Introduction to Lattices Attacks
“Cryptography”

Di Santi Giovanni

June 3, 2021

Contents

Chapter 1

Introduction

Lattices are powerful mathematical objects that can be used in computer science and mathematics to solves an extensive range of different problems. In the recents years many post-quantum cryptographic protocols candidates involve the use of hard lattices problems, for example the **Learning With Errors** [7]. Lattice constructions can also be used in cryptanalysis to break cryptographic schemes, typically using reduction algorithms as **LLL** [3].

To introduce the mathematics of lattices a basic linear algebra background is needed, so the second chapter is a summary of the basics properties needed to understand it. We used as main resource "An Introduction to Mathematical Cryptography"[2] to explain them, but we also tried to gives some geometrical intuition behind some operations.

Cryptographic protocol as **RSA** and **ECDSA** are considered secure, however there are various attacks against a relaxed-model of each of them, i.e., What if we know some bits of the secret key? What if we know a part of the message before the encryption? We present an introduction to the attacks which involves lattices against these protocols. The code used to confirm the correctness of the attacks is available on the repository [10] and the main reference was "Recovering cryptographic keys from partial information, by example"[5].

Chapter 2

Linear Algebra Background

We'll start with some definitions and properties needed to understand the lattices, to move later on the mathematics behind them and some hard problems related to it. At the end of the chapter the reader should be able to follow the math used in the other chapters.

2.1 Vector Spaces

Definition 2.1.1 ***Vector space***.

A *vector space* V is a subset of $\mathbb{R}^m$ which is closed under finite vector addition and scalar multiplication, with the property that

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

Definition 2.1.2 ***Linear Combinations***

Let $v_1, v_2, \dots, v_k \in V$. A *linear combination* of $v_1, v_2, \dots, v_k \in V$ is any vector of the form

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

Definition 2.1.3 ***Lineaer Independece***

A set of vectors $v_1, v_2, \dots, v_k \in V$ is *linearly independent* if the the only way to get

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

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

Definition 2.1.4 ***Bases***

Taken a set of linearly independent vectors $b = (v_1, \ldots, v_n) \in V$ we say that b is a *basis* of V if $\forall w \in V$ we can write

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

Definition 2.1.5 ***Vector's length***

The vector's length or *Euclidean norm* of $v = (x_1, x_2, \ldots, x_m)$ is

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

Definition 2.1.6 ***Dot Product***

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

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

where θ is the angle between v and w if we place the starting points of the vectors at the origin O.

Geometrically speaking $v \cdot m$ is the length of w projected to v multiplied by the length of v as shown in 2.1

Definition 2.1.7 ***Ortoghonal Basis***

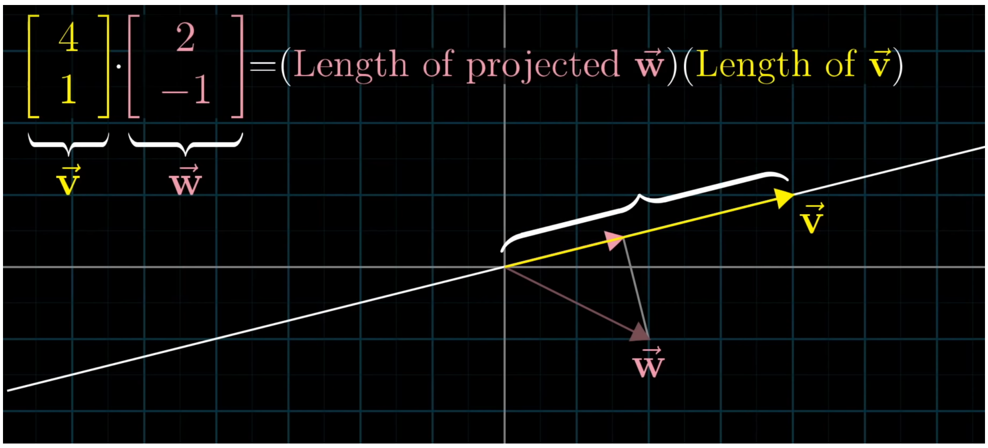

Figure 2.1: Dot Product By 3Blue1Brown [9]

An *ortoghonal basis* for a vector space V is a basis $v_1, \ldots, v_m$ with the property that

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

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

Algorithm 1 Gram-Schmidt Algorithm

$v_1^* = v1$
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

Let $b = (v_1, \ldots, v_n)$, be a basis for a vector space $V \subset \mathbb{R}^m$. There is an algorithm to create an orthogonal basis $b^* = (v_1^*, \ldots, v_n^*)$. The two bases have the property that $\text{Span}\{v_1, \ldots, v_i\} = \text{Span}\{v_1^*, \ldots, v_i^*\}$ for all $i = 1, 2, \ldots, n$

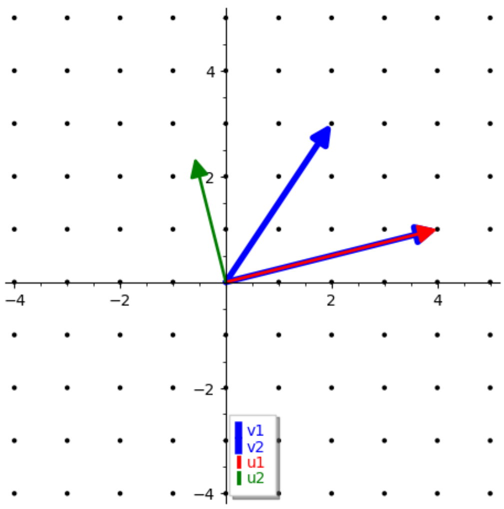

Figure 2.2: Gram Schmidt orthogonalization

If we take $v_1 = (4, 1), v_2 = (2, 3)$ as basis and apply gram schmidt we obtain $u_1 = v_1 = (4, 1), u_2 = (-10/17, 40/17)$ as shown in 2.2

Definition 2.1.8 ***Determinant***

The *determinant* of a square matrix is a function that satisfies the following properties:

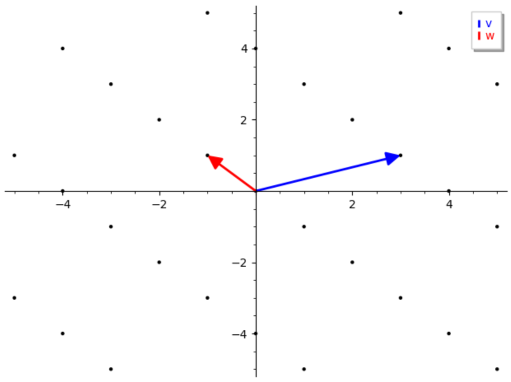

Figure 2.3: Lattice L spanned by v, w

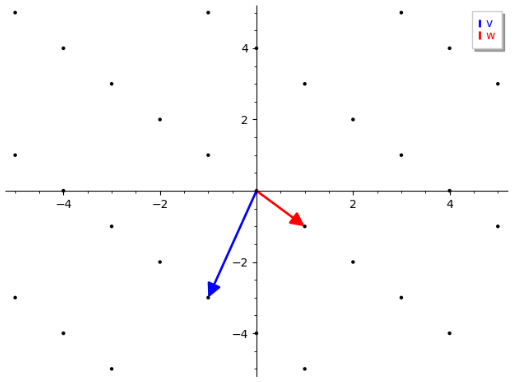

Figure 2.4: Lattice L spanned by v', w'

1. The determinant is linear in each row.
2. The determinant reverses sign if two rows are interchanged.
3. The determinant of the identity matrix is equal to 1.

If the determinant of a matrix is 0 then the matrix is called **singular** (without an inverse).

Grant Sanderson created fantastic animations to understand the geometrical intuition behind the *determinant*[8].

2.2 Lattices

Definition 2.2.1 ***Lattice***

Let $v_1, \ldots, v_n \in \mathbb{R}^m, m \geq n$ be linearly independent vectors. A *Lattice* L spanned by $\{v_1, \ldots, n_n\}$ is the set of all integer linear combinations of $v_1, \ldots, v_n$.

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

If v_i for every $i = 1, \ldots n$ has integer coordinates then the lattice is called *Integral Lattice*.

On the figure 2.3 we show a lattice L with bases $v = (3, 1)$ and $w = (-1, 1)$, and on 2.4 the same lattice L with a different basis.

2.3 Problems

2.3.1 SVP

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

Gauss's developed an algorithm to find an optimal basis for a two-dimensional lattice given an arbitrary basis. The output of the algorithm gives the shortest nonzero vector in L and in this way solves the `SVP`.

Algorithm 2 Gauss Basis Reduction

```
while true do
    if ||v2|| < ||v1|| then
        swap v1, v2
    end
    m = ⌊(v1 · v2)||v1||^(-2)⌉
    if m = 0 then
        return v1, v2
    end
    v2 = v2 − m v1
end
```

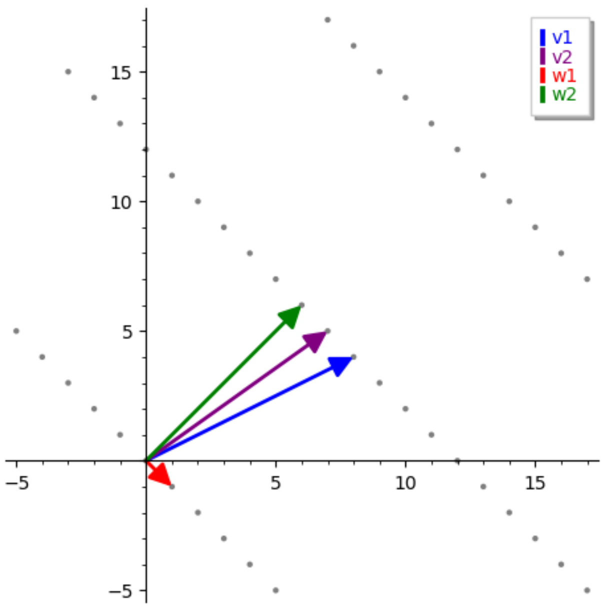

Figure 2.5: Gauss reduction

Example. Let L a lattice spanned by $v_1 = (8,4), v_2 = (7,5)$, if we apply the gauss reduction algorithm we obtain $w_1 = (1,-1), w_2 = (6,6)$ 2.5. w_1 is the shortest nonzero vector in the lattice L.

The bigger the dimension of the lattice, the harder is the problem and there isn't a polynomial algorithm to solve it.

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, in other words find a vector $v \in L$ that minimizes the Euclidean norm $\|t - v\|$.*

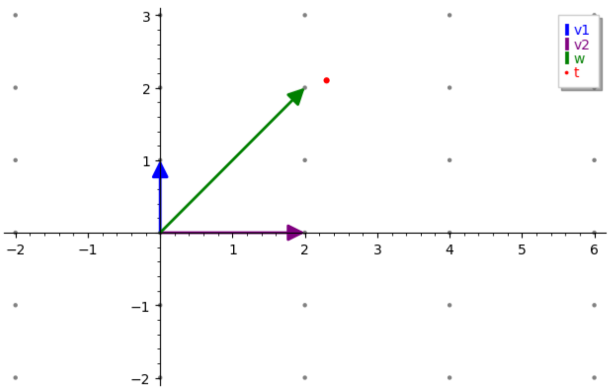

Figure 2.6: CVP

Example. Let L a lattice spanned by $v_1 = (0,1), v_2 = (2,0)$, and a target vector $t = (2.3, 2.4)$. The closest vector to t in the lattice L is $w = (2,2)$.

Both **CVP** and **SVP** are $\mathcal{NP}$-hard problems, thus the complexity of the problem increments with the dimension of the lattice. They can be used as primitive to create a trapdoor functions for a cryptographic scheme.

There are various algorithms that can output a solution under certain constraints for an approximation of both **SVP** and **CVP**, for example **LLL**[3] and **Babai's algorithm**[1] respectively.

Chapter 3

LLL

3.1 Introduction

The **Lenstra-Lenstra-Lovász** *LLL* or L^3 is a polynomial time algorithm to find a "shorter" basis.

Theorem 3.1.1 ***LLL***

Let $L \in \mathbb{Z}^n$ be a lattice spanned by $B = \{v_1, \ldots, v_n\}$. The LLL algorithm outputs a reduced lattice basis $\{w_1, \ldots, w_n\}$ with

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

in time polynomial in n and in the bit-size of the entries of the basis matrix B.

Basically the first vector of the new basis will be as short as possible, and the other will have increasing lengths. The new vectors will be as orthogonal as possible to one another, i.e., the dot product $w_i \cdot w_j$ will be close to zero.

Example

For example we can take the following basis (the rows are the vector) that span a lattice L.

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

Applying the LLL algorithm we obtain

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

Where the first row is the shortest vector in the lattice L, and so solves the **SVP** problem. For higher dimensions however the LLL algorithm outputs only an approximation for the **SVP** problem.

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 $(v_i \cdots v_j^*)/\|v_j^*\|^2$.
Output reduced basis $v_1, \ldots, v_n$

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

Kelby Ludwig explains an intuition behind every steps of the algorithm [4], and Oded Regev gives a more rigorous explanation of the algorithm and the properties [6].

3.3 Applications

There are many applications of LLL

1. Factoring polynomials over the integers. For example, given $x^2 - 1$ factor it into $x + 1$ and $x - 1$.
2. Integer Programming. This is a well-known **NP**-complete problem. Using LLL, one can obtain a polynomial time solution to integer programming with a fixed number of variables.
3. Approximation to the **CVP** or **SVP**, as well as other lattice problems.
4. Application in cryptanalysis.

Chapter 4

RSA Cryptanalysis

4.1 RSA Introduction

4.1.1 Algorithm

RSA is one of the earliest and most used asymmetric cryptosystem. The usual step to generate a public/private key for **RSA** is the following

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, i.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 the modulo N and we don't have a polynomial algorithm to factor it, so it's considered secure. However there are different attacks against textbook RSA or bad implementations:

- If $m < N^{\frac{1}{e}}$, then $m^e < N$ and the modulo operation is not applied and we only need to find the eth root of c over the integers to find m.
- If $q = p+x$ where x is small enough than it's easy to recover the factors of N computing $A = \sqrt{N}$ and compute $p = A - x$ for different value of x until $N \mod p = 0$, then we have found $q = N/p$
- Many more attacks can be found TODO.

4.2 Lattices against RSA

We want to attack a relaxed model of RSA where we know a part of the message m. We start introducing the **??** coppersmith attack and the math behind it.

4.2.1 Mathematical introduction

It's easy to find the roots of a univariate polynomial over the integers. Finding the roots of **modular** polynomial is hard, example:

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

Suppose N is an **RSA** modulus and we don't know the factorization of it. Let's have an univariate integer polynomial $f(x)$ with 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 showed how we can 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 an univariate polynomial with n monomials and m be a positive integer. If we have some restraint X and the following equations hold

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

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

Then $g(x_0) = 0$ holds over the integers.

This theorem states that is possible to compute the root of $f(x) \mod N$ if we can find a polynomial $g(x)$ that share the same root but modulo N^m. If 4.1 and 4.2 hold then we can simply compute the root of $g(x)$ over the integers to have the same root x_0 such that $f(x_0) \equiv 0 \mod N$.

Howgrave-Graham's idea is to find this polynomial g by combining polynomials p_i who also have x_0 as roots modulo N^m.

The **LLL** algorithm is fundamental because:

- It only does **integer linear operations** on the basis vectors. In this way even if the basis is different it's only a linear combination of vector that still have x_0 as root modulo N^m.
- If we craft the lattice properly, the norm of shortest vector on the reduced basis will satisfy 4.2. We know the length's bound of the shortest vector that LLL could find.

We can easily create 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:

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

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

Applying **LLL** to a lattice constructed by the coefficients of 4.3 we'll find a short vector $v = g(xX)$ that will satisfy 4.2. If we then take $g(x)$ we will be able to compute the root over the integer.

4.2.2 Example

We have a 100-bit **RSA** modulus

$$N = 0xf046522fb555a90bdc558fc93 \text{ and } e = 3.$$

Before the encryption the message m is padded as

$$z = pad||m = 0x74686973206b65793a||m$$

where || is the concatenation. The padding is the ascii encoding of "this key:"

The ciphertext is

$$c = z^e \mod N = 0x5b603cda4b72100c6f25954fc$$

Suppose that we don't know the factorization of N and we would like to know the message m. However we know the padding and that the length of $m < 2^{16}$.

Let's define

$$a = 0x74686973206b65793a0000.$$

which is the known padding string that got encrypted.

Thus we have that $c = (a+m)^3 \mod N$, for an unkown small m. We can define $f(x) = (a+x)^3 - c$, and so we setup the problem to find a small root m such that $f(m) \equiv 0 \mod N$

$$f(x) = x^3 + 0x15d393c596142306bae0000x^2 + 0x1b53c5e184a49b39f9ad9eedbx \\ +0x486a5d936fb568185c8ff0506$$

Lattice contruction. Let the coefficients of f be $f(x) = x^3 + f_2x^2 + f_1x + f_0$ and $X = 2^{16}$ be the upper bound of the size of the root m. We can construct the matrix

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

The rows of the matrix correspond to the coefficient vectors of the polynomials $f(x), Nx^2, Nx$ and N, furthermore we know that each polynomials will be 0 modulo N if evaluated at $x = m$. We applied Howgrave-Graham with $m = 1$ (the N^m parameter not the message).
With this lattice construction every vector is of the form $v = (v_1X^3, v_2X^2, v_1X, v_0)$,

because any integer linear combination of the vector of the lattice will keep the bound X^i for $i = 0, .., \dim(B) - 1$.

Apply LLL. We then apply LLL to find the shortest vector of the reduced basis:

$$v = (0x90843131bc53X^3 + 0x2736f60b1c7ba3294X^2,$$
$$-0x1bec331b20625341b6d73X, 0x47336b98335c143ac912ec9e)$$

We can construct the polynomial g using the coefficients of v

$$g(x) = 0x90843131bc53x^3 + 0x2736f60b1c7ba3294x^2$$
$$-0x1bec331b20625341b6d73x + 0x47336b98335c143ac912ec9e$$

We know that

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

What we need to prove is that

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

In this example, $\det B = X^6 N^3$, and LLL will find a short vector with $\|v\| \leq 2^{\frac{n(n-1)}{4(n)}} (\det B)^{\frac{1}{n}}$. If we ignore the $2^{\frac{3}{4}}$ factor (remember that $n = 4$), then we need to satisfy

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

We have $(\det B)^{\frac{1}{4}} = (X^6 N^3)^{\frac{1}{4}} < \frac{N}{\sqrt{4}}$, if we solves for X this will be satisfied when $X < (\frac{N}{16})^{\frac{1}{6}}$. With the numbers we have this inequality is verified, however even if the bound of the shortest vector is larger we still have some possibilities to find the correct root.

If we compute the root of $g(x)$ over the integers we obtain $m = 0x6162$ which is the correct result.

This specific lattice works to find roots up to size $N^{\frac{1}{6}}$, so the same construction will work if we want to find

- $\approx$ 170 unkown bits of message from an RSA 1024-bit modulus
- $\approx$ 341 unkown bits of message from an RSA 2048-bit modulus

- ≈ 683 unkown bits of message from an RSA 4096-bit modulus

To compute bigger root a bigger lattice with more polynomials generated with 4.3 is needed, this method is better described in TODO, but the principles are the same.

4.2.3 Factorize modulus with known MSB bits of p

Knowing the most significat bits of p can be used to factor the modulus N ref TODO coppersmith.

Let $a = 2^l b$ where b are the bits known of p, then $p = a + r$ for some small value of r.

Using as lattice basis

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

And applying LLL we can find r.

Chapter 5

ECDSA Cryptanalysis

5.1 ECDSA Introduction

5.1.1 Algorithm

ECDSA is a variant of the Digital Signature Algorithm (**DSA**) which uses elliptic curve cryptography. To digitally sign a message we have 3 public parameters

- The elliptic curve E.
- The generator point G.
- The generator's order n.

We also need to create a private key $d \in [1, n-1]$ and public key $Q = dG$. To digitally **sign** a message m:

1. Compute $h = \text{HASH}(m)$ where HASH is a cryptographic hash functions.
2. Select a random integer $k \in [1, n-1]$.
3. Calculate $P = kG = (x_1, y_1)$ and set $r = (x_1)$.
4. Compute $s = k^{-1}(h + dr) \mod n$, if $s = 0$ repeat the steps.
5. The signature is composed by (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 $r \equiv x_1 \mod n$ then the signature is valid.

5.1.2 Security

The security of **ECDSA** depends on the discrete logarithm problem. Given the points Q, G such that $Q = k * G$ it's considered an hard problem to find k. However as RSA there are lots of implementation attacks

- If the same k is used to generate two different signatures it's possible to recover the secret key d. This was a real bug discovered in the Playstation 3.
- It's possible to recover d if the parameter k is not generated with a cryptographically secure pseudo random generator.
- If the elliptic curve used is not standardized (custom) then it's possible that the discrete logarithm is easily solvable.

5.2 Lattices against ECDSA

5.2.1 Recover d from a pair of signatures with small k_1, k_2

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

We have two signature:

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

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

And we know that both of these signatures use 32-bit nonce k, note that n is a 64-bit number. We can set up the problem as a system of equation

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

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

We don't know d, k_1, k_2, but we can write

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

and rewrite the equation in

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

To simplify the equation we write $t = -s_1^{-1} s_2 r_1^{-1} k_2$ and $u = s_1^{-1} r_1 h_2 r_2^{-1} - s_1^{-1} h_1$. In this way we have

$$k_1 + tk_2 + u \equiv 0 \mod n$$
$$\text{or}$$
$$-k_1 = tk_2 + u - xn \text{ for some } x$$

We know 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 $v = (-k_1, k_2, K)$ is in the lattice because

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

for some $x, k_2 \in Z$.

Can we prove that v is a short vector that we can find with LLL? We have

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

And we expect the shortest vector to have length

$$\approx 2^{\frac{1}{2}} (\det B)^{\frac{1}{3}}$$
$$\approx 2^{\frac{1}{2}} (nK)^{\frac{1}{3}}$$

So we want that

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

And if we remove the smaller terms this will be satisfied when $K < (nK)^{\frac{1}{3}}$ or $K < \sqrt{n}$.

In this case if we apply LLL in the second row we obtain:

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

To retrieve d we just need to compute

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

Source of the method:

5.2.2 Recover d from many signatures with small k_i

Suppose we have many signatures $(r_1, s_1), \ldots, (r_m, s_m)$ with message hashes $h_1, \ldots, h_m$. We can write the equivalence $s_i \equiv k_i^{-1}(h_i + dr_i) \mod n$ for $i = 1, \ldots, m$ and we can remove d as before to get

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

And create the lattice B as

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

Same as before this lattice contains $v = (-k_1, -k_2, \ldots, k_m, K)$ and with probability TODO we can find that vector after applying LLL.

5.2.3 Recover d knowing MSB bits of each k_i

What if we know the firsts MSB bits of each k_i?

Well, we can write $k_i = (a_i + b_i)$, where a_i is the known part and b_i is the unkown part that satisfies $|b_i| < K$. If we plug these values into the equivalence we obtain

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

And we can set $u'_i = a_i + t_i a_m + u_i$ to be the value inside the lattice 5.2.2 instead of u_i. In this way the vector

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

is in the lattice and could be found using LLL. To recover the original $(k_1, \dots, k_m)$ we must add to v the vector $w = (a_1, \dots, a_m)$.

I was able to recover the nonces with the 10 MSB bits known on the curve `secp256r1` given 100 signatures, but it's also possible to recover the nonces with less known bits as shown in paper.

Conclusion

gg^2

Bibliography

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

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

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

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

[5] 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`.

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

[7] 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.

[8] Grant Sanderson. Determinant visualizatons, 2016. `https://www.youtube.com/watch?v=Ip3X9LOh2dk`.

[9] Grant Sanderson. Dot product visualizatons, 2016. `https://youtu.be/LyGKycYT2v0?t=105`.

[10] Giovanni Di Santi. Introduction to lattices attack, 2021. `https://github.com/meowmeowxw/introduction-to-lll`.