}400 \cdot 4) \bmod 435{,}600 \\ &= 221{,}149 \bmod 435{,}600, \end{aligned}$$

which is $x = 221{,}149$.

2.3 Side-channel Attacks

From time to time, an ingenious side-channel attack emerges. Examples are acoustic cryptanalysis that gained strength in 2014 [32], and speculative attacks such as Meltdown [64] and Spectre [57] in 2018. The following are the most popular categories of side-channel attacks, demonstrating the importance of concerning side-channel attacks when building a cryptographic model and carefully implementing it.

2.3.1 Timing Attacks

Timing attacks (TA) rely only on the algorithm execution time. This type of attack was introduced by Paul Kocher, who demonstrated its use on asymmetric cryptographic algorithms for recovering the private key [58]. Another classic timing attack relies on an authentication forge attack over HMACs[2], in which the attacker uses variances on the HMAC verification time to build a forged HMAC [61]. A countermeasure against timing attacks is the isochronous implementation of primitives that handle sensitive information.

2.3.2 Cache Attacks

Cache attacks (CA) are based on the ability to monitor the accesses to the victim's cache by sharing part of the same physical memory addresses. For example, both the victim and attacker can share the same library. Cache attacks can be considered a subcategory of

[2]HMAC, the acronym for Hash Message Authentication Code.

timing attacks, as they require comparing time measurements. An example is the attack FLUSH + RELOAD [97], which managed to retrieve an RSA key of 1024 bits by watching accesses to the cache of a GnuPG implementation[3].

2.3.3 Speculative Attacks

Speculative Attacks (SA) are based on the ability to monitor the speculative execution of the processor. Speculative attacks can be considered a subcategory of cache attacks, as they require comparing time measurements in the cache memory. Examples are Meltdown [64] and Spectre [57]. In particular, Meltdown breaks the mechanism that keeps applications from accessing the system's arbitrary memory (a.k.a, the kernel memory). In turn, Spectre tricks other applications into accessing arbitrary locations in their memory, allowing secret information to be leaked.

2.3.4 Simple Power Analysis

Simple Power Analysis (SPA) is a type of side-channel attack based on the observation of electromagnetic emanations that allows the reconstruction of related information. A classic example of a SPA attack was performed by Van Eck [90], which recovered what was being played on a monitor about 1 km away through the leakage of electromagnetic waves. This category of attacks is old, prior to the 80s, and contains examples of how spy agencies use it to retrieve information. In the case of the United States, the project dedicated to observing these leaks is known as TEMPEST [42] and is implemented by the National Security Agency (NSA).

2.3.5 Differential Power Analysis

Side-channel attacks using Differential Power Analysis (DPA) interconnect data dependencies by capturing electromagnetic emanations. In other words, several power measurements of the same stretch can make a difference and help infer some private data. An example of a DPA attack is the work of Nascimento et al., which used 1,000 measures to recover about 80% of a 256-bit key of an elliptic curve [60].

2.3.6 Differential Fault Analysis

Differential Fault Analysis (DFA) is a side-channel attack that introduces faults during computation. In the work of Tunstall et al. [88], for example, a 128-bit key was retrieved from AES by injecting faults through the clock.

2.3.7 Acoustic Cryptanalysis

Attacks based on Acoustic Cryptanalysis (AC) use the sound produced during computation. An example was given in 2014 by Genkin et al. when a 4096-bit RSA key was fully recovered through the sound emitted by the processor [32].

[3] `https://www.gnupg.org/`.

2.3.8 Remaining Data

Side-channel attacks can be performed based on retrieving sensitive data remaining in memory after being supposedly deleted. The work of Halderman et al. showed how to obtain the decryption key from an encrypted hard drive by freezing the memory in liquid nitrogen [37], characterizing a cold boot attack. Cold boot attacks can retrieve cryptographic keys in memory even when the computer screen is locked.

2.4 Fault Injection Attacks

The fault injection model works with four aspects [92]:

1. **Granularity:** how many bits are affected by the failure.
2. **Modification:** given a fixed bit, set as either zero or one, a modification can invert its value or make it random.
3. **Control:** guarantees location and when the failure will occur. The control can be precise, vague, or none.
4. **Duration or failure effect:** describes whether the failure is transient, permanent, or destructive.

A system can be attacked or analyzed in a non-invasive or invasive way. The classification is defined according to the operations performed on the chip. The best non-invasive techniques consist of reducing the supply voltage [4], changing the temperature [41], increasing supply voltages in small spikes [51, 82], and inducing fault-injection by clock [3]. On the other hand, the invasive technique can be considered an optical exposure [85, 91, 24]. For that, the attack requires uncapping the chip, causing permanent damage.

The Chinese Remainder Theorem (Subsection 2.2.6) is employed in implementations to accelerate computational calculations with large numbers. In the 90s, researchers began to worry about generating intentional failures [14, 13]. The RSA was the first cryptographic algorithm to suffer a fault-injection attack. In the implementation in question [14], RSA used the CRT for working with the primes that lead to the module $n = p \cdot q$ so that the decryption would follow the procedure described in Algorithm 2.4.

Algorithm 2.4 RSA decryption using CRT

Pre-computed: $p_{inv} \leftarrow p^{-1} \bmod q$, $n = p \cdot q$.
Require: Primes p and q, ciphertext c, and a private key given by the pairs $d_p = e^{-1} \bmod (p-1)$ and $d_q = e^{-1} \bmod (q-1)$, where e is the public key.
Ensure: Plaintext m.

```
1: procedure RSA_CRT(p, q, c, d_p, d_q)
2:     m_p ← c^{d_p} mod p
3:     m_q ← c^{d_q} mod q
4:     m ← (((m_q − m_p) · p_inv) mod q) · p + m_p mod n
5:     return m
6: end procedure
```

The attack [14] used the following procedures:

- Insert ciphertext c and collect plaintext m.
- Insert ciphertext c, inject any failure into m_p or m_q, and collect the plaintext $\widehat{m}$.
- Compute the greatest common divisor of $(m - \widehat{m})$ and n to factor the RSA modulus, since $(m - \widehat{m})$ contains a common factor with n.

Without loss of generality, suppose that the fault was injected in m_q. Thus,

$$\begin{aligned} m - \widehat{m} &= ((((m_q - m_p) \cdot p_{inv}) \bmod q) \cdot p + m_p) \\ &\quad - ((((\widehat{m_q} - m_p) \cdot p_{inv}) \bmod q) \cdot p + m_p) \bmod n \\ &= (m_q - \widehat{m_q}) \cdot p_{inv} \cdot p \bmod n \end{aligned}$$

Therefore, $\gcd(m - \widehat{m}, n) = p$.

In 2000, three fault injection attacks were proposed on elliptic curves in the work of Biehl et al. [12]:

1. Some implementations do not check if a certain point is on the desired curve. Thus, the attacker can use a point on another curve (a twist curve) and perform an invalid curve attack.
2. The second attack uses the fact that the implementation checks for an invalid point. The attacker injects faults during this verification to corrupt and therefore bypass it. After successfully bypassing the check, the attacker performs the first attack.
3. Finally, the last attack uses fault injection during point duplication and performs a differential analysis through these faults. This attack works similarly to the attack employed in the RSA + CRT described in Algorithm 2.4.

Higher-order attacks require capturing more data from side channels to retrieve the desired secret information. Suppose an attack on a given algorithm that needs to obtain or change x operations to be successful (i.e. first-order attack). Another attack that needs to obtain or change about $n \cdot x$ operations in the same algorithm by, for example,

inserting redundancy into the algorithm is called an attack of order n. Compared to their first proposal, the second attack of Biehl et al. [12] is an example of a second-order attack, since it needs to inject faults in the check phase and provide an invalid point.

Countermeasures against fault injections are always challenging, as preventing an attacker from trying to mount an attack is impossible. However, the attack can be made more demanding to succeed. A typical barrier against fault injection attacks is hiding sensitive hardware information as much as possible. These barriers can be built by encrypting the main memory and employing purposeful noises to make it difficult to read the bus. Moreover, one can enhance the metal layers to be a passive shield, add filters or security sensors such as power, time, light, and temperature, and wire the meshes to act as an active shield [66].

Computational barriers via software are also possible by performing several operations that lead to the same point and comparing the results. Alternatively, the inverse operation can be used to check if the result has not been changed, and both software countermeasures can be used concurrently. Nevertheless, second and higher-order attacks are still highly probable to succeed [12].

Chapter 3

DRAM Memory

Dynamic Random-Access Memory (DRAM) is the main memory in most computational systems. DRAMs are composed of memory cells, each having a capacitor responsible for holding a bit of information. Due to the capacitor's nature, the DRAM uses a process called refresh that updates memory cells frequently to preserve their charge and recover the stored logical bit.

In the computer system's, the main memory is critical to maintain its performance growth and technology scaling benefits. Lim et al. [63] showed that the number of processor cores in a computer doubles every two years, whereas the amount of DRAM memory doubles every three years. Thus, the memory capacity per core is affected by around 30% every two years, and even worse regarding memory bandwidth since the number of pins in a physical memory is limited and can not grow indefinitely. Figure 3.1, extracted from the work of Lim et al. [63], shows a projection for the growth in the number of processor cores and memory capacity over the past years.

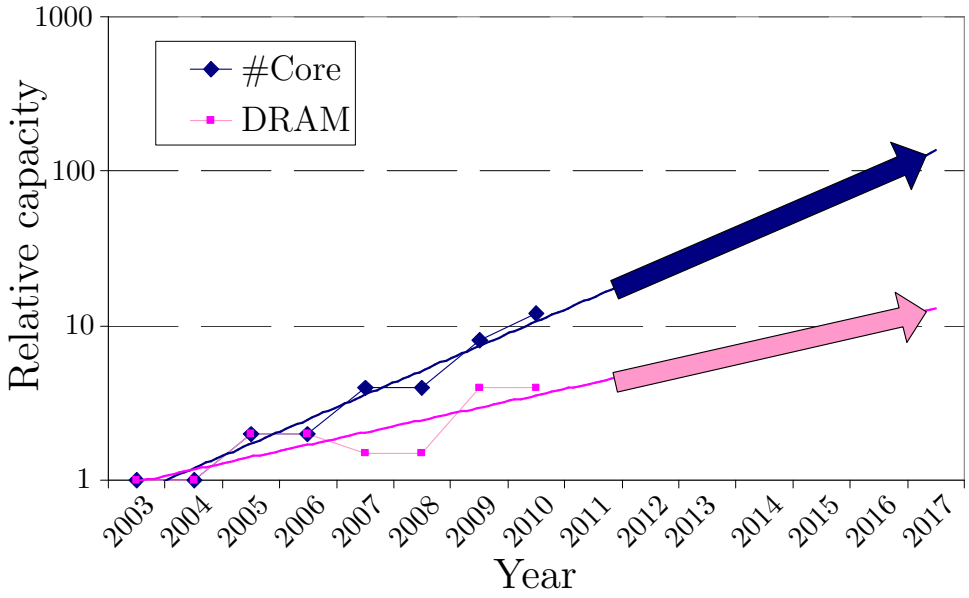


Figure 3.1: On average, the memory capacity per processor core is supposed to decrease 30% every two years [63].

A system's main memory must scale in size, technology, efficiency, cost, and algorithms

management to preserve performance even with its capacity growing slower than the number of processor cores. For this to be successful, many challenges need to be addressed since memory technology needs to handle multi-core accesses, data-intensive applications, and different technologies like GPUs[1], mobile, among others.

This chapter is organized as follows. Section 3.1 shows details about the DRAM architecture, including DRAM cell information, construction materials, and subsystem. Section 3.2 presents how the memory controller accesses DRAM information, detailing the required commands, the timings needed to execute each command, and the address mapping to distribute DRAM accesses.

3.1 DRAM Architecture

The following sections present the DRAM construction and architecture, showing its complexity in material selection, internal arrangement, and subsystem structure. For instance, each subsystem of the DRAM construction increments a specific size in its final configuration. Finally, an overview of the DRAM is presented to show its function and how large it can be.

3.1.1 DRAM Cells

DRAM memories can be seen as a two-dimensional array of cells, each cell composed of a transistor and a capacitor, known as *DRAM array* [47]. Part of a physical address contains the row and column number in the DRAM array. The DRAM array is composed of *DRAM rows*; otherwise, it is known as a *DRAM page*.

The Row Decoder selects the DRAM page in the DRAM array by activating a specific wordline. The Sense Amplifier amplify the signal through bitlines and copies the result of the cells selected by the wordline to the Row Buffer. Then, the row's content is entirely moved to the Row Buffer. This process is said to leave that row open. Any read or write in some column is processed over the Column Decoder directly affecting the Row Buffer. The row is closed – that is, the data kept by the Row Buffer is returned to the original row – only by command [47]. This entire process is illustrated in Figure 3.2.

[1]GPU, the acronym for Graphics Processing Unit.

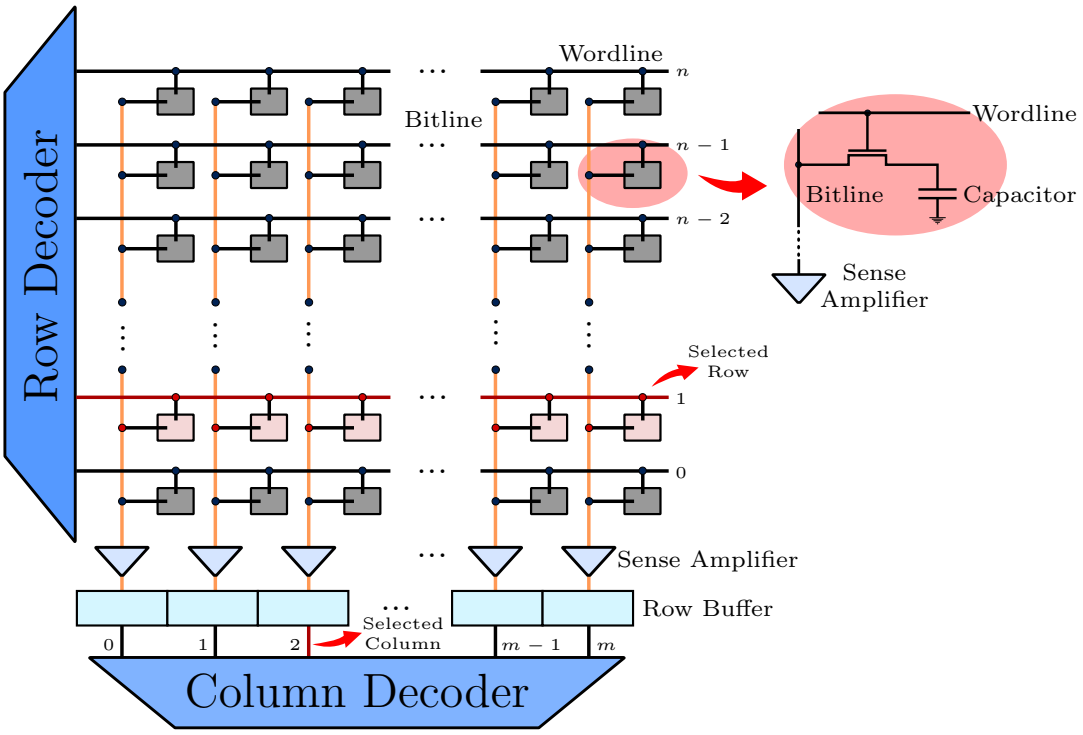


Figure 3.2: Memory arrays with an expansion of the memory cells in red.

Example 3.1 presents a decoder from a physical address. For better understanding, combine Figure 3.2 and Example 3.1.

Example 3.1 (Finding the physical address `0b000100001`)**.** Consider a DRAM array with five columns and four rows bits addresses respectively, in which the address `0bCCCCCRRRR` represents the column `0bCCCCC` and the row `0bRRRR`. Then, the address `0b000100001` needs to activate the wordline 1 (`0b0001`), bring the entire row to the Row Buffer, and then select column 2 (`0b00010`).

3.1.2 DRAM Construction

According to the International Technology Roadmap for Semiconductors (ITRS) [44], the scaling of the DRAM technology is ending since hardware projects will not be able to scale below a certain length in nanometers. The most used material for building a capacitor in DRAMs has a minimum distance between 5 and 7 nanometers of manufacturing concerning proper capacitance. If the capacitor has a higher capacitance, the data in memory does not need to be refreshed frequently because a more elevated capacitance implies the possibility of recovering the logic value of the capacitor, which is zero or one, even with a charge loss. On the other hand, a distance below 5 nanometers requires new materials for manufacturing the capacitor.

Recall that the capacitor is only a part of the memory cell, as shown in Figure 3.2. The entire DRAM cell can reach up to 28 nanometers using TiN (Titanium and Nitrogen), the currently most used material for building capacitors. In 2009, the ITRS [44] considered adopting other materials for 20 nanometers. From 2019 to 2024 the DRAM memories will be made it in 20 nanometers [45, Section 8.2]. Kim et al [52] claimed that will

be really challenging for further scaling toward 10 nanometers and DRAM may end at approximately 15 nanometer technology due to the capacitor constraints. The Figure 3.3 shows the evolution of the DRAM capacitors scaling.

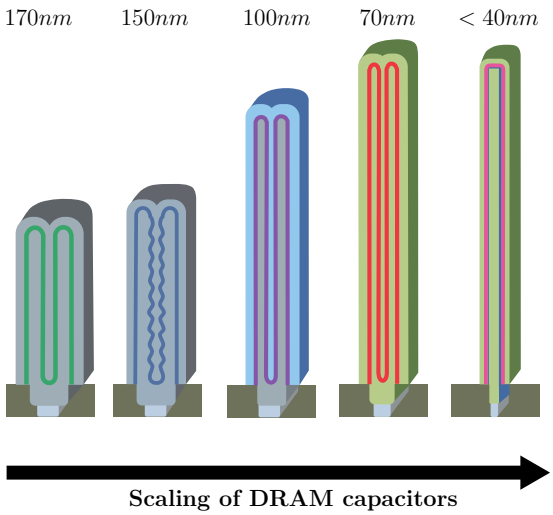


Figure 3.3: Scaling of DRAM capacitors over the years [53].

The time to access a transistor and capacitor based memory should be large enough to allow low leakage and high retention. However, creating a reliable sense amplifier is challenging since cell storage tends to reduce over time.

3.1.3 DRAM Subsystem

The DRAM subsystem is depicted in Figure 3.4, showing from its outer (channel) to the inner (column and row) composition. This section focuses on the DRAM Double Data Rate 4 (DDR4) standard [47], presenting each layer in a DRAM subsystem.

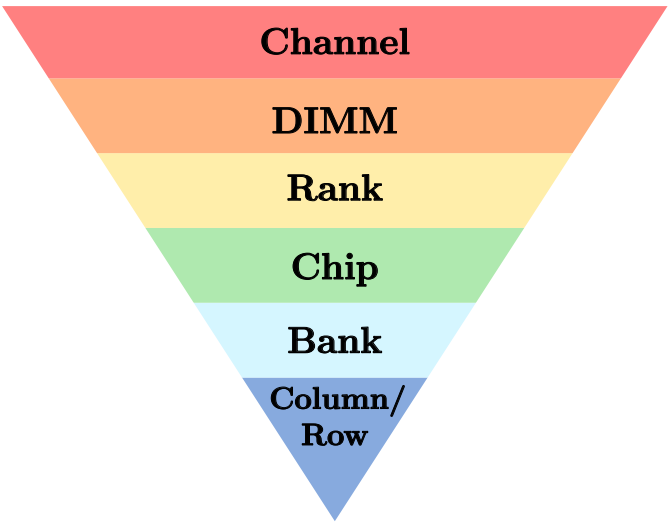


Figure 3.4: Memory subsystem organization from multiple modules to cells.

Column and Row

The DRAM row, or DRAM page, has a maximum of 18 physical pins for access. The maximum number of rows is $2^{18} = 256$K. Nowadays, the common number of rows is 32,768 or 32K – giving an 8GB memory. For reading and writing, a DRAM page is moved to the Row Buffer. Internally, the rows are split into small groups known as subarrays [55] to facilitate accesses on different rows of the bank. Therefore, each sub-array contains its Row Buffer that can then be copied to a global Row Buffer.

A DRAM column can be accessed from the Row Buffer and have a maximum of 10 physical pins for access. The maximum number of columns is $2^{10} = 1$K. Nowadays, a common number of columns is 1024 or 1K, and each column usually has 8 bits. With this information, it is possible to observe that a natural size for the Row Buffer is 8,192 bits or 8Kb.

Banks

A DRAM bank is a series of bi-dimensional arrays of cells in the form of rows and columns. Additionally, the bank has a Row Decoder, a Column Decoder, and a Row Buffer, as illustrated in Figure 3.5. Usually, the bank has an 8-bit input and output. Considering the values mentioned previously, the bank is composed of a three-dimensional matrix of bits, having 8 bits in depth, 1K addresses through the columns, and 32K addresses in the rows.

The Chip

A DRAM chip consists of multiple banks. The chip is composed of four bank groups, and each group has four banks. Resulting in 16 banks in a chip. The chip has a narrow interface providing from 4 bits to 16 bits per reading – the most common number is 8 bits.

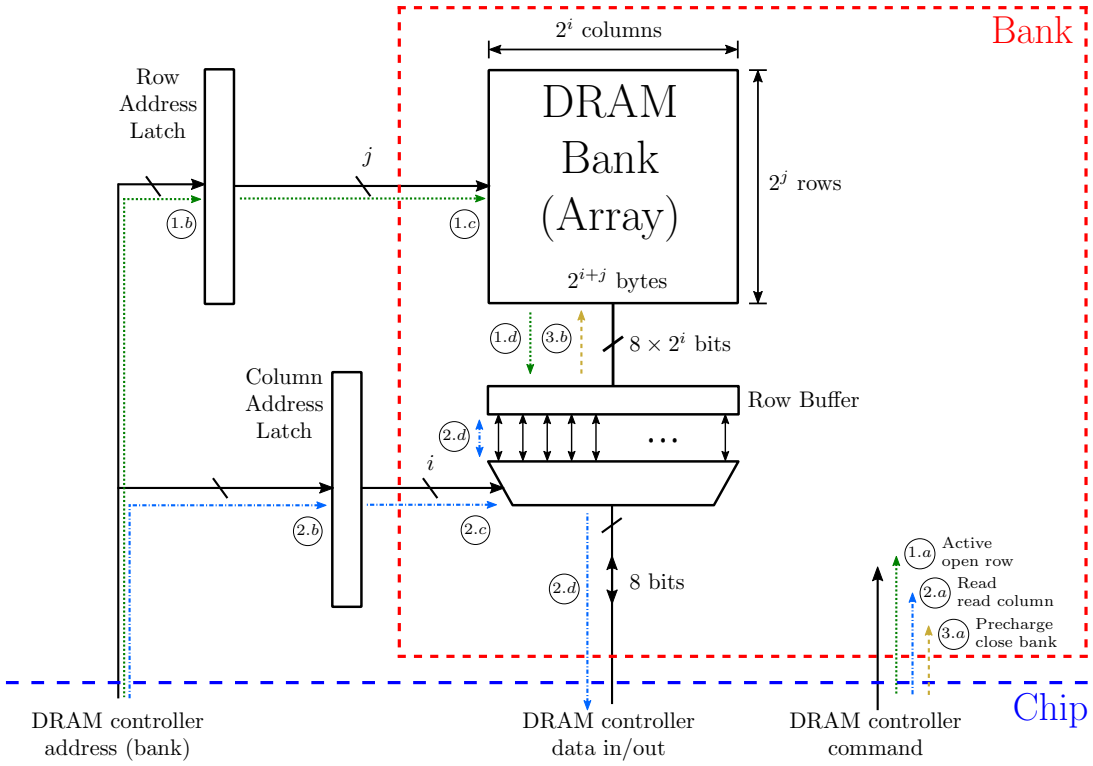


Figure 3.5: A high-level overview of a DRAM chip consisting of multiple banks sharing addresses, data, and command buses. Memory bank – array, row address, column address and sense amplifiers.

The Rank

The DRAM rank comprises multiple chips operating together by a single interface. All chips composing a rank are controlled concurrently by sharing the address and command buses, but each is provided with different data information. The rank has three physical pins to be selected, so the maximum number of ranks is eight. A typical number of chips in a rank is eight, and each chip in this configuration provides 8 bits, resulting in 64-bit wide data. Recall that a bank's standard number for the Row Buffer is 8Kb. However, as a rank opens a row in a bank on each chip, it is possible to consider a larger Row Buffer of $8 \times 8\text{Kb} = 8\text{KB}$, or 8,192 bytes.

The Dual Inline Memory Module

The Dual Inline Memory Module (DIMM), also called a module, consists of one or more ranks. The terminology dual inline came from the front and back pins of the module. In DDR4's case, there are 144 pins in the front and 144 pins in the back, apart from the Error Correction Code (ECC). A DIMM composed of ranks with 8 chips and providing 8 bits per chip has the advantage of high capacity and flexibility for the memory controller since it does not need to deal with individual chips. However, it has at least one disadvantage, which is the granularity, because the access cannot retrieve less than the interface width [69]. Figure 3.6 shows the chip and rank location in a DRAM module, allowing the visualization of the memory structure from the chip to the DIMM.

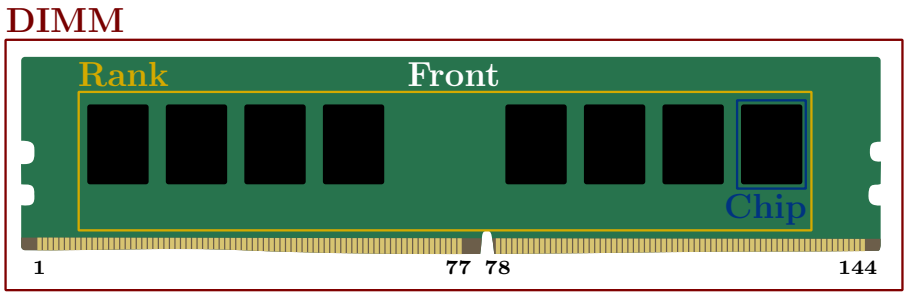


Figure 3.6: DDR4's DIMM without Error-Correcting Code (ECC). The figure shows the front of the modulo and half of the pins.

The Channel

The DRAM channel has one or more DIMMs. Each channel needs a different memory controller from the CPU [69]. The typical number of DIMMs per channel is two. This information can be retrieved from the processor and motherboard.

Example 3.2 (16GB DDR4 laptop)**.** Consider a laptop with one channel and two DDR4 slots, each having two DIMMs of 8GB. Each DIMM has 2 ranks having 8 chips. Note that each rank has 4GB (8GB/2) and each chip has 512MB (4GB/8). Thus, the computation of the number of banks, columns, and rows of the 8GB DIMM proceeds as follows.

- **Banks:** Each chip has 4 bank groups and 4 banks per bank group, leading to 16 banks in total. A physical address selecting the bank group $X \in \{1, 2, 3, 4\}$ and the bank $Y \in \{1, 2, 3, 4\}$ from the group X will select the same bank on all eight chips. Then, the number of banks in the DIMM can have two possible values: 16 banks, from a physical address perspective, or $2 \times 16 \times 8 = 256$ banks, considering all the chips in the two ranks. Thus, taking the physical address perspective, each bank has 32MB (512MB/16).

- **Columns:** The higher the number of columns, the greater the Row Buffer will be (Figure 3.5) and potentially larger the consecutive addresses that can be accessed. With 32MB remaining for each bank, it is possible to maximize the number of columns to 2^{10} or, equivalently, 10 bits in the physical address. Thus, each column will have 1,024 bytes.

- **Rows:** The bus data needs to provide 64-bit wide data, assuming eight chips and each chip giving 8 bits. The entire bank has $8 \times \#\text{rows} \times \#\text{columns}$ bits $= 32\text{MB} = 8 \times 32 \times 1{,}024 \times 1{,}024$ bits. Since the number of columns is 1,024 and each column provides 8 bits, the above computation ends up resulting in $32 \times 1{,}024 = 32{,}768 = 2^{15}$ rows, which represents 15 bits in the physical address.

3.2 Memory Controller

The memory controller is complex, comprehending more than fifty time constraints for its proper functioning [47, 62]. Examples of constraints are Read to Read Delay (tRTR),

Read to Write Delay (tRTW), and Refresh Period (tREF). Beyond the constraints, the memory controller needs a minimum of bank conflicts, a good rank selection, and knowing the open rows and refresh rate to avoid loss of information and save energy. Moreover, the controller still needs to maintain quality of service [62, 55].

Although having a high-level view of the system's memory, the memory controller needs to control some commands to prevent leaks to maintain reliability and security. DRAM and memory controllers, as known today, are unlikely to satisfy all requirements of capacity, bandwidth, and, especially, reliability and security [54]. Thus, rethinking the main memory system is necessary to fix DRAM issues and enable emerging technologies while satisfying all requirements.

3.2.1 Memory Controller Commands

Table 3.1 presents each memory controller command:

Table 3.1: DDR4 commands summarized [47].

Command	Abbreviation	$\overline{\text{CS}}$	BG_{1-0} BA_{1-0}	$\overline{\text{ACT}}$	A_{17}	A_{16} $\overline{\text{RAS}}$	A_{15} $\overline{\text{CAS}}$	A_{14} $\overline{\text{WE}}$	A_{13}	A_{12} BC	A_{11}	A_{10} AP	A_{9-0}
Deselect	DES	H	X										
Active (Row Opening)	ACT	L	Bank	L	Row Address								
No Operation	NOP	L	V	H	V	H	H	H	V	V	V	V	V
ZQ Calibration	ZQCL/ZQCS	L	V	H	V	H	H	L	V	V	V	Long	V
Read	RD/RDA RDS4/RDAS4 RDS8/RDAS8	L	Bank	H	V	H	L	H	V	Burst Chop	V	Auto-Precharge	Column Address
Write	WR/WRA WRS4/WRAS4 WRS8/WRAS8	L	Bank	H	V	H	L	L	V	Burst Chop	V	Auto-Precharge	Column Address
Reserved for Future Use	RFU	L	V	H	RFU	L	H	H	RFU				
Precharge All Banks	PREA	L	V	H	V	L	H	L	V	V	V	H	V
Single Bank Precharge	PRE	L	Bank	H	V	L	H	L	V	V	V	L	V
Refresh	REF	L	V	H	V	L	L	H	V	V	V	V	V
Mode Register Set	MRS	L	Register	H	Opcode	L	L	L	Opcode				

Signal level: high (H), low (L), valid signal (V) either low or high, and irrelevant (X). Logic level: Active, Inactive, Not interpreted. All other words have a specific value depending on the command.

- **Deselect:** A chip only recognizes a command when Chip Select ($\overline{\text{CS}}$) is low. Thus, this command deselects a chip by setting $\overline{\text{CS}}$ as high. Three more signals, C_0, C_1, and C_2, are required for selecting a chip, omitted in Table 3.1 for simplicity.

- **Active:** This command opens a specific row – that is, moves the DRAM row to the Row Buffer – of an idle given bank by selecting a Bank Group (BG_{1-0}) and a bank (BA_{1-0}) inside the bank group. The read and write commands are valid only if the banks row is open.

- **No Operation:** This command does not specify an operation, but the chip remains selected.

- **ZQ (or Zero-Quotient) Calibration:** This command has two versions. The first version is ZQCL, denoted long when the signal A_{10} is high, used during a power-up initialization sequence. In a final step of this command, the DRAM provides an impedance of transmission lines and output driver (signal after column decoder). The second version is ZQCS, called short (A_{10} is low), performed periodically to adjust for voltage and temperature variations.

- **Read:** This command reads from a specific bank and column address (A_{9-0}) and has six variations depending on the burst chop and auto-precharge modes. In modern processors, a typical cache line size is 64 bytes, but the memory only provides 8 bytes (64 bits) of data simultaneously. For convenience, the DRAM memory implements the burst chop that receives as input and provides as output 4 cycles of data (32 bytes) or 8 cycles of data (64 bytes) of sequential columns. The auto-precharge command finalizes a reading or writing with precharge. In other words, the precharge closes the row, returning the data from the Row Buffer to the original row. From hereon, commands terminated with 'A' mean auto-precharge (AP equals high), and 'S4' and 'S8' are meant for on-the-fly burst chop of size 4 and 8, respectively. The commands RD and RDA have fixed-size burst chop (4 or 8) configured by the register's mode.

- **Write:** This command works similarly to the read command, but, as the name indicates, it writes information in the DRAM.

- **Reserved for Future Use:** This command can extend the DRAM functionality in the future. All the address pins with RFU[2] (A_{13-0}, A_{17}) can be changed in the future.

- **Precharge All Banks:** This command closes all rows in all banks.

- **Single Bank Precharge:** This command closes a single row from a specific bank.

- **Refresh:** This command acts similarly to executing the active and precharge commands in a row. During refresh, the bank is unavailable, and each row typically requires refreshing at a specified number of milliseconds – this number is usually 64 milliseconds [47]. There exist two models of refresh: burst and distributed. The burst model refreshes all rows, one after the other. The distributed model refreshes each line at a different time and at regular intervals. Because of the refresh, the memory has a limited density, as refreshing needs to be fast and at a well-defined interval [65, 18]. The refresh command has two variations: refresh and self-refresh. The first variation refreshes a row using an internal row counter and auto-increments the counter in each requested refresh command. The second command orders the chip to self-refresh on schedule without an external clock.

- **Mode Register Set (MR0-MR6):** This command configures options such as normal operation, reset an operation, burst type, and burst length. It also defines some latency preferences.

[2]RFU, the acronym for Reserved for Future Use.

3.2.2 Memory Timings Explained

Although the DRAM memory has many different internal latencies, this section focuses on the most important timing parameters. In order of relevance, the memory timings are the CL, the tRCD, the tRP, and the tRAS. For example, a 4GB DDR4-2400 UDIMM 1.2V CL17 is a memory with 4GB of capacity, a clock rate of 2400 MHz, a UDIMM from an unbuffered module, a nominal voltage of 1.2V, and a CL that takes 17 clock tics for execution. The most crucial timings are detailed in the following:

- **CL (CAS Latency):** CAS stands for Column Address Strobe. After opening a row, it refers to how many clock cycles will be necessary until receiving a result after sending a column address to a read command. The write command has a similar latency, called CWL (CAS Write Latency), which is usually smaller than the CL. The time required to send the column address is called AL (Address Latency). The latency of AL is zero sometimes, and only the CL is relevant.

- **tRCD (RAS to CAS Delay time):** RAS stands for Row Address Strobe. Once the activation command is sent, one must wait for the tRCD to execute a read or write command.

- **tRP (RAS Precharge time):** tRP refers to the time required for closing a row in a bank.

- **tRAS (RAS Active time):** tRAS is the minimum number of cycles a row has to be active to ensure a certain amount of time to access the data. Usually, the tRAS needs to be greater or equal to the sum of the CL, tRCD, and tRP latencies.

Figure 3.7 exemplifies a writing on a CPU cache line, where memory controller uses the active command to denote that the row is already open before the writing. More columns can be read or written from this row before closing. Also, Figure 3.8 shows an example of reading a CPU cache line, where memory controller, after the active command also uses precharge command to close the row before finishing the read command. It is possible to start the precharge before finishing the read command, since the data will be present in the Row Buffer [44, Section 4.24.3]. For write command it is necessary finishing to start the precharge, because wrong data can be saved.

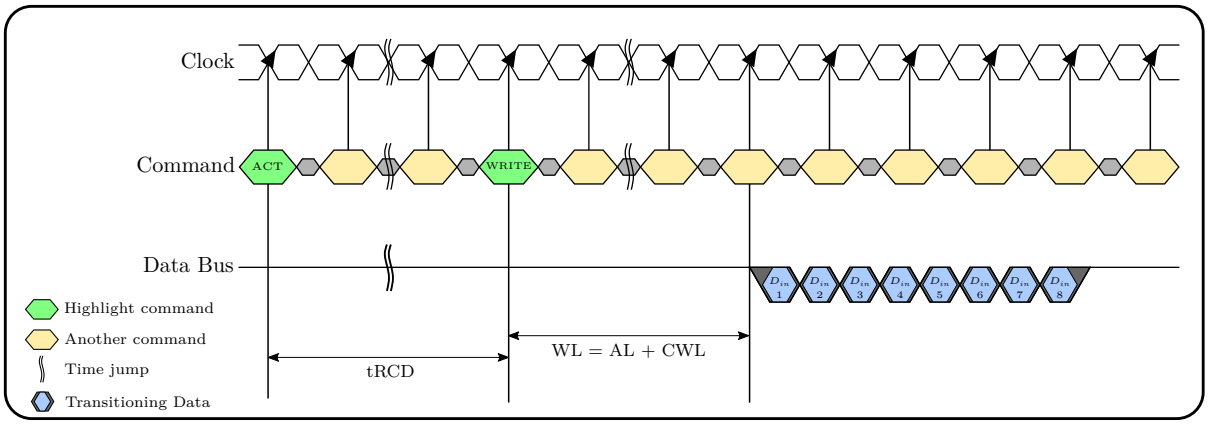


Figure 3.7: Memory controller executing a activate, and a write in a burst of 8 × 64-bit data.

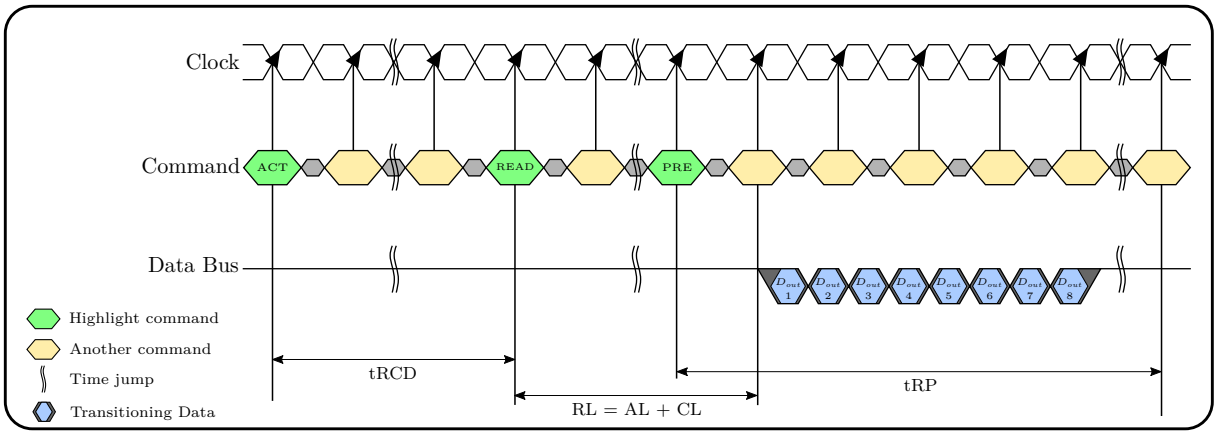


Figure 3.8: Memory controller executing a activate, a read in a burst of 8 × 64-bit data, and a precharge command for close the open row.

3.2.3 Address Mapping

When scheduling commands, the memory controller prioritizes older commands first and column commands (read/write) before row commands (active/precharge) [62].

Multiple banks and channels enable concurrent accesses in the DRAM since there is no intersection between the addresses of different banks. Moreover, channels enable concurrency because they have different physical buses, which also rises the bandwidth. The higher concurrence between banks and channels is necessary to minimize conflicts. In this sense, the controller tries to randomize the index selection of banks and channels [75]. In an address, the least significant bits have the largest entropy as they change faster than the most significant bits. Additionally, a way to select banks and channels is to perform a simple hash function that can be implemented by XORing the indexes.

Intel processors do not present the mapping of the physical address and the memory pins in their manual. Because of that, Pessl et al. [72] presented a side-channel attack for reverse engineering the mapping of the DRAM. The attack is based on time measurements

of memory accesses for two different addresses. In this case, three access possibilities exist: *i*) the addresses are in different banks, *ii*) the addresses are in the same bank and row, and *iii*) the addresses are in the same bank but in different rows. In the case of two addresses in the same bank but in different rows, accesses generate a bank conflict, thus having a more significant access latency since an active, a precharge, and a new active command are performed. In other cases, there is no precharge latency, and different banks can be accessed concurrently. Therefore, it is possible to conclude that any two addresses are in the same bank but in different rows by identifying a latency higher than the average. Moreover, doing this several times and creating sets of addresses residing in the same bank allows reverse engineering the mapping. Figure 3.9 shows a reversing of an address mapping in a server with four DIMMs of 8GB.

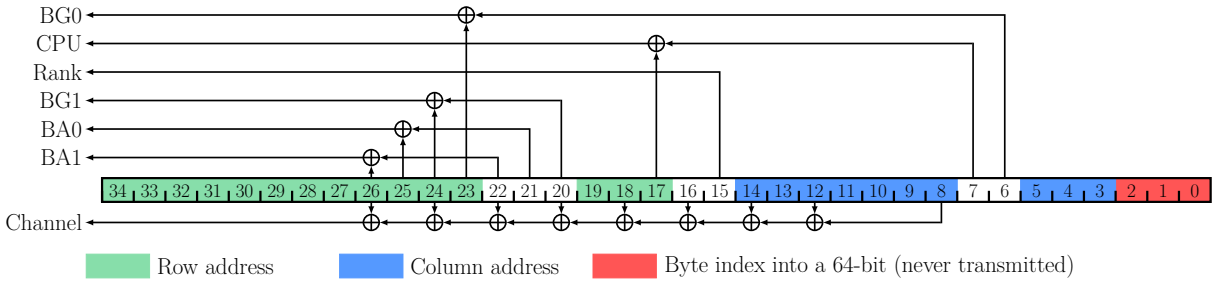


Figure 3.9: Memory controller mapping of a dual Haswell-EP Xeon E5-2630 v3 with 4x8GB dual rank DIMMs. Figure adapted from DRAMA paper [72]. DDR4 memorires contains four Bank Groups (BG1, BG0), four Banks (BA1, BA0) in each Bank Group.

3.2.4 The Error-Correcting Code Memory

In a 64-bit system, a DDR memory provides 64 bits of data [47], and 72 bits of data in the case of an Error-Correcting Code (ECC) memory. To simplify memory manufacturing, one more chip is added for the 8-bit correction [20]. A DRAM with ECC can detect single-error correction and double-bit error detection (SECDED) inspired by the extended Hamming code [20].

3.2.5 The Target Row Refresh Solution

For preventing memory corruption, DDR4 memories isolate rows neighboring the one that will perform an activation or precharge [47]. This technique is called Target Row Refresh (TRR), and each manufacturer implements their own solution [29].

Chapter 4

Rowhammer

Rowhammer is a security vulnerability that takes advantage of an unintended and undesirable side effect in DRAM memories construction in which memory cells interact electrically between themselves by leaking their charges, possibly changing the contents of nearby memory rows by specially crafted memory access patterns that rapidly activate the same memory rows in short time window.

The first study showing a more complex test for detecting Rowhammer appeared in 2012, focusing on embedded DRAMs (eDRAMs) [40]. The study found that alternate toggling between wordlines accelerates the flow of charge between two bridged cells, causing the cell to fail. In 2014, Kim et al. [54] published a well-documented study showing that Rowhammer is a severe security issue that can be activated from typical DRAM systems like DDR3. According to Kim et al., the continued reduction of DRAM memory cells created some advantages, such as reduced cost per memory bit and increased cell density. However, this same reduction assisted in the arise of Rowhammer, which occurred mainly because of three reasons:

1. The memory cell reduction implies the reduction of the capacitor size, and so the amount of stored charge. This fact hinders the sense amplifier's ability to determine the logical condition of a memory cell. In order to maintain the capacitor charge, even with its reduction, it is necessary to change the material used in its construction. However, the materials introduced the hot carrier injection, a phenomenon in solid electronic circuits in which an electron can gain enough kinetic energy to overcome the potential barrier needed to change a state.

2. The cells have become increasingly closer, which introduces the electromagnetic coupling effect, in which they unintentionally interact electromagnetically among themselves. In other words, during an active or precharge instruction, the chosen row influences the near rows by unintentionally opening them with less intensity, implying in loss of charge.

3. Due to variations in the technology adopted in the manufacturing process, an increased number of outlier cells interact among themselves by doing inter-cell crosstalk, implying the construction of conductive bridges.

According to a news story from Motherboard Vice [46], the Rowhammer attack is "like breaking into an apartment by repeatedly slamming the neighbor's door until the vibrations open the door you were after". In the context of memories, when performing readings by opening (activating) and closing (precharge) the same row several times in a small interval, the neighboring rows are affected, having their bits flipped. According to Kim et al. [54], the amount of opening and closing instructions required to perform a Rowhammer via bit flips is about 139K in old DDR3 memories.

In the study of Kim et al. [54], two addresses are in the same bank but different rows known as aggressors rows. As each address is in a different row, switching the address access requires toggling the active, read, and precharge commands at all times, maximizing the opening and closing rates of the same row. Thus, some observations in older DDR3 memories could be made:

- The shorter the access time between the aggressors, the greater the number of induced errors. The time of 55ns is the smallest that can be used to alternate access the two aggressors. However, about 65ns is enough to generate many errors. Above 500ns, the error rate is zero.
- If the refresh rate is reduced by about seven times, from 64 ms to an average of 10 ms, the error rate also becomes zero.
- If the two aggressor rows are adjacent to the victim row, the number of errors is hundreds of thousands of times greater than if the two aggressors are far from the victim.
- The data pattern in memory is also relevant when inducing errors. If the aggressor rows are filled with 'v' (0 or 1) and the victim row is filled with '$\neg v$' (1 or 0), the amount of generated errors is ten times greater than in other patterns.
- The errors were also repetitive: if a cell presented a failure when taking from 0 to 1, or contrariwise, the chance to repeat the same fault is 70%.
- In some cases, it was shown that some cache lines presented at least four errors, making it impossible to correct them via Error-Correcting Code (ECC) only. The ECC is a process based on redundant data that can be retrieved, and is able to correct n out of m bits.

Abstraction barriers between different security systems have been extensively broken by devastating attacks using Rowhammer. Examples of attacks include privilege escalation from native environments [83, 34], privilege escalation from a JavaScript browser's sandbox [36], privilege escalation from a virtual machine running on third-party computation clouds [96], mounting fault attacks on signatures using RSA public-key cryptography [11, 76], and obtaining root privileges on Android mobile phones [89]. Due to the large range of critical attacks, there were many techniques to mitigate Rowhammer. Hardware countermeasures such as Error-Correction Code (ECC-RAM) [26] appeared to make Rowhammer attacks harder [54], but not infeasible [71, 59, 20].

This chapter is organized as follows. Section 4.1 details the Rowhammer types, like double-sided, single-sided, one-location, and many-sided. Then, Section 4.2 presents the steps of a Rowhammer attack: finding a vulnerable region, putting a target in a vulnerable region, and flippling a bit (or bits) from the target. Finally, Section 4.3 shows how Rowhammer countermeasures work in software and hardware.

4.1 Types of Rowhammer

Rowhammer has some variations techniques used to generate bit flips. The most common are double-sided, single-sided, one-location, and many-sided.

4.1.1 Double-sided Rowhammer

The double and single-sided Rowhammer types were conceived by Kim et al. [54]. Regardless, that nomination was proposed by Seaborn and Dullien from Google Project Zero [83]. The double-sided Rowhammer targets both rows around the victim's row, also known as the aggressor rows, which requires knowing the mapping between virtual and physical addresses.

In 2015, Seaborn and Dullien [83] observed that, in a system with two DDR3 with 8 banks each, one aggressor row has 256KB more physical addresses than the victim's row, and the other has 256KB less physical addresses than the victim's row. They notice that the number of bit flips increased considerably during an experiment, as already predicted by Kim et al. [54]. In 2016, Gruss et al. [72] used the DRAMA attack to show that the mapping of a physical address to the memory pins can be precisely known through a timing side-channel attack as the one in Figure 3.9, Section Subsection 3.2.3. Due to Gruss et al. [72], the double-sided Rowhammer attack became facilitated and relevant as the DRAMA attack helped test DDR4 memories, which were also found to be vulnerable to Rowhammer.

Listing 1 shows how to execute a double-sided Rowhammer after knowing the aggressors' addresses of a target row. To be more clear if the row of the target is A, the aggressors addresses X and Y need to be located at the row $A-1$ and $A+1$ respectively.

```
1  code1a:
2      mov (X), %eax  # read from address X
3      mov (Y), %ebx  # read from address Y
4      clflush (X)    # flush cache for address X
5      clflush (Y)    # flush cache for address Y
6      jmp code1a
```

Listing 1: Double-sided Rowhammer in x86 assembly code. Aggressor lines are loaded in X and Y. The instruction `mov` reads from the memory, and `clflush` removes data from the CPU cache.

Notice that the attack presented in Listing 1 only succeeds in Intel x86-64 processors since the `clflush` instruction presents different results in AMD processors. In both AMD and Intel processors, the instruction `clflush` removes the data from the L3 cache, but in Intel processors, the caches are inclusive. In other words, the L1 cache content is also present at the L2 content, and the L2 content is present in the L3 cache. On the other hand, AMD processors have exclusive caches, meaning that L1 content is not necessarily in L2 content, and L2 content is not necessarily in L3 content. In ARM processors, the equivalent instruction is the Master Control Reset (MCR). However, the userland cannot be accessed without permission. Figure 4.1a shows, at a high level, how the attack works in DRAM memories.

4.1.2 Single-sided Rowhammer

Because of the restrictions inherent to double-sided Rowhammer, Seaborn and Dullie [83] proposed the single-sided Rowhammer. Unlike the double-sided, the single-sided only needs that a single aggressor row is near to the victim's. By experimental observation, they realized that Rowhammer can be triggered using four or eight accesses to random addresses without slowing down the time per iteration. Thus, single-sided Rowhammer was proposed using four or eight random addresses.

Also, single-sided Rowhammer allows testing if at least two random addresses are in the same bank but in different rows. This is done by measuring the access time to two uncached addresses using a fine-grained timer such as the `rdtsc` instruction in x86 processors. Then, pairs that do not satisfy the above property have faster access time than those that satisfy.

Listing 2 shows how to execute a single-sided Rowhammer after knowing the aggressors' addresses in the variable vector `addr`. Similar to double-sided Rowhammer (Listing 1), the single-sided Rowhammer in Listing 2 only works in Intel x86-64 processors. For clearness, the code in Listing 2 is presented at a higher level than the double-sided Rowhammer. Also, Figure 4.1b shows how the attack performs in DRAM memories.

4.1.3 One-location Rowhammer

One-location Rowhammer was discovered by Gruss et al. [34], who observed that the memory controller on modern processors uses a policy for closing an opened row after an amount of time. This policy improves the processor's performance because the probability of two address accesses falling in the same row is tiny. In this sense, Gruss et al. used only one address to access the memory: as the memory controller closes the row, a new command for opening a row will be triggered, leading to a Rowhammer attack. In other words, bit flips occur because this single address will open and close a row in the same bank.

Gruss et al. [34] also observed that a single 4KB page could be tested since a single address is needed for the access. Thus, a bit flip inside of a 4KB page could be tested, and, unlike double-sided Rowhammer, the mapping between virtual and physical memories does not need to be known for a one-location Rowhammer. Considering the amount of

```
long int * addr[NUMBER_ADDR]; // NUMBER_ADDR := 4 or 8
...
while(1) {
    for(int i=0; i < NUMBER_ADDR; i++) {
        volatile long int * access = addr[i];
        *access;
        __asm__ volatile (
            "clflush (%0);"
            : /* out  registers */
            : /* in   registers */  "r"(access)
            : /* used registers */  "memory"
        );
    }
}
```

Listing 2: Single-sided Rowhammer in C and x86 inline assembly. The variable vector `addr` has 4 or 8 address pointers to some memory location. For each address in this variable, one access is done using a volatile variable to make the compiler execute the code without being removed by any optimization. Also, one `clflush` is executed to take the address from the cache.

bit flips in a page of 4KB, the one-location Rowhammer influences much fewer bits than other techniques. Particularly, one-location Rowhammer flips about 36% of the bits in a 4KB page. On the other hand, double-sided and single-sided Rowhammer flip more than 76% of the bits on the same page.

Listing 3 shows how to execute one-location Rowhammer after knowing the address of the aggressor rows in the variable `X`. Similarly to other Rowhammer types, the attack in Listing 3 only works in Intel x86-64 processors. Figure 4.1c shows, at a high level, how the attack works in DRAM memories.

```
code1b:
    mov (X), %eax  # read from address X
    clflush (X)    # flush cache for address X
    jmp code1a
```

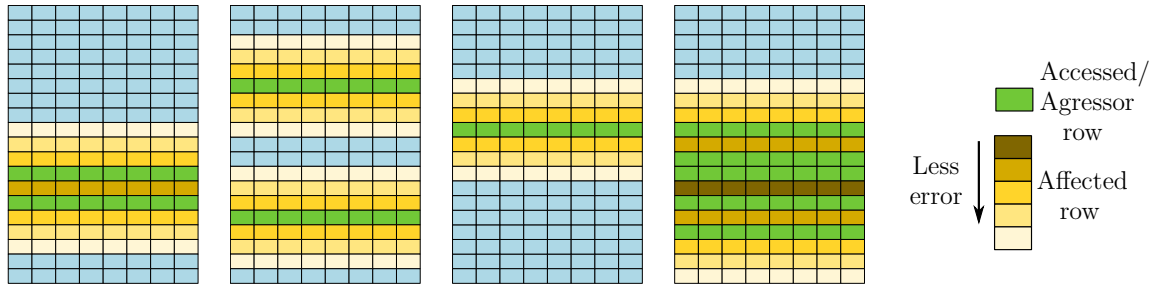
Listing 3: One-location Rowhammer in x86 assembly. The variable `X` has the address to some memory location known as the aggressor address. For each access in the address `X`, a `clflush` instruction is executed to take the address from the cache.

4.1.4 Many-sided Rowhammer

In 2020, Frigo and Vannacci et al. [29] defined a new category of Rowhammer, in which they explored the fact that bit flips in DDR4s can be done with less activation than

in DDR3s. They observed that DDR4s present bit flips with only 45K activations [29] instead of the 139K from the DDR3 [54]. In DDR4 memories, the shortest time between two activations is approximately 45ns, with a refresh rate of 64ms. Moreover, with 45K activations, a DDR4 can contain approximately 28 aggressor addresses accessing the same bank before some row is refreshed. In this sense, Frigo and Vannacci et al. created a technique similar to double-sided Rowhammer in which around 3 to 28 aggressor rows can be used in the same bank, bypassing some recent DDR4 countermeasures. Figure 4.1d shows, at a high level, how the attack works in DRAM memories.

Figure 4.1 summarizes the Rowhammer types described above. The attacker uses the accessed row through some access to an aggressor address. Also, the affected row indicates the possible location of the target row. Figure 4.1 was adapted from Gruss et al. work, describing one-location Rowhammer [34].



(a) Double-sided (b) Single-sided (c) One-location (d) Many-sided

Figure 4.1: The Rowhammer types: (a) two aggressors in the same bank and both adjacent to the victim's row, (b) at least two aggressors in the same bank and at least one adjacent to the victim's row, (c) just one aggressor in the same bank and preferably adjacent to the victim's row, and (d) many aggressors in the same bank and, as close as possible, to the victim's row.

4.2 Rowhammer Attack: Step-by-step

The Rowhammer attack consists of a few steps:

1. Find a vulnerable region:
   (a) Allocate memory;
   (b) Use a pointer for testing the allocated memory;
   (c) Check for vulnerable region;
   (d) Collect some memory information from the vulnerable region;
   (e) Release the vulnerable memory.
2. Put a target on the vulnerable region:
   (a) Force the target to go to the vulnerable region using some technique;

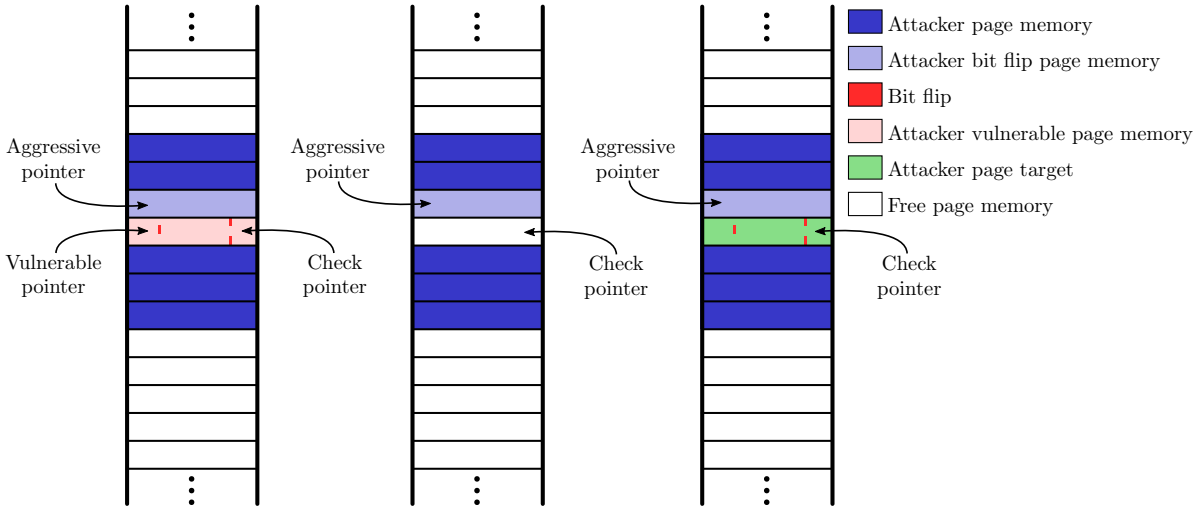
(b) Test if the target is in the vulnerable region.

3. Flip a bit (or bits) from the target:

   (a) Flip the desired bit (or bits);

   (b) Force the target to execute the changed part.

These steps have been developed since 2015 after the publication of the first successful attack [83]. Figure 4.2 shows, at a high level, the steps of a Rowhammer attack.



(a) Find vulnerable region (b) Put target on vulnerable region (c) Flip bits from the target

Figure 4.2: Step-by-step Rowhammer attack, passing from (a) finding a vulnerable region, (b) putting a target on the vulnerable region, and (c) flipping the bits from the target.

4.2.1 Finding a Vulnerable Region

Finding a vulnerable region is also known as Memory Massaging [15, 34, 89, 29]. This step is crucial for the attack's success, but it is not sufficient. The hardest part of the attack is not how to find the vulnerable region but how to hold information about that memory location. Finding the vulnerable region is not difficult, as the attacker can allocate several pages, carefully choose pointer addresses to be the aggressors, and finally test memory positions around the aggressors. Note that, at this step, the attacker controls both the aggressor's and the victim's regions using its memory. In the memory test phase, when the vulnerable location is found, the memory location needs to be stored somehow.

The first way to know the memory location was presented by Google's Project Zero [83], which consists in saving the target physical address by accessing the `/proc/PID/pagemap` file that holds the map from the virtual address to the physical address in the Linux operating system. This technique helps the attacker to save the memory address when it still belongs to him during the test phase. In this sense, a small amount of memory can be allocated for the attack, since the physical address is known and this help to bypass some defenses that check huge memory allocation in a short time. However, the Linux Kernel

solves this problem by limiting the access of non-privileged userspaces to the `pagemap`[1].

Consequently, a new technique should be designed to face the limitation imposed by the translation of virtual to physical addresses in Linux systems. Novel attacks arose, especially a prefetch side-channel attack by Gruss et al. [35]. Prefetch instructions are non-blocking memory loads that fill the cache with the requested addresses. In other words, a prefetch instruction gives a hint to the processor of an address that may be executed in the future [22]. Intel® states that prefetching addresses not mapped to physical pages can introduce non-deterministic performance penalties [22]. Thus, virtual addresses not belonging to the requesting process (for instance, a kernel virtual address) can be prefetched and used in timing side-channel attacks to recover an address that belongs to both the kernel and the process, breaking the kernel and user's memory isolation. Then, this attack allows holding information about some address since a virtual pointer inside the kernel with address X and a user space pointer with virtual address Y will point to the same physical memory address, as depicted in Figure 4.3.

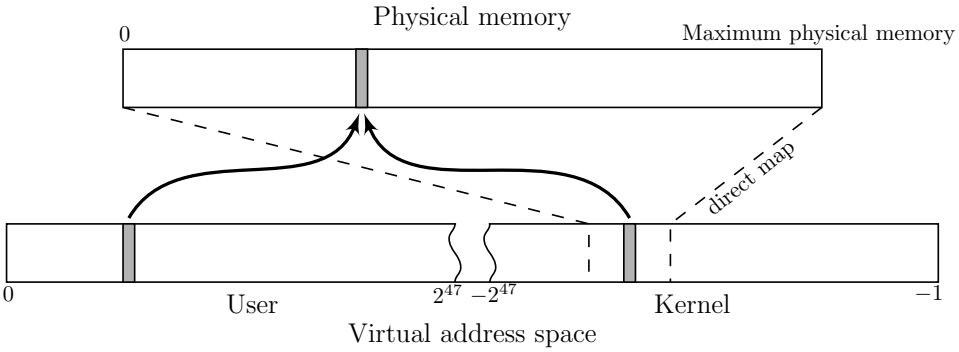


Figure 4.3: Mapping physical memory using Prefetch Attack [35].

The prefetch side-channel attack [35] helped the rise of new types of Rowhammer [34]. However, the issue of address space isolation was fixed by the Linux Kernel [33, 27], and the solution was named Kernel Address Isolation to have Side-channels Efficiently Removed (KAISER) by Gruss et al. [33] and Kernel Page-Table Isolation (KPTI) by the Linux Kernel [27] as shown in the Figure 4.4. Before the fix, adding a kernel base address and scanning a certain amount of addresses was enough for the attack's success. Now, the kernel pages are isolated from the pages of the user's process, and accessing some pages will only result in page fault without any gain of side information.

[1]Kernel commit: "pagemap: do not leak physical addresses to non-privileged userspace". Available at `https://git.kernel.org/pub/scm/linux/kernel/git/torvalds/linux.git/commit/?id=ab676b7d6fbf4b294bf198fb27ade5b0e865c7ce`.

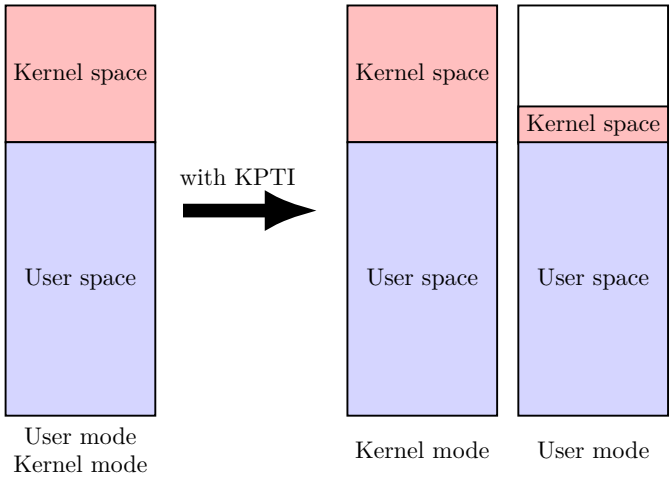


Figure 4.4: Kernel Page-Table Isolation in a high level overview in the actual Linux kernel. `CONFIG_PAGE_TABLE_ISOLATION=y` is used to enable this feature.

Nonetheless, address space isolation can be bypassed by allocating huge pages, with sizes like 2MB or 4MB, because huge pages are continuously allocated in memory, leaking the offset of the physical address. In particular, at least 21 and 22 bits are leaked in the case of 2MB and 4MB pages, respectively. The leaked offset, added with Address Mapping (Subsection 3.2.3), suffices to get a few 4KB pages within the same DRAM bank, allowing the collection of information from memory. On the other hand, if huge pages cannot be allocated, other techniques can be used as knowing the system allocator [89, 59]. Also, some information can be used to allocate memory blocks contiguously [89, 59], as several small blocks allocated in sequence can result in huge pages when combined.

4.2.2 Putting a Target on the Vulnerable Region

The first challenge that Rowhammer has faced so far is saving some information about the position of the vulnerable region. Thus, the second challenge becomes guiding the target to the vulnerable memory. Examples of techniques used to guide the target to the desired position are memory spraying [36, 84], memory grooming [89], page deduplication [15], memory page cache eviction [34], and Frame Feng Shui [59].

Memory Spraying

Memory spraying was the first public technique to guide the target to a specific memory [84, 36]. This technique works by first exhausting almost all the system's physical memory [98]. Then, it starts to replicate ("spraying") the target's memory page, expecting that a page to fall into the vulnerable region released for the target. For instance, the attacker can make many forks of the target.

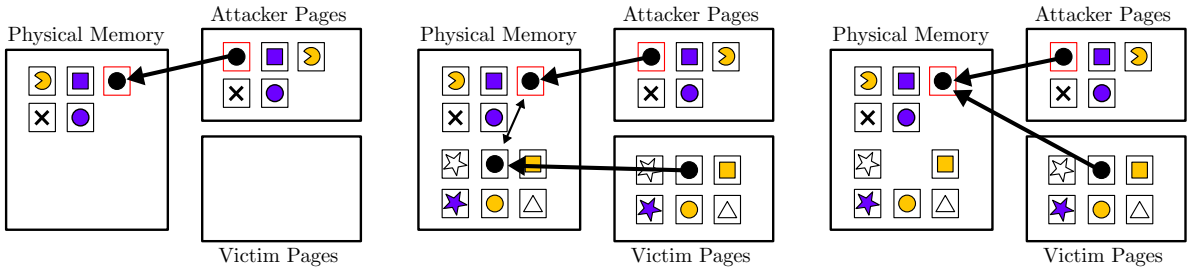
Memory Grooming

Similarly to memory spraying [89], memory grooming works by first exhausting almost all physical memory. Then, only the vulnerable region is released, with just enough memory

to load the target. When loading the target, it will get the vulnerable page as it is one of the few memory pages available.

Page Deduplication

Memory pages with the same content can be shared across independent processes to reduce the total memory footprint of a running system [15]. Memory deduplication is known as Kernel Samepage Merging (KSM) in Linux and as memory combining in Windows operating system [15]. Deduplication uses an algorithm that periodically merges memory pages with the same content into the same physical memory page. Figure 4.5 is an adaptation from [15] depicting the page deduplication technique.



(a) After massaging step (b) After victim load (c) After system merging

Figure 4.5: Page deduplication adapted from [15]. (a) Physical and virtual memory layout after the massaging step. The red square indicates the copy of the victim's page. (b) Load the victim's page in memory, each duplicated page points to different places in the physical memory. (c) After the system merging, the victim's and attacker's pages point to the same physical page.

Then, the attacker can put the same content from the target page on the vulnerable page, hoping that the system will merge the physical page with the vulnerable page. If this does not happen, the attack is restarted until the attacker gets the desired result. The deduplication algorithm merges pages in about 15 minutes, so this technique requires the attacker not to be in a hurry.

Page Cache Eviction

The operating system contains a cache page for the most used memory page [34]. For instance, an attacker can use a fork from the target until the desired target page gets in the vulnerable region, force the eviction from the cache page, and `unmmap` and `mmap` the binary target. Thus, when the target is executed, it will use the same physical page. Some information on the address resolution is required to know if the target falls into the vulnerable region. In the case of Gruss et al. [34], they used the prefetch side-channel attack [35].

Frame Feng Shui

Kwong et al. [59] developed the Frame Feng Shui technique exploring the Linux Buddy allocator operation. When a process requests memory, the Buddy allocator provides the

smallest available pages that meet the request. The smallest available blocks can be checked via `/proc/pagetypeinfo`. Then, the attack consists of requesting the smallest available blocks during the memory massaging step, releasing the vulnerable page (block) to the system, and immediately executing the target. Thus, the target will acquire the vulnerable region.

4.2.3 Flipping Bits From the Target

Unlike the previous steps, flipping bits from the target does not contain any challenges. The memory region was already confirmed as vulnerable in step one, and the target was guaranteed to be in the vulnerable region in step two. At this point, it only reuses the aggressor pointers and flips the bits from the target. As it is possible to observe at Figure 4.2 (c) where the attacker access the aggressive pointer several times until trigger the same bit flips pattern from the vulnerable region.

4.3 Countermeasures

The first Rowhammer attacks targeted DDR3 memories. Because of that, the DDR4 standard was built, and many thought the Rowhammer problem was solved. However, recent attacks showed that much still needs to be done to address this issue [59, 29]. In this sense, this section covers some software and hardware countermeasures against Rowhammer.

4.3.1 Software Countermeasures

Software solutions are much slower than hardware solutions but are malleable and much simpler to change than hardware solutions. This section lists some software countermeasures against Rowhammer, such as static analysis and memory encryption.

Static Analysis

One way of employing Rowhammer is to use specific statements like the `clflush` instruction from Intel. For this reason, a system can statically parse the binary code looking for the instructions used by Rowhammer [43]. However, this countermeasure is insufficient since some attacks do not adopt these instructions [34].

Hardware Performance Counters

Processors come with several performance counters, such as the cache miss counter. These counters can be used to prevent Rowhammer attacks [2]. However, this defense mechanism is insufficient to avoid Rowhammer, as shown by Gruss et al. [34].

Memory Access Pattern Analysis

Techniques like double-sided Rowhammer use a specific memory access pattern that can be identified and prevented [2]. However, Rowhammer attacks such as one-location still can bypass this countermeasure [34].

Physical Proximity Prevention

Still, one can completely isolate some memories. For instance, the Kernel and User memories can be placed as far as possible in the physical memory [16, 27]. However, this countermeasure only prevents the user from attacking kernel addresses. For example, the attacker can perform bit flips of binaries in the userland using the `sudo` command to elevate privilege [34].

Prevention of Near-out-of-memory Situations

Preventing the system from getting too close to exhausting all physical memory [36] can mitigate attack techniques like memory spraying and grooming, which are used to put the intended target in place. Other techniques for putting the target in place are page deduplication [15], page cache eviction [34], and Frame Feng Shui [59].

Memory Encryption

Another technique is to encrypt the memory that must be protected since a bit flip in this region can lead to a result entirely different from the expected by the attacker [59]. Nonetheless, a caveat is a latency of encrypting the entire memory region via software.

4.3.2 Hardware Countermeasures

Considering that a hardware issue generated the Rowhammer, one may also try to solve this problem in the hardware sphere by employing some of the countermeasures listed in the following.

Doubling Refresh Rate

One of the first hardware countermeasures was to double the refresh rate [2]. Instead of refreshing the entire memory at every 64ms, perform it at 32ms. However, that was not enough to prevent Rowhammer attacks [29].

Memory Encryption

Using hardware enclaves to encrypt memory spaces is also an alternative to prevent Rowhammer attacks. However, some works use this countermeasure to perform Denial of Service attacks on the system [34]. Flipping bits in the encrypted region will be seen as an integrity violation, leading the processor to stop working until it is manually reset.

ECC Memories

Memories with ECC are commonly used in servers and can detect two bit flips but correct only one. However, what happens if three bit flips are found when transmitting 72 bits to the processor? Cojocar et al. [20] demonstrated with reverse engineering the functioning of the bit correction system. Thus, by carefully choosing three bits to be flipped, they showed that one could bypass the ECC system. Indeed, Cojocar et al. [19] showed that Rowhammer could be used against a system using ECC memories.

Moreover, most works are only concerned with system integrity since flipping a bit would, in theory, ends with system integrity. Kwong et al. [59] showed that the latency resulted from the ECC system could be used to create a timing side-channel attack to break the system confidentiality, retrieving information in memory such as passwords, cryptography keys, and others.

TRR Memories

Memories with Target Row Refresh (TRR) were sold as the "silver bullet" [29] against Rowhammer attacks since target rows are refreshed with the highest priority. However, this countermeasure was also bypassed by Frigo et al. [29].

Chapter 5

ECDSA, secp256k1, and Bitcoin

A major problem with digital money is preventing it from being copied [30]. Traditionally, the coins used as money could not be easily counterfeited like the coins-commodities, such as cocoa, salt, barley, and other valuable objects. Metals such as gold and silver plate are rare enough not to be found on a large scale, thus preventing the currency from losing or inflating its value.

Clay tablets were used as tenure records, marking how much silver or grains one would receive from the government. The problem arose when the government had to prevent people from duplicating the record. The Chinese took advantage of material they had invented, paper, to start making receipts. Then, this paper money was getting more elaborate and fabricated with exclusive materials that could guarantee that no one could copy the currency.

Copying or fraud has always been a problem when managing financial transactions. In the digital age, controlling this type of copying is even more challenging because information about a transaction can be easily copied as music in digital format. For example, when using a credit card, there is a mediator, the card operator, to ensure a fund for the purchase and that no spend was duplicated. Thus, banks, card companies, and financial institutions act as arbiters for electronic transactions to ensure that there has been no fraud.

In 2008, banks continued to lend money mainly for unsustainable real-estate financing, leading to a significant drop in confidence in the banking systems. Therefore, a fall took the price of real estate and the economy to the ground. Along with this financial crisis, the American government spent considerable money to prevent more institutions from failing, generating the highest inflation in the country since the 90s. In this context of digital advancement, with the distribution of computing power and distrust of the financial system, Satoshi Nakamoto created Bitcoin.

In the article "Bitcoin: A Peer-to-Peer Electronic Cash System" [70], Nakamoto presented a system based on cryptography that does not need an intermediary like a financial institution and that is practically irreversible due to its mathematical nature. The mathematical framework is known as the blockchain, which audits all transactions on the Bitcoin network. Participants in the Bitcoin network become a node of the network and record all transactions, performing accounts and keeping the system running without a central system. In the Bitcoin network, every transaction uses the public-private key system to

prevent a transaction from duplicating. Thus, the receiver shares his public key, and the sender must sign the transaction with his private key. Bitcoin is the most popular cryptocurrency, and a single coin is currently estimated to be worth tens of thousands of dollars.

This chapter is organized as follows. Section 5.1 details the ECDSA protocol, showing invalid-curve attacks and introduces the curve `secp256k1`. Then, Section 5.2 presents the Bitcoin Core wallet with the implementation of countermeasures against fault injection attacks in the library `libsecp256k1` and a possible fault injection attack in the ECDSA protocol of the curve `secp256k1`.

5.1 ECDSA

In 1991, The Digital Signature Algorithm (DSA) was proposed by the U.S. National Institute of Standards and Technology (NIST) and specified in a U.S. Government Federal Information Processing Standard (FIPS 186) called Digital Signature Standard (DSS) [38]. The Elliptic Curve Digital Signature Algorithm (ECDSA) is the elliptic curve analog of the Digital Signature Algorithm (DSA). It is the most widely standardized elliptic curve-based signature scheme, appearing in the ANSI[1] X9.62, FIPS 186-2, IEEE 1363-2000, and ISO/IEC 15946-2 standards, as several draft standards [38]. The procedures for signature generation and verification are presented in Algorithm A.1 and Algorithm A.2, respectively.

The ECDSA protocol needs to handle random number generation to create keys and unique signatures. It also has to handle big numbers and modular arithmetic since the curve points are represented by several bits. Moreover, the protocol handles curve arithmetic, that is, point doubling ($2P = P + P$), and point addition ($3P = 2P + P$). The curve arithmetic is performed in a large prime field because the coordinates of the curve points belong to this field. The ECDSA support modules are summarized in Figure 5.1, adapted from [38].

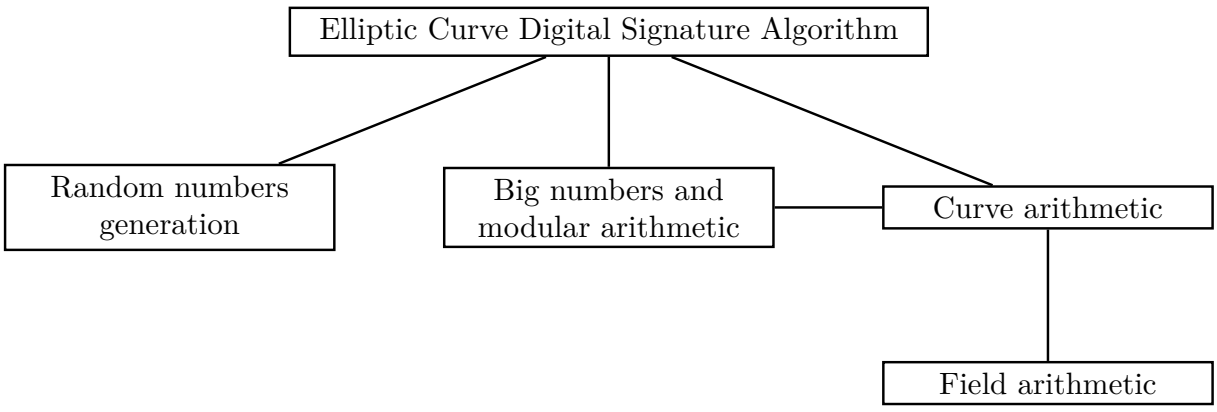


Figure 5.1: Module framework required by the Elliptic Curve Digital Signature Algorithm (ECDSA) protocol, adapted from [38].

[1] ANSI, the acronym for American National Standards Institute.

5.1.1 Invalid Curve Attacks

The main observation in invalid-curve attacks is that the usual formula for adding points on a short Weierstrass elliptic curve $E : y^2 = x^3 + ax + b$ defined over $\mathbb{F}_q$ does not involve the coefficient b (see Section 2.2.3) [38]. Thus, an attacker can replace G in Algorithm A.1, an elliptic curve point generating a subgroup with order n, by G', which does not belong to E, but rather to an unsafe curve E' carrying many small subgroups [8] (see Figure 2.6). Also, the curve E' contains an order $h'\ell'$ with many smaller subgroups ($h' \gg 1$), unlike the curve E of order $h\ell$ (with small h). Normally, G contains the order $n = \ell$, and G' is chosen to have order $n' \approx h'\ell' \ll \ell$. Thus, $r = x_1 \bmod n$ is delivered as $r = x'_1 \bmod n$, in which $P = (x'_1, y'_1) = kG'$. With enough invalid curves E', E'_2, $\cdots$, E'_m the attacker will recover k partially by solving m instances of the Elliptic Curve Discrete Logarithm Problem, to obtain $k \bmod n'$, $k \bmod n'_2$, $\cdots$, $k \bmod n'_m$. With the Chinese Remainder Theorem the attacker will recover $k \bmod n$, and thus the private key d_A.

Recall from Section 2.2.3, Short Weierstrass, that if all curves with the same equation E do not contain many small subgroups, changing G to G' will not help recover the private key. For example, consider that the group order of G' is prime and much larger than that of G. Then, it would be more productive to attack k directly using the point order of G, which is chosen to generate a high bit security.

Consider the situation that an attacker wants to change one of the public parameters inherent to the implementation. An approach is to inject faults in the victim's implementation. However, injecting faults in the implementation can open possibilities beyond attacking the parameters. The attacker can control the message by requesting multiple signs of different messages to the victim. The other parameters are the curve E, the generator point G, the order n, and the algorithm itself. The attacker could choose to change E, but, in this case, he would have to change many instructions that perform the calculations, such as the functions of point doubling and addition. Moreover, the algorithm can be changed by injecting faults in the instructions. For instance, the multiplication parameters can be changed using just a few bits, and secret parameters like the private key d_A could be leaked.

5.1.2 The Elliptic Curve secp256k1

The elliptic curve `secp256k1` is the curve adopted by Bitcoin to implement its public-key cryptography. All points on this curve are valid Bitcoin public keys. For generating the public key, a user multiplies the private key, a large number, by the generator point, a point defined in the `secp256k1` curve. Due to the Discrete Logarithm Problem, recovering the private key using the public key is a hard task using classical computers.

`secp256k1` is a short Weierstrass curve ($y^2 = x^3 + ax + b$). Specifically, $a = 0$ and $b = 7$, yielding the equation $y^2 = x^3 + 7$ [17]. The order of the generator point G is a large prime p. Since the y equation component is a square, `secp256k1` is symmetric across the x-axis, which means that, for each value of x, two values of y exist, one is even, and the other is odd. For instance, if $y = \pm 2 \bmod p$, then $y = 2 \bmod p$ (even) and $y = p - 2 \bmod p$ (odd). This fact allows public keys to be identified simply by the x-coordinate and the parity of the y-coordinate, saving significant data usage on the blockchain.

Particularly, this curve has a weakness that helps attackers to pursue invalid curve attacks, one of them being the controlled injection of faults. For instance, injecting a fault in the generator point G leads to a new curve equation that can be cryptographically weaker than the original curve. Toward the curve `secp256k1`, any fault injected in the generator point conducts to an isomorphic curve, which comes from the Short Weierstrass equation (Equation 2.2), in which $a = 0$ implies that any curve of the type $y^2 = x^3 + b$ will have the same j-invariant. Still, evaluating Theorem 2.1 results in six different curves, all others equivalent to these six. Table 5.1 shows all the six twists of `secp256k1`, and the `secp256k1` curve itself, over the prime field $\mathbb{F}_p$ with $p = 2^{256} - 2^{32} - 2^9 - 2^8 - 2^7 - 2^6 - 2^4 - 1$.

Table 5.1: Equations and order of the twists of `secp256k1`, and the `secp256k1` itself. All the orders are evaluated over the field $\mathbb{F}_p$ with p being the prime $p = 2^{256} - 2^{32} - 2^9 - 2^8 - 2^7 - 2^6 - 2^4 - 1$.

Curve Equation ($E : y^2 = x^3 + b$)	Order (n)
$E_1 : y^2 = x^3 + 1$	$2^2 \times 3 \times 20412485227 \times 83380711482738671590122559 \times 566938778783345283642190524432767265205 9$
$E_2 : y^2 = x^3 + 2$	$3^2 \times 13^2 \times 3319 \times 22639 \times 10131766773001318469008702396060356387381009972480920692566974370 31$
$E_3 : y^2 = x^3 + 3$	$109903 \times 12977017 \times 383229727 \times 211853322379233867315890044223858703031485253961775684523$
$E_4 : y^2 = x^3 + 4$	$3 \times 199 \times 18979 \times 5128356331187950431517 \times 19927510177695253241189007035359757442641709999 67$
$E_6 : y^2 = x^3 + 6$	$2^2 \times 7^2 \times 10903 \times 5290657 \times 10833080827 \times 22921299619447 \times 41245443549316649091297836755593555342121$
$E_7 : y^2 = x^3 + 7$	$11579208923731619542357098500868790785283756427907490438260516314151816149433 7$

5.2 Bitcoin Core Wallet

Bitcoin Core[2], formerly Bitcoin-Qt, is a Bitcoin client developed by Wladimir van der Laan based on the original reference code by Satoshi Nakamoto dated in 2009. The Bitcoin Core can be used as a desktop client for payments or as a server utility for merchants. Satoshi Nakamoto initially used Bitcoin-Qt, but the name has been rebranded to Bitcoin Core since version 0.9.0. Bitcoin Core uses the ECDSA protocol to sign a transaction. The elliptic curve used in the protocol is an implementation of the `secp256k1` in the library `libsecp256k1`[3].

5.2.1 Countermeasures Against Fault Injections

The `libsecp256k1` library implements several countermeasures that harden the introduction of fault injections using Rowhammer or similar fault attacks. These countermeasures are used to handle other types of side-channel attacks, such as timing attacks. Hence, implementations try to perform uniform accesses to the memory to prevent timing attacks, making it difficult to recover any private data.

Nonce Generation

A crucial aspect of ECDSA is the generation of nonces, which corresponds to the random number k in line 4 of Algorithm A.1. `libsecp256k1` solves this problem by relying on

[2]Bitcoin Core - Bitcoin Wiki `https://en.bitcoin.it/wiki/Bitcoin_Core`

[3]`libsecp256k1`'s GitHub repository: `https://github.com/bitcoin-core/secp256k1`.

the RFC6979 [74], which specifies how to generate derandomized nonces for ECDSA. The nonces are computed deterministically, given the message and the private key d as input to SHA256 HMAC [7], a pseudo-random function. The conventional approach of `libsecp256k1` is to recompute the nonce periodically, making it difficult for an attacker to generate several signatures with the same nonce. This countermeasure makes it difficult to keep the nonce as a constant, hampering the retrieval of several signatures generated with the same nonce and, thus, the recovery of the private key d.

Precomputed Table

Another countermeasure implemented by `libsecp256k1` is generating a precomputed table to evaluate the nonce multiplication. Thus, the signature generation algorithm does not need to evaluate several point additions. Equation 5.1 exemplifies a precomputed table for nonce multiplication. Notice that, after the addition at each line, the point P tends to disappear, leaving only a $k \cdot G$ term. In Equation 5.1, G is the generator point, and P is a fixed point $P = (x_1, y_1) \leftarrow k \cdot G$ for k a random number in the interval $[1, n-1]$. Thus, by construction, this fixed point does not allow the derivation of the scalar k used to produce the fixed point P.

$$\begin{bmatrix} P, & P+G, & ..., & P+15\cdot G \\ 2\cdot P, & 2\cdot P+16\cdot G, & ..., & 2\cdot P+31\cdot G \\ & & \vdots & \\ (2^{62})\cdot P, & (2^{62})\cdot P+(16^{62})\cdot G, & ..., & (2^{62})\cdot P+(16^{62}+15)\cdot G \\ (1-2^{63})\cdot P, & (1-16^{63})\cdot P+(16^{63})\cdot G, & ..., & (1-2^{63})\cdot P+(16^{63}+15)\cdot G \end{bmatrix} \tag{5.1}$$

This countermeasure is enabled by default in Bitcoin Core using `libsecp256k1`, hampering fault injections in just one point, forcing the attacker to inject failures throughout the table to succeed. If this table is disabled by default, a fault injection attack still needs to conduce the point P to infinity to avoid affecting the point G' with the fault, which is possible by inverting a single bit from P.

Blinding Point

The blinding point [21] is a scalar number used to prevent timing side-channel attacks. Instead of evaluating $k \cdot G$ for calculating the signature, use the computation $(k-b) \cdot G + b \cdot G$. This type of computation helps to avoid the leakage of any private data in the array of indexes. The Bitcoin Core wallet saves the result of the $b \cdot G$ computation to make it easier to compute only $(k-b) \cdot G$. Thus, it would be necessary to modify $b \cdot G$ to inject a fault, moving this point to the desired fake curve. A fault can also be injected by preventing the blind computation from happening, which could be done by avoiding the function call responsible for using the blind computation.

5.2.2 A Fault Injection in secp256k1

All the countermeasures in a `secp256k1` implementation (Subsection 5.2.1) makes it harder to perform an invalid curve attack. However, supposing that each of the countermeasures mentioned above can be circumvented, the protocol itself still needs to be bypassed. For instance, flipping the y-coordinate first bit from `secp256k1`'s generator G will guide the curve E_7 to E_3, as shown in Table 5.1. Moreover, notice that flipping any bit of the `secp256k1` generator guides the curve E_7 to one of those six twists listed in Table 5.1.

From the attacker's perspective, faults can be injected into the generator in a non-controlled way, and luckily the fault will guide the original curve to one of its five insecure twists. For verifying that the point is in one of the insecure twist curves, the attacker can receive the malformed signature and check if the point is in one of the curves in Table 5.1. On the other hand, if the attacker can control the fault injection, the fault can be injected in the curve order n, which is a value used during the signature generation for modular reduction when computing the values of r (line 6) and s (line 10) in Algorithm A.1. For instance, flippling the 237th bit of the original n (Table 5.1) generates the curve order

$$\begin{aligned} n = 5 \times 19 \times 937 \times 55409197 \times 26698940729717 \times \\ 16417030323195811079 \times 53560469775481898997442373619121, \end{aligned}$$

which can leak some bits of the private key. For instance this will allow the attacker to obtain r, Algorithm A.1 line 6, by small factors modulus, such as, $r = x \bmod 5$, $r = x \bmod 19$ and so on.

In Algorithm 5.1 we propose a procedure for recovering a private key d by injecting faults in the original generator $G = (G_x, G_y)$. For instance, flipping the 6th, 5th, 1st, 3rd, and 24th bits from the G_y coordinate generates a point in the curves E_1, E_2, E_3, E_4, and E_6 respectively. Then, the private key d can be recovered by collecting all forged signatures and the original signature using the E_7 curve. For each x-coordinate of r in the signature (r, s), four possible points can be generated, as described from line 10 to line 12 in Algorithm 5.1. This occurs because each square root can generate an odd and an even point. Also, the reduction $r_i = x_i \bmod n_7$ can generate a different value for $r_i = x_i \bmod n_i$ for all i in $\{1, 2, 3, 4, 6\}$.

The curve $E_1 : y_1^2 = x_1^3 + 1$ is the only one with an order smaller than that of $E_7 : y_7^2 = x_7^3 + 7$. All other twists have order n_i greater than n_7, the order of the E_7 curve. When performing the attack and jumping to a curve E_i ($i \neq 7$), the result will be delivered to the attacker in the form of $x_i \bmod n_7$. However, x_i is in the E_i curve. So $x_i \bmod n_7$ can be different from $x_i \bmod n_i$, which can lead to a failure when trying to retrieve the integer square root $y_i = (x_i^3 + i)^{(1/2)}$. In this case, one can use the value of $x_i' = x_i + n_7$ that contains a square root. Because, except for n_1 all values respect the interval $n_7 < n_i < 2n_7$

Algorithm 5.1 ECDSA key recovering algorithm

Require:

E_i: twist elliptic curve $E_i : y^2 = x^3 + i$, $\forall i \in \{1, 2, 3, 4, 6, 7\}$,
G_i: generator of each twist curve E_i,
n_i: order of each curve E_i as in Table 5.1,
h: truncated hash used for computing the signature,
(r_i, s_i): signatures in all twist curves, $\forall i \in \{1, 2, 3, 4, 6, 7\}$.

procedure ECDSA_RECOVERY($\{E_i, G_i, n_i, (r_i, s_i), \forall i \in \{1, 2, 3, 4, 6, 7\}\}, h$)
  $m_1 \leftarrow \{20412485227\}$ ▷ Small cofactors from unsafe curve orders
  $m_2 \leftarrow \{3319, 22639\}$
  $m_3 \leftarrow \{109903, 12977017, 383229727\}$
  $m_4 \leftarrow \{199, 18979\}$
  $m_6 \leftarrow \{10903, 5290657, 10833080827, 22921299619447\}$
  $m \leftarrow m_1 \cup m_2 \cup m_3 \cup m_4 \cup m_6$
  for $\forall i \in \{1, 2, 3, 4, 6\}$ **do**
    $Ps \leftarrow \Big(x = r_i, y = (r_i^3 + i)^{\frac{1}{2}} \text{ odd}\Big) \cup$
      $\Big(x = r_i, y = (r_i^3 + i)^{\frac{1}{2}} \text{ even}\Big) \cup$
      $\Big(x = r_i + n_7, y = ((r_i + n_7)^3 + i)^{\frac{1}{2}} \text{ odd}\Big) \cup$
      $\Big(x = r_i + n_7, y = ((r_i + n_7)^3 + i)^{\frac{1}{2}} \text{ even}\Big)$
    for $P \in Ps$ **do**
      $f_i \leftarrow \emptyset$
      for $h \in m_i$ **do**
        $f_i \leftarrow f_i \cup (n_i/h) \cdot G_i$ discrete log with $(n_i/h) \cdot P$
      end for
    end for
  end for
  for $\forall v = \{v_1, v_2, v_3, v_4, v_6\}, \forall v_i \in f_i$ **do**
    $k' \leftarrow \text{CRT}(v, m)$
    $P' = (x'_7, y'_7) \leftarrow k' \cdot G_7$
    if $x'_7 = x_7$ **then**
      $k \leftarrow k'$
    end if
  end for
  $d \leftarrow s_7 \cdot k \bmod n_7$
  $d \leftarrow (d - h) \bmod n_7$
  $d \leftarrow d \cdot x_7^{-1} \bmod n_7$
  return d
end procedure

This attack is hard to conduct in practice, so for demonstration purposes a wrapper was built around only the `libsecp256k1` code with all countermeasures from Subsection 5.2.1 disabled. The attack consists in injecting a fault in the curve generator, one for each

twist curve, for collecting five corrupted signatures and one valid signature to employ Algorithm 5.1. This was sufficient to recover the private signing key used in the victim's wallet.

5.2.3 A Fault Injection in ECDSA

This proposed attack does not affect only the `libsecp256k1` library but also the ECDSA protocol. This attack considers that instructions can be changed through bit flips. Instead of evaluating s as $k^{-1}(h+rd)$ mod n, a fault injection can lead this computation to another that facilitates the recovery of the private key d. At some point, the implementation needs to multiply k^{-1} and $(h+rd)$ to obtain s. However, the call to the multiplication function can be changed, so it computes the multiplication of $(h+rd)$ and $(h+rd)$. Then, the value of s will be $(h+rd)^2$, allowing a direct recovery of the private key d. In an x86-64 platform, changing just a `mov` instruction can lead to the desired evaluation. For instance, given a version of the Bitcoin Core wallet, it suffices to change the instruction `4c 89 fe (mov rsi, r15)` by only three bits to `48 89 d6 (mov rsi, rdx)` as can be seen in Figure 5.2. In Table 5.2 representing the x86-64 calling convention it is possible verify why the changing from `r15` to `rdx`, since the `multiply_function(output, num1, num2)` uses the `rdi` for `output` parameter, `rsi` and `rdx` for `num1` and `num2` parameters respectively, where `multiply_function` makes `output` = `num1` × `num2`.

Table 5.2: x86-64 calling conventions. Conventions primarily intended for C/C++ compilers. Other languages may use other formats and conventions in their implementations. In case of more parameters than registers the remaining are saved on the stack.

Systems	Register Parameters
Windows	rcx, rdx, r8, r9
Linux, macOS	rdi, rsi, rdx, rcx, r8, r9

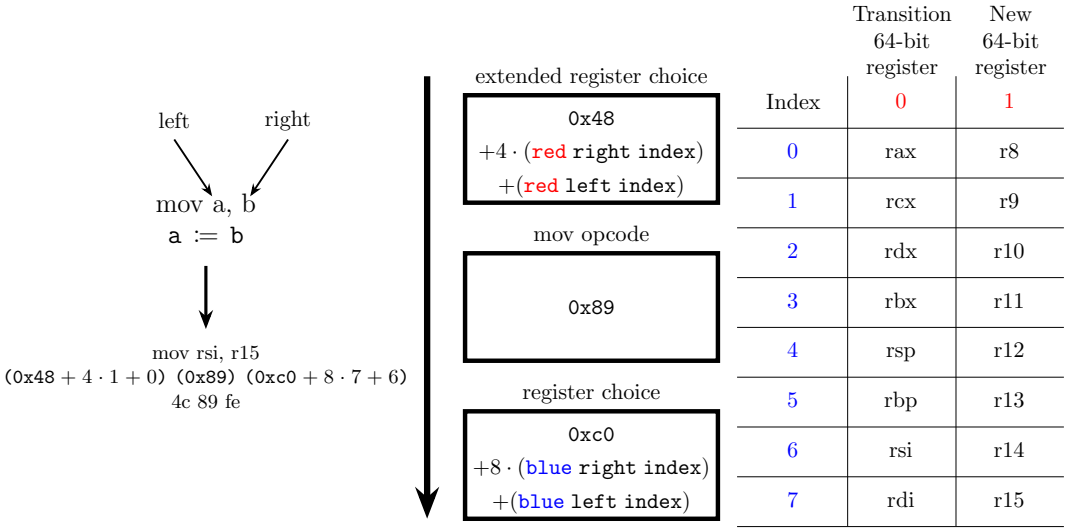


Figure 5.2: x86-64 mov operation. For 32-bits system all transition register replace letter 'r' by letter 'e' (eax, ecx, ...) and keep register choice field.

This technique can be applied not only against the `secp256k1` curve, but also in any curve that can be used with the ECDSA protocol. Given that this technique targets line 10 in Algorithm A.1, the attacker just needs to find the call of the multiplication of $k^{-1}(h + rd) \bmod n$ in target binary, and introduce faults to changes to generate $(h + rd)^2 \bmod n$. The private key can then be recovered directly by solving for d.

Chapter 6

DRAM Aging

Accelerated aging is a general technique for product testing to help determine the long-term effects of expected stress levels within a shorter time [77, 81]. A test may use aggravated conditions of heat, humidity, oxygen, sunlight, vibration, and others to speed up the normal aging processes of items. Accelerated aging tests are done on products that have not existed long enough, such as a new car engine or battery, to predict an approximated lifespan since the actual lifespan data is unavailable.

A prominent example of accelerated aging occurred during land grabbing in Brazil when large-scale land acquisitions were made. In Portuguese, land grabbing is translated as "grilagem" [80], meaning the "cricket way". In Brazil, pieces of land were distributed to beneficiaries in the name of Portugal's king. However, after a 1850 law, the lands started to be registered, and the old sheets of paper were considered as documents delivered directly by the Portuguese king. Thus, some people made fake copies of these old papers and put them in a drawer full of crickets. This way, a brand-new sheet of paper became artificially aged, appearing as an original old document.

The accelerated aging of paper sheets is also applied in libraries and archives. In this context, the paper is subjected to elevated temperature, concentrated pollutants, and intense light [73]. These tests try to predict the long-term effects of particular conservation treatments, the fundamental processes of paper decay, and to predict the lifespan of a particular paper type.

Accelerated aging does not have a recommended set of guidelines for its execution. In fact, in semiconductors, it is possible to use temperature variations, voltage changing, relative humidity ranging from 1% to 100%, and timing varying from hours to days.

Chapter 4 presented how Rowhammer can harm DRAM memories. This chapter starts a long yet unfinished research project that seeks to answer whether DRAM aging leads to a memory vulnerable to Rowhammer. In other words, this chapter seeks to answer if a brand new memory, which does not present a Rowhammer problem, will start presenting issues after a while, which can be months or even years.

This chapter is organized as follows. Section 6.1 details how some types of memory aging are measured. Then, Section 6.2 illustrates the setup adopted in this work to age memories, including the machine configuration, memory type, and how to produce heat. Finally, Section 6.3 presents our initial results from memory aging, such as temperature thresholds for the occurrence of failures.

6.1 DRAM Aging Evaluation

Conceiving a mathematical model for complex devices like DRAM memory is challenging, even having several mathematical models for the individual components [77, 81].

Some works claim to use high temperatures to accelerate the aging of DRAMs [87, 86]. Tehranipoor et al. [86] started the argument that memories can present problems with aging. In the first part of their paper, they aimed to evaluate stable bits in the nominal operation of the DRAM memory – nominal voltage and 25°C – and measure the charge retention factor on top of these bits during the time. Also, half of the bits were assigned to one and the other half to zero. As a result, about 64% to 81% of the charges remained stable, and those bits were counted stable. After that, they started to use a high (80°C) and a low temperature (0°C) to check how many of those bits would remain stable compared with the nominal operation. In both scenarios, there was a significant drop in the number of stable bits. For the low temperature, it was shown that about 50% of bits start to lose their stability. The experiment was repeated, varying the nominal voltage by ±10%. Similarly, the low voltage showed that more than 50% of bits lose their stability. Therefore, the voltage showed to be more aggressive when related to the bits' stability.

Wang et al. [93] discussed the Very Large Scale Integrated (VLSI) phenomena with Bias Temperature Instability (BTI). In this case, aging was evaluated using high temperature and voltage, and this combination resulted in a higher threshold voltage for charge retention while decreasing the speed of transistors. They also created a circuit called Radic that works with different clock frequencies. Thus, the aging rate is calculated based on the difference between the clocks. It seems that Tehranipoor et al. [86] used this method to claim that 8 hours of high temperature (80°C) generates an effect of 6 months of aging.

Table 6.1 summarizes the results of Tehranipoor et al. [86]. This table shows that aging also affects stable bits that may indicate unusual access, such as those generated by Rowhammer, that leads to bit flips errors. As the process technology scales further, aging in integrated circuits becomes a major challenge [93], causing an increased mismatch between modeled and actual silicon behavior. However, only a few works discuss this mismatch in behavior and its implications.

Table 6.1: Stable bits under different operating conditions. Adapted from Tehranipoor et al. [86]. The first column shows different conditions, and the remaining columns show three types of vendors.

Condition		**% of stable bits**		
Type	**Operation**	**DRAM 1**	**DRAM 2**	**DRAM 3**
Nominal Operation	1.50V, 25°C, 0 year	81.4%	65.5%	64.3%
High Temperature	1.50V, 80°C, 0 year	78.8%	64.4%	49.8%
Low Temperature	1.50V, 0°C, 0 year	49.9%	54.4%	43.3%
High Voltage	1.65V, 25°C, 0 year	55.4%	54.3%	30.5%
Low Voltage	1.35V, 25°C, 0 year	43.3%	36.7%	23.8%
Aged	1.50V, 25°C, 1 year	71.0%	65.0%	55.1%

6.2 Methodology Setup

This section discusses the setup adopted for DRAM aging and testing, detailing each step of the proposed experiment.

6.2.1 Experimental Setup

This study focused on a low-cost setup in a Desktop instead of a server, so no Xeon processor was used. At first, the chosen processor was an Intel Kaby Lake i5-7400, with four cores running at 3GHz, with a memory known to be vulnerable to Rowhammer. However, even after several hours of running Rowhammer at a high temperature, no flipped bits were found. So, the processor was replaced by a more powerful one, an Intel Skylake i7-6700k, with four cores running at 4GHz. Besides the higher frequency, it is also 5% faster in the memory access latency and contains an overall effective speed 14% greater than Kaby Lake. The performance comparison between both processors can be see at Figure 6.1.

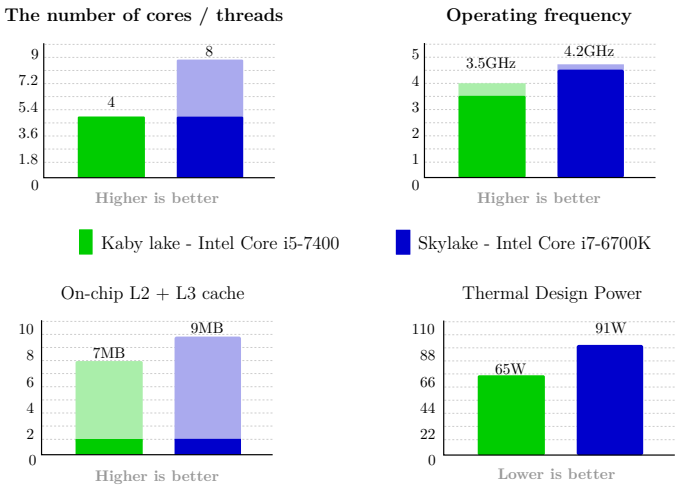


Figure 6.1: Performance comparison between the Kaby Lake i5-7400 and Skylake i7-6700k. Some processors do not have enough power to trigger Rowhammer, in this case we were able to flip bits only in the Skylake i7-6700k.

Then, the DRAMA attack [72] was used to find the memory mapping for the following main purposes:

1. Use an access pattern that would increase the number of row misses, inducing some types of damage to memory components such as the Row Buffer;
2. Find bit flips via Rowhammer with high precision;
3. Change the `memtest` instruction so it would run for several hours in a loop, performing access patterns through all memory.

The focus was in control voltage and temperature as showed in Tehranipoor et al. [86]. So the motherboard was used to control voltage changing and was used an external circuit for heating control. Due to a large number of timing parameters to control the memory operation it was decided not address frequency changing.

Brand new memories were used for the experiment due to some difficulties, such as only a few DDR4 memories at beginning of 2016 were known to be vulnerable to Rowhammer. With artificial aging would it be possible to compare in a short time how much worse it the memory state became with respect to Rowhammer.

6.2.2 Heating Setup

This experiment focuses on creating a low-cost but efficient setup for heating. First, the memory cases were constructed using Peltier plates and a 3D printer. In this first setup, it was also expected to observe the impacts of cooling the memory. However, it was not possible to reach high temperatures since it was not possible to create a 3D case with sufficient thermal isolation, due the amount of air between case with Peltier and the memory.

Because of that, the initial idea of cooling was discarded. Then, a molded copper plate was used to surround a memory DIMM and a hair straightener resistance was used in contact with the copper plate. Thermal isolation was done outside with a fiber thermal blanket, a plate of aluminum, and Kapton tape for fixation. The resistance's current flow was controlled using a dimmer, and the complete setup was controlled via a thermostatic. The final setup can be seen in Figure 6.2 and a block diagram can be seen in Figure 6.3.

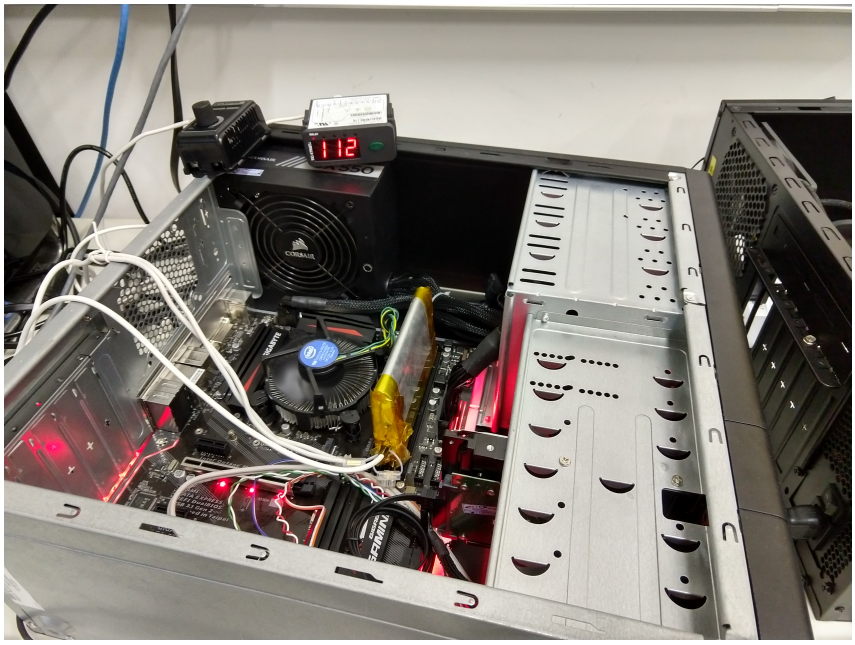


Figure 6.2: DRAM aging setup using a plate of copper in contact with the memory, a hair straightener resistance in contact with the plate of copper, a thermal sensor in contact with the plate of copper, a fiber thermal blanket in contact with the plate of copper, the resistance and the aluminum plate. Finally, a Kapton tape was used to keep all parts connected and isolated. A dimmer was adopted to control the resistance's current flow, and a thermostatic was used to control the temperature.

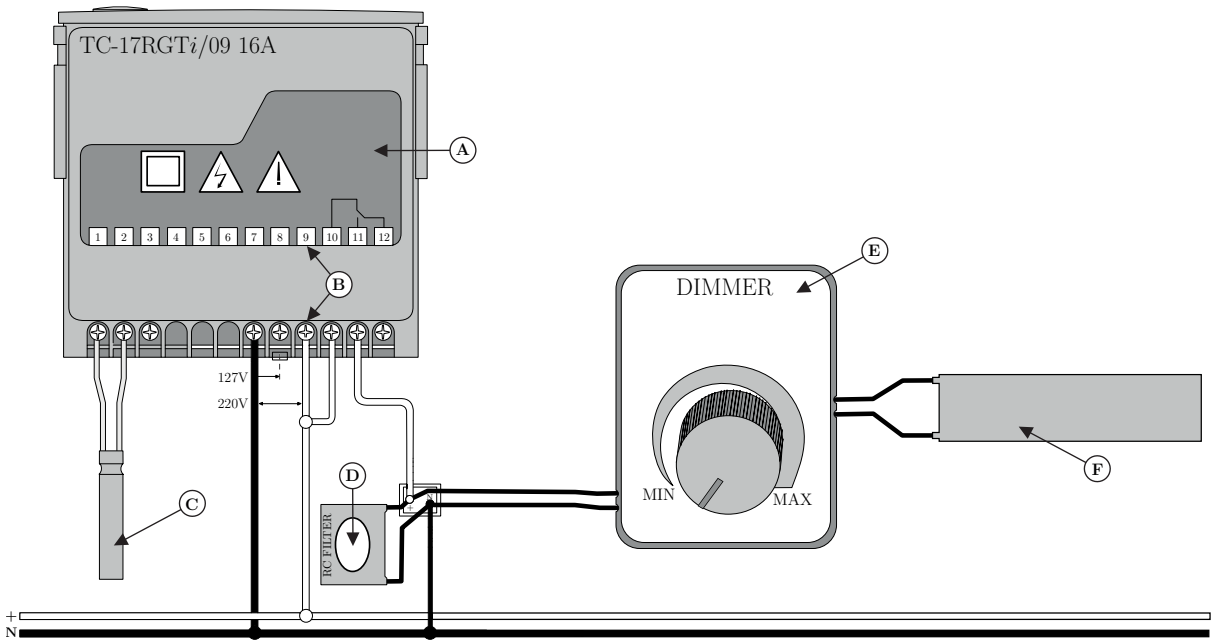


Figure 6.3: Diagram setup used to heat memories. (A) digital thermostat Full Gauge TIC-17RGTi, (B) thermostat terminals from 1 to 12, (C) thermistor sb59 sensor from -50 to $200°C$, (D) RC 5W suppressor filter, (E) dimmer for power control, and (F) heat resistance from a hair straightener.

6.3 Preliminary Results

Several preliminary tests were performed using the setup presented above, some of them to answer how long it takes for a memory to stop working under extreme conditions. At first, the following patterns of reading and writing were applied:

`0xaaaaa...aa` and `0x55555...55`.

These patterns consist of the intercalated bit sequences `1010` and `0101`, respectively. Thus, a zero bit is surrounded by ones, or contrariwise, increasing the chance of a bit being flipped.

A 4GB DDR4 memory using 2400MHz and 1.2V for nominal operation and 125°C was chosen for the first experiment. These two patterns were read and written in the computer setup, presented in the Figure 6.2, at every 7 seconds. In this context, one can argue whether the setup memory could be burned by using a temperature close to 125°C during a certain time. This first test observed if a reading or writing error occurred when running this memory under 125°C for over 800 hours. As a result, the memory kept working well, and no problem was detected using `memtest` for memory testing. After that, some types of failures were observed by changing the nominal voltage. For instance, the nominal operation of 1.2V, 2400MHz, and 25°C was adjusted to a configuration close to 1.65V, 2700MHz, and 125°C. The frequency only helped to perform more reading and writing in the same amount of time, but, excluding this test, the default configuration was the nominal frequency.

In a second test with a brand new 4GB DDR4 2400MHz 1.2V memory, the goal was to search for operating ranges using the same reading/writing pattern above. For operations in the setting 1.4V and 127°C, the memory started to present errors under 3 minutes. However, adopting the 1.4V and 100°C configuration, the same memory did not present any error also under 3 minutes. On the other hand, the computer could not be turned on with 1.4V and 155°C. Due to the motherboard limitation, only some points could be measured. Figure 6.4 exemplifies this setup, and the graph can be interpreted as follows:

- The black curve is the lower bound, where the memory starts presenting errors. The curve was extrapolated in regions where points could not be collected.
- The red curve defines the upper bound in which the memory stops working; that is, the computer can no longer be turned on.
- The red and black points represented the collected data.

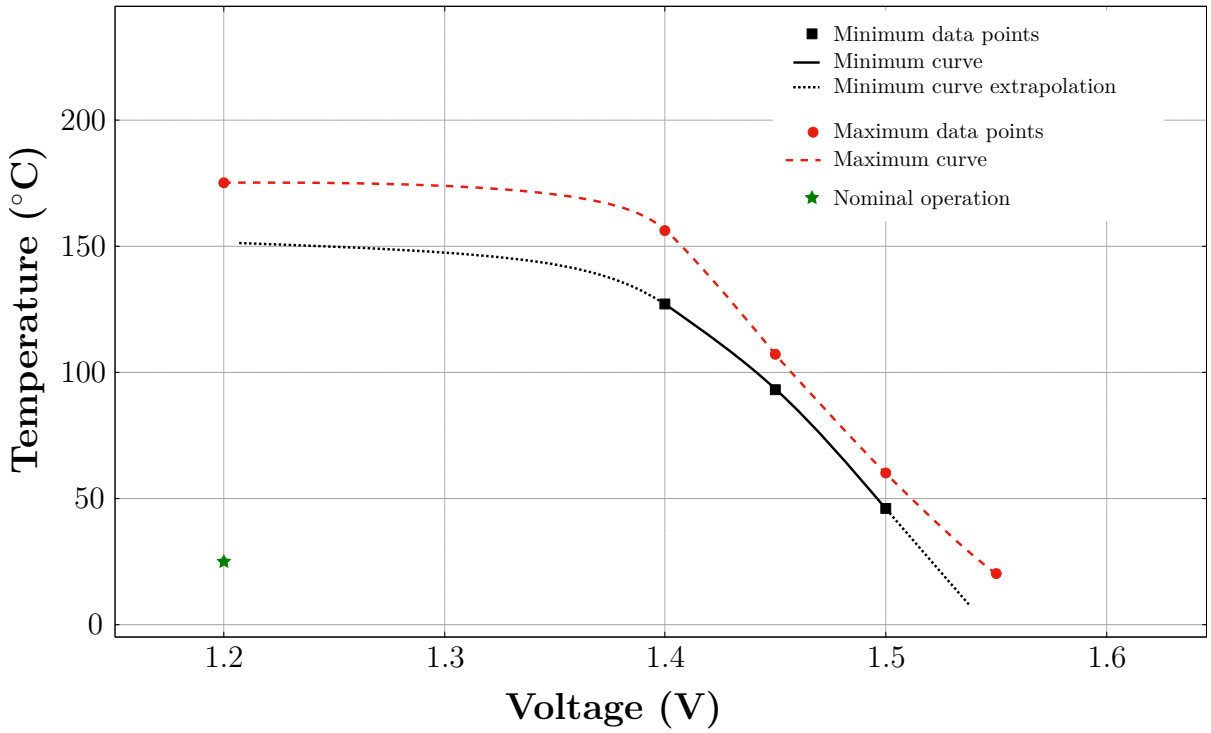


Figure 6.4: Operation of a 4GB DDR4 2400MHz 1.2V memory. Below the black curve, the memory works normally. The memory presents read and write errors between the black and red curves. After the red curve, the memory does not work. The points are the collected data, and the black and red dot line are the curve extrapolation.

Another focus of attention was the temperature at 175°C, in which the memory got damaged on cooling and stopped working, implying that the red curve is an extreme case. In summary, Figure 6.4 shows that the memory works perfectly below the black curve, the memory presents read and write errors between the black and red curves, and the memory does not work above the red curve.

Chapter 7

Conclusions

This work provided a survey on DRAM memory working, including the DRAM architecture with design and construction and the memory controller responsible for managing all operations over DRAM. In addition, this work showed how challenging it is to keep DRAM security at a higher level due to its high complexity and the need for new memory-building technologies to speed up access and security. Finally, it showed the significant academic effort to reverse engineering the DRAM details, such as the memory controller mapping. Notice that some memory manufacturers are unwilling to publish architectural details, trying to protect the technology involved or using security through obscurity, which relies on the idea that a system can be safe if vulnerabilities are secret or hidden.

This work also surveyed Rowhammer, detailing its types, how to execute the attack step-by-step, and some countermeasures. The survey showed how powerful Rowhammer is as a software fault injection attack. Since Rowhammer still works on modern DRAM memories, it implies that new memory-building technologies need to be developed to improve access time and security. Finally, the survey presented how the dispute attackers against defenders works, showing the countermeasures' evolution.

After that, this work focused on fault injection using Rowhammer in the Bitcoin Core wallet, the Elliptic Curve Digital Signature Algorithm (ECDSA) protocol, and the elliptic curve `secp256k1` implementation. This approach presented how invalid curve attacks can be implemented, especially how the `libsecp256k1` library can be subverted to leak the private key d by using twist curves. On the other hand, it also exposes how fragile the ECDSA protocol can be if the attacker injects faults in certain computations.

Finally, this work initiated research on DRAM aging and its effects on attacks such as Rowhammer. The primary goal of this research was to claim that, after some time, the memory will start to present vulnerabilities. DRAM aging uses extreme operation conditions, such as high temperature and voltage.

7.1 Future Work

On the Rowhammer front, new variants can still be researched, keeping the push for new hardware updates. The attack variants now needs to bypass hardware countermeasures, such as the attack focusing on the Target Row Refresh (TRR), which is one of the features

in DDR4 memories, or the attack focusing on Error-Correcting Code (ECC) memories. On the other hand the attack may also become stronger with the arrival of 3D memory printing, where layers of memory arrays begin to stack, meaning that the aggressor line also interacts with lines in other arrays.

In the ECDSA protocol, the recommendation is to add more countermeasures against fault injection attacks in real-world implementations like the Bitcoin Core wallet. In particular, high-level multiplications performed by the protocol must be protected, as it was proved to be highly vulnerable to Rowhammer. For instance instead of evaluate $s = k^{-1}(h+rd_A) \bmod n$ change this computation for a masked one like $s = s_1+s_2+\cdots+s_n$.

Finally, the research on DRAM memory aging can be continued by analyzing more evidence on aging effects. The research still requires the creation of new patterns for the Rowhammer test and a better strategy for testing the DRAM memory aging.

Bibliography

[1] Karim M. Abdellatif and Olivier Hériveaux. Silicontoaster: A cheap and programmable em injector for extracting secrets. Cryptology ePrint Archive, Paper 2020/1115, 2020. https://eprint.iacr.org/2020/1115.

[2] Zelalem Birhanu Aweke, Salessawi Ferede Yitbarek, Rui Qiao, Reetuparna Das, Matthew Hicks, Yossi Oren, and Todd Austin. Anvil: Software-based protection against next-generation rowhammer attacks. In *Proceedings of the Twenty-First International Conference on Architectural Support for Programming Languages and Operating Systems*, ASPLOS '16, page 743–755, New York, NY, USA, 2016. Association for Computing Machinery.

[3] Josep Balasch, Benedikt Gierlichs, and Ingrid Verbauwhede. An In-depth and Black-box Characterization of the Effects of Clock Glitches on 8-bit MCUs. In *2011 Workshop on Fault Diagnosis and Tolerance in Cryptography, FDTC 2011, Tokyo, Japan, September 29, 2011*, pages 105–114, 2011.

[4] Josep Balasch, Benedikt Gierlichs, Roel Verdult, Lejla Batina, and Ingrid Verbauwhede. *Power Analysis of Atmel CryptoMemory – Recovering Keys from Secure EEPROMs*, pages 19–34. Springer Berlin Heidelberg, Berlin, Heidelberg, 2012.

[5] James Ball, Julian Borger, and Glenn Greenwald. Revealed: how US and UK spy agencies defeat internet privacy and security, 2013. https://www.theguardian.com/world/2013/sep/05/nsa-gchq-encryption-codes-security.

[6] Elaine Barker. Recommendation for key management, part 1: General, 2016-01-28 2016.

[7] Elaine Barker and John Kelsey. Recommendation for random number generation using deterministic random bit generators, 2015-06-24 2015.

[8] Daniel J. Bernstein and Tanja Lange. SafeCurves: choosing safe curves for elliptic-curve cryptography, 2014. https://safecurves.cr.yp.to/twist.html.

[9] Daniel J. Bernstein and Tanja Lange. Montgomery curves and the montgomery ladder. Cryptology ePrint Archive, Report 2017/293, 2017. https://ia.cr/2017/293.

[10] Sarani Bhattacharya and Debdeep Mukhopadhyay. *Curious Case of Rowhammer: Flipping Secret Exponent Bits Using Timing Analysis*, pages 602–624. Springer Berlin Heidelberg, Berlin, Heidelberg, 2016.

[11] Sarani Bhattacharya and Debdeep Mukhopadhyay. Curious case of rowhammer: Flipping secret exponent bits using timing analysis. In *Cryptographic Hardware and Embedded Systems - CHES 2016 - 18th International Conference, Santa Barbara, CA, USA, August 17-19, 2016, Proceedings*, pages 602–624, 2016.

[12] Ingrid Biehl, Bernd Meyer, and Volker Müller. Differential Fault Attacks on Elliptic Curve Cryptosystems. In *Proceedings of the 20th Annual International Cryptology Conference on Advances in Cryptology*, CRYPTO '00, pages 131–146, London, UK, UK, 2000. Springer-Verlag.

[13] Eli Biham and Adi Shamir. Differential Fault Analysis of Secret Key Cryptosystems. In *Proceedings of the 17th Annual International Cryptology Conference on Advances in Cryptology*, CRYPTO '97, pages 513–525, London, UK, UK, 1997. Springer-Verlag.

[14] Dan Boneh, Richard A. DeMillo, and Richard J. Lipton. *On the Importance of Checking Cryptographic Protocols for Faults*, pages 37–51. Springer Berlin Heidelberg, Berlin, Heidelberg, 1997.

[15] Erik Bosman, Kaveh Razavi, Herbert Bos, and Cristiano Giuffrida. Dedup est machina: Memory deduplication as an advanced exploitation vector. In *2016 IEEE Symposium on Security and Privacy (SP)*, pages 987–1004, 2016.

[16] Ferdinand Brasser, Lucas Davi, David Gens, Christopher Liebchen, and Ahmad-Reza Sadeghi. Can't touch this: Software-only mitigation against rowhammer attacks targeting kernel memory. In *26th USENIX Security Symposium (USENIX Security 17)*, pages 117–130, Vancouver, BC, August 2017. USENIX Association.

[17] Daniel R. L. Brown. SEC 2: Recommended Elliptic Curve Domain Parameters, 2010.

[18] Kevin Kai-Wei Chang, Donghyuk Lee, Zeshan Chishti, Alaa R. Alameldeen, Chris Wilkerson, Yoongu Kim, and Onur Mutlu. Improving dram performance by parallelizing refreshes with accesses. *2014 IEEE 20th International Symposium on High Performance Computer Architecture (HPCA)*, Feb 2014.

[19] Lucian Cojocar, Kaveh Razavi, Cristiano Giuffrida, and Herbert Bos. Exploiting Correcting Codes: On the Effectiveness of ECC Memory Against Rowhammer Attacks. In *S&P*, May 2019. Best Practical Paper Award, Pwnie Award Nomination for Most Innovative Research.

[20] Lucian Cojocar, Kaveh Razavi, Cristiano Giuffrida, and Herbert Bos. Exploiting correcting codes: On the effectiveness of ecc memory against rowhammer attacks. In *2019 IEEE Symposium on Security and Privacy (SP)*, pages 55–71, 2019.

[21] Jean-Sébastien Coron. Resistance against differential power analysis for elliptic curve cryptosystems. In Çetin K. Koç and Christof Paar, editors, *Cryptographic Hardware and Embedded Systems*, pages 292–302, Berlin, Heidelberg, 1999. Springer Berlin Heidelberg.

[22] Intel Corporation. *Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 3 (3A, 3B & 3C): System Programming Guide*. Intel Corporation, September 2016.

[23] Craig Costello and Benjamin Smith. Montgomery curves and their arithmetic, 2017.

[24] Franck Courbon, Philippe Loubet-Moundi, Jacques J. A. Fournier, and Assia Tria. *Adjusting Laser Injections for Fully Controlled Faults*, pages 229–242. Springer International Publishing, Cham, 2014.

[25] Joan Daemen and Vincent Rijmen. *The Design of Rijndael*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2002.

[26] Timothy J. Dell. A white paper on the benefits of chipkill-correct ecc for pc server main memory. In *IBM Microelectronics Division - Rev. 11/19/97*, 1997.

[27] Changbin Du. Page Table Isolation (PTI), 2019.

[28] Armando Faz-Hernández, Sam Scott, Nick Sullivan, Riad S. Wahby, and Christopher A. Wood. Hashing to Elliptic Curves. Internet-Draft draft-irtf-cfrg-hash-to-curve-12, Internet Engineering Task Force, September 2021. Work in Progress.

[29] Pietro Frigo, Emanuele Vannacci, Hasan Hassan, Victor van der Veen, Onur Mutlu, Cristiano Giuffrida, Herbert Bos, and Kaveh Razavi. TRRespass: Exploiting the Many Sides of Target Row Refresh. In *S&P*, May 2020. Best Paper Award, Pwnie Award for Most Innovative Research.

[30] Neil Gandal, J. T. Hamrick, Tyler Moore, and Tali Oberman. Price manipulation in the bitcoin ecosystem. *Journal of Monetary Economics*, 95:86–96, May 2018. Publisher Copyright: © 2017.

[31] Geoff Gasior. The SSD Endurance Experiment: 200TB update - The first bad blocks appear, 2013. `https://techreport.com/review/25559/the-ssd-endurance-experiment-200tb-update`.

[32] Daniel Genkin, Adi Shamir, and Eran Tromer. *RSA Key Extraction via Low-Bandwidth Acoustic Cryptanalysis*, pages 444–461. Springer Berlin Heidelberg, Berlin, Heidelberg, 2014.

[33] Daniel Gruss, Moritz Lipp, Michael Schwarz, Richard Fellner, Clémentine Maurice, and Stefan Mangard. Kaslr is dead: Long live kaslr. In Eric Bodden, Mathias Payer, and Elias Athanasopoulos, editors, *Engineering Secure Software and Systems*, pages 161–176, Cham, 2017. Springer International Publishing.

[34] Daniel Gruss, Moritz Lipp, Michael Schwarz, Daniel Genkin, Jonas Juffinger, Sioli O'Connell, Wolfgang Schoechl, and Yuval Yarom. Another flip in the wall of rowhammer defenses. In *2018 IEEE Symposium on Security and Privacy, SP 2018, Proceedings, 21-23 May 2018, San Francisco, California, USA*, pages 245–261, 2018.

[35] Daniel Gruss, Clémentine Maurice, Anders Fogh, Moritz Lipp, and Stefan Mangard. Prefetch side-channel attacks: Bypassing smap and kernel aslr. In *Proceedings of the 2016 ACM SIGSAC conference on computer and communications security*, pages 368–379, 2016.

[36] Daniel Gruss, Clémentine Maurice, and Stefan Mangard. Rowhammer.js: A remote software-induced fault attack in javascript. In *Detection of Intrusions and Malware, and Vulnerability Assessment - 13th International Conference, DIMVA 2016, San Sebastián, Spain, July 7-8, 2016, Proceedings*, pages 300–321, 2016.

[37] J. Alex Halderman, Seth D. Schoen, Nadia Heninger, William Clarkson, William Paul, Joseph A. Calandrino, Ariel J. Feldman, Jacob Appelbaum, and Edward W. Felten. Lest We Remember: Cold-boot Attacks on Encryption Keys. *Commun. ACM*, 52(5):91–98, May 2009.

[38] Darrel Hankerson, Alfred J. Menezes, and Scott Vanstone. *Guide to Elliptic Curve Cryptography*. Springer-Verlag, Berlin, Heidelberg, 2003.

[39] H. M. Heys and S. E. Tavares. The design of substitution-permutation networks resistant to differential and linear cryptanalysis. In *Proceedings of the 2nd ACM Conference on Computer and Communications Security*, CCS '94, page 148–155, New York, NY, USA, 1994. Association for Computing Machinery.

[40] Rei-Fu Huang, Hao-Yu Yang, Mango Chia-Tso Chao, and Cojocar Shih-Chin Lin. Alternate hammering test for application-specific drams and an industrial case study. In *The 49th Annual Design Automation Conference 2012, DAC '12, San Francisco, CA, USA, June 3-7, 2012*, pages 1012–1017, 2012.

[41] Michael Hutter and Jörn-Marc Schmidt. *The Temperature Side Channel and Heating Fault Attacks*, pages 219–235. Springer International Publishing, Cham, 2014.

[42] SANS Institute. An Introduction to TEMPEST, 2017. `https://www.sans.org/reading-room/whitepapers/privacy/introduction-tempest-981`.

[43] Gorka Irazoqui, Thomas Eisenbarth, and Berk Sunar. Mascat: Preventing microarchitectural attacks before distribution. In *Proceedings of the Eighth ACM Conference on Data and Application Security and Privacy*, CODASPY '18, page 377–388, New York, NY, USA, 2018. Association for Computing Machinery.

[44] ITRS. Front end processes. In *The International Technology Roadmap for Semiconductors*, ITRS '09. ITRS, 2009.

[45] ITRS. Executive report. In *International Technology Roadmap for Semiconductors 2.0*, ITRS '15. ITRS, 2015.

[46] Alix Jean-Pharuns. Motherboard vice: Rowhammer.js is the most ingenious hack i've ever seen, 2015.

[47] JEDEC. Ddr4 sdram. In *JEDEC STANDARD*, JEDEC '12. JEDEC Solid State Technology Association, 2012.

[48] Pascal Junod. On the complexity of matsui's attack. In Serge Vaudenay and Amr M. Youssef, editors, *Selected Areas in Cryptography*, pages 199–211, Berlin, Heidelberg, 2001. Springer Berlin Heidelberg.

[49] Irving Kaplansky. *Fields and rings*. University of Chicago Press, Chicago, 1972.

[50] Cameron F. Kerry, Acting Secretary, and Charles Romine Director. Fips pub 186-4 federal information processing standards publication digital signature standard (dss), 2013.

[51] Chong Hee Kim and Jean-Jacques Quisquater. *Fault Attacks for CRT Based RSA: New Attacks, New Results, and New Countermeasures*, pages 215–228. Springer Berlin Heidelberg, Berlin, Heidelberg, 2007.

[52] Seong Keun Kim and Mihaela Popovici. Future of dynamic random-access memory as main memory. *MRS Bulletin*, 43(5):334–339, 2018.

[53] Seong Keun Kim and Mihaela Popovici. Future of dynamic random-access memory as main memory. *MRS Bulletin*, 43(5):334–339, 2018.

[54] Yoongu Kim, Ross Daly, Jeremie Kim, Chris Fallin, Ji Hye Lee, Donghyuk Lee, Chris Wilkerson, Konrad Lai, and Onur Mutlu. Flipping bits in memory without accessing them: An experimental study of dram disturbance errors. *SIGARCH Comput. Archit. News*, 42(3):361–372, June 2014.

[55] Yoongu Kim, Vivek Seshadri, Donghyuk Lee, Jamie Liu, and Onur Mutlu. A case for exploiting subarray-level parallelism (salp) in dram. *SIGARCH Comput. Archit. News*, 40(3):368–379, June 2012.

[56] Neal Koblitz. Elliptic curve cryptosystems. *Mathematics of Computation*, 48(177):203–209, January 1987.

[57] Paul Kocher, Daniel Genkin, Daniel Gruss, Werner Haas, Mike Hamburg, Moritz Lipp, Stefan Mangard, Thomas Prescher, Michael Schwarz, and Yuval Yarom. Spectre attacks: Exploiting speculative execution, 2018.

[58] Paul C. Kocher. Timing Attacks on Implementations of Diffie-Hellman, RSA, DSS, and Other Systems. In *Proceedings of the 16th Annual International Cryptology Conference on Advances in Cryptology*, CRYPTO '96, pages 104–113, London, UK, UK, 1996. Springer-Verlag.

[59] Andrew Kwong, Daniel Genkin, Daniel Gruss, and Yuval Yarom. Rambleed: Reading bits in memory without accessing them. In *41st IEEE Symposium on Security and Privacy (S&P)*, 2020.

[60] Lucas Ladeira, Erick Nascimento, João Paulo Fernandes Ventura, Ricardo Dahab, Diego Aranha, and Julio Hernandez. Canais laterais em criptografia simétrica e de curvas elípticas: ataques e contra medidas, 2016. `http://sbseg2016.ic.uff.br/pt/files/MC3-SBSeg2016.pdf`.

[61] Nate Lawson. Timing attack in Google Keyczar library , 2009.

[62] Donghyuk Lee, Yoongu Kim, Vivek Seshadri, Jamie Liu, Lavanya Subramanian, and Onur Mutlu. Tiered-latency dram: A low latency and low cost dram architecture. In *Proceedings of the 2013 IEEE 19th International Symposium on High Performance Computer Architecture (HPCA)*, HPCA '13, pages 615–626, Washington, DC, USA, 2013. IEEE Computer Society.

[63] Kevin Lim, Jichuan Chang, Trevor Mudge, Parthasarathy Ranganathan, Steven K. Reinhardt, and Thomas F. Wenisch. Disaggregated memory for expansion and sharing in blade servers. *SIGARCH Comput. Archit. News*, 37(3):267–278, June 2009.

[64] Moritz Lipp, Michael Schwarz, Daniel Gruss, Thomas Prescher, Werner Haas, Anders Fogh, Jann Horn, Stefan Mangard, Paul Kocher, Daniel Genkin, Yuval Yarom, and Mike Hamburg. Meltdown: Reading kernel memory from user space. In *27th USENIX Security Symposium (USENIX Security 18)*, pages 973–990, Baltimore, MD, August 2018. USENIX Association.

[65] Jamie Liu, Ben Jaiyen, Richard Veras, and Onur Mutlu. Raidr: Retention-aware intelligent dram refresh. *SIGARCH Comput. Archit. News*, 40(3):1–12, June 2012.

[66] Victor Lomme. Countermeasures against Fault Attacks, 2014. `http://sips.inesc-id.pt/~trudevice/presentations/Countermeasures%20against%20Fault%20Attacks.pdf`.

[67] A.J. Menezes, T. Okamoto, and S.A. Vanstone. Reducing elliptic curve logarithms to logarithms in a finite field. *IEEE Transactions on Information Theory*, 39(5):1639–1646, 1993.

[68] Alfred J. Menezes, Scott A. Vanstone, and Paul C. Van Oorschot. *Handbook of Applied Cryptography*. CRC Press, Inc., Boca Raton, FL, USA, 1st edition, 1996.

[69] Onur Mutlu and Thomas Moscibroda. Stall-time fair memory access scheduling for chip multiprocessors. In *Proceedings of the 40th Annual IEEE/ACM International Symposium on Microarchitecture*, MICRO 40, pages 146–160, Washington, DC, USA, 2007. IEEE Computer Society.

[70] Satoshi Nakamoto. Bitcoin: A peer-to-peer electronic cash system, 2009.

[71] Shaun Nichols. The register: 3 is the magic number (of bits): Flip 'em at once and your ECC protection can be rowhammer'd, 2018. Dutch boffins prove it is possible to evade memory-busting attack mitigations.

[72] Peter Pessl, Daniel Gruss, Clémentine Maurice, Michael Schwarz, and Stefan Mangard. DRAMA: Exploiting DRAM addressing for cross-cpu attacks. In *25th USENIX Security Symposium (USENIX Security 16)*, pages 565–581, Austin, TX, 2016. USENIX Association.

[73] Henk Porck, Biblioteca Espaa, Biblioteca Portugal, and Liechtensteinisches Landesarchiv. Rate of paper degradation the predictive value of artificial aging tests, 11 2000.

[74] Thomas Pornin. Deterministic Usage of the Digital Signature Algorithm (DSA) and Elliptic Curve Digital Signature Algorithm (ECDSA). RFC 6979, August 2013.

[75] B. Ramakrishna Rau. Pseudo-randomly interleaved memory. *SIGARCH Comput. Archit. News*, 19(3):74–83, April 1991.

[76] Kaveh Razavi, Ben Gras, Erik Bosman, Bart Preneel, Cristiano Giuffrida, and Herbert Bos. Flip feng shui: Hammering a needle in the software stack. In *25th USENIX Security Symposium, USENIX Security 16, Austin, TX, USA, August 10-12, 2016.*, pages 1–18, 2016.

[77] F.H. Reynolds. Thermally accelerated aging of semiconductor components. *Proceedings of the IEEE*, 62(2):212–222, 1974.

[78] R. L. Rivest, A. Shamir, and L. Adleman. A Method for Obtaining Digital Signatures and Public-key Cryptosystems. *Commun. ACM*, 21(2):120–126, February 1978.

[79] Ronald L. Rivest. Cryptography. In Jan van Leeuwen, editor, *Handbook of Theoretical Computer Science, Volume A: Algorithms and Complexity*, pages 717–755. Elsevier and MIT Press, 1990.

[80] Rogério Reis Devisate. *Grilagem das Terras e da Soberania*. Imagem Art Studio, NITEROI, RJ, Brazil, 1st edition, 2017.

[81] Bhaskar Saha, Jose R. Celaya, Philip F. Wysocki, and Kai F. Goebel. Towards prognostics for electronics components. In *2009 IEEE Aerospace conference*, pages 1–7, 2009.

[82] Jörn-Marc Schmidt and Christoph Herbst. A Practical Fault Attack on Square and Multiply. *2013 Workshop on Fault Diagnosis and Tolerance in Cryptography*, 00:53–58, 2008.

[83] Mark Seaborn and Thomas Dullien. Exploiting the dram rowhammer bug to gain kernel privileges. In *Google Project Zero*, Project Zero '15. Google blog, 2015.

[84] Mark Seaborn and Thomas Dullien. Exploiting the DRAM rowhammer bug to gain kernel privileges: How to cause and exploit single bit errors, 2015.

[85] Sergei P. Skorobogatov and Ross J. Anderson. Optical Fault Induction Attacks. In *Revised Papers from the 4th International Workshop on Cryptographic Hardware and Embedded Systems*, CHES '02, pages 2–12, London, UK, UK, 2003. Springer-Verlag.

[86] Fatemeh Tehranipoor, Nima Karimian, Wei Yan, and John A. Chandy. Dram-based intrinsic physically unclonable functions for system-level security and authentication. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 25(3):1085–1097, 2017.

[87] Fatemeh Tehranipoor, Nima Karimian, Wei Yan, and John A. Chandy. Investigation of dram pufs reliability under device accelerated aging effects, 2017.

[88] Michael Tunstall, Debdeep Mukhopadhyay, and Subidh Ali. *Differential Fault Analysis of the Advanced Encryption Standard Using a Single Fault*, pages 224–233. Springer Berlin Heidelberg, Berlin, Heidelberg, 2011.

[89] Victor van der Veen, Yanick Fratantonio, Martina Lindorfer, Daniel Gruss, Clémentine Maurice, Giovanni Vigna, Herbert Bos, Kaveh Razavi, and Cristiano Giuffrida. Drammer: Deterministic rowhammer attacks on mobile platforms. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security, Vienna, Austria, October 24-28, 2016*, pages 1675–1689, 2016.

[90] Wim van Eck. Electromagnetic Radiation from Video Display Units: An Eavesdropping Risk? *Comput. Secur.*, 4(4):269–286, December 1985.

[91] Jasper G. J. van Woudenberg, Marc F. Witteman, and Federico Menarini. Practical Optical Fault Injection on Secure Microcontrollers. In *FDTC*, pages 91–99. IEEE Computer Society, 2011.

[92] Ingrid Verbauwhede, Dusko Karaklajic, and Jorn-Marc Schmidt. The Fault Attack Jungle - A Classification Model to Guide You. In *Proceedings of the 2011 Workshop on Fault Diagnosis and Tolerance in Cryptography*, FDTC '11, pages 3–8, Washington, DC, USA, 2011. IEEE Computer Society.

[93] Xiaoxiao Wang, LeRoy Winemberg, Donglin Su, Dat Tran, Saji George, Nisar Ahmed, Steve Palosh, Allan Dobin, and Mohammad Tehranipoor. Aging adaption in integrated circuits using a novel built-in sensor. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 34(1):109–121, 2015.

[94] Lawrence C. Washington. *Elliptic Curves: Number Theory and Cryptography, Second Edition*. Chapman & Hall/CRC, 2 edition, 2008.

[95] Kelce S. Wilson. Conflicts among the pillars of information assurance. *IT Professional*, 15(4):44–49, 2013.

[96] Yuan Xiao, Xiaokuan Zhang, Yinqian Zhang, and Radu Teodorescu. One bit flips, one cloud flops: Cross-vm row hammer attacks and privilege escalation. In *25th USENIX Security Symposium, USENIX Security 16, Austin, TX, USA, August 10-12, 2016.*, pages 19–35, 2016.

[97] Yuval Yarom and Katrina Falkner. FLUSH+RELOAD: A High Resolution, Low Noise, L3 Cache Side-Channel Attack. In *23rd USENIX Security Symposium (USENIX Security 14)*, pages 719–732, San Diego, CA, 2014. USENIX Association.

[98] Keun Soo Yim. The rowhammer attack injection methodology. In *2016 IEEE 35th Symposium on Reliable Distributed Systems (SRDS)*, pages 1–10, 2016.

[99] J. F. Ziegler and W. A. Lanford. Effect of Cosmic Rays on Computer Memories. In *Science*, volume 206, pages 776–788, 1979.

Appendix A

ECDSA Algorithms

Algorithm A.1 ECDSA signature generation

Require:

E: elliptic curve field and equation,

G: elliptic curve point that generates a subgroup with large prime order n,

n: order of the subgroup generated by G, meaning that $nG = \mathcal{O}$, where $\mathcal{O}$ is the identity element,

L_n: bit length of the group order n, which means that $L_n = (1 \ll (\lfloor \log_2(n) \rfloor + 1)) - 1$,

m: a message to be signed,

d_A: a randomly selected private key.

Ensure:

(r, s): a signature represented by a pair of integers, each in the interval $[1, n-1]$.

1: **procedure** ECDSA_SIGN(E, G, n, L_n, m, d_A)
2: $\alpha \leftarrow \text{HASH}(m)$ ▷ Cryptographic hash with the output converted into an integer
3: $h \leftarrow L_n$ bits of α ▷ Note that h can be greater than n but not longer in bit size [50]
4: $k \leftarrow (\text{random}() \mod (n-1)) + 1$ ▷ Selects a random number from $[1, n-1]$
5: $P = (x, y) \leftarrow k \cdot G$
6: $r \leftarrow x \bmod n$
7: **if** $r = 0$ **then**
8: **go to** 4
9: **end if**
10: $s \leftarrow k^{-1}(h + rd_A) \bmod n$
11: **if** $s = 0$ **then**
12: **go to** 4
13: **end if**
14: **return** Signature pair (r, s)
15: **end procedure**

Algorithm A.2 ECDSA signature verification

Require:

E: elliptic curve field and equation,
G: elliptic curve point that generates a subgroup with large prime order n,
n: order of the subgroup generated by G, meaning that $nG = \mathcal{O}$, where $\mathcal{O}$ is the identity element,
L_n: bit length of the group order n, which means that $L_n = (1 \ll (\lfloor \log_2(n) \rfloor + 1)) - 1$,
Q_A: the public key computed by the elliptic curve as $Q_A = d_A G$, with d_A the private key,
m: the message used to generate the signature,
(r, s): a signature to be verified.

Ensure:

A flag indicating if the signature is valid.

procedure ECDSA_VERIFY($E, G, n, L_n, Q_A, m, r, s$)
  if $Q_A = \mathcal{O}$ **or** $Q_A \notin E$ **or** $n \cdot Q_A \neq \mathcal{O}$ **then**
    return Invalid!
  end if
  if $r \notin [1, n-1]$ **or** $s \notin [1, n-1]$ **then**
    return Invalid!
  end if
  $\alpha \leftarrow \text{HASH}(m)$ ▷ Same hash function used in Algorithm A.1
  $h \leftarrow L_n$ bits of α
  $u_1 = hs^{-1} \bmod n$
  $u_2 = rs^{-1} \bmod n$
  $P = (x, y) \leftarrow u_1 \cdot G + u_2 \cdot Q_A$
  if $P = \mathcal{O}$ **then**
    return Invalid!
  end if
  if $x \equiv r \bmod n$ **then**
    return Valid!
  else
    return Invalid!
  end if
end procedure