Introduction to Lattice-based Attacks

Version 2

Giovanni Di Santi

Contents

1 Introduction **2**

2 Linear Algebra Background **3**
2.1 Vector Spaces 3
2.2 Lattices 6
2.3 Lattice Problems 7
2.3.1 SVP 7
2.3.2 CVP 8

3 LLL **9**
3.1 Introduction 9
3.2 Algorithm 10

4 RSA Cryptanalysis **12**
4.1 RSA Introduction 12
4.1.1 Algorithm 12
4.1.2 Security 13
4.2 Lattices against RSA 13
4.2.1 Mathematical Introduction 13
4.2.2 Recovering Plaintext with Known MSB Bits of m . . . 15
4.2.3 Factorizing the Modulus with Known MSB Bits of p . 17
4.2.4 Real World Case Study: ROCA 18

5 ECDSA Cryptanalysis **20**
5.1 ECDSA Introduction 20
5.1.1 Algorithm 20
5.1.2 Security 21
5.2 Lattices against ECDSA 21
5.2.1 Recover d from a Pair of Signatures with Small k_1, k_2 . 22
5.2.2 Recover d from Many Signatures with Small k_i 23
5.2.3 Recover d Knowing MSB Bits of Each k_i 24

Chapter 1

Introduction

Lattices are potent mathematical structures widely utilized in computer science and mathematics to solve a broad spectrum of problems. In recent years, numerous post-quantum cryptographic protocol candidates have employed the challenging problems associated with hard lattices, for example, the **Learning With Errors** [20]. Additionally, lattice constructions are leveraged in cryptanalysis to compromise cryptographic schemes, typically through the use of reduction algorithms such as **LLL** [9].

To comprehend the mathematics of lattices, a foundational understanding of linear algebra is essential. Therefore, the second chapter provides a summary of the basic properties necessary for this understanding. We primarily referenced "An Introduction to Mathematical Cryptography"[8] to elucidate these concepts, while also providing some geometric intuition behind certain operations.

Cryptographic protocols such as **RSA** and **ECDSA** are considered secure; however, various attacks can exploit a relaxed model of each, e.g., what if some bits of the secret key are known? What if a portion of the message is known prior to encryption? We introduce lattice-based attacks on these protocols and provide code in our repository [23] to verify the effectiveness of these attacks. The main reference for this discussion is "Recovering cryptographic keys from partial information, by example"[14].

Chapter 2

Linear Algebra Background

This chapter introduces key definitions and properties essential for understanding lattices. It progresses to cover the mathematics of lattices and related complex problems. By the end of this chapter, the reader should be well-equipped to follow the mathematical discussions in subsequent chapters.

2.1 Vector Spaces

Definition 2.1.1 ***Vector space.***

A *vector space* V over $\mathbb{R}^m$ is a set closed under finite vector addition and scalar multiplication. This closure is characterized by:

$$a_1 v_1 + a_2 v_2 \in V \text{ for any } v_1, v_2 \in V \text{ and } a_1, a_2 \in \mathbb{R}$$

Definition 2.1.2 ***Linear Combinations***

Consider any vectors $v_1, v_2, \ldots, v_k \in V$. A *linear combination* of these vectors is any vector expressed as:

$$\alpha_1 v_1 + \alpha_2 v_2 + \cdots + \alpha_k v_k \text{ where } \alpha_1, \ldots, \alpha_k \in \mathbb{R}$$

Definition 2.1.3 ***Linear Independence***

A set of vectors $v_1, v_2, \ldots, v_k \in V$ is *linearly independent* if the only solution to the equation

$$a_1 v_1 + a_2 v_2 + \cdots + a_k v_k = 0$$

is $a_1 = a_2 = \cdots = a_k = 0$.

Definition 2.1.4 ***Bases***

A set of linearly independent vectors $b = (v_1, \ldots, v_n) \in V$ constitutes a *basis* of V if every vector $w \in V$ can be represented as:

$$w = a_1 v_1 + a_2 v_2 + \cdots + a_n v_n$$

Definition 2.1.5 ***Vector's Length***

The *Euclidean norm* or length of a vector $v = (x_1, x_2, \ldots, x_m)$ is given by:

$$\|v\| = \sqrt{x_1^2 + x_2^2 + \cdots + x_m^2}$$

Definition 2.1.6 ***Dot Product***

Given vectors $v, w \in V \subset \mathbb{R}^m$ where $v = (x_1, x_2, \ldots, x_m)$ and $w = (y_1, y_2, \ldots, y_m)$, the *dot product* of v and w is defined as:

$$v \cdot w = x_1 y_1 + x_2 y_2 + \cdots + x_m y_m$$
or
$$v \cdot w = \|v\| \|w\| \cos\theta$$

where θ is the angle between v and w when their starting points are positioned at the origin O.

Geometrically, $v \cdot w$ represents the length of the projection of w onto v multiplied by the length of v, as illustrated in 2.1.

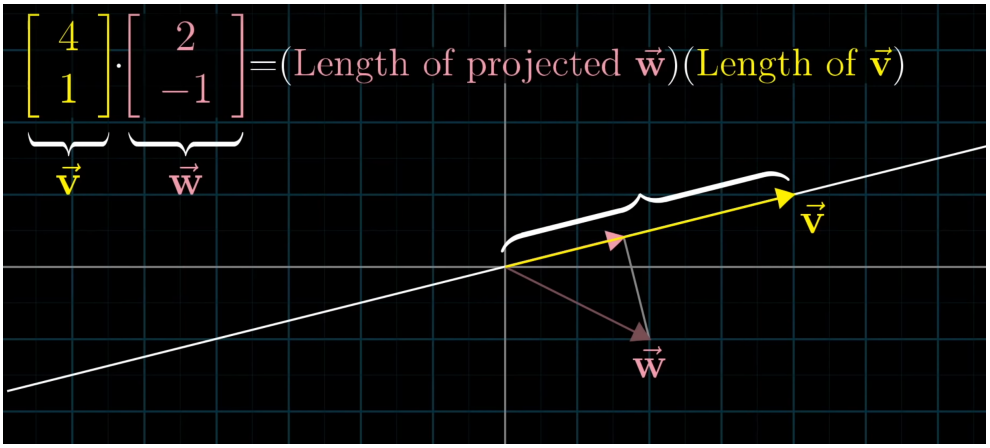

Figure 2.1: Dot Product By 3Blue1Brown [21]

Definition 2.1.7 ***Orthogonal Basis***

An *orthogonal basis* for a vector space V consists of vectors $v_1, \ldots, v_m$ such that:

$$v_i \cdot v_j = 0 \text{ for all } i \neq j$$

If $\|v_i\| = 1$ for each i, then the basis is termed *orthonormal*.

Algorithm 1 Gram-Schmidt Algorithm

$v_1^* = v_1$
for $i \leftarrow 2$ **to** n **do**
  $\mu_{ij} = v_i \cdot v_j^* / \|v_j^*\|^2$ for $1 \leq j < i$
  $v_i^* = v_i - \sum_{j=1}^{i-1} \mu_{ij} v_j^*$
end

Given a basis $b = (v_1, \ldots, v_n)$ of a vector space $V \subset \mathbb{R}^m$, the Gram-Schmidt algorithm generates an orthogonal basis $b^* = (v_1^*, \ldots, v_n^*)$. These two bases satisfy the property that $\text{Span}\{v_1, \ldots, v_i\} = \text{Span}\{v_1^*, \ldots, v_i^*\}$ for all $i = 1, 2, \ldots, n$.

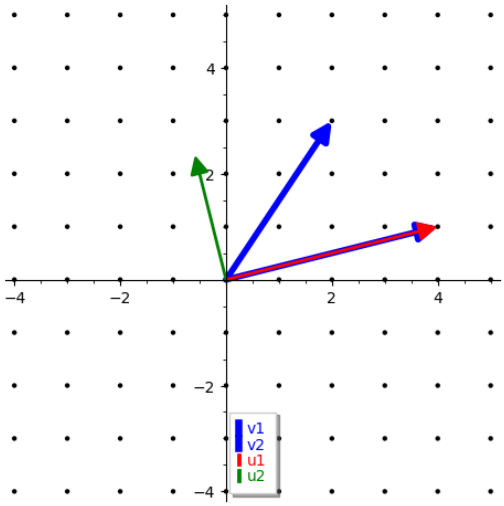

Figure 2.2: Gram Schmidt Orthogonalization

Using vectors $v_1 = (4, 1)$ and $v_2 = (2, 3)$ as a basis and applying the Gram-Schmidt process results in $u_1 = v_1 = (4, 1), u_2 = (-10/17, 40/17)$, as depicted in 2.2.

Definition 2.1.8 ***Determinant***

The *determinant* of a square matrix is a scalar value that characterizes the matrix and is defined by the following properties:

1. The determinant is linear in each row.
2. The determinant changes sign when two rows are swapped.
3. The determinant of the identity matrix is 1.

A matrix is termed **singular** if its determinant is 0, indicating it lacks an inverse.

Animations created by Grant Sanderson provide a fantastic visual understanding of the geometric intuition behind the *determinant*[22].

2.2 Lattices

Definition 2.2.1 ***Lattice***

Consider $v_1, \ldots, v_n \in \mathbb{R}^m$ with $m \geq n$, which are linearly independent vectors. A *lattice* L, spanned by $\{v_1, \ldots, v_n\}$, comprises all integer linear combinations of these vectors:

$$L = \left\{ \sum_{i=1}^{n} a_i v_i, a_i \in \mathbb{Z} \right\}$$

If every v_i $(i = 1, \ldots, n)$ has integer coordinates, the lattice is termed an *integral lattice*.

Figures 2.3 and 2.4 depict the lattice L with bases $v = (3, 1)$ and $w = (-1, 1)$, and another basis configuration, respectively.

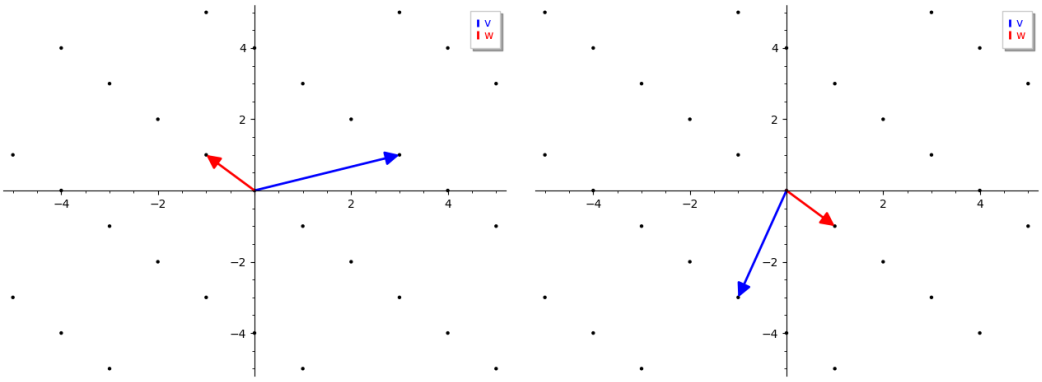

Figure 2.3: Lattice L spanned by v, w Figure 2.4: Lattice L spanned by v', w'

2.3 Lattice Problems

2.3.1 SVP

The Shortest Vector Problem (SVP): *Find a nonzero vector $v \in L$ that minimizes the Euclidean norm $\|v\|$.*

Gauss developed an algorithm to determine an optimal basis for a two-dimensional lattice from any given basis. This method outputs the shortest nonzero vector in L, thereby solving the SVP.

Algorithm 2 Gauss Basis Reduction

while *true* **do**
  if $\|v_2\| < \|v_1\|$ **then**
    swap v_1, v_2
  end
  m $= \lfloor (v_1 \cdot v_2)\|v_1\|^{-2} \rfloor$ **if** $m = 0$ **then**
    return v_1, v_2
  end
  $v_2 = v_2 - mv_1$
end

Example: In a lattice L spanned by $v_1 = (8, 4), v_2 = (7, 5)$, applying the

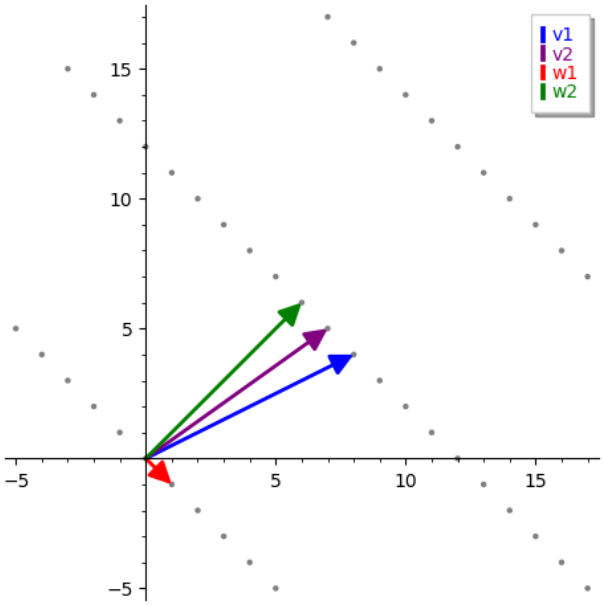

Figure 2.5: Gauss reduction

Gauss reduction algorithm results in $w_1 = (1, -1), w_2 = (6, 6)$ as shown in 2.5. w_1 represents the shortest nonzero vector in lattice L.

The complexity of solving SVP increases with the dimension of the lattice, making it a challenging problem without a known polynomial-time solution.

2.3.2 CVP

The Closest Vector Problem (CVP): *Given a vector $t \in \mathbb{R}^m$ that is not in L, find the vector $v \in L$ closest to t, minimizing the Euclidean norm $\|t - v\|$.*

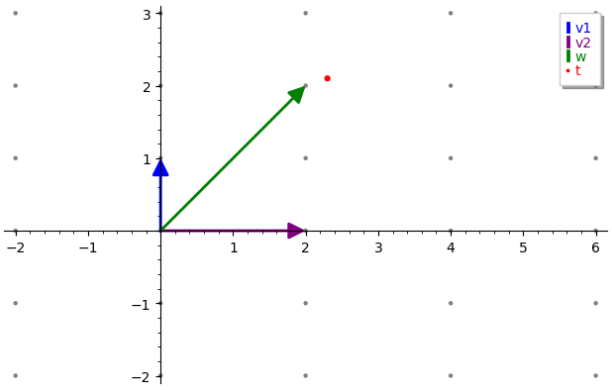

Figure 2.6: CVP

Example: For a lattice L spanned by $v_1 = (0, 1), v_2 = (2, 0)$, with target vector $t = (2.3, 2.4)$, the closest vector in the lattice L is $w = (2, 2)$, as depicted in 2.6.

Both **CVP** and **SVP** are considered $\mathcal{NP}$-hard problems, and their complexity increases with the dimension of the lattice. They serve as primitives for creating trapdoor functions in cryptographic schemes.

Several algorithms can approximate solutions for both **SVP** and **CVP**, such as **LLL**[9] and **Babai's algorithm**[3] respectively.

Chapter 3

LLL

3.1 Introduction

The **Lenstra-Lenstra-Lovász (LLL)** algorithm is a polynomial-time method designed to compute a "short" basis for a lattice.

Theorem 3.1.1 ***LLL***

Given a lattice $L \in \mathbb{Z}^n$ spanned by $B = \{v_1, \ldots, v_n\}$, the LLL algorithm produces a reduced basis $\{w_1, \ldots, w_n\}$ such that:

$$\|w_i\| \leq 2^{\frac{n(n-1)}{4(n-i+1)}} \cdot det(L)^{\frac{1}{n-i+1}} \text{ for } i = 1, \ldots, n$$

This process is executed in time polynomial in n and the bit-size of the entries in the basis matrix B.

In this reduced basis, the vectors are as short as possible starting from the first, with each subsequent vector being longer. Additionally, these vectors are almost orthogonal to each other, meaning the dot product $w_i \cdot w_j$ is close to zero.

Example

Consider the following basis, represented in matrix form, that spans a lattice L:

$$L = \begin{pmatrix} 4 & 9 & 10 \\ 2 & 1 & 30 \\ 3 & 7 & 9 \end{pmatrix}$$

After applying the LLL algorithm, the basis is transformed to:

$$LLL(L) = \begin{pmatrix} -1 & -2 & -1 \\ 3 & -2 & 1 \\ -1 & -1 & 5 \end{pmatrix}$$

Here, the first row is the shortest vector in the lattice L, thus addressing the **SVP** problem. For higher dimensions, however, the LLL algorithm provides only an approximation to the **SVP**.

3.2 Algorithm

Algorithm 3 LLL reduction algorithm

Input basis $v_1, \ldots, v_n$
$k = 2$
$v_1* = v_1$
while $k \leq n$ **do**
  for $j = k - 1$ **to** 1 *step* -1 **do**
    $v_k = v_k - \lfloor \mu_{k,j} \rceil \, \boldsymbol{v}_j$ [size-reduction]
  end
  if $\|v_k^*\|^2 \geq (\frac{3}{4} - \mu_{k,k-1}^2)\|v_{k-1}^2\|$ **then**
    $k = k + 1$ [Lovàsz-condition]
  else
    Swap v_{k-1} and v_k [swap-step]
    $k = \max(k - 1, 2)$
  end
end
Note: At each step, $v_1^*, \ldots, v_k^*$ is the orthogonalized set of vectors obtained with the Gram-Schmidt 1.
$\mu_{i,j}$ is the quantity $(v_i \cdot v_j^*)/\|v_j^*\|^2$.
Output reduced basis $v_1, \ldots, v_n$

The running time of the algorithm is polynomial and completes the main loop in no more than $\mathcal{O}(n^2 \log n + n^2 \log M)$ steps, where $M = max\|v_i\|$.

Kelby Ludwig elucidates the intuition behind each step of the algorithm [11], and Oded Regev provides a detailed explanation of its properties [19].

Applications of the LLL algorithm include:

1. Factoring polynomials over the integers, e.g., factorizing $x^2 - 1$ into $(x + 1)(x - 1)$.

2. Solving Integer Programming, a renowned $\mathcal{NP}$-complete problem, in polynomial time when the number of variables is fixed.

3. Approximating solutions to the **CVP** and **SVP**, among other lattice-based challenges.

4. Cryptanalysis, specifically breaking cryptographic protocols.

Chapter 4

RSA Cryptanalysis

4.1 RSA Introduction

4.1.1 Algorithm

RSA is one of the earliest and most used asymmetric cryptosystem. The steps to generate a public/private key for **RSA** are:

1. Fix $e = 65537$ or $e = 3$ (public).
2. Find two primes p, q such that $p - 1$ and $q - 1$ are relatively prime to e, $\gcd(e, p - 1) = 1$ and $\gcd(e, q - 1) = 1$.
3. Compute $N = p * q$ and $\phi(n) = (p - 1) * (q - 1)$
4. Calculate d (private) as the multiplicative inverse of e modulo $\phi(n)$.
5. (N, e) is the public key, (N, d) is the private key.

To encrypt a message m with **textbook RSA**:

$$c = m^e \mod N$$

To decrypt a ciphertext c:

$$m = c^d \mod N$$

4.1.2 Security

RSA relies on the hardness of factoring N, and there isn't a polynomial algorithm for factorization, so it's considered secure. However, there are different attacks against textbook RSA or bad implementations:

- If $m < N^{\frac{1}{e}}$, then $m^e < N$ (the modulo operation is not applied), and we only need to find the eth root of c over the integers to find m. This is one of the reasons why we pad the message before the encryption.
- If $q = p + x$ where x is small enough, then it's easy to recover the factors of N computing $A = \sqrt{N}$ and $p = A - x$ for different values of x until $N \mod p = 0$, which means that $q = N/p$.
- It's possible to recover d if it is too small (in practice it could be used to accelerate decryption), using the Wiener's attack [24].
- Other attacks can be found on the excellent survey by Dan Boneh [4].

4.2 Lattices against RSA

We begin by discussing the main concepts behind Coppersmith's attack [6] and the underlying mathematics. This attack is applicable to **RSA** when a small exponent e is used, and it can also be used to factorize the modulus if partial information about p or q is known. In 2017, this method was utilized to exploit CVE-2017-15361, also known as **ROCA** (Return Of Coppersmith's Attack) [15].

4.2.1 Mathematical Introduction

Finding roots of a univariate polynomial over the integers is straightforward. However, finding roots of a modular polynomial, such as:

$$f(x) \equiv 0 \mod N$$

poses significant challenges. Assume N is an **RSA** modulus whose factorization is unknown. Consider a univariate integer polynomial $f(x)$ of degree n:

$$f(x) = x^n + a_{n-1}x^{n-1} + a_{n-2}x^{n-2} + \cdots + a_1 n + a_0$$

Coppersmith demonstrated that it is possible to recover the value x_0 such that $f(x_0) \equiv 0 \mod N$, with $x_0 < N^{\frac{1}{n}}$, in polynomial time using the following theorem:

Theorem 4.2.1 ***Howgrave-Graham***

Let $g(x)$ be a univariate polynomial with n monomials and m be a positive integer. If we establish a constraint X and the following conditions are satisfied:

$$g(x_0) \equiv 0 \mod N^m, |x_0| \leq X \tag{4.1}$$

$$\|g(xX)\| < \frac{N^m}{\sqrt{n}} \tag{4.2}$$

Then $g(x_0) = 0$ is also valid over the integers.

This theorem enables the computation of the root of $f(x) \mod N$ by finding a polynomial $g(x)$ that shares the same root, but modulo N^m. If the conditions 4.1 and 4.2 are met, the root of $g(x)$ over the integers corresponds to the same root x_0 such that $f(x_0) \equiv 0 \mod N$.

The idea behind Howgrave-Graham's method involves generating polynomials p_i that also have x_0 as a root modulo N^m.

The significance of the **LLL** algorithm in this context is two-fold:

- It conducts **integer linear operations** on basis vectors, ensuring that even if the basis changes, it remains a linear combination of vectors retaining x_0 as a root modulo N^m.
- If the lattice is properly constructed, the norm of the shortest vector in the reduced basis will satisfy 4.2, allowing the algorithm to predict the length's bound of the shortest vector it can find.

Polynomials p_i ($g_{i,j}$ and h_i) sharing the same root x_0 over N^m of f, where δ is the degree of f, can be crafted as follows:

$$g_{i,j}(x) = x^j \cdot N^i \cdot f^{m-i}(x) \text{ for } i = 0, \ldots, m-1, \ j = 0, \ldots, \delta - 1 \tag{4.3}$$

$$h_i(x) = x^i \cdot f^m(x) \text{ for } i = 0, \ldots, t-1 \tag{4.4}$$

Applying **LLL** to a lattice built from the coefficients of 4.3, we'll find a short vector $v = g(xX)$ that will satisfy 4.2. Consequently, if we then compute $g(x)$, we will be able to determine the root over the integers.

4.2.2 Recovering Plaintext with Known MSB Bits of m

We will employ a simplified version of Coppersmith's attack, effective when the root is approximately less than $N^{\frac{1}{6}}$.

Problem Setup. Consider an **RSA** modulus N of 97 bits:

$$N = \texttt{0x1a34992f64135aedaa0fe2bfd} \text{ and } e = 3.$$

Before encryption, the message m undergoes padding:

$$z = pad||m = \texttt{0xffffffffffffffffffffff}||m$$

where $||$ denotes concatenation.
The resulting ciphertext is:

$$c = z^e \mod N = \texttt{0x11701d5479bcdef207cba6937}$$

Assuming the factorization of N is unknown, our goal is to deduce the message m. We are aware of the padding and that the message m is 2 bytes long, consisting only of ASCII characters, so $m < \texttt{0x7b7b}$.

Let's define:

$$a = \texttt{0xffffffffffffffffffffff0000}$$

as the known padded string that was encrypted.

This leads to the equation $c = (a + m)^3 \mod N$, for an unknown small m. We can then define $f(x) = (a + x)^3 - c$ and set up the problem to find a small root m such that $f(m) \equiv 0 \mod N$:

$$f(x) = x^3 + \texttt{0x17d7f55d0c9dc5851af54957c}x^2 + \texttt{0x3cf96fed2ea927527a2ed329}x + \texttt{0x32882e547964b3bcf75d315e}$$

Lattice Construction. Let the coefficients of $f(x)$ be $f(x) = x^3 + f_2x^2 + f_1x + f_0$ and $X = \texttt{0x7b7b}$ be the upper bound for the root m. We can construct the matrix:

$$B = \begin{pmatrix} X^3 & f_2X^2 & f_1X & f_0 \\ 0 & NX^2 & 0 & 0 \\ 0 & 0 & NX & 0 \\ 0 & 0 & 0 & N \end{pmatrix}$$

Each row of this matrix corresponds to the coefficient vectors of the polynomials $f(x), Nx^2, Nx$, and N, all evaluated at $x = m$ being 0 modulo N. We apply Howgrave-Graham with $m = 1$ (the N^m parameter, not the message).

In this lattice setup, every vector is of the form $v = (v_3X^3, v_2X^2, v_1X, v_0)$, because any integer linear combination of the vectors in the lattice will respect the bound X^i for $i = \dim(B) - 1, .., 0$.

Apply LLL. We apply LLL to identify the shortest vector in the reduced basis:

$$v = (-\texttt{0xd55d67baf71d}X^3, \texttt{0x3cf3cca3200a58bf}X^2, \texttt{0x3854d44211e80f248449}X, -\texttt{0xec93bf51cf766f7b9f1e5e6})$$

Using the coefficients of v, we construct the polynomial g:

$$g(x) = -\texttt{0xd55d67baf71d}x^3 + \texttt{0x3cf3cca3200a58bf}x^2 \texttt{0x3854d44211e80f248449}x - \texttt{0xec93bf51cf766f7b9f1e5e6}$$

Knowing that:

$$g(x_0) \equiv 0 \mod N, |x_0| \leq X$$

We need to verify that:

$$\|g(xX)\| \leq \frac{N}{\sqrt{n}}$$

In this scenario, the determinant of B, $\det B = X^6N^3$, and LLL is expected to find a short vector with $\|v\| \leq 2^{\frac{n(n-1)}{4(n)}}(\det B)^{\frac{1}{n}}$. Neglecting the factor $2^{\frac{3}{4}}$ (given that $n = 4$), the requirement simplifies to:

$$g(m) \leq \|v\| \leq (\det B)^{\frac{1}{4}} < \frac{N}{\sqrt{4}}$$

We have $(\det B)^{\frac{1}{4}} = (X^6N^3)^{\frac{1}{4}} < \frac{N}{\sqrt{4}}$. Solving for X under this constraint implies $X < (\frac{N}{16})^{\frac{1}{6}}$. Adjusting for the $\sqrt{4}$ as an error term allows $X < N^{\frac{1}{6}}$. This inequality holds with the given numbers; however, even if the bound of the shortest vector is larger, there remains a chance to find the correct root.

Computing the root of $g(x)$ over the integers, we find $m = \texttt{0x4142} =$ "AB", which is the correct result.

This specific lattice construction is effective for finding roots up to size $N^{\frac{1}{6}}$. Therefore, the same approach could be applied to recover:

- Approximately 170 unknown bits of a message from an RSA 1024-bit modulus.
- Approximately 341 unknown bits of a message from an RSA 2048-bit modulus.
- Approximately 683 unknown bits of a message from an RSA 4096-bit modulus.

For larger roots, a more extensive lattice incorporating additional polynomials generated with 4.3 is required. This methodology is further elaborated in [12], with a detailed implementation in SageMath by David Wong [25].

4.2.3 Factorizing the Modulus with Known MSB Bits of p

Lattices can also assist in factorizing a modulus N if the most significant bits of one of its factors are known. Let $a = 2^l b$, where b represents the known bits of p, then $p = a + r$ for some small r. We can reformulate this to find the root r of $f(x) = a + x \mod p$ such that $f(r) = p \equiv 0 \mod p$. Assume we know r's upper bound, $|r| < X$.

We can configure the following lattice:

$$B = \begin{pmatrix} X^2 & Xa & 0 \\ 0 & X & a \\ 0 & 0 & N \end{pmatrix}$$

Each row corresponds to the polynomials $x(x + a)$, $(x + a)$, and N, each evaluated at $x = r$ being $0 \mod p$. Similar to 4.2.2, every polynomial is scaled by X, the upper bound.

The shortest vector found by **LLL** should be less than p because if $g(x) < p$, then by construction $g(r) = 0$ over the integers can be computed. We calculate:

$$(detB)^{\frac{1}{\dim L}} = (X^3 N)^{\frac{1}{3}} < p$$

Solving for X gives $X < p^{\frac{1}{3}}$, indicating that this method is valid for every $|r| < p^{\frac{1}{3}}$.

To pinpoint r, simply apply **LLL** and take the first vector (the shortest), which will have the form $v = (v_2 X^2, v_1 X, v_0)$.

Finally, construct $g(x) = v_2 x^2 + v_1 x + v_0$ (adjust each coefficient by dividing by X^i) and compute the roots over the integers to find r. We executed

this attack against an **RSA**-1024 modulus knowing the 160 most significant bits of p. This strategy can be extended to $X < p^{\frac{1}{2}}$ with a larger lattice.

4.2.4 Real World Case Study: ROCA

ROCA[15] is a vulnerability discovered in 2017 in a software library *RSALib*, which affects the generation of the public modulus for **RSA**. This issue is believed to impact millions of smart cards. The vulnerability arises from the method used to generate prime numbers; since embedded devices lack extensive computational resources, a rapid generation method was necessary. Each prime number generated adheres to the following structure:

$$p = kM + (g^a \mod M)$$

Here, k, a are random values unique for each prime (unknown), $g = 65537$, and M is a *primorial* number. A *primorial* number M is defined as the product of the first n successive primes:

$$M = P_n\# = \prod_{i=1}^{n} P_i$$

where n corresponds to the key size. In *RSALib*, the primorial number used is of similar size to p, and the values of a and k are low in entropy. The modulus is expressed as:

$$N = (kM + g^a \mod M)(lM + g^b \mod M)$$

for $a, b, k, l \in \mathbb{Z}$. This can be simplified to:

$$N \equiv g^{a+b} \equiv g^c \mod M$$

for some $c \in \mathbb{Z}$, as all multiples of kM or lM reduce to $0 \mod M$.

Computing the discrete logarithm of $g^c \mod M$ is feasible due to the smoothness of $|M|$ and the effectiveness of the Pohlig-Hellman algorithm [17]. With c, we can establish a boundary for the value of a as follows:

$$\frac{c}{2} \leq a \leq \frac{c + ord}{2}$$

where ord is the multiplicative order of the generator g within Z_M.

Once the bounds for a are known, it becomes possible to determine the value of k using Coppersmith's method [6] for each plausible a.

The conceptual flow of the algorithm is outlined below:

Algorithm 4 Exploit idea

Input: N, M, g, ord, X
$c = \log_g N \mod M$
for $a \leftarrow (\frac{c}{2}, \frac{c+ord}{2})$ **do**
    $f(x) = xM + (g^a \mod M)$
    $g(x) = \frac{1}{M} f(x)$ [make monic, the roots are the same]
    $k = \text{coppersmith}(g(x), N, X)$ [find k]
    $p = kM + (g^a \mod M)$ [candidate]
    if $N \mod p = 0$ **then**
        return p
    end
end

However, in practice, the bound of a is too large, leading the authors to devise a method to transpose the problem to a smaller M' since the bit size of M is more than sufficient to make the attack viable with Coppersmith's algorithm. Their optimization finds a new M' such that:

- (p, q) maintain the form specified in 4.2.4, hence M' must be a divisor of M.
- Coppersmith's algorithm can determine k' for the accurate guess of a'.
- A smaller order of g in $\mathbb{Z}_{M'}$ is found to reduce the size of a'.

We successfully replicated the attack in SageMath against **RSA**-512, using M' as specified by Bruno Produit [18] in his thesis and implementing David Wong's version of Coppersmith's attack [25].

Chapter 5

ECDSA Cryptanalysis

5.1 ECDSA Introduction

5.1.1 Algorithm

ECDSA is a variant of the Digital Signature Algorithm (**DSA**) that incorporates elliptic curve cryptography. It relies on three public parameters:

- The elliptic curve E.
- The generator point G.
- The order of the generator n.

Additionally, a private key $d \in [1, n-1]$ and a corresponding public key $Q = dG$ are required. To **sign** a message m:

1. Compute $h = \text{MSB}(\text{HASH}(m))$, where HASH is a cryptographic hash function, such as SHA-256, and MSB extracts the n most significant bits.
2. Select a random integer $k \in [1, n-1]$ (nonce).
3. Calculate $P = kG = (x_1, y_1)$ and set $r = x_1$; if $x_1 = 0$, repeat the steps.
4. Compute $s = k^{-1}(h + dr) \mod n$; if $s = 0$, repeat the steps.
5. The signature comprises (r, s).

To **verify** the signature:

1. Compute $h = \text{HASH}(m)$.
2. Calculate $u_1 = hs^{-1} \mod n$ and $u_2 = rs^{-1} \mod n$.
3. Compute $P = u_1G + u_2Q = (x_1, y_1)$; if $P = O$, the signature is invalid.
4. If $r \equiv x_1 \mod n$, the signature is valid.

5.1.2 Security

The security of **ECDSA** hinges on the discrete logarithm problem. Given the points Q and G such that $Q = kG$, it is a challenging problem to discover k. However, similar to RSA, ECDSA is susceptible to various implementation attacks:

- If the same k is used for generating two different signatures, it is possible to recover the private key d. This vulnerability was notably exploited in the PlayStation 3.
- The private key d can be compromised if the nonce k is not produced by a cryptographically secure pseudo-random number generator.
- If a non-standard (custom) elliptic curve is used, the discrete logarithm problem may become tractable, potentially undermining the security of the system [7], [10], [2].

5.2 Lattices against ECDSA

The lattice-based attacks discussed here derive from the *Hidden Number Problem (HNP)*, initially introduced by Boneh and Venkatesan [5]. This concept was later extended to demonstrate that even partial information about the nonce k, specifically its most significant bits, can compromise the entire nonce [16]. In 2020, researchers successfully extracted the most significant bits of *DH* and *DHE* secrets in **TLS** $\leq$ **1.2** and employed a lattice construction similar to the one described here to solve the *HNP* and thereby breach the protocol [13] (Raccoon Attack).

5.2.1 Recover d from a Pair of Signatures with Small k_1, k_2

Problem setup. Let $p = \texttt{0xffffffffffffd21f}$, and consider an elliptic curve $E : y^2 = x^3 + 3$ over $\mathbb{F}_p$ with the generator $G = (\texttt{0xcc3b3d1a0c4938ef}, \texttt{0x4ab35ff66f8194fa})$ of order $n = \texttt{0xfffffffefa23f437}$.

We analyze two signatures:

$$(r_1, s_1) = (\texttt{0x269fa43451c5ff3c}, \texttt{0x1184ec0a74d4be7c}) \\ \text{and hash } h_1 = \texttt{0xb526aef1a341cfe6}$$

$$(r_2, s_2) = (\texttt{0xf77cda14f5bf50a2}, \texttt{0xcd1143ccc1516b02}) \\ \text{and hash } h_2 = \texttt{0x84768ddee659efea}$$

Both signatures use a 32-bit nonce k, whereas n is a 64-bit number. We can establish a system of equations:

$$s_1 \equiv \frac{h_1 + dr_1}{k_1} \mod n$$

$$s_2 \equiv \frac{h_2 + dr_2}{k_2} \mod n$$

Although d, k_1, k_2 are unknown, we can express:

$$d = \frac{s_1 k_1 - h_1}{r_1}$$

and rewrite the equation as:

$$k_1 - s_1^{-1} s_2 r_1^{-1} k_2 + s_1^{-1} r_1 h_2 r_2^{-1} - s_1^{-1} h_1 \equiv 0 \mod n$$

To simplify, we denote $t = -s_1^{-1} s_2 r_1^{-1} k_2$ and $u = s_1^{-1} r_1 h_2 r_2^{-1} - s_1^{-1} h_1$. This leads to:

$$k_1 + t k_2 + u \equiv 0 \mod n$$
$$\text{or}$$
$$-k_1 = t k_2 + u - xn \text{ for some } x$$

Given that $|k_1|, |k_2| < K = 2^{32}$.

Lattice construction. We construct the lattice $B = (b_0, b_1, b_2)$:

$$B = \begin{pmatrix} n & 0 & 0 \\ t & 1 & 0 \\ u & 0 & K \end{pmatrix}$$

Note that the vector $v = (-k_1, k_2, K)$ lies within the lattice because:

$$v = -xb_0 + k_2 b_1 + b_2 = (-xn + tk_2 + u = -k_1, k_2, K)$$

for some integers x, k_2.

Can we ascertain that v is a short vector discoverable through LLL? We have:

$$\|v\| = \sqrt{k_1^2 + k_2^2 + K^2} \leq \sqrt{3K^2} = \sqrt{3}K$$

The expected length of the shortest vector is:

$$\begin{aligned} &\approx 2^{\frac{1}{2}} (\det B)^{\frac{1}{3}} \\ &\approx 2^{\frac{1}{2}} (nK)^{\frac{1}{3}} \end{aligned}$$

Thus, the criterion:

$$\|v\| \leq \sqrt{3}K < \sqrt{2}(nK)^{\frac{1}{3}}$$

is satisfied when $K < (nK)^{\frac{1}{3}}$ or $K < \sqrt{n}$.
Applying LLL to the second row, we extract:

$$v = (-k_1, k_2, K) = (\texttt{-0x50a65330, 0x1f5b977a, 0x100000000})$$

To recover d, we compute:

$$d = r_1^{-1}(k_1 s_1 - h_1) = \texttt{0xf00e5fb275bfd304}$$

5.2.2 Recover d from Many Signatures with Small k_i

Suppose we have multiple signatures $(r_1, s_1), \ldots, (r_m, s_m)$ along with their corresponding message hashes $h_1, \ldots, h_m$. We can express the relationship: $s_i \equiv k_i^{-1}(h_i + dr_i) \mod n$ for $i = 1, \ldots, m$, and eliminate d as previously to obtain:

$$\begin{aligned} k_1 + t_1 k_m + u_1 &\equiv 0 \mod n \\ k_2 + t_2 k_m + u_2 &\equiv 0 \mod n \\ &\vdots \\ k_{m-1} + t_{m-1} k_m + u_{m-1} &\equiv 0 \mod n \end{aligned}$$

We then construct the lattice B as follows:

$$B = \begin{pmatrix} n & & & & & \\ & n & & & & \\ & & \ddots & & & \\ & & & n & & \\ t_1 & t_2 & \cdots & t_{m-1} & 1 & 0 \\ u_1 & u_2 & \cdots & u_{m-1} & 0 & K \end{pmatrix}$$

This lattice includes the vector $v = (-k_1, -k_2, \ldots, k_m, K)$, which can be identified using LLL.

5.2.3 Recover d Knowing MSB Bits of Each k_i

What if the most significant bits (MSBs) of each k_i are known?

In such cases, we express $k_i = (a_i + b_i)$, where a_i is the known part, and b_i is the unknown part satisfying $|b_i| < K$. Incorporating these values into the equivalences yields:

$$\begin{gathered} k_i + t_i k_m + u_i \equiv 0 \mod n \\ (a_i + b_i) + t_i(a_m + b_m) + u_i \equiv 0 \mod n \\ b_i + t_i b_m + a_i + t_i a_m + u_i \equiv 0 \mod n \end{gathered}$$

We then define $u'_i = a_i + t_i a_m + u_i$ as the new term in the lattice 5.2.2 in place of u_i. Consequently, the vector:

$$v = (-b_1, -b_2, \ldots, b_m, K)$$

resides within the lattice and can be detected using LLL. To restore the original nonces $(k_1, \ldots, k_m)$, we simply add the vector $w = (a_1, \ldots, a_m)$ to v.

Without specific optimizations, we successfully retrieved the nonces using only the 8 MSB bits on the curve `secp256r1` given 100 signatures, achieving over a 50% success rate with 7 MSB bits known.

Further refinements to enhance the efficacy of this attack, allowing for fewer known bits, have been demonstrated in the findings by Albrecht and Heninger [1].

Conclusion

The results from implementing these attacks are satisfactory and applicable to real-world scenarios. The success of these attacks often relies on a combination of probability and heuristic approaches, which are not discussed in detail here. As with other branches of mathematics, not all theorems and properties prove immediately useful upon their introduction. However, they may become valuable later, as illustrated by the Hidden Number Problem, which was crucial for the Raccoon Attack.

The exploration of how to exploit side-channel attacks, such as knowing some bits of p in **RSA**, has proven fruitful, as evidenced by the **ROCA** vulnerability. The most commonly used cryptographic protocols are generally considered secure; however, improper implementation can easily introduce vulnerabilities. Side-channel attacks, in particular, can significantly undermine the se

Bibliography

[1] Martin R. Albrecht and Nadia Heninger. On bounded distance decoding with predicate: Breaking the "lattice barrier" for the hidden number problem. Cryptology ePrint Archive, Report 2020/1540, 2020. `https://eprint.iacr.org/2020/1540`.

[2] Araki, Satoh, Semaev, and Smart. Smart-ass attack: For elliptic curves e/fp of size p, the ecdlp can be solved very efficiently, 1997.

[3] L. Babai. On lovász' lattice reduction and the nearest lattice point problem. *Combinatorica*, 6(1):1–13, Mar 1986.

[4] Dan Boneh. Twenty years of attacks on the rsa cryptosystem, 1999. `https://crypto.stanford.edu/~dabo/papers/RSA-survey.pdf`.

[5] Dan Boneh and Ramarathnam Venkatesan. Hardness of computing the most significant bits of secret keys in diffie-hellman and related schemes. In Neal Koblitz, editor, *Advances in Cryptology — CRYPTO '96*, pages 129–142, Berlin, Heidelberg, 1996. Springer Berlin Heidelberg.

[6] Don Coppersmith. Finding a small root of a bivariate integer equation; factoring with high bits known. In Ueli Maurer, editor, *Advances in Cryptology — EUROCRYPT '96*, pages 178–189, Berlin, Heidelberg, 1996. Springer Berlin Heidelberg.

[7] Kenneth Giuliani. Attacks on the elliptic curve discrete logarithm problem, 1999.

[8] Jeffrey Hoffstein, Jill Pipher, and J.H. Silverman. *An Introduction to Mathematical Cryptography*. Springer Publishing Company, Incorporated, 2 edition, 2014.

[9] H.W. jr. Lenstra, A.K. Lenstra, and L. Lovász. Factoring polynomials with rational coefficients. *Mathematische Annalen*, 261:515–534, 1982.

[10] Florian Luca, David Mireles Morales, and Igor Shparlinski. Mov attack in various subgroups on elliptic curves. *Illinois Journal of Mathematics - ILL J MATH*, 48, 07 2004.

[11] Kelby Ludwig. Building lattice reduction (lll) intuition, 2017. `https://kel.bz/post/lll/`.

[12] Alexander May. *Using LLL-Reduction for Solving RSA and Factorization Problems*, pages 315–348. Springer Berlin Heidelberg, Berlin, Heidelberg, 2010.

[13] Robert Merget, Marcus Brinkmann, Nimrod Aviram, Juraj Somorovsky, Johannes Mittmann, and Jörg Schwenk. Raccoon attack: Finding and exploiting most-significant-bit-oracles in tls-dh(e). Cryptology ePrint Archive, Report 2020/1151, 2020. `https://eprint.iacr.org/2020/1151`.

[14] Gabrielle De Micheli and Nadia Heninger. Recovering cryptographic keys from partial information, by example. Cryptology ePrint Archive, Report 2020/1506, 2020. `https://eprint.iacr.org/2020/1506`.

[15] Matus Nemec, Marek Sys, Petr Svenda, Dusan Klinec, and Vashek Matyas. The Return of Coppersmith's Attack: Practical Factorization of Widely Used RSA Moduli. In *24th ACM Conference on Computer and Communications Security (CCS'2017)*, pages 1631–1648. ACM, 2017.

[16] Phong Q. Nguyen and Igor E. Shparlinski. The insecurity of the elliptic curve digital signature algorithm with partially known nonces. *Des. Codes Cryptography*, 30(2):201–217, September 2003.

[17] S. Pohlig and M. Hellman. An improved algorithm for computing logarithms over gf(p) and its cryptographic significance (corresp.). *IEEE Trans. Inf. Theory*, 24:106–110, 1978.

[18] Bruno Produit. Optimization of the roca (cve-2017-15361) attack, 2019.

[19] Oded Regev. Lll algorithm, 2004. `https://cims.nyu.edu/~regev/teaching/lattices_fall_2004/ln/lll.pdf`.

[20] Oded Regev. On lattices, learning with errors, random linear codes, and cryptography. In *Proceedings of the Thirty-Seventh Annual ACM Symposium on Theory of Computing*, STOC '05, page 84–93, New York, NY, USA, 2005. Association for Computing Machinery.

[21] Grant Sanderson. Dot product plot, 2016. https://youtu.be/LyGKycYT2v0?t=105.

[22] Grant Sanderson. Geometric explanation of determinant, 2016. https://www.youtube.com/watch?v=Ip3X9LOh2dk.

[23] Giovanni Di Santi. Introduction to lattices attack, 2021. https://github.com/meowmeowxw/lattice-based-attacks.

[24] M.J. Wiener. Cryptanalysis of short rsa secret exponents. *IEEE Transactions on Information Theory*, 36(3):553–558, 1990.

[25] David Wong. Lattice based attacks on rsa, 2015. https://github.com/mimoo/RSA-and-LLL-attacks.