

Introduction to LLL “Cryptography”

Di Santi Giovanni

June 1, 2021

Contents

1	Linear Algebra Background	2
1.1	Vector Spaces	2
1.2	Lattices	4
1.3	Problems	5
1.3.1	SVP	5
1.3.2	CVP	5
2	LLL	7
2.1	Introduction	7
2.2	Algorithm	8
2.3	Applications	8
3	Cryptanalysis	9
3.1	RSA Introduction	9
3.1.1	Algorithm	9
3.1.2	Security	10
3.2	Lattices against RSA	10
3.2.1	Mathematical introduction	10
3.2.2	Example	11
3.3	ECDSA Introduction	14
3.3.1	Algorithm	14
3.3.2	Security	15
3.4	Lattices against ECDSA	15
3.4.1	Example	15

Chapter 1

Linear Algebra Background

1.1 Vector Spaces

Definition 1.1.1 *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

$$a_1v_1 + a_2v_2 \in V \text{ for all } v_1, v_2 \in V \text{ and all } a_1, a_2 \in \mathbb{R}$$

Definition 1.1.2 *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

$$\alpha_1v_1 + \alpha_2v_2 + \dots + \alpha_kv_k \text{ with } \alpha_1, \dots, \alpha_k \in \mathbb{R}$$

Definition 1.1.3 *Linear Independence*

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

$$a_1v_1 + a_2v_2 + \dots + a_kv_k = 0$$

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

Definition 1.1.4 *Bases*

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

$$w = a_1v_1 + a_2v_2 + \cdots + a_nv_n$$

Definition 1.1.5 *Vector's length*

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

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

Definition 1.1.6 *Dot Product*

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

$$v \cdot w = x_1y_1 + x_2y_2 + \cdots + x_my_m$$

or

$$v \cdot w = \|v\|\|w\|\cos\theta$$

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

Geometrically speaking $v \cdot w$ is the length of w projected to v multiplied by the length of v as shown in 1.1

Definition 1.1.7 *Orthogonal Basis*

An orthogonal basis for a vector space V is a basis v_1, \dots, v_m with the property that

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

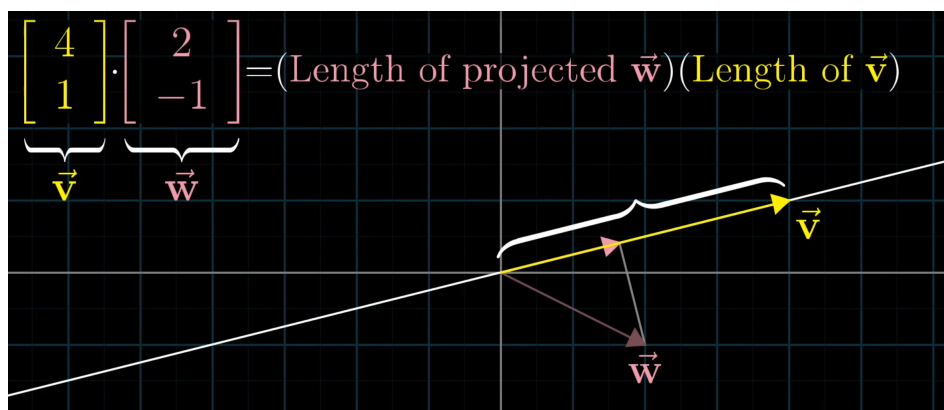


Figure 1.1: Dot Product By 3Blue1Brown

Gram-Schmidt Algorithm

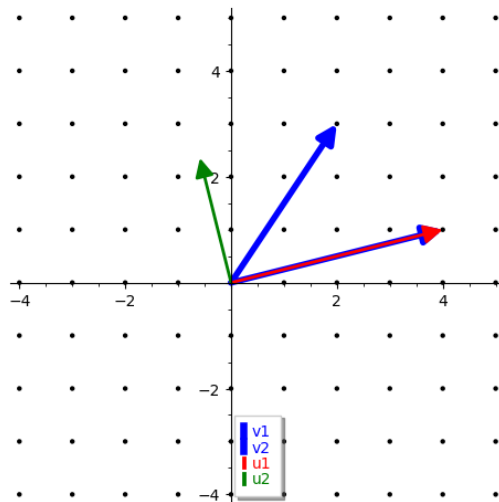


Figure 1.2: Gram Schmidt orthogonalization

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

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

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 1.2

1.2 Lattices

Definition 1.2.1 *Lattice*

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

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

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

On the figure 1.3 we show a lattice L with bases $v = (3, 1)$ and $w = (-1, 1)$, and on 1.4 the same lattice L with a different basis.

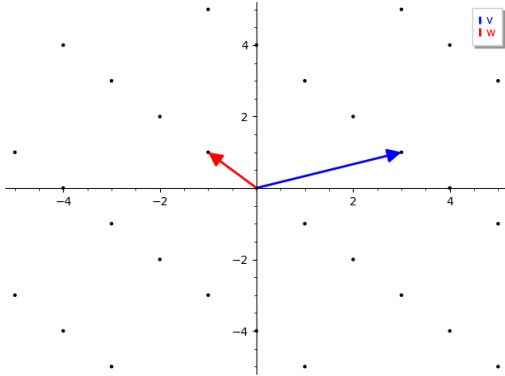


Figure 1.3: Lattice L spanned by v, w

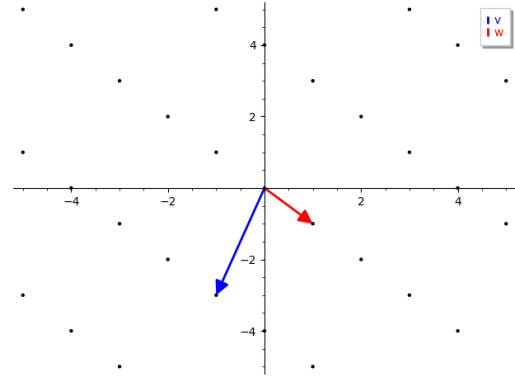


Figure 1.4: Lattice L spanned by v', w'

1.3 Problems

1.3.1 SVP

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

Gauss Reduction

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.

If we take for example $v_1 = (8, 4), v_2 = (7, 5)$ and apply the gauss reduction algorithm we obtain $w_1 = (1, -1), w_2 = (6, 6)$. w_1 is the shortest nonzero vector in the lattice L spanned by v_1, v_2 .

However the bigger the dimension of the lattice, the harder is the problem and there isn't a polynomial algorithm to find such vector.

1.3.2 CVP

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

TODO FIX 1.6 is wrong, the lattice is different.

TODO: CVP and SVP are related.

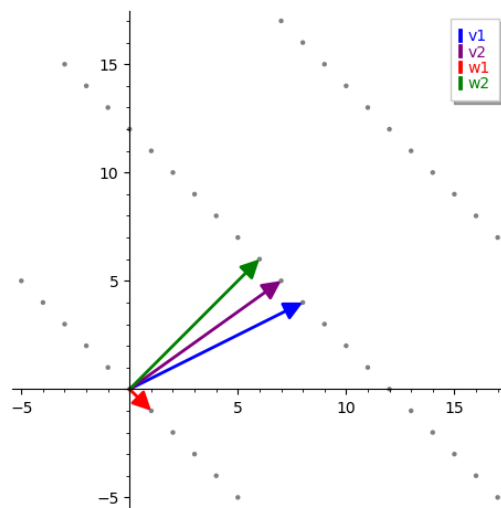


Figure 1.5: Gauss reduction

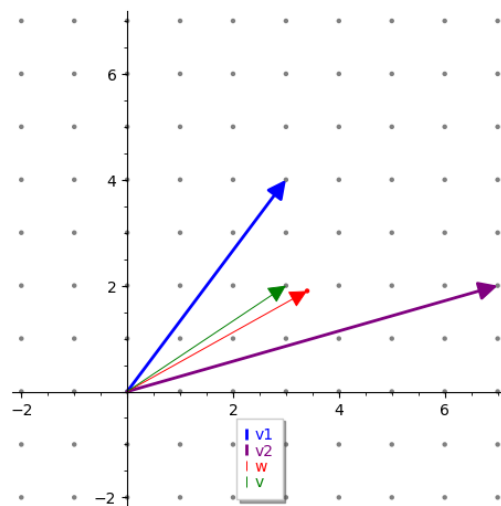


Figure 1.6: CVP

Chapter 2

LLL

2.1 Introduction

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

Theorem 2.1.1 *LLL*

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

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

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.

2.2 Algorithm

TODO: Write algorithm and explain some steps

2.3 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 3

Cryptanalysis

3.1 RSA Introduction

3.1.1 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

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

3.1.2 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 a relaxed model of RSA:

1. If $m < N^{\frac{1}{e}}$, then $m^e < N$ and the modulo operation is not applied and we only need to find the e th root of c over the integers to find m .
2. 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 $N \bmod p = 0$, then we have found $q = N/p$
3. Many more attacks can be found TODO.

3.2 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.

3.2.1 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:

$$f(x) \equiv 0 \pmod{N}$$

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

$$f(x) = x^n + a_{n-1}x^{n-1} + a_{n-2}x^{n-2} + \dots + a_1x + a_0$$

Coppersmith showed how we can recover the value x_0 such that $f(x_0) \equiv 0 \pmod{N}$, with $x_0 < N^{\frac{1}{n}}$ in polynomial time using the following theorem

Theorem 3.2.1 *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

$$g(x_0) \equiv 0 \pmod{N^m}, |x_0| \leq X \quad (3.1)$$

$$\|g(xX)\| < \frac{N^m}{\sqrt{n}} \quad (3.2)$$

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

This theorem states that is possible to compute the root of $f(x) \pmod{N}$ if we can find a polynomial $g(x)$ that share the same root but modulo N^m . If 3.1 and 3.2 hold then we can simply compute the root of $g(x)$ over the integers to have the same root x_0 such that $f(x_0) \equiv 0 \pmod{N}$.

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 3.2. 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 :

$$g_{i,j}(x) = x^j \cdot N^i \cdot f^{m-i}(x) \text{ for } i = 0, \dots, m-1, j = 0, \dots, \delta-1 \quad (3.3)$$

$$h_i(x) = x^i \cdot f^m(x) \text{ for } i = 0, \dots, t-1 \quad (3.4)$$

Applying **LLL** to a lattice constructed by the coefficients of 3.3 we'll find a short vector $v = g(xX)$ that will satisfy 3.2. If we then take $g(x)$ we will be able to compute the root over the integer.

3.2.2 Example

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

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

Before the encryption the message m is padded as

$$z = \text{pad} || m = 0x74686973206b65793a || m$$

where $||$ is the concatenation. The padding is the ascii encoding of “this key:”

The ciphertext is

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

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 $c = (a + m)^3 \bmod N$, for an unknown 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 $f(m) \equiv 0 \bmod N$

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

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 $i = 0, \dots, \dim(B) - 1$.

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

$$g(x_0) \equiv 0 \pmod{N}, |x_0| \leq X$$

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 solve 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

- ~170 unknown bits of message from an RSA 1024-bit modulus
- ~341 unknown bits of message from an RSA 2048-bit modulus

- ~683 unknown bits of message from an RSA 4096-bit modulus

To compute bigger root a bigger lattice with more polynomials generated with 3.3 is needed, this method is better described in TODO, but the principles are the same.

3.3 ECDSA Introduction

3.3.1 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 $s = k^{-1}(h + dr) \pmod n$.

To **verify** the signature:

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

3.3.2 Security

3.4 Lattices against ECDSA

3.4.1 Example

End of Paper

gg^2

Bibliography