TÉCNICO LISBOA

Attacks on ECDSA when the nonces are generated with a weak pseudo random number generator

Pedro Afonso Gama Caldas de Miranda

Thesis to obtain the Master of Science Degree in

Mathematics

Supervisor: Prof. Paulo Alexandre Carreira Mateus

Examination Committee

Chairperson: Prof. Maria Cristina De Sales Viana Serôdio Sernadas
Supervisor: Prof. Paulo Alexandre Carreira Mateus
Member of the Committee: Prof. André Nuno Carvalho Souto

July 2019

Acknowledgments

I would like to thank Instituto Superior Técnico and the great professors that I had during the undergraduate and master's degree. I would especially like to thank professor Paulo Mateus for suggesting the theme for this thesis, and for his help during the process of writing it. I would also like to thank everyone in the examination committee for reading and grading my thesis. Last, but certainly not least, I need to thank my family, for the constant emotional and financial support.

Resumo

Nesta tese, são descritos vários ataques contra o ECDSA. A maioria deles são baseados nos ataques de rede diagonal usados para resolver o ”hidden number problem”. Começamos por apresentar o clássico ataque de rede diagonal contra o ECDSA. Seguidamente, apresentamos dois novos ataques, um que recupera uma equação linear entre duas chaves privadas diferentes e outro que usa a solução para o ”hidden number problem” em duas variáveis para recuperar duas chaves privadas. Também descrevemos como usar o ataque com duas chaves privadas para recuperar uma única chave privada que, segundo os nossos resultados, tem uma maior probabilidade de sucesso que o ataque para uma variável que é classicamente usado. No final da tese, implementamos e testamos estes ataques para a curva secp256k1, a curva usada nas assinaturas de Bitcoins, usando o algoritmo LLL.

Também descrevemos ataques que começam por guardar alguns pontos da curva elíptica numa tabela de dispersão e, depois, usam a operação de grupo da curva elíptica para recuperar o ”nonce” usado numa assinatura.

Keywords: ECDSA, Hidden Number Problem, Rede Diagonal, LLL, BKZ.

Abstract

In this thesis, we describe various attacks against the ECDSA. Most of them are based on the lattice attack used to solve the hidden number problem. We start by describing the usual lattice attack against the ECDSA. Then, we design two new attacks, the first one recovers a linear equation between two different private keys and the second one uses a solution to the two variables hidden number to recover two private keys. Moreover, we describe how to use that two variables lattice attack to design an attack which recovers a single private key, which, according to our results, has a better probability of success than the usual one variable lattice attack. These attacks were implemented and tested for the secp256k1 curve, the curve used in Bitcoin's public key-cryptography, using the LLL algorithm.

We also describe some attacks that start by storing some of the elliptic curve's points in a hash table and then, use the elliptic curve's group operation in order to try to recover the nonce used in a signature.

Keywords: ECDSA, Hidden Number Problem, Lattice, LLL, BKZ.

Contents

Acknowledgments . . . iii
Resumo . . . v
Abstract . . . vii
List of Tables . . . xi
List of Figures . . . xiii
Nomenclature . . . xv
Glossary . . . xvii

1 Introduction **1**
1.1 Related work . . . 1

2 Background **3**
2.1 Elliptic Curves . . . 3
2.2 ECDSA . . . 6
2.3 Lattice . . . 8
2.3.1 Lattice attacks . . . 8

3 Attacks on ECDSA **13**
3.1 Lattice attacks . . . 13
3.1.1 Computing private key through small nonces . . . 13
3.1.2 Computing private key through linear equations between nonces . . . 16
3.1.3 Computing the private key with most significant bits known . . . 17
3.1.4 Computing the private key with least significant bits known . . . 20
3.1.5 Computing linear equation between different private keys . . . 22
3.1.6 Computing two private keys through linear relation between nonces . . . 25
3.2 Group attacks . . . 31
3.2.1 Attack when the same nonce is used twice by the same user . . . 31
3.2.2 Attack when there is a known linear equation between two nonces used by the same user . . . 31
3.2.3 Attack when there is a known linear equation between two nonces used by different users . . . 32
3.2.4 Using a hash table to get a nonce . . . 32

3.2.5 Using a hash table to a get a linear relation between two nonces for the same user 33
3.2.6 Creating an hash table with a multi-core processor 34

4 Results **35**
4.1 One variable lattice attack 35
4.1.1 Fixed limit on the norm of the solution to the equation 36
4.1.2 Fixed number of equations 38
4.2 Two variables lattice attack 40
4.2.1 Fixed limit on the norm of the solution to the equation 41
4.2.2 Fixed number of equations 44
4.3 Comparing methods 46
4.3.1 User with a random solution to the equation 47
4.3.2 User whose solution to the equation is zeros 49

5 Conclusions **53**
5.1 Future Work 53

Bibliography **55**

List of Tables

4.1 Theoretical number of equations needed for one variable lattice attack, as a function of l. 36
4.2 Number of equations to be tested for one variable lattice attack, as a function of l. 36
4.3 Theoretical values of $\log C$ needed as a function of m, for the one lattice attack. 38
4.4 Tested values of $\log C$ as a function of m, for the one lattice attack. 38
4.5 Maximum values of $\log C$ that have at least 90% probability of success for the one variable lattice attack. 39
4.6 Real probability of success for optimal choice of $\log C$ for the one lattice attack. 39
4.7 Theoretical number of equations needed for the two variables lattice attack. 41
4.8 Number of equations tested for the two variables lattice attack. 41
4.9 Theoretical values of $\log C$ needed for the two variables lattice attack. 44
4.10 Values of $\log C$ tested for the two variables lattice attack. 44
4.11 Maximum values of $\log C$ that have at least 90% probability of success for the two variables lattice attack. 44
4.12 Real probability of success for optimal choice of $\log C$ for the two variables lattice attack. . 44
4.13 Experimental optimal values of m as a function of l, for the two variables lattice attack. . . 46
4.14 Theoretical values of n needed as a function of l for the two variables lattice attack. 46
4.15 Tested values of n as a function of l for the two variables lattice attack. 47

List of Figures

4.1 Probability of success as a function of the number equations for the one variable lattice attack. The left figure has $C = 2^{249}$ and the right one has $C = 2^{248}$. 37

4.2 Probability of success as a function of the number equations for the one variable lattice attack. The left figure has $C = 2^{247}$ and the right one has $C = 2^{246}$. 37

4.3 Probability of success as a function of the number equations for the one variable lattice attack with $C = 2^{245}$. 37

4.4 Probability of success as a function of $\log C$, for the one variable lattice attack. The left figure has $m = 2$ and the right one has $m = 3$. 39

4.5 Probability of success as a function of $\log C$, for the one variable lattice attack. The left figure has $m = 4$ and the right one has $m = 5$. 39

4.6 Probability of success as a function of $\log C$, for the one variable lattice attack. The left figure has $m = 6$ and the right one has $m = 7$. 40

4.7 Probability of success as a function of $\log C$, for the one variable lattice attack. The left figure has $m = 8$ and the right one has $m = 9$. 40

4.8 Probability of success as a function of $\log C$, for the one variable lattice attack with $m = 10$. 40

4.9 Probability of success a function of the number of equations, for the two variables lattice attack. The left figure has $l = 10$ and the right one has $l = 11$. 42

4.10 Probability of success a function of the number of equations, for the two variables lattice attack. The left figure has $l = 12$ and the right one has $l = 13$. 42

4.11 Probability of success a function of the number of equations, for the two variables lattice attack. The left figure has $l = 14$ and the right one has $l = 15$. 42

4.12 Probability of success a function of the number of equations, for the two variables lattice attack. The left figure has $l = 16$ and the right one has $m = 17$. 43

4.13 Probability of success a function of the number of equations, for the two variables lattice attack. The left figure has $l = 18$ and the right one has $m = 19$. 43

4.14 Probability of success a function of the number of equations, for the two variables lattice attack with $l = 20$. 43

4.15 Probability of success as a function of $\log C$, for the two variables lattice attack. The left figure has $m = 3$ and the right one has $m = 4$. 45

4.16 Probability of success as a function of $\log C$, for the two variables lattice attack. The left figure has $m = 5$ and the right one has $m = 6$. 45
4.17 Probability of success as a function of $\log C$, for the two variables lattice attack. The left figure has $m = 7$ and the right one has $m = 8$. 45
4.18 Probability of success as a function of $\log C$, for the two variables lattice attack. The left figure has $m = 9$ and the right one has $m = 10$. 46
4.19 Comparison between the one and two variables lattice attack for a user with a random solution to the equation. The left figure has $l = 10$ and the right one has $l = 11$. 47
4.20 Comparison between the one and two variables lattice attack for a user with a random solution to the equation. The left figure has $l = 12$ and the right one has $l = 13$. 48
4.21 Comparison between the one and two variables lattice attack for a user with a random solution to the equation. The left figure has $l = 14$ and the right one has $l = 15$. 48
4.22 Comparison between the one and two variables lattice attack for a user with a random solution to the equation. The left figure has $l = 16$ and the right one has $l = 17$. 48
4.23 Comparison between the one and two variables lattice attack for a user with a random solution to the equation. The left figure has $l = 18$ and the right one has $l = 19$. 49
4.24 Comparison between the one and two variables lattice attack for a user with a random solution to the equation with $l = 20$. 49
4.25 Comparison between the one and two variables lattice attack for a user with a zero solution to the equation. The left figure has $l = 10$ and the right one has $l = 11$. 51
4.26 Comparison between the one and two variables lattice attack for a user with a zero solution to the equation. The left figure has $l = 12$ and the right one has $l = 13$. 51
4.27 Comparison between the one and two variables lattice attack for a user with a zero solution to the equation. The left figure has $l = 14$ and the right one has $l = 15$. 51
4.28 Comparison between the one and two variables lattice attack for a user with a zero solution to the equation. The left figure has $l = 16$ and the right one has $l = 17$. 52
4.29 Comparison between the one and two variables lattice attack for a user with a zero solution to the equation. The left figure has $l = 18$ and the right one has $l = 19$. 52
4.30 Comparison between the one and two variables lattice attack for a user with a zero solution to the equation with $l = 20$. 52

Nomenclature

$\lvert b \rvert_q$	$\min\{b \bmod q, q - b \bmod q\}$.
$b \bmod q$	The integer $y \in [0, q-1]$ such that $y \equiv b \bmod q$.

Chapter 1

Introduction

Due to the increase in the amount of online transactions, digital signatures are being used more frequently than ever. One of the most popular schemes is the Elliptic Curve Digital Signature Algorithm (ECDSA) [1] which is used, for example, in Bitcoin transactions [2]. In order to ensure that the ECDSA is secure against total break attacks, many attacks have been designed that try to recover a user's private key, since if an attacker finds a user's private key he can steal all the money from that user's account. Some of those attacks start by mapping the problem of finding the user's private key to a hidden number problem (HNP) [3] and then, using a lattice reduction algorithm such as LLL [4] or BKZ [5]. These are polynomial time attacks and they can recover the private key if there is a side-channel attack which leaks a portion of the nonces' bits or if there is a known weakness in the pseudorandom number generator (PRNG) used to generate the nonces.

In this thesis we will study two different types of attacks, lattice attacks and group attacks. The former will use the solution to HNP in [6] to recover a private key, a linear equation between two different private keys or two private keys from a list of ECDSA signatures. The latter tries to either recover a private key or a linear equation between two different private keys using a hash table and the elliptic curve's group operation. The attacks that recovers a linear equation between two private keys and the attack which recovers two private keys are two new attacks against the ECDSA.

In chapter 2 we will give some background in elliptic curves, ECDSA, lattices, the HNP for one and two variables and their lattice solution. In chapter 3 we present the attacks and we will prove conditions in which these attacks are successful. In chapter 4 we implement and test the lattice attacks. Lastly, in section 5 we analyze our results and discuss ideas for future works.

1.1 Related work

In [3] Boneh and Venkatesan solved the HNP by mapping it to a closest vector problem (CVP) and then using the LLL and Babai's nearest plane algorithm [7]. Moreover, they proved that if there was an oracle which leaked the most significant bits (MSB) of the secret key in a Diffie-Hellman key-exchange protocol [8], the problem of recovering the secret key can be reduced to the HNP.

In [9] Howgrave-Graham and Smart applied similar techniques to the ones in [3] to attack a 160-bit DSA assuming that there existed an oracle which recovered a portion of the nonces' bits.

In [10] Nguyen and Shparlinski presented the first polynomial-time algorithm that provably recovers the signer's secret ECDSA key when a few bits of the random nonces are known for a number of signatures.

In [11] the authors applied the FLUSH+RELOAD side-channel attack based on cache hits/misses to extract a small amount of data from OpenSSL ECDSA signature requests. Then, they applied a lattice attack to extract the private key. This side-channel attack was able to recover information from almost all of the observed nonces.

In [12] the authors described an attack against nonce leaks in 384-bit ECDSA using an FFT-based attack due to Bleichenbacher [13, 14].

In [6] the authors designed a lattice attack which was able to compute hundreds of Bitcoin private keys and dozens of Ethereum, Ripple, SSH, and HTTPS private keys.

Chapter 2

Background

2.1 Elliptic Curves

An elliptic curve E over a field $\mathbb{F}$ is a smooth curve with equation

$$Y^2 + a_1XY + a_2Y = X^3 + a_3X^2 + a_4X + a_5, \quad a_i \in \mathbb{F}. \tag{2.1}$$

We define $E(\mathbb{F})$ as the set of points $(x, y) \in \mathbb{F}^2$ which are a solution to (2.1), along with a point at infinity O.

Lemma 2.1.1. *Let E be an elliptic over a field $\mathbb{F}$. If the characteristic of $\mathbb{F}$ is neither 2 nor 3,* (2.1) *can be written as*

$$Y^2 = X^3 + aX + b^3 \tag{2.2}$$

for some $a, b \in \mathbb{F}$, through a linear change of variables.

Proof. Let the variables Z, W be defined as

$$\begin{cases} Z = X + \dfrac{a_3 + \left(\frac{a_1}{2}\right)^2}{3}, \\ W = Y + \dfrac{a_1}{2}X + \dfrac{a_2}{2}. \end{cases} \tag{2.3}$$

This is a linear change of variables, thus, if we can find $a, b \in \mathbb{F}$ such that $W^2 = Z^3 + aZ + b$ the proof is complete. For these variables we have

$$\begin{aligned} W^2 &= \left(Y + \frac{a_1}{2}X + \frac{a_2}{2}\right)^2 \\ &= Y^2 + 2\frac{a_1}{2}XY + 2\frac{a_2}{2}Y + \left(\frac{a_1}{2}\right)^2 X^2 + 2\frac{a_1}{2}\frac{a_2}{2}X + \left(\frac{a_2}{2}\right)^2 \\ &= Y^2 + a_1XY + a_2Y + \left(\frac{a_1}{2}\right)^2 X^2 + \frac{a_1a_2}{2}X + \left(\frac{a_2}{2}\right)^2 \\ &= X^3 + a_3X^2 + a_4X + a_5 + \left(\frac{a_1}{2}\right)^2 X^2 + \frac{a_1a_2}{2}X + \left(\frac{a_2}{2}\right)^2 \end{aligned}$$

$$
\begin{aligned}
&= X^3 + \left(a_3 + \left(\frac{a_1}{2}\right)^2\right) X^2 + \frac{2a_4 + a_1 a_2}{2} X + a_5 + \left(\frac{a_2}{2}\right)^2 \\
&= \left(X + \frac{a_3 + \left(\frac{a_1}{2}\right)^2}{3}\right)^3 + aX + a_6 \\
&= \left(X + \frac{a_3 + \left(\frac{a_1}{2}\right)^2}{3}\right)^3 + a\left(X + \frac{a_3 + \left(\frac{a_1}{2}\right)^2}{3}\right) + b \\
&= Z^3 + aZ + b. \qquad (2.4)
\end{aligned}
$$

□

In this thesis, the fields studied will have characteristic which is neither 2 nor 3, thus, from now on, it will be assumed that the equation of the elliptic curve is written as (2.2).

We can define an operation in $E(\mathbb{F})$ as follows

1. $O + P = P + O = P$, for any $P \in E(\mathbb{F})$.
2. Given $P = (x_1, y_1) \in E(\mathbb{F}) \setminus \{O\}$, we define $-P$ as $(x_1, -y_1)$.
3. Given $P, Q \in E(\mathbb{F})$ with different x coordinates, let us define l as the line that goes through P and Q. There are three possible options
   (a) l is tangent to E at P.
   Then, we define $P + Q$ as $-P$.
   (b) l is tangent to E at Q.
   Then, we define $P + Q$ as $-Q$.
   (c) l is neither tangent to E at P nor tangent to E at Q.
   Then, l intersects E at a third point R and $P + Q$ is defined as $-R$.
4. Given $P \in E(\mathbb{F})$, let us define l as the line tangent to E at P. There are two possible options
   (a) P is an inflection point.
   Then, we define $P + P$ as $-P$.
   (b) P is not an inflection point.
   Then, l intersects E at another point R and $P + P$ is defined as $-R$.

Theorem 2.1.2. *Let E be an elliptic curve over a field $\mathbb{F}$. The operation $+$ is well defined. Moreover, $(E(\mathbb{F}), +)$ is an abelian group whose identity element is O.*

Proof. See [15]. □

Theorem 2.1.3. *Given $P = (x_1, y_1)$ and $Q = (x_2, y_2)$ such that $x_1 \neq x_2$, the coordinates of $P + Q$ are*

$$
\begin{aligned}
x_3 &= \left(\frac{y_2 - y_1}{x_2 - x_1}\right)^2 - x_1 - x_2; \\
y_3 &= -y_1 + \left(\frac{y_2 - y_1}{x_2 - x_1}\right)(x_1 - x_3).
\end{aligned} \qquad (2.5)
$$

Proof. See [15]. □

Theorem 2.1.4. *Given $P = (x_1, y_1)$ the coordinates of $P + P$ are*

$$\begin{aligned} x_3 &= \left(\frac{3{x_1}^2 + a}{2y_1}\right)^2 - 2x_1; \\ y_3 &= -y_1 + \left(\frac{3{x_1}^2 + a}{2y_1}\right)(x_1 - x_3), \end{aligned} \tag{2.6}$$

where this point should be interpreted as O when $y_1 = 0$.

Proof. See [15]. □

Since $E(\mathbb{F})$ is an abelian group, given a point $P \in E(\mathbb{F})$ and an integer k, we can define the point kP.

Lemma 2.1.5. *Let E be an elliptic curve over a field $\mathbb{F}$ and P be a point in the elliptic curve with order q. If $kP = (x_1, y_1)$, then $(q - k)P = (x_1, -y_1)$.*

Proof. For $kP = (x_1, y_1)$ we have $-kP = (x_1, -y_1)$, thus we need to prove that $(q - k)P = -kP$. Since the order of P is q, we have

$$\begin{aligned} & qP = O \\ \iff & kP + (q-k)P = O \\ \iff & (q-k)P = -kP. \end{aligned} \tag{2.7}$$

□

Theorem 2.1.6. *Let E be an elliptic curve over $\mathbb{Z}_p$, where p is a prime number such that $p \equiv 3 \mod 4$. Then, for each $P = (x_1, y_1) \in E(\mathbb{Z}_p)$, we either have $y_1 \equiv \left(x_1^3 + ax_1 + b\right)^{\frac{p+1}{4}} \mod p$ or $y_1 \equiv -\left(x_1^3 + ax_1 + b\right)^{\frac{p+1}{4}} \mod p$.*

Proof. Since P is a point of the elliptic curve E, we have $y_1^2 \equiv x_1^3 + ax_1 + b \mod p$. Moreover, for each x_1 there at most two solutions to $y^2 \equiv x_1^3 + ax_1 + b \mod p$ since $\mathbb{Z}_p$ is a field. Thus, if we can prove that for the two values of y_1 presented in the theorem we have $y_1^2 \equiv x_1^3 + ax_1 + b \mod p$ the proof is complete. For each of these values of y_1 we have

$$\begin{aligned} y_1^2 &\equiv \left(\left(x_1^3 + ax_1 + b\right)^{\frac{p+1}{4}}\right)^2 \mod p \\ &\equiv \left(x_1^3 + ax_1 + b\right)^{\frac{p+1}{2}} \mod p \\ &\equiv \left(x_1^3 + ax_1 + b\right)^{\frac{p-1}{2}} \left(x_1^3 + ax_1 + b\right) \mod p \\ &\equiv \left(x_1^3 + ax_1 + b\right) \mod p \end{aligned} \tag{2.8}$$

by Euler's criterion. □

Lemma 2.1.7. *Let E be an elliptic curve over a field $\mathbb{F}$ and P be a point in the elliptic curve. Given an integer k, kP can be computed using $2\lfloor \log k \rfloor$ additions.*

Proof. Let us write k in binary form as

$$2^{\lfloor \log k \rfloor} + \sum_{i=0}^{\lfloor \log k \rfloor - 1} 2^i \sigma_i, \quad \sigma_i \in \{0, 1\}. \tag{2.9}$$

Since $E(\mathbb{F})$ is a group, we have

$$kP = 2^{\lfloor \log k \rfloor} P + \sum_{i=0}^{\lfloor \log k \rfloor - 1} 2^i \sigma_i P. \tag{2.10}$$

Obviously, we have $2^i P = 2^{i-1}P + 2^{i-1}P$. So, we can start by computing $2^i P$ for $i \in \{1, \ldots, \lfloor \log k \rfloor\}$, which can be computed using $\lfloor \log k \rfloor$ additions. Next, we compute (2.10) which takes at most $\lfloor \log k \rfloor$ additions. Hence, the operation can be computed using at most $2 \lfloor \log k \rfloor$ additions. □

Theorem 2.1.8. *Let E be an elliptic curve over $\mathbb{Z}_p$, where p is a prime number. Given an integer $k \leq p$, the operation kP can be computed in time $O(\log(p)^4)$ for any $P \in E(\mathbb{Z}_p)$.*

Proof. Each addition in E involves a constant number of additions, multiplications and divisions in $\mathbb{Z}_p$. Hence, an addition in E can be computed in $O(\log(p)^3)$ time. By the previous lemma, kP can be computed using at most $2 \lfloor \log(k) \rfloor$ additions. Therefore, since $k \leq p$, the algorithm runs in $O(\log(p)^4)$ time. □

This method is called the double and add method. There are methods to compute kP which are faster, but they will not be presented in this thesis.

2.2 ECDSA

Let E be an elliptic over $\mathbb{Z}_p$, where $p \geq 3$ is a prime number. Given P and Q in $E(\mathbb{Z}_p)$, if there exists an integer k such that $kP = Q$, k is called the discrete logarithm of Q base P.

Definition 2.2.1. Let P be a generator of $E(\mathbb{Z}_p)$. The discrete logarithm problem is the problem of given Q, finding k such that $kP = Q$.

There are no efficient algorithms known for the discrete logarithm problem. This fact can be used to design a signature scheme, the ECDSA. The ECDSA has a prime number $p > 3$, an elliptic curve E over $\mathbb{Z}_p$ and a generator of $E(\mathbb{Z}_p)$, P, with order q, where q is a prime number. These are system-wide parameters. Each user Alice selects a random integer x in the interval $[1, q-1]$ and computes $Q = xP$. Alice's private key is x and her public key is Q.

ECDSA Signature Generation. To sign a message m Alice does the following

1) She computes a random integer k in the interval $[1, q-1]$.
2) She computes $kP = (x_1, y_1)$ and defines $r = x_1 \mod q$. If $r = 0$, return to 1).
3) She computes $s = k^{-1}(H(m) + xr) \mod q$, where $H(m)$ is the hash value of the message m. If $s = 0$, return to 1).
4) The signature is the pair (r, s).

ECDSA Signature Verification. To verify Alice's signature (r, s) of the message m, Bob should do the following.

1) Obtain Alice's public key Q and verify that

   (a) Q is different than O.

   (b) Q lies on E.

   (c) $qQ = O$.

2) Verify that r and s are integers in the interval $[1, q-1]$.

3) Compute $w = s^{-1} \mod q$ and $H(m)$.

4) Compute $u_1 = H(m)w \mod q$ and $u_2 = rw \mod q$.

5) Compute $u_1P + u_2Q = (x_0, y_0)$ and $v = x_0 \mod q$.

6) Accept the signature if $v = r$.

Let us assume that $p \equiv 3 \mod 4$. If we have the x coordinate of kP we can compute a point which is equal to either kP or $-kP$ by **Theorem 2.1.6** and **Lemma 2.1.5**. However, in the ECDSA signature we are not given the x coordinate of kP since in step $2)$ of the ECDSA signature generation Alice computes $r = x_1 \mod q$, so, it may happen that $r \not\equiv x_1 \mod p$. We want to find conditions such that $r \not\equiv x_1 \mod p$ only happens with negligible probability, assuming that the nonces are randomly generated.

Lemma 2.2.1. *Let E be an elliptic curve over $\mathbb{Z}_p$ where $p \geq 3$ is a prime number, $P \in E(\mathbb{Z}_p)$ be a point with order q and (r, s) be a signature generated by a nonce k such that $kP = (x_1, y_1)$. If p and q have more than 3 bits, $p/q \leq 1 + \varepsilon$ and the nonces are randomly generated,*

$$P(r = x_1 \bmod p) \geq 1 - 4\varepsilon. \tag{2.11}$$

Proof. The only way that we can have $r \neq x_1 \bmod p$ is if $r \in [q, p-1]$. Thus, there are $p - q$ different possibilities of the x coordinate of kP such that $r \neq x_1 \bmod p$. Hence, since $\mathbb{Z}_p$ is a field, there at most $2(p-q)$ possibilities of k such that $r \neq x_1 \bmod p$. If we assume that the nonces are randomly generated we have

$$\begin{aligned} P(r \neq x_1 \mod p) &\leq \frac{2(p-q)}{q-3} \\ &= \frac{2(p-q)}{q}\frac{q}{q-3} \\ &\leq 4\varepsilon. \end{aligned} \tag{2.12}$$

□

For the primes p and q in the secp256k1 curve, $p \equiv 3 \bmod 4$ and ε can be 10^{-38}, thus, $r = x_1 \bmod p$ almost surely. Therefore, given a signature with nonce k, we can compute a point which is almost surely equal to either kP or $-kP$.

2.3 Lattice

Definition 2.3.1 (Lattice)**.** Given m linearly independent vectors $b_1, \ldots, b_m \in \mathbb{R}^n$, the lattice generated by them is

$$\mathcal{L}(b_1, \ldots, b_m) = \left\{ \sum_1^m k_i b_i : k_i \in \mathbb{Z} \right\}. \tag{2.13}$$

When $m = n$ we call the lattice a full-rank lattice. Throughout this thesis, only full rank lattices will be used. Let M be a matrix, we define $\mathcal{L}(M)$ as the lattice generated by the rows of M.

Given a lattice L, we define $\lambda(\mathcal{L})$ as

$$\lambda(\mathcal{L}) = \inf \{ \|v\| : v \in \mathcal{L} \setminus \{0\} \}. \tag{2.14}$$

It can be proved that $\lambda(\mathcal{L}) > 0$ and that there is a vector $v \in \mathcal{L}$ such that $\|v\| = \lambda(\mathcal{L})$. This vector will be called the shortest vector of the lattice. The shortest vector problem (SVP) is the problem of given a lattice, finding its shortest vector. There are no efficient algorithms known to solve SVP. However, there are polynomial time algorithms which return a vector whose norm is close the norm of the shortest vector, such as the LLL algorithm. Let the output of LLL to $\mathcal{L}$ be $b_1', \ldots, b_n'$. In [16] it was proved that

$$\left\| b_1' \right\| \leq 1.02^{(n-1)/2} \lambda(L), \tag{2.15}$$

so, for small values of n, the norm of that vector is close to the norm of the shortest vector.

2.3.1 Lattice attacks

2.3.1.1 One variable

Let α be a secret integer modulo a public prime q. In the HNP an attacker is assumed to be given an oracle that given a random sequence of integers t_i, for $i \in \{1, \ldots, m\}$, returns a sequence a_i such that $|t_i \alpha - a_i|_q \leq C$, for some $C \leq q$, the task is to recover α. That problem can be reduced to finding a solution $x_1 = b_1, \ldots, x_m = b_m, y = \alpha$ such that $|b_i| \leq C$ to the following system of equations

$$\begin{aligned} x_1 - t_1 y + a_1 &\equiv 0 \mod q, \\ &\vdots \\ x_m - t_m y + a_m &\equiv 0 \mod q. \end{aligned} \tag{2.16}$$

Let us a define matrix M as

$$\begin{bmatrix} q & & & & \\ & \ddots & & & \\ & & q & & \\ t_1 & & t_m & \frac{C}{q} & \\ a_1 & & a_m & & C \end{bmatrix}. \tag{2.17}$$

Then, we have the following theorem

Theorem 2.3.1. *Let $v = (b_1, \ldots, b_m, \alpha\ C/q\ , -C)$ be a vector such that $x_1 = b_1, \ldots x_m = b_m, y = \alpha$ is a solution to* (2.16)*. Furthermore, let us assume that $|b_i| \leq C$ for $i = 1, \ldots, m$, where C is an integer such that*

$$\log C \leq \frac{m-1}{m} \log q - \frac{m+2}{2m} \log (m+2). \tag{2.18}$$

Then, for small values of m, either v or $-v$ will be a vector in the LLL-reduced basis of $\mathcal{L}(M)$.

Proof. For each $i \in \{1, \ldots, m\}$ there is an equation

$$b_i = t_i \alpha - a_i + k_i q, \tag{2.19}$$

where $k_i \in \mathbb{Z}$. We have

$$\begin{aligned} v &= (b_1, \ldots, b_m, \alpha\ C/q\ , -C) \\ &= (t_1\alpha - a_1 + k_1 q, \ldots, t_m \alpha - a_m + k_m q, \alpha\ C/q\ , -C) \\ &= \alpha\,(t_1, \ldots, t_m,\ C/q\ , 0) - (a_1, \ldots, a_m, 0, C) + \sum_{i=1}^{m} k_i q e_i, \end{aligned} \tag{2.20}$$

where $e_i \in \mathbb{R}^{m+2}$ is the vector with a one in the $i-$coordinate and zeros elsewhere. Hence, v is an integer combination of the rows of M, thus $v \in \mathcal{L}(M)$. It remains to be proven that either v or $-v$ is one of the vectors in the LLL-reduced basis.

If v is a vector in the lattice $\mathcal{L}(M)$, m is small and $\|v\| \leq \det M^{1/\dim M}$, either v or $-v$ is one of the vectors in the LLL-reduced basis [6]. Therefore, we only need to prove that

$$\begin{aligned} &\|v\| \leq \left(C^2 q^{m-1}\right)^{\frac{1}{m+2}}. \\ \iff &\log\left(\|v\|\right) \leq \frac{2}{m+2} \log C + \frac{m-1}{m+2} \log q. \end{aligned} \tag{2.21}$$

Since $\alpha \leq q$ we have

$$\begin{aligned} \log(\|v\|) &= \log(\|(b_1, \ldots, b_m, \alpha\ C/q\ , C)\|) \\ &\leq \log(\|(C, \ldots, C, C, C)\|) \\ &= \log(\sqrt{m+2}C) \\ &= \frac{log(m+2)}{2} + \log(C) \\ &= \frac{log(m+2)}{2} + \frac{2}{m+2} \log C + \frac{m}{m+2} \log C \\ &\leq \frac{log(m+2)}{2} + \frac{2}{m+2} \log C + \frac{m}{m+2} \left(\frac{m-1}{m} \log q - \frac{m+2}{2m} \log (m+2) \right) \\ &= \frac{2}{m+2} \log C + \frac{m-1}{m+2} \log q. \end{aligned} \tag{2.22}$$

□

Given the vector v the secret α can immediately be computed. Therefore, if m is small and C and m

satisfy 2.18, the hidden number problem can be solved using a lattice reduction algorithm.

2.3.1.2 Two variables

Let α and β be secret integers modulo a public prime q. In the two variables HNP an attacker is assumed to be given an oracle that given two random sequence of integers $t_i, t_i^{'}$ for $i \in \{1, \ldots, m\}$, returns a sequence a_i for $i \in \{1, \ldots, m\}$ such that $|t_i\alpha + t_i^{'}\beta - a_i|_q \leq C$, for some $C \leq q$, the task is to recover α and β. That problem can be reduced to finding a solution $x_1 = b_1, \ldots, x_m = b_m, y = \alpha, z = \beta$ such that $|b_i| \leq C$ to the following system of equations

$$\begin{aligned} x_1 - t_1 y - t_1^{'} z + a_1 &\equiv 0 \mod q, \\ &\vdots \\ x_m - t_m y - t_m^{'} z + a_m &\equiv 0 \mod q. \end{aligned} \tag{2.23}$$

Let us define a matrix M as

$$\begin{bmatrix} q & & & & & \\ & \ddots & & & & \\ & & q & & & \\ t_1 & & t_m & \frac{C}{q} & & \\ t_1^{'} & & t_m^{'} & & \frac{C}{q} & \\ a_1 & & a_m & & & C \end{bmatrix}. \tag{2.24}$$

Theorem 2.3.2. *Let $v = (b_1, \ldots, b_m, \alpha\ C/q\ , \beta\ C/q\ , -C)$ be a vector such that $x_1 = b_1, \ldots x_m = b_m, y = \alpha, z = \beta$ is a solution to* (2.23)*. Moreover, let us assume that $|b_i| \leq C$ for $i = 1, \ldots, m$, where C is an integer such that*

$$\log C \leq \frac{m-2}{m} \log q - \frac{m+3}{2m} \log (m+3). \tag{2.25}$$

Then, for small values of m, either v or $-v$ will be a vector in the LLL-reduced basis of $\mathcal{L}(M)$.

Proof. For each $i \in \{1, \ldots, m\}$ there is an equation

$$b_i = t_i\alpha + t_i^{'}\beta - a_i + k_i q, \tag{2.26}$$

where $k_i \in \mathbb{Z}$. We have

$$\begin{aligned} v &= (b_1, \ldots, b_m, \alpha\ C/q\ , \beta\ C/q\ , -C) \\ &= \left(t_1\alpha + t_1^{'}\beta - a_1 + k_1 q, \ldots, t_m\alpha + t_m^{'}\beta - a_m + k_m q, \alpha\ C/q\ , \beta\ C/q\ , -C\right) \\ &= \alpha\,(t_1, \ldots, t_m,\ C/q\ , 0, 0) + \beta\left(t_1^{'}, \ldots, t_m^{'}, 0,\ C/q\ , 0\right) - (a_1, \ldots, a_m, 0, 0, C) + \sum_{i=1}^{m} k_i q e_i, \end{aligned} \tag{2.27}$$

where $e_i \in \mathbb{R}^{m+3}$ is the vector with a one in the $i-$coordinate and zeros elsewhere. Hence, v is an integer combination of the rows of M, thus $v \in \mathcal{L}(M)$. It remains to be proven that either v or $-v$ is one of the vectors in the LLL- reduced basis.

As it was stated in the proof of **Theorem 2.3.1** if $\|v\| \leq \det M^{1/\dim M}$ and m is small either v or $-v$ is vector in the LLL-reduced basis of $\mathcal{L}(M)$. Therefore, we need to prove that

$$\|v\| \leq \left(C^3 q^{m-2}\right)^{\frac{1}{m+3}} . \tag{2.28}$$

Equivalently, we can prove that

$$\log\left(\|v\|\right) \leq \frac{3}{m+3}\log C + \frac{m-2}{m+3}\log q. \tag{2.29}$$

Since $\alpha, \beta \leq q$ we have

$$\begin{aligned}
\log(\|v\|) &= \log(\|(b_1, \ldots, b_m, \alpha\ C/q\ , \beta\ C/q\ , C)\|)\,, \\
&\leq \log(\|(C, \ldots, C, C, C, C)\|)\,, \\
&= \log(\sqrt{m+3}C), \\
&= \frac{log(m+3)}{2} + \log(C), \\
&= \frac{log(m+3)}{2} + \frac{3}{m+3}\log C + \frac{m}{m+3}\log C, \\
&\leq \frac{log(m+3)}{2} + \frac{3}{m+3}\log C + \frac{m}{m+3}\left(\frac{m-2}{m}\log q - \frac{m+3}{2m}\log\left(m+3\right)\right), \\
&= \frac{3}{m+3}\log C + \frac{m-2}{m+3}\log q.
\end{aligned} \tag{2.30}$$

□

Therefore, similarly to the one variable HNP, the two variables HNP can be solved using a lattice reduction algorithm.

Chapter 3

Attacks on ECDSA

In this section we will design attacks to recover a user's private key from its list of ECDSA signatures. The attacks are divided into two categories, lattice attacks, which are variants of the solution to the hidden number problem, and group attacks where properties of the operation in elliptic curves will be used as a tool to recover the private key.

3.1 Lattice attacks

3.1.1 Computing private key through small nonces

Let Alice be a user whose private key is y_A, with signatures $\{(r_1, s_1), \ldots, (r_n, s_n)\}$ on the messages $\{m_1, \ldots, m_n\}$. For each $i \in \{1, \ldots, n\}$ we have

$$\begin{cases} s_i \equiv {k_i}^{-1}(m_i + yr_i) \mod q, \end{cases} \tag{3.1}$$

$$\implies \begin{cases} k_i - {s_i}^{-1} r_i y_A - {s_i}^{-1} m_i \equiv 0 \mod q, \end{cases} \tag{3.2}$$

thus $x_1 = k_1, \ldots, x_n = k_n, y = y_A$ is a solution to the system of equations (2.16) where

$$\begin{cases} t_i = s_i^{-1} r_i, \\ a_i = -s_i^{-1} m_i. \end{cases} \tag{3.3}$$

Let us assume that we know a constant $C \leq q$ such that $|k_i|_q \leq C$ for $i = 1, \ldots, n$. If n and C satisfy (2.18), a lattice reduction algorithm can be used to recover the private key, according to **Theorem 2.3.1**.

Let $m \leq n$ be an integer such that m and C satisfy (2.18). If instead of using n signatures, we start by randomly picking m signatures and then using the lattice reduction algorithm on the matrix generated by the m signatures, the method should still work. Furthermore, the probability of success of having $|k_i|_q \leq C$ for $i \in \{1, \ldots, m\}$ is at least as high as the probability of having $|k_i|_q \leq C$ for $i \in \{1, \ldots, n\}$. Therefore, if such a value of m exists, it should be used instead of n. Moreover, the optimal value of m

for C is

$$m = \min\{k : k \text{ and } C \text{ satisfy (2.18)}\}. \tag{3.4}$$

Algorithm 1 implements the method described. Its input is a prime number, a list of Alice's signatures, Alice's public key, an integer C and m, the number of signatures to be used. The algorithm starts by randomly selecting m signatures from the list using **Algorithm 2**, then it uses the lattice reduction algorithm on the matrix generated by those m signatures. Finally, a cycle goes though the lattice reduced basis and it returns the private key if it can be computed, and false otherwise.

Algorithm 1: Computing a private key through small nonces.

input : q, a prime number.
$ListSignA$, a list with n signatures from Alice .
Q_A, Alice's public key.
m, the number of equations to be used in the lattice reduction algorithm.
C, an integer, smaller than q.
output: The algorithm returns Alice's private key if it finds it and false otherwise.

$t_i \leftarrow \{\}$;
$a_i \leftarrow \{\}$;
$ReducedList \leftarrow$ List with m distinct random numbers between 1 and n;
for $i \leftarrow 1$ **to** m **do**
  $j \leftarrow$ `Get(`$ReducedList$`,`i`)`;
  $t_i \leftarrow$ `Add(`t_i`,` $s_j^{-1} r_j \mod q$`)`;
  $a_i \leftarrow$ `Add(`a_i`,` $-s_j^{-1} m_j \mod q$`)`;
end
$M \leftarrow$`MatrixCreation(`m`,`q`,`t_i`,`a_i`,`C`)`;
$ReducedM \leftarrow$`LLL(`M`)`;
$i \leftarrow 1$;
while $i \leq m + 2$ **do**
  $l_i \leftarrow$ Vector i of $ReducedM$;
  $z \leftarrow$`Get(`l_i`,`$m+2$`)`$/C$;
  if $|z| = 1$ **then**
    $x \leftarrow$`Get(`l_i`,`$m+1$`)`;
    $y \leftarrow -z * x * q/C \mod q$;
    $Q_y \leftarrow y\, P$;
    if $Q_y = Q_A$ **then**
      return y;
    else
      $i \leftarrow i + 1$
    end
  else
    $i \leftarrow i + 1$
  end
end
return false;

Algorithm 2: Robert Floyd's sampling algorithm .

input : N, an integer number.
M, an integer number.
output: $S \subset \{1, \ldots, N\}$ such that $\#S = M$.

```
S ← {}.;
for i ← N − M + 1 to N do
    r ←RandomInteger(1, i);
    if r ∉ S then
        S ←Add(S, r);
    else
        S ←Add(S, i);
    end
end
```

Algorithm 3: MatrixCreation.

input : m, an integer number.
q, a prime number.
t_i, a vector with m entries.
a_i, a vector with m entries.
C, an integer number
output: A matrix M like the one in (2.17).

```
S ← {}.;
M ← Null (m + 2) × (m + 2) matrix;
for j ← 1 to m do
    M(j, j) ← q;
    M(m + 1, j) ← t_i(j);
    M(m + 2, j) ← a_i(j);
end
M(m + 1, m + 1) ← C/q;
M(m + 2, m + 2) ← C;
```

Lemma 3.1.1. *If **Algorithm 1** does not return false, it returns the private key. Moreover, if m is small, C and m satisfy* (2.18) *and $|k_j|_q \leq C$ for $j \in ReducedList$, we hope to recover the private key. Furthermore, the algorithm runs in worst-case time $O(m^6 \log(q)^3 + m^6 \log(m)^3 + m \log(q)^4)$.*

Proof. First, let us prove that if the algorithm does not return false, it returns the private key. If the algorithm does not return false, it returns y, thus

$$\begin{aligned} & Q_y = Q_A \\ \iff & yP = y_A P \\ \iff & y \equiv y_A \mod q. \end{aligned} \tag{3.5}$$

By line 17, $y < q$. Therefore, $y = y_A$.

Let us now assume that m is small, $|k_j|_q \leq C$ for $j \in ReducedList$ and that C and m satisfy (2.18).

The matrix M is equal to

$$\begin{bmatrix} q & & & & \\ & \ddots & & & \\ & & q & & \\ t_{j_1} & & t_{j_m} & \frac{C}{q} & \\ a_{j_1} & & a_{j_m} & & C \end{bmatrix},$$

where the t_i and a_i are defined in (3.3). Since $x_i = k_i$ and $y = y_A$ is a solution to (2.16), either the vector $v = (k_{j_1}, \dots k_{j_m}, y_A\ C/q\ , -C)$ or $-v$ is a vector in the LLL-reduced basis of M, by **Theorem 2.3.1**. If v is one of those vectors the cycle starting in line 12 will eventually reach it. For v we have

$$\begin{cases} z = -1, \\ x = y_A \frac{C}{q}, \\ y = y_A \mod q. \end{cases} \tag{3.6}$$

Analogously, we can prove that if the vector in the reduced basis is $-v$ we also have $y = y_A \mod q$. Therefore, the algorithm returns the private key.

Finally, we need to prove that the algorithm runs in worst-case time $O(m^6 \log(q)^3 + m^6 \log(m)^3 + m \log(q)^4)$.

Computing the $ReducedList$ is done using **Algorithm 2**. This algorithm runs in $O(m^2 \log(q))$ worst-case time. The for cycle starting in line 4 runs m times, each operation inside the cycle can be computed in $O(\log(q)^3)$ worst-case time, so, the for cycle takes $O(m \log(q)^3)$ time. Hence, the total time until line 8 of the algorithm is at most $O(m^2 \log(q) + m \log(q)^3)$. Creating the matrix can be done in $O(m^2 \log(q))$ time. The LLL algorithm is computed in $O(m^6 \log(B)^3)$ time where B is the norm of the largest vector in the original basis. That norm is smaller than $\sqrt{m+1}q$, thus, the LLL algorithm can be run in $O(m^6 \log(\sqrt{m+1}q)^3)$ time, hence, in $O(m^6 \log(m)^3 + m^6 \log(q)^3)$ time. Computing Q can be done in $O(\log(q)^4)$ time, according to **Theorem 2.1.8**, all other operations inside the while cycle can also be computed in $O(\log(q)^4)$ time. Therefore, the while cycle takes $O(m \log(q)^4)$ worst-case time, so, the whole algorithm runs is worst-case time $O(m^6 \log(q)^3 + m^6 \log(m)^3 + m \log(q)^4)$. □

3.1.2 Computing private key through linear equations between nonces

Let Alice be a user whose private key is y_A, with signatures $\{(r_1, s_1), \dots, (r_n, s_n)\}$ on the messages $\{m_1, \dots, m_n\}$. Given $a \in \{1, \dots, q-1\}$ we have

$$ak_i - as_i{}^{-1} r_i y_A - as_i{}^{-1} m_i \equiv 0 \mod q \tag{3.7}$$

for each $i \in \{1, \dots n\}$. Hence, given $a, b \in \{1, \dots, q-1\}$, we have

$$ak_i - as_i{}^{-1} r_i y_A - as_i{}^{-1} m_i + bk_{i+1} - bs_{i+1}{}^{-1} r_{i+1} y_A - bs_{i+1}{}^{-1} m_{i+1} \equiv 0 \mod q \tag{3.8}$$

for each $i \in \{1, \ldots, n-1\}$.

Thus, $x_1 = ak_1 + bk_2, x_2 = ak_2 + bk_3, \ldots, x_{n-1} = ak_{n-1} + bk_n, y = y_A$ is a solution to the system of equations (2.16) where

$$\begin{cases} t_i = as_i^{-1} r_i + bs_{i+1}^{-1} r_{i+1}, \\ a_i = -as_i^{-1} m_i - bs_{i+1}^{-1} m_{i+1}. \end{cases} \tag{3.9}$$

Let us assume that we know a constant $C \leq q$ such that $|ak_i + bk_{i+1}|_q \leq C$ for $i \in \{1, \ldots, n-1\}$. If $n-1$ and C satisfy (2.18), a lattice reduction algorithm can be used to recover the private key. Similarly to what was done in 3.1.1, we can start by selecting $m \leq n$ such that $m-1$ and C satisfy (2.18) and then use the lattice reduction algorithm on the matrix generated by those m signatures.

Algorithm 4 implements the method described. Its input is a prime number, a list of Alice's signatures, Alice's public key, an integer C and m, the number of signatures to be used. The algorithm starts by randomly selecting m signatures from the list using **Algorithm 2**, then it uses the lattice reduction algorithm on the matrix generated by those m signatures. Finally, a cycle goes though the lattice reduced basis and it returns the private key if it can be computed, and false otherwise.

Lemma 3.1.2. *If **Algorithm 4** does not return false, it returns the private key. Moreover, if m is small, C and $m-1$ satisfy* (2.18) *and $\left|ak_{j_i} + bk_{j_{i+1}}\right|_q \leq C$ for $i \in \{1, \ldots, m-1\}$ where $ReducedList = \{j_1, \ldots, j_m\}$, we hope to recover the private key. Furthermore, the algorithm runs in worst-case time $O(m^6 \log(q)^3 + m^6 \log(m)^3 + m \log(q)^4)$.*

Proof. The proof of this lemma is analogous to the proof of **Lemma 3.1.1**. □

The nonces are generated with a PNRG. The attack in **Algorithm 1** computes the private key when the numbers generated by that PNRG have a disproportionately high probability of being numbers close to 0 or q. However, that does not have to be true for the PNRG. If it is known that there exists $a, b \in \{1, \ldots, q-1\}$, such that given two nonces k_1 and k_2, generated by the PNRG, there is a disproportionately high probability of $ak_1 + bk_2$ being close to 0 or q, we should use the attack in **Algorithm 4**. If we do not know such a and b, we will use $a = 1$ and $b = -1$.

3.1.3 Computing the private key with most significant bits known

Definition 3.1.1. Given a positive integer k, written in binary as

$$2^{\lfloor \log k \rfloor} + \sum_{i=0}^{\lfloor \log k \rfloor - 1} a_i 2^i, \quad a_i \in \{0, 1\}, \tag{3.10}$$

the l most significant bits of k are defined as the vector $(1, a_{\lfloor \log k \rfloor - 1}, \ldots, a_{\lfloor \log k \rfloor - l + 1})$, along with an index equal to $\lfloor \log k \rfloor$.

The l most significant bits represent the l left-most bits of k in binary representation, along with the position of the left-most bit.

Algorithm 4: Computing private key through linear equations between nonces.

input : q, a prime number.
$ListSignA$, a list with n signatures from Alice .
Q_A, Alice's public key.
m, the number of equations to be used in the lattice reduction algorithm.
C, an integer, smaller than q.
a, an integer.
b, an integer.

output: The algorithm returns Alice's private key if it finds it and false otherwise.

$t_i \leftarrow \{\}$;
$a_i \leftarrow \{\}$;
$ReducedList \leftarrow$ List with $m+1$ distinct random numbers between 1 and n;
for $i \leftarrow 1$ **to** m **do**
  $j \leftarrow$ `Get(`$ReducedList$`,`i`)`;
  $k \leftarrow$ `Get(`$ReducedList$`,`$i+1$`)`;
  $t_i \leftarrow$ `Add(`t_i`,` $as_j^{-1}r_j + bs_k^{-1}r_k \mod q$`)`;
  $a_i \leftarrow$ `Add(`a_i`,` $-as_j^{-1}r_j - bs_k^{-1}r_k \mod q$`)`;
end
$M \leftarrow$`MatrixCreation(`q`,`t_i`,`a_i`,`m`,`C`)`;
$ReducedM \leftarrow$`LLL(`M`)`;
$i \leftarrow 1$;
while $i \leq m+2$ **do**
  $l_i \leftarrow$ Line i of $ReducedM$;
  $z \leftarrow$`Get(`l_i`,`$m+2$`)`$/C$;
  if $|z| = 1$ **then**
    $x \leftarrow$`Get(`l_i`,`$m+1$`)`;
    $y \leftarrow -z * x * q/C \mod q$;
    $Q_y \leftarrow y\,P$;
    if $Q_y = Q_A$ **then**
      return y;
    else
      $i \leftarrow i+1$
    end
  else
    $i \leftarrow i+1$
  end
end
return false;

Definition 3.1.2. Let k be an integer whose l most significant bits are $(1, a_{\lfloor \log k \rfloor -1}, \ldots, a_{\lfloor \log k \rfloor -l+1})$ along with $\lfloor \log k \rfloor$. We define $MSB_l(k)$ as

$$2^{\lfloor \log k \rfloor} + \sum_{i=\lfloor \log k \rfloor -l+1}^{\lfloor \log k \rfloor -1} a_i 2^i + 2^{\lfloor \log k \rfloor -l}. \tag{3.11}$$

Theorem 3.1.3. *Let k and l be positive integers,*

$$|k - MSB_l(k)| \leq 2^{\lfloor \log k \rfloor -l}. \tag{3.12}$$

Proof. For positive integers k and l we have

$$\begin{cases} k = 2^{\lfloor \log k \rfloor} + \sum_{i=0}^{\lfloor \log k \rfloor - 1} a_i 2^i, \\ MSB_l(k) = 2^{\lfloor \log k \rfloor} + \sum_{i=\lfloor \log k \rfloor - l + 1}^{\lfloor \log k \rfloor - 1} a_i 2^i + 2^{\lfloor \log k \rfloor - l}, \end{cases} \tag{3.13}$$

$$\implies \begin{cases} k - MSB_l(k) = (a_{\lfloor \log k \rfloor - l} - 1) 2^{\lfloor \log k \rfloor - l} + \sum_{i=0}^{\lfloor \log k \rfloor - l - 1} a_i 2^i, \end{cases} \tag{3.14}$$

$$\implies \begin{cases} k - MSB_l(k) = -2^{\lfloor \log k \rfloor - l} + \sum_{i=0}^{\lfloor \log k \rfloor - l - 1} a_i 2^i, & \text{if } a_{\lfloor \log k \rfloor - l} = 0, \\ k - MSB_l(k) = \sum_{i=0}^{\lfloor \log k \rfloor - l - 1} a_i 2^i, & \text{if } a_{\lfloor \log k \rfloor - l} = 1. \end{cases} \tag{3.15}$$

Since $a_i \in \{0, 1\}$ for $i \in \{0, 1, \ldots, \lfloor \log k \rfloor - l - 1\}$, we have

$$|k - MSB_l(k)| \leq 2^{\lfloor \log k \rfloor - l}. \tag{3.16}$$

□

Corollary 3.1.3.1. *Let $q > 2$ be a prime number with n bits and k be an integer such that $k \leq q$. Then*

$$|k - MSB_l(k)|_q \leq \frac{2^n}{2^{l+1}}. \tag{3.17}$$

.

Proof. Since $q > 2$ and q is a prime number we have

$$\begin{cases} \lfloor \log q \rfloor = \lceil \log q \rceil - 1, \\ \lceil \log q \rceil = n. \end{cases} \tag{3.18}$$

Thus, by **Theorem 3.1.3** we have

$$\begin{aligned} |k - MSB_l(k)|_q &\leq |k - MSB_l(k)| \\ &\leq 2^{\lfloor \log k \rfloor - l} \\ &\leq \frac{2^{\lfloor \log q \rfloor}}{2^l} \\ &= \frac{2^n}{2^{l+1}}. \end{aligned} \tag{3.19}$$

□

In this thesis we will be studying the $256-$bit ECDSA, thus, if a side channel attack can recover the l most significant bits of the nonces used, we will have

$$|k - MSB_l(k)| \leq 2^{255 - l}. \tag{3.20}$$

Let Alice be a user whose private key is y_A, with signatures $\{(r_1, s_1), \ldots, (r_n, s_n)\}$ on the messages

$\{m_1, \ldots, m_n\}$. For each $i \in \{1, \ldots, n\}$ we have

$$k_i - MSB_l(k_i) - {s_i}^{-1} r_i y_A - {s_i}^{-1} m_i + MSB_l(k_i) \equiv 0 \mod q. \tag{3.21}$$

Hence, $x_1 = k_1 - MSB_l(k_1), \ldots, x_n = k_n - MSB_l(k_n), y = y_A$ is a solution to (2.16) where

$$\begin{cases} t_i = {s_i}^{-1} r_i, \\ a_i = -{s_i}^{-1} m_i + MSB_l(k_i). \end{cases} \tag{3.22}$$

Let us assume that there exists a side-channel which recovers the nonces' l most significant bits. We define C as 2^{255-l}. If n and C satisfy (2.18) and n is small, the private key can be recovered using a lattice reduction algorithm. Moreover, as it was done in the previous attacks, if there is $m \leq n$ such that m and C satisfy (2.18), we can start by selecting m signatures and then using the lattice reduction algorithm on the matrix generated by those m signatures. **Algorithm 5** implements this method. Its input is a prime number, Alice's list of signatures, Alice's public key, the number of equations m, and l, the number of most significant bits known for each nonce. The algorithm starts by selecting m random signatures from the list using **Algorithm 2**, then it uses the lattice reduction algorithm on the matrix generated by those m signatures. Finally, a cycle goes through the lattice reduced basis and it returns the private key if it can be computed, and false otherwise.

3.1.4 Computing the private key with least significant bits known

Definition 3.1.3. Given a positive integer k written in binary as

$$2^{\lfloor \log k \rfloor} + \sum_{i=0}^{\lfloor \log k \rfloor - 1} a_i 2^i, \quad a_i \in \{0, 1\}, \tag{3.23}$$

the l most significant bits are defined as the vector $(a_{l-1}, a_{l-2} \ldots, a_0)$. We define $LSB_l(k)$ as the integer

$$\sum_{i=0}^{l-1} a_i 2^i. \tag{3.24}$$

Theorem 3.1.4. *Let k and l be positive integers, $2^{-l}(k - LSB_l(k))$ is also a positive integer. Moreover,*

$$\left| 2^{-l}(k - LSB_l(k)) - 2^{\lfloor \log k \rfloor - l} \right| \leq 2^{\lfloor \log k \rfloor - l}. \tag{3.25}$$

Proof. For positive integers k and l we have

$$\begin{cases} k = 2^{\lfloor \log k \rfloor} + \sum\limits_{i=0}^{\lfloor \log k \rfloor - 1} a_i 2^i \\ LSB_l(k) = \sum\limits_{i=0}^{l-1} a_i 2^i \end{cases}$$

Algorithm 5: Computing a private key with MSB known.

input : q, a prime number with k bits.
$ListSignA$, a list with n signatures from Alice .
Q_A, Alice's public key.
m, the number of equations to be used in the lattice reduction algorithm.
l, number of most significant bits known.
output: The algorithm returns Alice's private key if it finds it and false otherwise.

$C = 2^{k-l-1}$;
$t_i \leftarrow \{\}$;
$a_i \leftarrow \{\}$;
$ReducedList \leftarrow$ List with m distinct random numbers between 1 and n;
for $i \leftarrow 1$ **to** m **do**
  $j \leftarrow$ `Get(`$ReducedList$`,`i`)`;
  $t_i \leftarrow$ `Add(`t_i`,` $s_j^{-1} r_j \mod q$`)`;
  $a_i \leftarrow$ `Add(`a_i`,` $-s_j^{-1} m_j + MSB_l(k_j) \mod q$`)`;
end
$M \leftarrow$`MatrixCreation(`q`,`t_i`,`a_i`,`m`,`C`)`;
$ReducedM \leftarrow$`LLL(`M`)`;
$i \leftarrow 1$;
while $i \leq m + 2$ **do**
  $l_i \leftarrow$ Line i of $ReducedM$;
  $z \leftarrow$`Get(`l_i`,`$m+2$`)`$/C$;
  if $|z| = 1$ **then**
    $x \leftarrow$`Get(`l_i`,`$m+1$`)`;
    $y \leftarrow -z * x * q/C \mod q$;
    $Q_y \leftarrow y\, P$;
    if $Q_y = Q_A$ **then**
      return y;
    else
      $i \leftarrow i + 1$
    end
  else
    $i \leftarrow i + 1$
  end
end
return false;

$$\Longrightarrow k - LSB_l(k) = 2^{\lfloor \log k \rfloor} + \sum_{i=l}^{\lfloor \log k \rfloor - 1} a_i 2^i$$

$$\Longrightarrow 2^{-l}\left(k - LSB_l(k)\right) = 2^{\lfloor \log k \rfloor - l} + \sum_{i=0}^{\lfloor \log k \rfloor - l - 1} a_{i+l} 2^i. \tag{3.26}$$

where $a_i \in \{0, 1\}$. Thus, $2^{-l}\left(k - LSB_l(k)\right)$ is a positive integer and

$$\left|2^{-l}(k - LSB_l(k)) - 2^{\lfloor \log k \rfloor - l}\right| \leq 2^{\lfloor \log k \rfloor - l}. \tag{3.27}$$

□

Corollary 3.1.4.1. *Let $q > 2$ be a prime number with n bits and k be an integer such that $k \leq q$. Then*

$$\left|2^{-l}(k - LSB_l(k)) - 2^{\lfloor \log k \rfloor - l}\right| \leq \frac{2^n}{2^{l+1}}. \tag{3.28}$$

Proof. This proof is analogous to the proof of **Corollary 3.1.3.1**. □

So, if a side-channel attack can recover the l least significant bits of the nonces used in the $256-$bit ECDSA, we will have

$$\left|2^{-l}(k - LSB_l(k)) - 2^{\lfloor \log k \rfloor - l}\right| \leq 2^{255-l}. \tag{3.29}$$

Let Alice be a user whose private key y_A, with signatures $\{(r_1, s_1), \ldots, (r_n, s_n)\}$ on the messages $\{m_1, \ldots, m_n\}$. For each $i \in \{1, \ldots, n\}$ we have

$$\begin{aligned}
&k_i - {s_i}^{-1} r_i y_A - {s_i}^{-1} m_i \equiv 0 \mod q, \\
\Longrightarrow &k_i - LSB_l(k_i) - {s_i}^{-1} r_i y_A - {s_i}^{-1} m_i + LSB_l(k_i) \equiv 0 \mod q, \\
\Longrightarrow &2^{-l}\left(k_i - LSB_l(k_i)\right) - 2^{-l} {s_i}^{-1} r_i y_A + 2^{-l}\left(-{s_i}^{-1} m_i + LSB_l(k_i)\right) \equiv 0 \mod q, \\
\Longrightarrow &2^{-l}\left(k_i - LSB_l(k_i)\right) - 2^{\lfloor \log k \rfloor - l} - 2^{-l} {s_i}^{-1} r_i y_A + 2^{-l}\left(-{s_i}^{-1} m_i + LSB_l(k_i)\right) + 2^{\lfloor \log k \rfloor - l} \equiv 0 \mod q,
\end{aligned} \tag{3.30}$$

hence, $x_1 = 2^{-l}\left(k_1 - LSB_l(k_1)\right) - 2^{\lfloor \log k \rfloor - l}, \ldots, x_n = 2^{-l}\left(k_n - LSB_l(k_n)\right) - 2^{\lfloor \log k \rfloor - l}, y = y_A$ is a solution to (2.16) where

$$\begin{cases} t_i = 2^{-l} {s_i}^{-1} r_i, \\ a_i = 2^{-l}(-{s_i}^{-1} m_i + LSB_l(k_i)) + 2^{\lfloor \log k \rfloor - l}. \end{cases} \tag{3.31}$$

Let us assume that there exists a side-channel which recovers the nonce's l least significant bits . We define C as 2^{255-l}. If n and C satisfy (2.18) and n is small, the private key can be recovered using a lattice reduction algorithm. Moreover, as it was done in the previous attacks, if there is $m \leq n$ such that m and C satisfy (2.18), we can start by selecting m signatures and then using the lattice reduction algorithm on the matrix generated by those m signatures. **Algorithm 6** implements this method. Its input is a prime number, Alice's list of signatures, Alice's public key, the number of equations m, and l, the number of least significant bits known for each nonce. The algorithm starts by selecting m random signatures from the list using **Algorithm 2**, then it uses the lattice reduction algorithm on the matrix generated by those m signatures. Finally, a cycle goes through the lattice reduced basis and it returns the private key if it can be computed, and false otherwise.

3.1.5 Computing linear equation between different private keys

Let Alice be a user whose private key is y_A with signatures $\{(r_{A,1}, s_{A,1}), \ldots, (r_{A,n_A}, s_{A,n_A})\}$ on the messages $\{m_{A,1}, \ldots, m_{A,n_A}\}$ and Bob be a user whose private key is y_B with signatures $\{(r_{B,1}, s_{B,1}), \ldots, (r_{B,n_B}, s_{B,n_B})\}$ on the messages $\{m_{B,1}, \ldots, m_{B,n_B}\}$. We define Max as $\max\{n_A, n_B\}$. We want to extend the list of signatures to Max signatures for both users. In order to do that, we select the user with $min = \min\{n_A, n_B\}$ signatures and append to its list of signatures the user's first signature, and to the list of messages the user's first message. That is, we define

Algorithm 6: Recovering a private key with LSB known.

input : q, a prime number with k bits.
$ListSignA$, a list with n signatures from Alice .
Q_A, Alice's public key.
m, the number of equations to be used in the lattice reduction algorithm.
l, number of most significant bits known.
output: The algorithm returns Alice's private key if it finds it and false otherwise.

$C = 2^{k-l-1}$;
$t_i \leftarrow \{\}$;
$a_i \leftarrow \{\}$;
$ReducedList \leftarrow$ List with m distinct random numbers between 1 and n;
for $i \leftarrow 1$ **to** m **do**
  $j \leftarrow$ `Get(`$ReducedList$`,`i`)`;
  $t_i \leftarrow$ `Add(`t_i`,` $2^{-l}s_j^{-1}r_j \mod q$`)`;
  $a_i \leftarrow$ `Add(`a_i`,` $2^{-l}(-s_j^{-1}m_j + LSB_l(k_j)) + 2^{\lfloor \log k \rfloor - l} \mod q$`)`;
end
$M \leftarrow$`MatrixCreation(`q`,`t_i`,`a_i`,`m`,`C`)`;
$ReducedM \leftarrow$`LLL(`M`)`;
$i \leftarrow 1$;
while $i \leq m + 2$ **do**
  $l_i \leftarrow$ Line i of $ReducedM$;
  $z \leftarrow$`Get(`l_i`,`$m+2$`)`$/C$;
  if $|z| = 1$ **then**
    $x \leftarrow$`Get(`l_i`,`$m+1$`)`;
    $y \leftarrow -z * x * q/C \mod q$;
    $Q_y \leftarrow y\ P$;
    if $Q_y = Q_A$ **then**
      return y;
    else
      $i \leftarrow i + 1$
    end
  else
    $i \leftarrow i + 1$
  end
end
return false;

$$\begin{cases} k_{A,i} = k_{A,1}, s_{A,i} = s_{A,1}, r_{A,i} = r_{A,1}, m_{A,i} = m_{A,1} & \text{for } i = n_A + 1, \ldots, Max, \text{ if } Max > n_A, \\ k_{B,i} = k_{B,1}, s_{B,i} = s_{B,1}, r_{B,i} = r_{B,1}, m_{B,i} = m_{B,1} & \text{for } i = n_B + 1, \ldots, Max, \text{ if } Max > n_B. \end{cases} \tag{3.32}$$

This extension, obviously, adds no new information to either of the users. However, having both list of signatures with the same size makes the subsequent attack easier to describe and read. We have

$$\begin{cases} s_{A,i} \equiv k_{A,i}^{-1}(m_{A,i} + y_A r_{A,i}) \mod q, \\ s_{B,i} \equiv k_{B,i}^{-1}(m_{B,i} + y_B r_{B,i}) \mod q, \end{cases}$$
$$\iff \begin{cases} k_{A,i}s_{A,i} \equiv (m_{A,i} + y_A r_{A,i}) \mod q, \\ k_{B,i}s_{B_i} \equiv (m_{B,i} + y_B r_{B,i}) \mod q, \end{cases}$$

$$
\begin{aligned}
&\iff \begin{cases} k_{A,i}s_{A,i} - y_A r_{A,i} - m_{A,i} \equiv 0 \mod q, \\ k_{B,i}s_{B,i} - y_B r_{B,i} - m_{B,i} \equiv 0 \mod q, \end{cases} \\
&\iff \begin{cases} k_{A,i}s_{A,i}r_{B,i} - y_A r_{A,i}r_{B,i} - m_{A,i}r_{B,i} \equiv 0 \mod q, \\ k_{B,i}s_{B,i}r_{A,i} - y_B r_{A,i}r_{B,i} - m_{B,i}r_{A,i} \equiv 0 \mod q. \end{cases}
\end{aligned} \tag{3.33}
$$

for each $i \in \{1, \ldots, Max\}$. Thus, given $a, b \in \{1, \ldots, q-1\}$, we have

$$
ak_{A,i}s_{A,i}r_{B,i} + bk_{B,i}s_{B,i}r_{A,i} - (ay_A + by_B)r_{A,i}r_{B,i} - am_{A,i}r_{B,i} - bm_{B,i}r_{A,i} \equiv 0 \mod q \tag{3.34}
$$

for each $i \in \{1, \ldots, Max\}$. Hence, $x_i = ak_{A,i}s_{A,i}r_{B,i} + bk_{B,i}s_{B,i}r_{A,i}, y = ay_A + by_B$ for $i = 1, \ldots, Max$ is a solution to (2.16), where

$$
\begin{cases} t_i = r_{A,i}r_{B,i}, \\ a_i = -am_{A,i}r_{B,i} - bm_{B,i}r_{A,i}. \end{cases} \tag{3.35}
$$

If there is a known constant C such that $|ak_{A,i}s_{A,i}r_{B,i} + bk_{B,i}s_{B,i}r_{A,i}|_q \leq C$ for $i \in \{1, \ldots, Max\}$, and C and Max satisfy (2.16), we can compute $ay_A + by_B \mod q$ using a lattice reduction algorithm, so we get an equation

$$
ay_A + by_B \equiv c \mod q, \quad a, b \not\equiv 0 \mod q. \tag{3.36}
$$

That is not enough to recover either y_A or y_B. Nonetheless, that information can still be used to recover the private keys.

One way is to use the method described but instead of using $\{a, b\}$ in (3.34), using $\{a', b'\}$ such that $a'a^{-1} \not\equiv b'b^{-1} \mod q$. If the attack is successful we have

$$
\begin{aligned}
&\begin{bmatrix} a & b \\ a' & b' \end{bmatrix} \begin{bmatrix} y_A \\ y_B \end{bmatrix} \equiv \begin{bmatrix} c \\ c' \end{bmatrix} \mod q, \\
\iff &(ab' - a'b)^{-1} \begin{bmatrix} b' & -b \\ -a' & a \end{bmatrix} \begin{bmatrix} c \\ c' \end{bmatrix} \equiv \begin{bmatrix} y_A \\ y_B \end{bmatrix} \mod q.
\end{aligned} \tag{3.37}
$$

Therefore, we can recover both private keys and

$$
\begin{cases} y_A \equiv (ab' - a'b)^{-1}(b'c - bc') \mod q, \\ y_B \equiv (ab' - a'b)^{-1}(a'c - ac') \mod q. \end{cases} \tag{3.38}
$$

Similarly to what was done in the previous attacks, we can start by selecting $m \leq Max$ such that m and C satisfy (2.18), selecting m signatures and then using the lattice reduction algorithm on the matrix generated by those m equations.

Algorithm 7 implements the method described. Its input is a prime number, Alice's and Bob's list of

signatures, Alice's and Bob's public keys, two integers a and b, an integer C, and m, the number of signatures to be used. The algorithm starts by randomly selecting m signatures from the list using **Algorithm 2**, then it uses the lattice reduction algorithm on the matrix generated by those m signatures. Finally, a cycle goes though the lattice reduced basis and it returns $ay_A + by_B \mod q$ if it can be computed, and false otherwise.

Lemma 3.1.5. *If **Algorithm 7** does not return false, it returns $ay_A + by_B \mod q$. Moreover, if m is small, C and m satisfy* (2.18) *and $|ak_{A,i}s_{A,i}r_{B,i} + bk_{B,i}s_{B,i}r_{A,i}|_q \leq C$ for $i \in ReducedList$ we hope to recover $ay_A + by_B \mod q$. Furthermore, the algorithm runs in worst-case time $O(m^6 \log(q)^3 + m^6 \log(m)^3 + m \log(q)^4)$.*

Proof. The proof of this lemma is analogous to the proof of **Lemma 3.1.1**. □

As mentioned before, if we can use the method and compute two different linear equations for y_A and y_B, we can recover both y_A and y_B. However, computing one linear equation can be useful even if we can not compute another linear equation between the private keys. In Bob's equations, we can replace y_B by $b^{-1}(c - ay_A) \mod q$. After doing it, we get $n_A + n_B$ equations for y_A when before we only had n_A. Then, we can use one of the previous attacks to the $n_A + n_B$ equations. This is especially useful if **Algorithm 7** returns a linear equation between a user with a small amount of signatures and one with a large amount. In real world applications there are going to be many user which can have as little as one signature and there are also going to be users with millions of signatures. If this method can find a linear equation between the private keys of these two different types of users, the user with the small amount of signatures gets a great increase on the number of equations for its private key.

If we have a user Charlie, whose private key is y_C and we can compute one linear equation between y_A and y_B, one between y_A and y_C and one between y_B and y_C and all the linear equations are independent, we can recover y_A, y_B and y_C. Therefore, we can compute y_A, y_B, y_C without having to know two different linear equations between any two private keys. This method can be extended to as many users as we want to. That is, we can almost surely recover $y_1, \dots y_n$. if we can find a system of equations

$$\begin{cases} a_1 y_1 + b_1 y_2 \equiv c_1 \mod q, \\ \quad \vdots \\ a_n y_n + b_n y_1 \equiv c_n \mod q. \end{cases} \tag{3.39}$$

3.1.6 Computing two private keys through linear relation between nonces

In order to recover Alice's and Bob's private keys using the attack in 3.1.5 we need two successful lattice reduction algorithms. Moreover, we need to have $|ak_{A,i}s_{A,i}r_{B,i} + bk_{B,i}s_{B,i}r_{A,i}|_q \leq C$ in order for the algorithm to be successful. Thus, the values which we need to be small depend on $s_{A,i}$ and $s_{B,i}$ which behave as random variables, hence, the probability of the algorithm being successful is not higher when

Algorithm 7: Computing a linear equation between two private keys.

input : q, a prime number.
$ListSignA$, a list with n_A signatures from Alice .
$ListSignB$, a list with n_B signatures from Bob .
Q_A and Q_B, which are, respectively, the public keys of Alice and Bob.
m, the number of equations to be used in the lattice reduction algorithm.
C, an integer, smaller than q.
a, an integer between 1 and $q-1$.
b, an integer between 1 and $q-1$.

output: The algorithm returns $[ay_A + by_B]_q$ if it finds it and false otherwise.

$ReducedList \leftarrow$ List with m distinct random numbers between 1 and $\max\{n_A, n_B\}$;
for $i \leftarrow 1$ **to** m **do**
  $j \leftarrow$ `Get(`$ReducedList$`,`i`)`;
  if $j > n_A$ **then**
    $t_i \leftarrow$ `Add(`t_i`,`$r_{A,1}r_{B,j} \mod q$`)`;
    $a_i \leftarrow$ `Add(`a_i`,` $-am_{A,1}r_{B,j} - bm_{B,j}r_{A,1} \mod q$`)`;
  else if $j > n_B$ **then**
    $t_i \leftarrow$ `Add(`t_i`,`$r_{A,j}r_{B,1} \mod q$`)`;
    $a_i \leftarrow$ `Add(`a_i`,` $-am_{A,j}r_{B,1} - bm_{B,1}r_{A,j} \mod q$`)`;
  else
    $t_i \leftarrow$ `Add(`t_i`,`$r_{A,j}r_{B,j} \mod q$`)`;
    $a_i \leftarrow$ `Add(`a_i`,` $-am_{A,j}r_{B,j} - bm_{B,j}r_{A,j} \mod q$`)`;
  end
end
$M \leftarrow$`MatrixCreation(`q`,`t_i`,`a_i`,`m`,`C`)`;
$ReducedM \leftarrow$`LLL(`M`)`;
$Q \leftarrow a\,Q_A + b\,Q_B$;
$i \leftarrow 1$;
while $i \leq m+2$ **do**
  $l_i \leftarrow$ Line i of $ReducedM$;
  $z \leftarrow$`Get(`l_i`,`$m+2$`)`$/C$;
  if $|z| = 1$ **then**
    $x \leftarrow$`Get(`l_i`,`$m+1$`)`;
    $y \leftarrow -z * x * q/C \mod q$;
    $Q_y \leftarrow y\,P$;
    if $Q_y = Q$ **then**
      return y;
    else
      $i \leftarrow i+1$
    end
  else
    $i \leftarrow i+1$
  end
end
return false;

a bad PRNG is used to generate the nonces which is another problem with the attack. In order to try to fix both these problems this new method is implemented.

Let Alice be a user whose private key is y_A with signatures $\{(r_{A,1}, s_{A,1}), \ldots, (r_{A,n}, s_{A,n})\}$ on the messages $\{m_{A,1}, \ldots, m_{A,n}\}$ and Bob be a user whose private key is y_B with signatures $\{(r_{B,1}, s_{B,1}), \ldots, (r_{B,n}, s_{B,n})\}$ on the messages $\{m_{B,1}, \ldots, m_{B,n}\}$.
Given $a, b \in \{1, \ldots, q-1\}$, we have

$$\begin{aligned} & \begin{cases} k_{A,i} - s_{A,i}^{-1} r_{A,i} y_A - s_{A,i}^{-1} m_{A,i} \equiv 0 \mod q, \\ k_{B,i} - s_{B,i}^{-1} r_{B,i} y_B - s_{B,i}^{-1} m_{B,i} \equiv 0 \mod q, \end{cases} \\ \iff & \begin{cases} ak_{A,i} - as_{A,i}^{-1} r_{A,i} y_A - as_{A,i}^{-1} m_{A,i} \equiv 0 \mod q, \\ bk_{B,i} - bs_{B,i}^{-1} r_{B,i} y_B - bs_{B,i}^{-1} m_{B,i} \equiv 0 \mod q, \end{cases} \end{aligned} \tag{3.40}$$

for each $i \in \{1, \ldots, n\}$, thus

$$ak_{A,i} + bk_{B,i} - as_{A,i}^{-1} r_{A,i} y_A - bs_{B,i}^{-1} r_{B,i} y_B - as_{A,i}^{-1} m_{A,i} - bs_{B,i}^{-1} m_{B,i} \equiv 0 \mod q. \tag{3.41}$$

Therefore, $x_1 = ak_{A,1} + bk_{B,1}, \ldots, x_n = ak_{A,n} + bk_{B,n}, y = y_A, z = y_B$ is a solution to (2.23) where

$$\begin{cases} t_i = as_{A,i}^{-1} r_{A,i}, \\ t_i' = bs_{B,i}^{-1} r_{B,i} y_B, \\ a_i = -as_{A,i}^{-1} m_{A,i} - bs_{B,i}^{-1} m_{B,i}. \end{cases} . \tag{3.42}$$

Let us now assume that Alice is a user with n_A signatures and Bob is a user with n_B signatures, and $n_B > n_A$. Similarly to what was done in 3.1.5 we want to extend Alice's list of signatures to n_B signatures. For that attack, n_A could be as small as one and it still could be enough to successfully use the lattice attack. That was possible because the randomness of the t_i depended on $r_{A,i} r_{B,i}$, hence, as long as the sequence $r_{B,i}$ was random, the sequence $r_{A,i}$ did not have to be random. In this method if only one signature was used for Alice, t_i would be constant, thus there would be an integer x such that $|t_i x - t_i y_A|_p \leq 1$ for $i \in \{1, \ldots, n_B\}$, which, in turn, means that if $(b_1, \ldots, b_{n_B}, y_A, y_B, 1)M$ is a small vector in the lattice generated by M, the vector $(b_1, \ldots, b_{n_B}, x, y_B, 1)M$ will also be a small vector in the lattice generated by M. Thus, the algorithm might return y_B but the probability of it returning y_A is negligible. Therefore, n_A can not be as small as we want. As far as we know, there are no known inequalities relating n_A and C which would guarantee that the attack is successful, but we expect that if n_A and C satisfy (2.18) the attack is successful.

In a lattice reduction we want t_i to behave as randomly as possible, thus, instead of only repeating one signature, as it was done in 3.1.5, we will repeat the maximum possible number of signatures. In order to achieve that, we append Alice's list of signatures to itself $\lfloor n_B / n_A \rfloor$ times and then take its first

n_B entries. That is, for $i = \{n_A + 1, \ldots, n_B\}$ we define

$$\begin{cases} k_{A,i} = k_{A,j}, s_{A,i} = s_{A,j}, r_{A,i} = r_{A,j}, m_{A,i} = m_{A,j}, \text{ for } i \not\equiv 0 \bmod n_A, \text{ where } j = i \bmod n_A, \\ k_{A,i} = k_{A,n_A}, s_{A,i} = s_{A,n_A}, r_{A,i} = r_{A,n_A}, m_{A,i} = m_{A,n_A}, \text{ for } i \equiv 0 \bmod n_A \end{cases} \tag{3.43}$$

and then use the attack on the corresponding equations. Similarly to what was done in the previous attacks, we can start by selecting $m \leq n_B$ such that m and C satisfy (2.25), selecting m equations and then using the lattice reduction algorithm on the matrix generated by those m equations.

Algorithm 9 implements the method described. Its input is a prime number, Alice's and Bob's list of signatures, Alice's and Bob's public keys, two integers a and b, an integer C, the number of Alice's equations to be used m_A and the total number of signatures to be used m. The algorithm starts by randomly selecting m_A signatures from Alice's list using **Algorithm 2**, then it randomly selects m signatures from Bob's list of signatures also using **Algorithm 2**. Next, it uses the lattice reduction algorithm on the matrix generated by those m equations. Finally, a cycle goes though the lattice reduced basis and it returns both private keys if they can be computed, and false otherwise.

Lemma 3.1.6. *If **Algorithm 9** does not return false, it returns both private keys. Moreover, if m is small, C and m satisfy* (2.25)*, $m = m_A$ and $|ak_{A,j_i} + bk_{B,l_i}|_q \leq C$ for $i = \{1, \ldots, m\}$ where $j_i = ReducedListA[i]$ and $l_i = ReducedListB[i]$, we hope to recover both private keys. Furthermore, the algorithm runs in worst-case time $O(m^6 \log(q)^3 + m^6 \log(m)^3 + m \log(q)^4)$.*

Proof. If the algorithm does not return false, we have

$$\begin{aligned} & \begin{cases} Q_{Y,A} = Q_A, \\ Q_{Y,B} = Q_B, \end{cases} \\ \iff & \begin{cases} yP = y_A P, \\ zP = y_B P, \end{cases} \\ \iff & \begin{cases} y \equiv y_A \mod q, \\ z \equiv y_B \mod q. \end{cases} \end{aligned} \tag{3.44}$$

Thus, the algorithm returns the private keys. The matrix M is equal to

$$\begin{bmatrix} q & & & & & \\ & \ddots & & & & \\ & & q & & & \\ t_{j_1} & \ldots & t_{j_m} & \frac{C}{q} & & \\ t'_{l_1} & \ldots & t'_{l_m} & & \frac{C}{q} & \\ a_1 & \ldots & a_m & & & C \end{bmatrix} \tag{3.45}$$

where $t_{j_i}, t_{l_i}^{'}$ are the ones defined in (3.42) and $a_i = -as_{A,j_i}m_{A,j_i} - bs_{A,l_i}m_{A,l_i}$. By (3.41) $x_i = ak_{A,j_i} + bk_{B,l_i}, y = y_A, z = y_B$ is a solution to

$$x_i - t_{j_i}y - t_{l_i}^{'}y + a_i \equiv 0 \mod q. \tag{3.46}$$

Let us now assume that m is small, m and C satisfy (2.18), $m = mA$ and $|ak_{A,j_i} + bk_{B,l_i}|_q \leq C$ for $i \in \{1, \ldots, m\}$ where $j_i = ReducedListA[i]$ and $l_i = ReducedListB[i]$. Either $v = (ak_{A,j_1} + bk_{B,l_1}, \ldots, ak_{A,j_m} + bk_{B,l_m}, y_A\ C/q\ , y_B\ C/q\ , -C)$ or $-v$ will be a vector in the LLL-reduced basis of M, according to **Theorem 2.3.2**. The while cycle in line 19 will eventually reach that vector. If v is that vector, we have

$$\begin{cases} u = -1 \\ x_1 = y_A \frac{C}{q} \\ x_2 = y_B \frac{C}{q} \end{cases} . \tag{3.47}$$
$$\Rightarrow \begin{cases} y = y_A \mod q \\ z = y_B \mod q \end{cases}$$

If the vector is $-v$ we also have $y = y_A \bmod q, z = y_B \bmod q$.

The proof for the worst-case time is analogous to the one in **Lemma 3.1.1**. □

Similarly to **Algorithm 4**, if it is known that there exists $a, b \in \{1, \ldots, q-1\}$, such that given two nonces k_1 and k_2, generated by the PNRG, there is a disproportionately high probability of $ak_1 + bk_2$ being close to 0 or q, this attack should be used. If we do not know such a and b, we will try to use the method with $a = 1$ and $b = -1$.

Algorithm 8: MatrixCreation2V.

input : m, an integer number.
q, a prime number.
t_i, a vector with m entries.
$t_i^{'}$, a vector with m entries.
a_i, a vector with m entries.
C, an integer number
output: A matrix M like the one in (2.24).

1 $S \leftarrow \{\}$.;
2 $M \leftarrow$ Null $(m+3) \times (m+3)$ matrix;
3 **for** $j \leftarrow 1$ **to** m **do**
4 $\quad M(j,j) \leftarrow q$;
5 $\quad M(m+1,j) \leftarrow t_i(j)$;
6 $\quad M(m+2,j) \leftarrow t_i^{'}(j)$;
7 $\quad M(m+3,j) \leftarrow a_i(j)$;
8 **end**
9 $M(m+1,m+1) \leftarrow C/q$;
10 $M(m+2,m+2) \leftarrow C/q$;
11 $M(m+3,m+3) \leftarrow C$;

Algorithm 9: Computing two private keys through linear relation between nonces.

input : q, a prime number.
$ListSignA$, a list with n_A signatures from Alice .
$ListSignB$, a list with n_B signatures from Bob .
Q_A and Q_B, which are, respectively, the public keys of Alice and Bob.
m, the number of equations to be used in the lattice reduction algorithm.
m_A, the number of Alice's equations used.
C, an integer, smaller than q.
a, an integer between 1 and $q-1$.
b, an integer between 1 and $q-1$.

output: The algorithm returns y_A and y_B if it finds them and false otherwise.

$t_i \leftarrow \{\}$;
$t_i^{'} \leftarrow \{\}$;
$a_i \leftarrow \{\}$;
$ReducedListA \leftarrow$ List with m_A distinct random numbers between 1 and n_A;
$ReducedListB \leftarrow$ List with m distinct random numbers between 1 and n_B;
for $j \leftarrow 1$ **to** $\lfloor m/m_A \rfloor$ **do**
  $ReducedListA \leftarrow$`Append(`$ReducedListA$,$ReducedListA$`)`;
end
$ReducedListA \leftarrow$Take m first entries of $ReducedListA$;
for $i \leftarrow 1$ **to** m **do**
  $j \leftarrow$ `Get(`$ReducedListA$,i`)`;
  $k \leftarrow$ `Get(`$ReducedListB$,i`)`;
  $t_i \leftarrow$ `Add(`t_i,$a\ s_{A,j}^{-1} r_{A,j} \mod q$`)`;
  $t_i^{'} \leftarrow$ `Add(`$t_i^{'}$,$b\ s_{B,k}^{-1} r_{B,k} \mod q$`)`;
  $a_i \leftarrow$ `Add(`a_i,$-as_{A,j}^{-1} m_{A,j} - bs_{B,k}^{-1} m_{B,k} \mod q$`)`;
end
$M \leftarrow$`MatrixCreation2V(`q, t_i,$t_i^{'}$,a_i,m,C`)`;
$ReducedM \leftarrow$`LLL(`M`)`;
$i \leftarrow 1$;
while $i \leq m+3$ **do**
  $l_i \leftarrow$ Line i of $ReducedM$;
  $u \leftarrow$`Get(`l_i,$m+3$`)`$/C$;
  if $|u| = 1$ **then**
    $x_1 \leftarrow$`Get(`l_i,$m+1$`)`;
    $x_2 \leftarrow$ `Get(`l_i,$m+2$`)`;
    $y \leftarrow -u * x_1 * q/C \mod q$;
    $z \leftarrow -u * x_2 * q/C \mod q$;
    $Q_{Y,A} \leftarrow yP$;
    $Q_{Y,B} \leftarrow zP$;
    if $Q_{Y,A} = Q_A$ *and* $Q_{Y,A} = Q_B$ **then**
      return $\{y, z\}$;
    else
      $i \leftarrow i+1$
    end
  else
    $i \leftarrow i+1$
  end
end
return false;

3.2 Group attacks

The knowledge of a nonce used in a signature is enough to compute the respective private key. So, in the subsequent attacks we will try to compute the nonces used instead of the private keys. In the following attacks we will assume that $p/q \approx 1$ and that $p \equiv 3 \mod 4$.

3.2.1 Attack when the same nonce is used twice by the same user

Let (r_1, s_1) and (r_2, s_2) be two signatures on the messages $\{m_1, m_2\}$ with the same nonce, from one user whose private key is y. Since the nonces are equal, r_1 is equal to r_2. Thus, we have

$$\begin{aligned}
&\begin{cases} k_1 - {s_1}^{-1} r_1 y - {s_1}^{-1} m_1 \equiv 0 \mod q, \\ k_1 - {s_2}^{-1} r_1 y - {s_2}^{-1} m_2 \equiv 0 \mod q, \end{cases} \\
\Longrightarrow & {s_1}^{-1} r_1 y + {s_1}^{-1} m_1 \equiv {s_2}^{-1} r_1 y + {s_2}^{-1} m_2 \mod q, \\
\Longrightarrow & y r_1 ({s_1}^{-1} - {s_2}^{-1}) \equiv {s_2}^{-1} m_2 - {s_1}^{-1} m_1 \mod q, \\
\Longrightarrow & y \equiv r_1^{-1} ({s_1}^{-1} - {s_2}^{-1})^{-1} ({s_2}^{-1} m_2 - {s_1}^{-1} m_1) \mod q.
\end{aligned} \tag{3.48}$$

If $m_1 \neq m_2 \mod q$, we have $s_1 \neq s_2 \mod q$, thus, (3.48) returns the private key.

Let (r_1, s_1) and (r_2, s_2) be two signatures on the messages $\{m_1, m_2\}$, with nonces k_1 and k_2, respectively, from a user whose public key is Q. If $k_1 = k_2$ we have $r_1 = r_2$ but we can have $r_1 = r_2$ and $k_1 \neq k_2$. However, since $p \equiv 3 \mod 4$ and $p/q \approx 1$ we almost surely either have $k_1 = k_2$ or $k_1 = q - k_2$ as it was explained in 2.2. So, given two signatures (r_1, s_1) and (r_1, s_2) we can start by computing the y in (3.48) and test if $yP = Q$. If it is, y is the private key of the user, otherwise we must have $k_1 = q - k_2$. The pair $(r_1, -s_1)$ is the signature on the message m_1 with nonce equal to $q - k_1$. Thus, the signatures $(r_1, -s_1)$ and (r_2, s_2) will have the same nonce. Therefore, the y in (3.48) is the user's private key almost surely.

3.2.2 Attack when there is a known linear equation between two nonces used by the same user

Let (r_1, s_1) and (r_2, s_2) be two signatures on the messages $\{m_1, m_2\}$ with nonces k_1 and k_2, respectively, from a user whose private key is y. We have

$$\begin{aligned}
&\begin{cases} s_1 \equiv k_1^{-1}(m_1 + y r_1) \mod q, \\ s_2 \equiv k_2^{-1}(m_2 + y r_2) \mod q, \end{cases} \\
\Longleftrightarrow & \begin{cases} r_1^{-1}(k_1 s_1 - m_1) \equiv y \mod q, \\ r_2^{-1}(k_2 s_2 - m_2) \equiv y \mod q, \end{cases} \\
\Longrightarrow & \begin{cases} r_1^{-1}(k_1 s_1 - m_1) \equiv r_2^{-1}(k_2 s_2 - m_2) \mod q, \end{cases}
\end{aligned} \tag{3.49}$$

Thus, with two different signatures we can compute some a_1, b_1, c_1 such that $a_1k_1+b_1k_2 \equiv c_1 \mod q$. If we know other a_2, b_2, c_2 such that $a_2k_1 + b_2k_2 \equiv c_2 \mod q$ we can compute both nonces unless the equations are linearly dependent, which has a negligible probability if we assume that a_2 and b_2 are random. Therefore, knowing a linear equation between two nonces for the same user is enough to almost surely compute that user's private key.

3.2.3 Attack when there is a known linear equation between two nonces used by different users

Let (r_1, s_1) be a signature on a message m_1 with nonce k_1 from one user whose private key is y_A and (r_2, s_2) be a signature on a message m_2 with nonce k_2 from one user whose private key is y_B. Let us also assume that we know $a, b, c \in \{1, \ldots, q-1\}$ such that $ak_1 + bk_2 \equiv c \mod q$. We have

$$\begin{cases} s_1 \equiv k_1^{-1}(m_1 + y_A r_1) \mod q, \\ s_2 \equiv k_2^{-1}(m_2 + y_B r_2) \mod q, \end{cases} \tag{3.50}$$

$$\iff \begin{cases} k_1 \equiv s_1^{-1}(m_1 + y_A r_1) \mod q, \\ k_2 \equiv s_2^{-1}(m_2 + y_B r_2) \mod q, \end{cases} \tag{3.51}$$

$$\iff \begin{cases} ak_1 \equiv as_1^{-1}(m_1 + y_A r_1) \mod q, \\ bk_2 \equiv bs_2^{-1}(m_2 + y_B r_2) \mod q, \end{cases} \tag{3.52}$$

$$\implies \begin{cases} c \equiv as_1^{-1}(m_1 + y_A r_1) + bs_2^{-1}(m_2 + y_B r_2) \mod q. \end{cases} \tag{3.53}$$

Thus, from a linear equation between the nonces we get a linear equation between the private keys. If we can find another linear equation for two nonces we can compute both private keys. However, as it was described in 3.1.5, even if we are not able to find another linear equation, the knowledge of the linear equation between the private keys can still be useful to compute the private keys.

3.2.4 Using a hash table to get a nonce

Let (r_1, s_1) be a signature. In order to check if the nonce used for this signature is k_1, we start by computing $k_1P = (x_1, y_1)$, where P is the system-wide point of the elliptic curve Then, we check if $x_1 = r_1$, if it is we are almost surely guaranteed that the nonce used was k_1 or $q-k_1$, otherwise we know that neither k_1 nor $q-k_1$ was the nonce used for the signature. An important property about this process is that it does not depend on the private key. Thus, we can store the coordinate x_1 as a variable and if we find a user which has a signature equal to (x_1, s) we are almost surely guaranteed that the signature was generated by the nonce k_1 or by $q - k_1$, and, consequently, we can compute that user's private key.

So, for this attack, we start by selecting a set $S = \{k_1, \ldots, k_m\}$ and computing the values $k_iP = (x_i, y_i)$ for $i \in \{1, \ldots, m\}$. Then, we store the vectors (x_i, y_i, k_i) in a hash table. The key used to select where to store (x_i, y_i, k_i) is $f(x_i)$, where f is the hash table's hash function. Then, given a signature (r, s) we compute $f(r)$ and search the hash table for a pair with (r, y, k) in the values with index $f(r)$. If

such a value exists we can compute the private key almost surely, otherwise we test different signatures.

The downsides of this attack is that for large values of m the computation of S is very slow and storing the hash table is unfeasible. The values of m that are still physically feasible depend on the computational power, but for example $m = 10^9$ is a value which for modern computers is feasible.

Suppose that we know a set $\{a, a+1, \ldots, b\}$ such that for the PNRG used to generate the nonces the probability of the nonce being in that set is higher than if a true RNG was used. Then, we can use $\{a, \ldots, b\}$ as the set S. In [6] it was observed that the nonces are equal to $(q-1)/2$ with much higher probability than expected in Bitcoin transactions' signatures. Therefore, the set S used can be $\{(q-1)/2 - n, \ldots, (q-1)/2 + n\}$ for some positive integer n.

Let us assume that we were able to compute the private key of a user with m signatures. Then, for each of his signatures the nonces used can be computed. Since the nonces are generated using a PNRG, if k was used as a nonce the probability of another nonce being equal to k is higher than $1/q$. Thus, adding the m nonces used by that user to the set S increases the probability of another user finding his nonce in S more than if another random subset of m integers was added. If that user is one of the users with millions of signatures, S can be replaced by the user's list of nonces and the probability of another user being able to compute his private key might increase immensely.

3.2.5 Using a hash table to a get a linear relation between two nonces for the same user

Let (r_1, s_1) and (r_2, s_2) be two signatures on messages $\{m_1, m_2\}$ with nonces k_1 and k_2, respectively, from one user whose private key is y. We want to find a linear equation between k_1 and k_2 and then, use the attack in 3.2.2. We can compute y_1 from r_1 using **Theorem 2.1.6**, and either k_1P is equal to (r_1, y_1) or it is equal to $(r_1, -y_1)$ almost surely. We can also compute y_2 from r_2 using **Theorem 2.1.6**, and either k_2P is equal to (r_2, y_2) or it is equal to $(r_2, -y_2)$ almost surely. So, we have four possibilities

1. $k_1P = (r_1, y_1)$ and $k_2P = (r_2, y_2)$;
2. $k_1P = (r_1, y_1)$ and $k_2P = (r_2, -y_2)$;
3. $k_1P = (r_1, -y_1)$ and $k_2P = (r_2, y_2)$;
4. $k_1P = (r_1, -y_1)$ and $k_2P = (r_2, -y_2)$.

Given $a, b \in \{1, \ldots, q-1\}$ we start by computing the point $a(r_1, y_1) + b(r_2, y_2) = (x, y)$ and then we search the hash table for a point with first coordinate equal to x. If there exists one, it can be written as (x, y_1, k) for some y_1 and k. If $y = y_1$ we have $a + b \equiv k \mod q$, otherwise we must have $y \equiv -y_1 \mod p$ and $a + b \equiv -k \mod q$. Let us define a variable k' such that $k' = k$ if $y = y_1$ and $k' = q - k$ if $y = -y_1$. We have one of the four possibilities

1. $ak_1 + bk_2 \equiv k' \mod q$;
2. $ak_1 - bk_2 \equiv k' \mod q$;

3. $-ak_1 + bk_2 \equiv k^{'} \mod q$;

4. $-ak_1 - bk_2 \equiv k^{'} \mod q$.

For each of these possibilities we use the method in 3.2.2 to return some y, then, for each y we compute yP and if is equal to Q we know that y is the private key. Since we are assuming that there exists a point (x, y_1, k) in S, one of these four possibilities will return the private key.

If the method does not find a point with first coordinate equal to x in the hash table, we can try the method but with the point $a(r_1, y_1) + b(r_2, -y_2)$ instead of $a(r_1, y_1) + b(r_2, y_2)$. If with that point it still does not find the private key we try with $a(r_1, -y_1) + b(r_2, y_2)$ and, finally, if that one is still not successful we try the method with the point $a(r_1, -y_1) + b(r_2, -y_2)$. If for either of the four points there is an element in the hash table with the same x coordinate we can almost surely find a linear equation between the nonces and, consequently, recover the the private key. Otherwise that means that $(\pm ak_1 \pm bk_2)P$ is not one of the points in the hash table, thus, we either need to select different values of a and b or to select different nonces.

Using a hash table to a get a linear equation between two nonces for different users can be done in a very similar way.

The choice of a and b depends on the PNRG used. Similarly to what was done in 3.1.2 if there are known $a, b \in \{1, \ldots, q-1\}$, such that given two nonces k_1 and k_2, generated by the PNRG, there is a disproportionately high probability of $ak_1 + bk_2$ being in a set S_1, we use those values of a and b and take S_1 to be the set S for the hash table. If we do not know such a and b, we will use $a = 1$ and $b = -1$ and the set S can be any set.

3.2.6 Creating an hash table with a multi-core processor

Let S be the set that we want to use for the hash table. Let us denote $\#S$ by m and $\max\{x : x \in S\}$ by M. In order to compute jP for $j \in S$ we could start by computing $2^i P$ for $0 \leq i \leq 2^{\lfloor \log M \rfloor}$ and then using the double and add method for each $j \in S$. If this process is done in a single processor it takes at most $(m+1)\lfloor \log M \rfloor T$ time, where T is the time that it takes to compute an elliptic curve addition. Let us now assume that we have a multi-core processor with n processors. We can start by dividing S into n sets $S_1, \ldots, S_n$ in a way that $\#S_i \leq \lceil \frac{m}{n} \rceil$ for $i = 1, \ldots, n$. In order to compute jP for $j \in S$, one processor starts by computing $2^i P$ for $0 \leq i \leq 2^{\lfloor \log M \rfloor}$ and send that list to the other $n-1$ processors. Then, each processor i computes jP for $j \in S_i$ and send the list to the first processor. Each of these processors can be run in parallel, thus, if we ignore the time it takes to send the lists, which is negligible compared to the other operations done, this whole process takes at most $(1 + \lceil \frac{m}{n} \rceil)\lfloor \log M \rfloor T$ time. So, the process can be approximately n times faster for a multi-core processor with n processors.

Chapter 4

Results

We wish to compute the probability of success of the lattice attacks as a function of m and C, where m is the number of equations used, and C is a constant such that $|b_i|_q \leq C$ for $i = 1, \ldots, m$.

It was proved in the previous sections that the attack in **3.1.6** is reducible to the one in **2.3.1.2**, and the ones in **3.1.1,3.1.2,3.1.3,3.1.4,3.1.5** are reducible to **2.3.1.1**. Thus, in order to test the attacks, we will test the attacks in **2.3.1.1** and **2.3.1.2**. Two different types of tests will be done, in the first one we will fix C and compute the probability of success as a function of m. This test is useful, for example, to find out how many equations should be used if there is a side-channel attack which can recover a number of most, or least, significant bits. The second type will fix a small m and compute the probability of success as a function of C. This test will be used to test the probability of success for users with a very small number of equations, up to ten. Moreover, it will be used to decided how many equations should be used to have the highest probability of success if the b_i are randomly generated.

The prime number q used in these tests will be the order of P in the elliptic curve secp256k1, the curve used in Bitcoin's public key-cryptography. The numbers t_i and b_i will be generated with the RandomInteger function in Mathematica and the lattice reduction algorithm used is also the one defined in Mathematica. These test were run in a computer with 8 GB RAM and a i7-7500U processor.

4.1 One variable lattice attack

Given C and m we want to compute the probability of the one variable hidden number problem being successful. In order to do that, we design an algorithm which starts by randomly generating an integer $1 \leq \alpha \leq q-1$, the vector $(t_1, \ldots, t_m)$, where $1 \leq t_i \leq q-1$, and the vector $(b_1, \ldots, b_m)$ where $-C \leq b_i \leq C$. Then, it computes $(a_1, \ldots, a_m)$, in such a way that $x_i = b_i, y = \alpha$ is a solution to (2.16). Lastly, the algorithm uses LLL on $\mathcal{L}(M)$ where M is the matrix (2.17), and it returns 1 if either the vector $v = (b_1, \ldots, b_m, \alpha C/q, -C)$ or $-v$ is a vector in the reduced basis and 0 otherwise. Each of these tests were run thirty times. The average of those thirty tests was taken, and that average is the experimental probability of success.

4.1.1 Fixed limit on the norm of the solution to the equation

Since the prime number q has 256 bits, C can be smaller than 2^{255}. The values of C that will be tested are 2^{255-l}, for $l = \{6, \ldots, 10\}$ which, according to **Corollary 3.1.3.1** and **Corollary 3.1.4.1**, correspond to side-channel attacks which recover $\{6, \ldots, 10\}$ most, or least, significant bits. For each value of l, the value of m presented in Table 4.1 is the smallest integer such that C and m satisfy (2.18). Theoretically, for these values of l and m, we have a high probability of recovering α using the LLL-reduction algorithm.

l	6	7	8	9	10
m	67	51	42	36	31

Table 4.1: Theoretical number of equations needed for one variable lattice attack, as a function of l.

In **Theorem 2.3.1** we assumed that m was small but we did not specify how small m has to be. Thus, for each l, the value of m in Table 4.1 might no be enough to have a high probability of success, we might need more equations. So, for each l, an interval $\{\min m, \ldots, \max m\}$ was defined, presented in Table 4.2, and for each $n \in \{\min m, \ldots, \max m\}$ we tested the probability of success for $m = n$ and $C = 2^{255-l}$. The results of these tests are presented in Figures 4.1, 4.2, 4.3.

l	6	7	8	9	10
$\min m$	63	45	35	30	23
$\max m$	90	70	55	50	35

Table 4.2: Number of equations to be tested for one variable lattice attack, as a function of l.

The results for $l = 6$ are very poor. For most number of equations tested the probability of success is equal to zero, and it is smaller than 7% for all number of equations tested. We could not run the tests for values of m greater than 90, since the tests were already too slow for $m = 90$, so we are not sure if this is a problem which could be corrected if more equations were used, or if for $l = 6$ the attack can not recover the private key with a high probability of success. For $l = 7$, if $m \geq 59$, the probability of success is at least 45%. The number of equations needed is slightly greater than expected and the probability of success does not converge as m grows, however, it is enough to greatly endanger the security of the private keys if there exists a side-channel attack which can recover 7 MSB or LSB of the nonces. For $l = 8$, if $m \geq 42$ there is at least a 70% probability of success and for $m \geq 45$ there is at least a 90% probability of success. For $l = 9$, if $m \geq 36$ there is at least a 85% probability of success and for $m \geq 38$ there is a 100% probability of success. For $l = 10$, if $m \geq 31$ there is at least a 90% probability of success. As expected, for $l \in \{8, 9, 10\}$ the attack has a higher probability of success than for $l \in \{6, 7\}$. Moreover, for each $l \in \{8, 9, 10\}$ a number of equations can be chosen such that the probability of success of the attack is at least 90%.

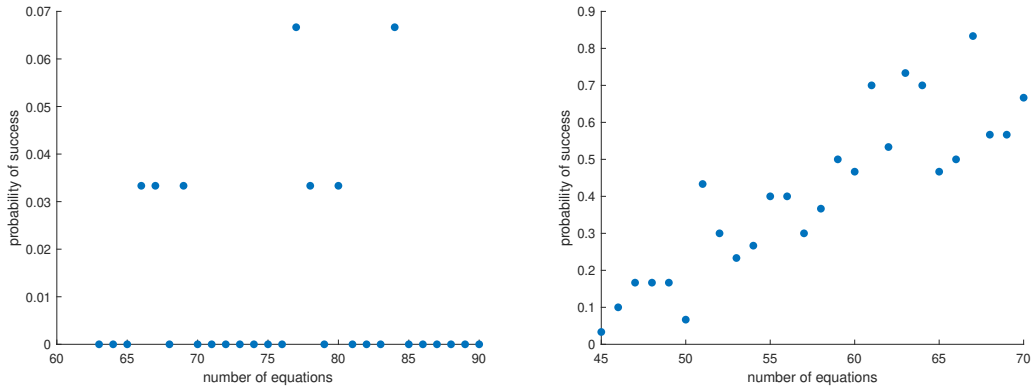

Figure 4.1: Probability of success as a function of the number equations for the one variable lattice attack. The left figure has $C = 2^{249}$ and the right one has $C = 2^{248}$.

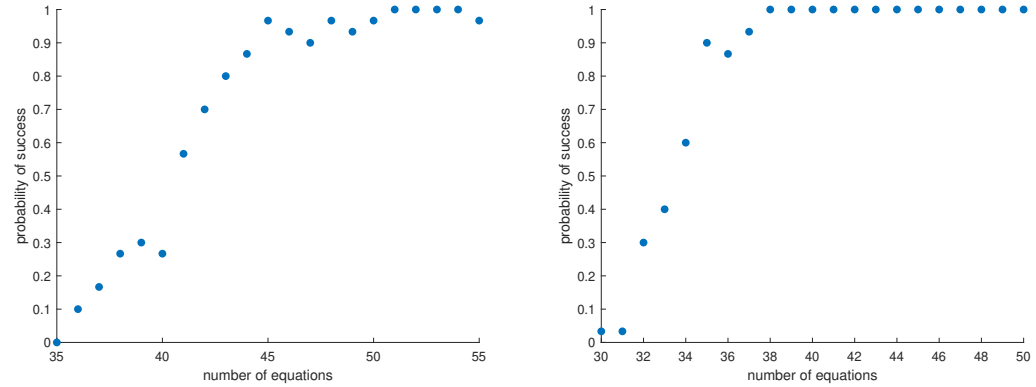

Figure 4.2: Probability of success as a function of the number equations for the one variable lattice attack. The left figure has $C = 2^{247}$ and the right one has $C = 2^{246}$.

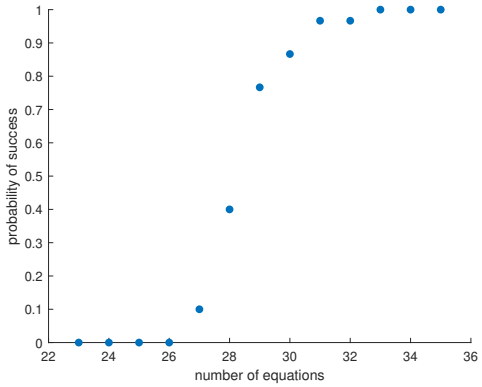

Figure 4.3: Probability of success as a function of the number equations for the one variable lattice attack with $C = 2^{245}$.

4.1.2 Fixed number of equations

Let us assume that the b_i are randomly generated in the interval $\{0, \ldots, q-1\}$. Then, the probability of $|b_i|_q \leq C$ for $i = 1, \ldots, m$ is

$$\left(\frac{2C+1}{q}\right)^m. \tag{4.1}$$

Moreover, let us assume that if $|b_i|_q \leq C$ for $i = 1, \ldots, m$ the probability of success of the one variable lattice attack is p_1. Then, the probability of success of the one variable lattice attack for random b_i, which we will denote by real probability of success for C and m, is at least

$$\left(\frac{2C+1}{q}\right)^m p_1. \tag{4.2}$$

In the following tests, we want to find out what is the pair (m, C) which has the highest real probability of success for $m \in \{2, \ldots, 10\}$. In order to do that, we started by fixing $m \in \{2, \ldots, 10\}$ and then, computing the probability of success as a function of C. Since (2.18) relates $\log C$ with m, the values used for C will all be powers of two. For each value of m, the value of $\log C$ presented in Table 4.3 is the smallest integer such that $\log C$ and m satisfy (2.18). Theoretically, for these values of m and C, we have a high probability of computing α using the LLL-reduction algorithm.

m	2	3	4	5	6	7	8	9	10
$\log C$	126	168	190	202	211	217	221	225	228

Table 4.3: Theoretical values of $\log C$ needed as a function of m, for the one lattice attack.

For each m, the values of $\log C$ tested are the values between $\min \log C$ and $\max \log C$ in Table 4.4. The results of these tests are presented in Figures 4.4, 4.5, 4.6,4.7,4.8.

m	2	3	4	5	6	7	8	9	10
$\min \log C$	115	160	180	195	200	205	210	215	220
$\max \log C$	135	180	200	215	220	225	230	235	240

Table 4.4: Tested values of $\log C$ as a function of m, for the one lattice attack.

As expected, for these values of m the probability of success as a function of $\log C$ is decreasing. Moreover, it is a steep decrease from the points where the probability is equal to 1 to the points where it is equal to 0. The decrease is steeper for greater values of m. Comparing Table 4.5 with Table 4.3, we can conclude that for all values of m the theoretical values of $\log C$ needed are enough to have at least a 90% probability of success. Furthermore, for some values of m, the value of $\log C$ can be greater than the one predicted by (2.18) and still have a probability of success greater than 90%.

For each m, we denote the value of $\log C$ which has the greatest real probability of success by the optimal value of $\log C$ for m. The values of these optimal values are presented in Table 4.6.

Number of equations	2	3	4	5	6	7	8	9	10
$\max \log C$ with 90% probability of success	126	169	190	203	212	218	222	226	229

Table 4.5: Maximum values of $\log C$ that have at least 90% probability of success for the one variable lattice attack.

Number of equations	2	3	4	5	6	7	8	9	10
Optimal choice of $\log C$	130	171	191	204	213	219	223	227	230
Real probability of success($\times 10^{-77}$)	3.7	2.3	0.58	1.2	2.8	2.8	0.66	1.4	1.8

Table 4.6: Real probability of success for optimal choice of $\log C$ for the one lattice attack.

Therefore, according to these tests, if the b_i are randomly generated and we have at most ten signatures, we should only select two of those signatures and use the lattice reduction algorithm for the matrix associated with those two equations. We also conclude that if the b_i are random the real probability of success is negligible. So, in order to have a non negligible real probability of success, we need to know a weakness in the PNRG used to generate the nonces and design a way to exploit it.

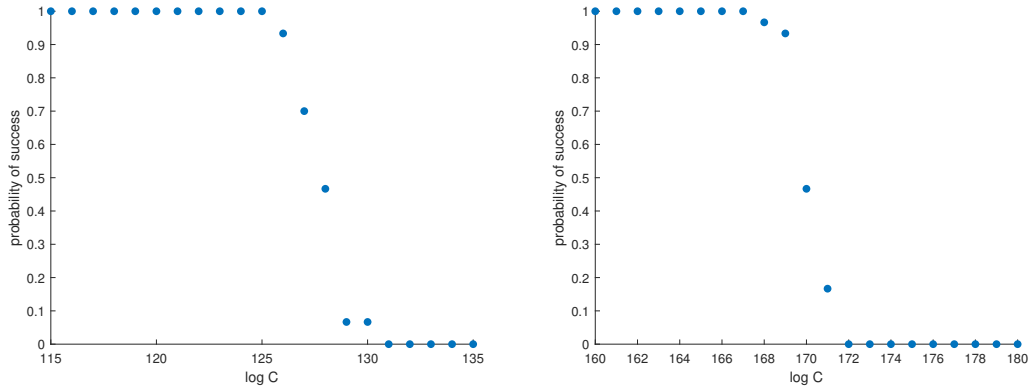

Figure 4.4: Probability of success as a function of $\log C$, for the one variable lattice attack. The left figure has $m = 2$ and the right one has $m = 3$.

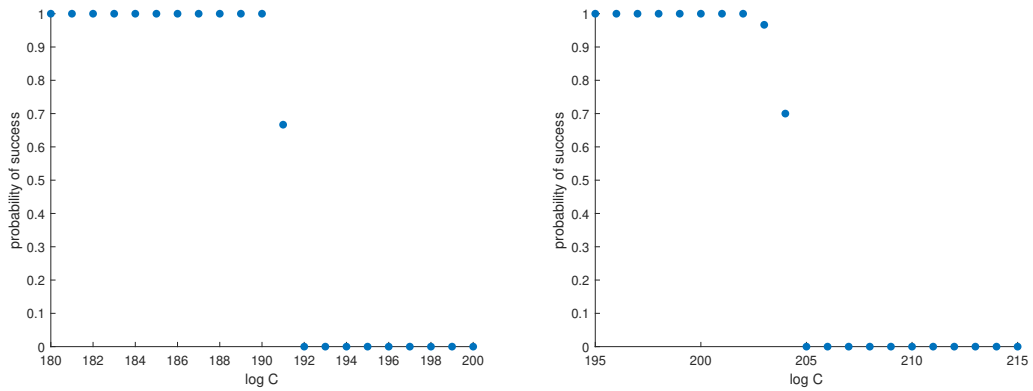

Figure 4.5: Probability of success as a function of $\log C$, for the one variable lattice attack. The left figure has $m = 4$ and the right one has $m = 5$.

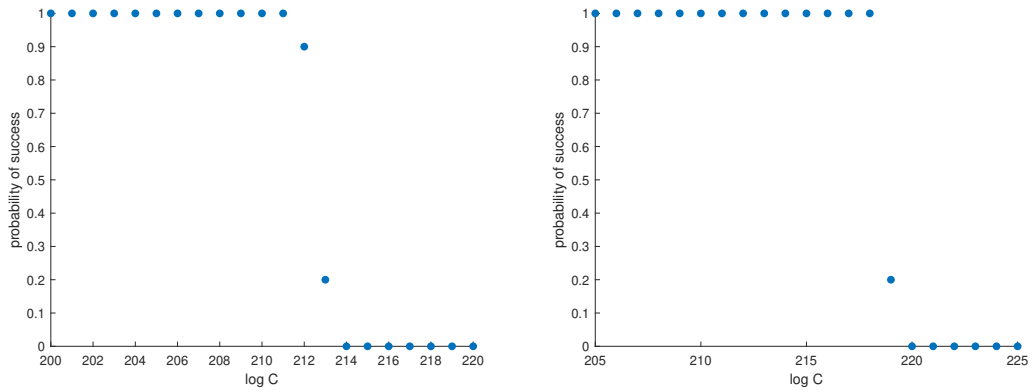

Figure 4.6: Probability of success as a function of $\log C$, for the one variable lattice attack. The left figure has $m = 6$ and the right one has $m = 7$.

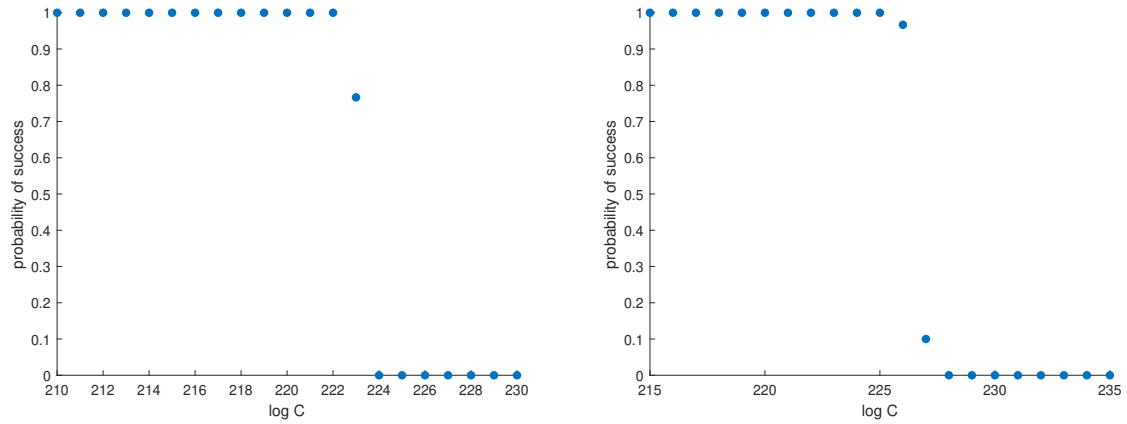

Figure 4.7: Probability of success as a function of $\log C$, for the one variable lattice attack. The left figure has $m = 8$ and the right one has $m = 9$.

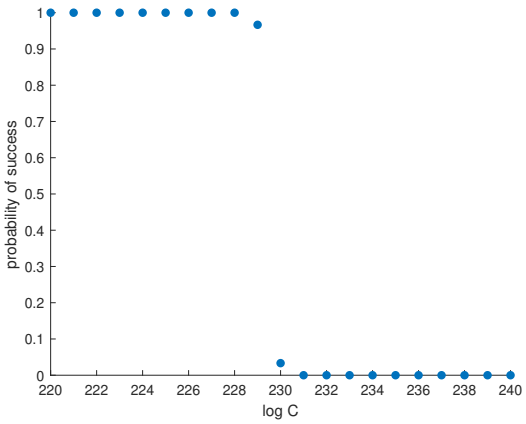

Figure 4.8: Probability of success as a function of $\log C$, for the one variable lattice attack with $m = 10$.

4.2 Two variables lattice attack

Given C and m we want to compute the probability of the two variables lattice attack computing the private keys. In order to do that we design an algorithm which starts by randomly generating two integers

$1 \leq \alpha, \beta \leq q-1$, the vectors $(t_1, \ldots, t_m)$ and $(t_1^{'}, \ldots, t_m^{'})$, where $1 \leq t_i, t_i^{'} \leq q-1$, and the vector $(b_1, \ldots, b_m)$ where $-C \leq b_i \leq C$. Then, it computes $(a_1, \ldots, a_m)$, in such a way that $x_i = b_i, y = \alpha, z = \beta$ is a solution to (2.23). Lastly, the algorithm uses LLL on $\mathcal{L}(M)$ where M is the matrix (2.24), and it returns 1 if either the vector $v = (b_1, \ldots, b_m, \alpha C/q, \beta C/q, -C)$ or $-v$ is a vector in the reduced basis and 0 otherwise. Each of these tests were run thirty times. The average of those thirty tests was taken, and that average is the experimental probability of success.

4.2.1 Fixed limit on the norm of the solution to the equation

Similarly to what it was done for one variable, the values of C tested will be powers of two. Those values will be 2^{255-l} for $l = \{10, \ldots, 20\}$. These values of l are greater compared to the ones used for one variable since the number of equations needed for the same C is considerably greater for two variables, thus, using the same values of l would make the attack too slow to compute.

For each value of l, the value of m presented in Table 4.7 is the smallest integer such that C and m satisfy (2.25). Theoretically, for those values of l and m, we have a high probability of computing α and β using the LLL-reduction algorithm.

For each l, the number of equations tested are the values between $\min m$ and $\max m$ in Table 4.8. The results of these tests are presented in Figures 4.9, 4.10, 4.11, 4.12,4.13, 4.14.

l	10	11	12	13	14	15	16	17	18	19	20
m	66	58	52	47	43	40	37	34	32	30	29

Table 4.7: Theoretical number of equations needed for the two variables lattice attack.

l	10	11	12	13	14	15	16	17	18	19	20
$\min m$	77	55	50	45	40	35	30	30	25	20	20
$\max m$	100	80	75	65	60	55	50	50	45	40	40

Table 4.8: Number of equations tested for the two variables lattice attack.

For $l = 10$ the probability of success does not seem to converge as m grows, moreover the probability of success is smaller than 60% for all values of m tested. These poor results are expected since the number of equations used are close to 100. For $l = 11$, the probability of success as a function of m is not an increasing function and it does not seem to converge as m grows . However, for $m \geq 66$ the probability of success is greater than 70% and for $m \geq 73$ it is greater than 80%. Thus, for a large enough number of equations, we have a high probability of success of recovering α and β assuming that $|b_i|_q \leq 2^{244}$. For each other value of l tested, there is an integer n_l, such that for $m \geq n_l$, the probability of success for m equations with $C = 2^{255-l}$ is equal to 1. Therefore, for $l \in \{12, \ldots, 20\}$ the two variables lattice attack will recover both private keys if $|b_i|_q \leq 2^{255-l}$ for $i = 1, \ldots, n_l$.

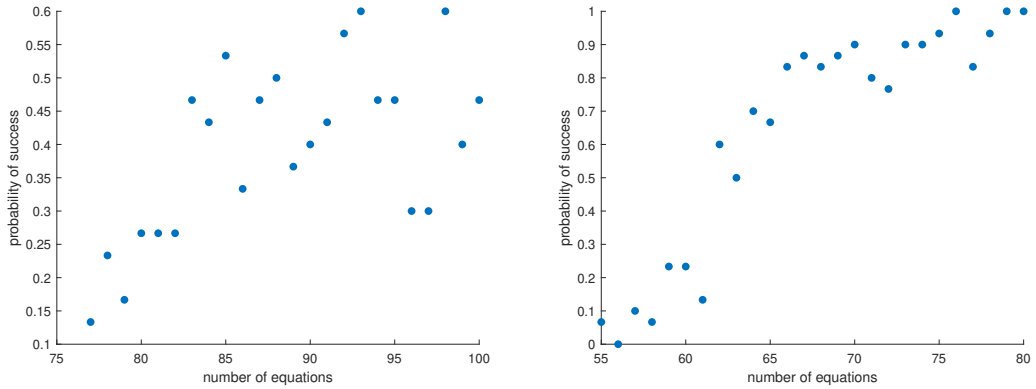

Figure 4.9: Probability of success a function of the number of equations, for the two variables lattice attack. The left figure has $l = 10$ and the right one has $l = 11$.

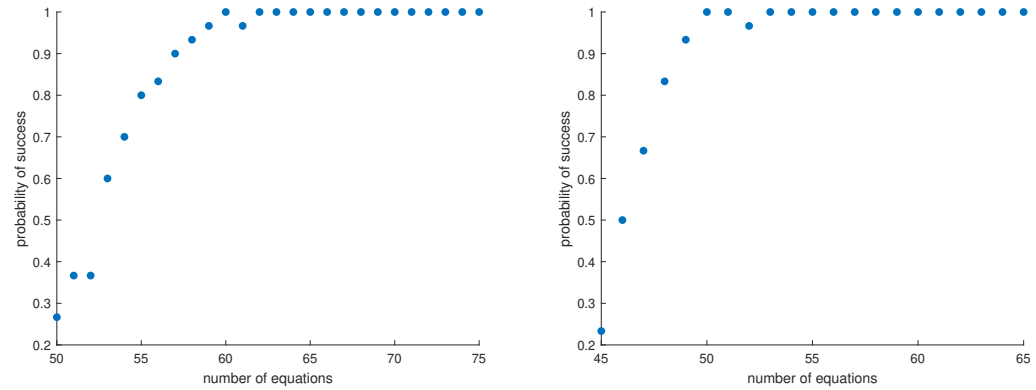

Figure 4.10: Probability of success a function of the number of equations, for the two variables lattice attack. The left figure has $l = 12$ and the right one has $l = 13$.

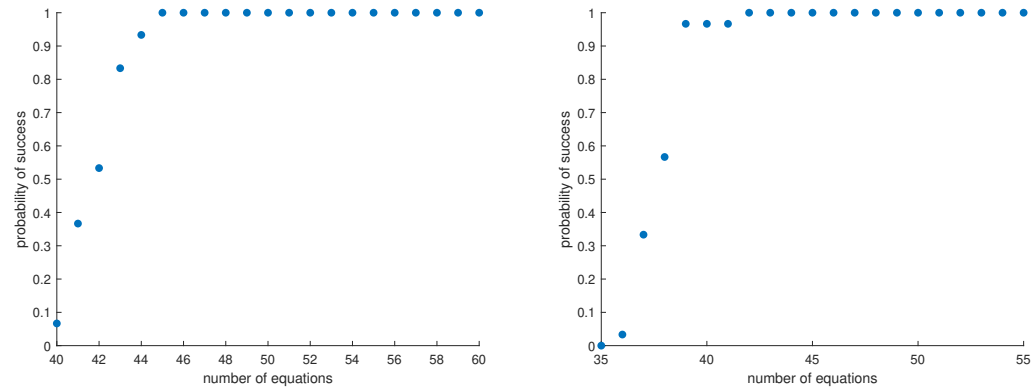

Figure 4.11: Probability of success a function of the number of equations, for the two variables lattice attack. The left figure has $l = 14$ and the right one has $l = 15$.

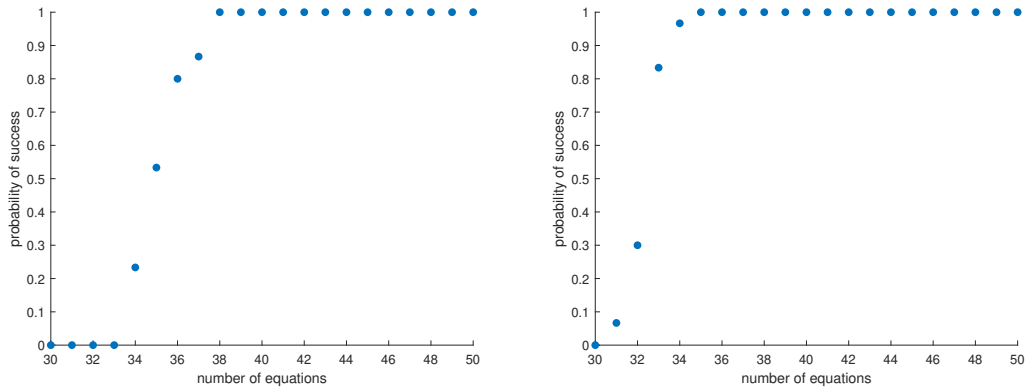

Figure 4.12: Probability of success a function of the number of equations, for the two variables lattice attack. The left figure has $l = 16$ and the right one has $m = 17$.

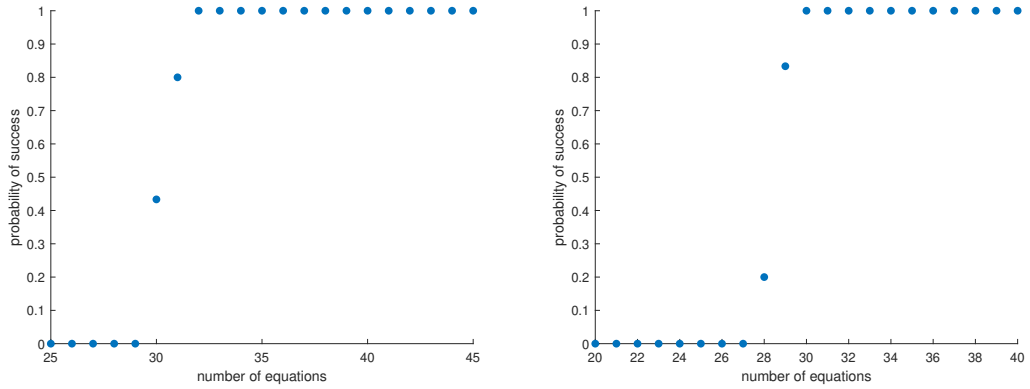

Figure 4.13: Probability of success a function of the number of equations, for the two variables lattice attack. The left figure has $l = 18$ and the right one has $m = 19$.

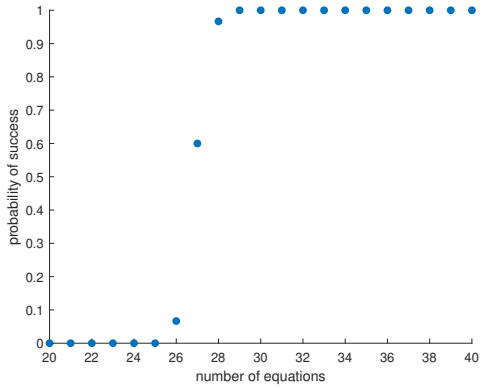

Figure 4.14: Probability of success a function of the number of equations, for the two variables lattice attack with $l = 20$.

4.2.2 Fixed number of equations

Analogously to what was done for one variable, we want to find out how many equations should be used to achieve the highest real probability of success when the number of equations is small. Since in (2.23) there are $n+2$ variables for n equations, the two variables lattice reduction algorithm needs at least three equations. The values of m tested range from three to ten. For each m, the value of $\log C$ presented in Table 4.9 is the smallest integer such that $\log C$ and m satisfy (2.25).

m	3	4	5	6	7	8	9	10
$\log C$	82	125	151	168	180	189	196	202

Table 4.9: Theoretical values of $\log C$ needed for the two variables lattice attack.

For each m, the values of $\log C$ tested are the values between $\min \log C$ and $\max \log C$ in Table 4.10. The results of these tests are presented in Figures 4.15, 4.16, 4.17,4.18.

m	3	4	5	6	7	8	9	10
$\min \log C$	75	115	145	160	170	180	190	195
$\max \log C$	95	135	165	180	190	200	210	215

Table 4.10: Values of $\log C$ tested for the two variables lattice attack.

The behaviour of these graphics is very similar to the ones for one variable. For each m, the probability of success as a function of $\log C$ is also decreasing and the decrease from the points where the probability is equal to 1 to the points where it is equal to 0 is steep, and the greater the m, the steeper it is. Comparing Table 4.9 and Table 4.11, we infer that for each m, the values of C can be two times greater than the ones expected and the method still has, at least, a 90% probability of success.

Number of equations	3	4	5	6	7	8	9	10
$\max \log C$ with 90% probability of success	83	126	152	169	181	190	197	203

Table 4.11: Maximum values of $\log C$ that have at least 90% probability of success for the two variables lattice attack.

For each m, we denote the value of $\log C$ which has the greatest real probability of success by the optimal value of $\log C$ for m. The values of these optimal values are presented in Table 4.12.

Number of equations	3	4	5	6	7	8	9	10
Optimal choice of $\log C$	86	128	153	170	182	191	198	204
Real probability of success($\times 10^{-154}$)	7.2	6.4	1.1	2.9	0.80	0.75	0.37	3.0

Table 4.12: Real probability of success for optimal choice of $\log C$ for the two variables lattice attack.

Therefore, according to these tests, if the b_i are generated randomly and we have at most ten signatures, we should only select three of those signatures and use the lattice reduction algorithm for the

matrix associated with these three equations. We also conclude that if the b_i are random the real probability of success is very low, randomly generating an integer in $[0, q-1]$ and testing if it is the private key has a higher probability of success than this method. Hence, in order to have a non negligible real probability of success, we need to know a weakness in the PNRG used to generate the nonces and design a way to exploit it.

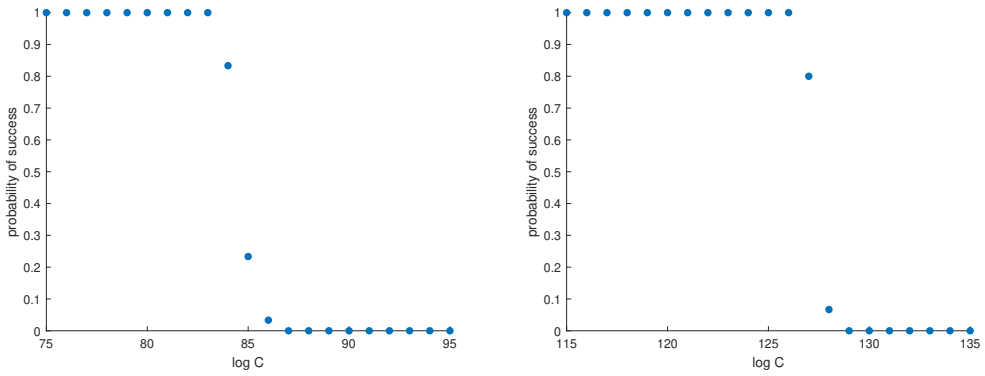

Figure 4.15: Probability of success as a function of $\log C$, for the two variables lattice attack. The left figure has $m = 3$ and the right one has $m = 4$.

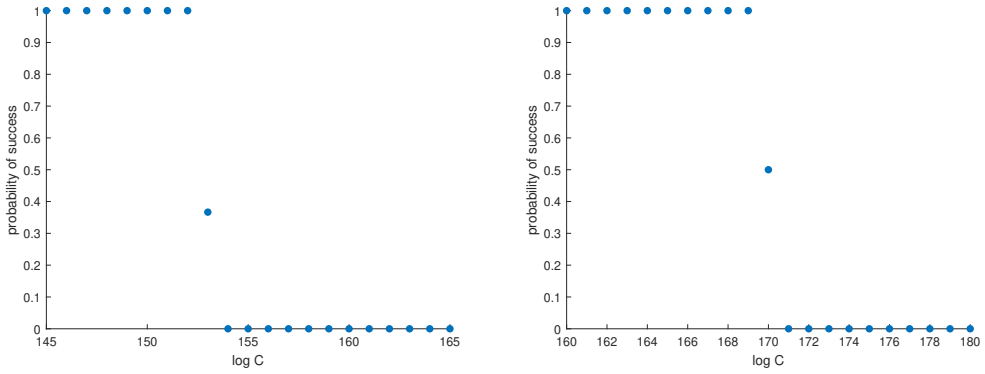

Figure 4.16: Probability of success as a function of $\log C$, for the two variables lattice attack. The left figure has $m = 5$ and the right one has $m = 6$.

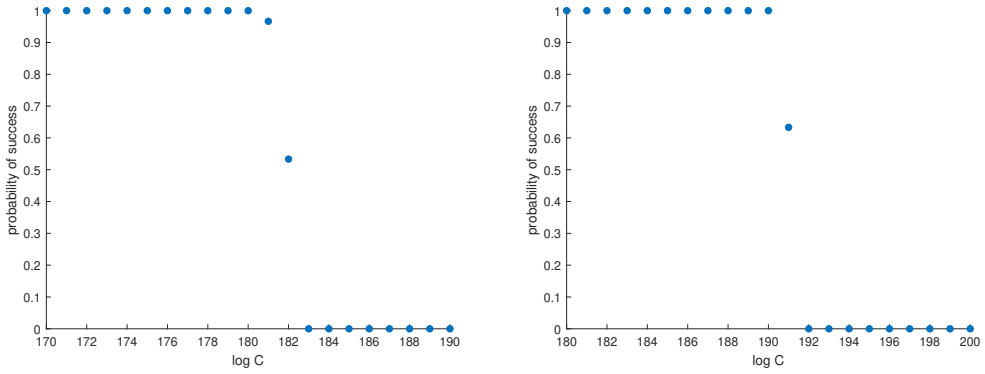

Figure 4.17: Probability of success as a function of $\log C$, for the two variables lattice attack. The left figure has $m = 7$ and the right one has $m = 8$.

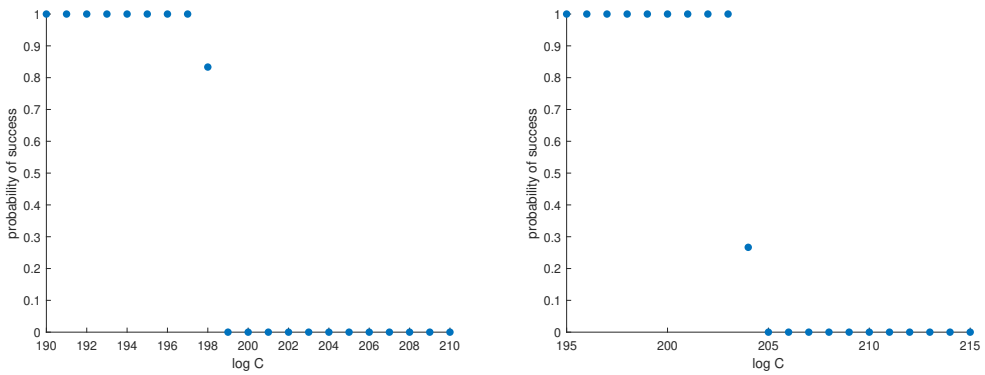

Figure 4.18: Probability of success as a function of $\log C$, for the two variables lattice attack. The left figure has $m = 9$ and the right one has $m = 10$.

4.3 Comparing methods

As it was described in 3.1.6 the number of equations used for Alice and Bob do not always have to be the same. Let us assume that Alice has n signatures and Bob has m signatures, where $n < m$. For each C, we will fix m and compute the probability of success of the two variables lattice attack as a function of n. Moreover, we will also compute the probability of success for the one variable lattice attack for the same C and n. Then, we will compare the results of both methods.

In section 4.2.1 we computed the probabilities of success for the two variables method for $C = 2^{255-l}$ for $l \in \{10, \ldots, 20\}$. Those will be the values used for C in these tests as well. According to the results in 4.2.1, for each $l \in \{11, \ldots, 20\}$ there is an integer m_l, such that, for all $x \geq m_l$ the probability of success for the two variables lattice attack with C and x is equal to 1. So, for each $l \in \{11, \ldots, 20\}$ the values of m used to test the two variables lattice attack will be m_l. For $l = 10$, the number of equations with the highest probability of success for the two variables lattice attack are 93 and 98, the value of m chosen was 93. The values of m used for each l are presented in Table 4.13. For each value of l, the value of n presented in Table 4.14 is the smallest integer such that C and n satisfy (2.18). The values of n used to test the attacks are the ones between $nmin$ and $nmax$ in Table 4.15.

l	10	11	12	13	14	15	16	17	18	19	20
m	93	79	62	53	45	42	38	35	32	30	29

Table 4.13: Experimental optimal values of m as a function of l, for the two variables lattice attack.

l	10	11	12	13	14	15	16	17	18	19	20
n	31	28	25	23	21	19	18	17	16	15	14

Table 4.14: Theoretical values of n needed as a function of l for the two variables lattice attack.

l	10	11	12	13	14	15	16	17	18	19	20
$\max n$	23	20	18	15	15	14	13	12	11	10	9
$\min n$	35	30	28	25	25	24	23	22	21	20	19

Table 4.15: Tested values of n as a function of l for the two variables lattice attack.

4.3.1 User with a random solution to the equation

Given C, m, n we want to to compute the probability of success of **Algorithm 9** when $|ak_{A,j_i} + bk_{B,l_i}|_q \leq C$ for $i \in \{1, \ldots, m\}$ where $j_i = ReducedListA(i)$ and $l_i = ReducedListB(i)$. In order to do that, we design an algorithm which starts by randomly generating two integers $1 \leq \alpha, \beta \leq q-1$, the vectors $t = (t_1, \ldots, t_m)$, $t' = (t'_1, \ldots, t'_n)$ and the vector $(b_1, \ldots, b_m)$, where $-C \leq b_i \leq C$ for $i = 1, \ldots, m$. Then, it appends t' with itself $\lfloor \frac{m}{n} \rfloor$ times, and it takes its first m entries and updates t' to be equal to this vector. Next, the algorithm computes $a = (a_1, \ldots, a_m)$ in such way that $x_i = b_i, y = \alpha, z = \beta$ is a solution to (2.23). Lastly, the algorithm uses LLL on $\mathcal{L}(M)$ where M is the matrix (2.24), and it returns 1 if either $v = (b_1, \ldots, b_m, \alpha C/q, \beta C/q, -C)$ or $-v$ is a vector in the reduced basis and 0 otherwise. Each of these tests were run thirty times. The average of those thirty tests was taken, and that average is the experimental probability of success.

We tested the method for the values of l, m and n presented in Table 4.13 and Table 4.15 . The test results are presented in Figures 4.19, 4.20,4.21,4.22,4.23,4.24.

For all values of l tested the probability of success for the one variable method converges to 1 as n grows. The same does not happen for the two variables method. For $l = 10$ and $l = 11$, the probability of success for this method does not converge to 1 as n grows, in fact, for the largest values of n tested, there is a decrease in the probability of success, this is most noticeable for $l = 10$. For $l \in \{12, \ldots, 20\}$ the probability of success for the two variables method seems to converge to 1 as n grows.

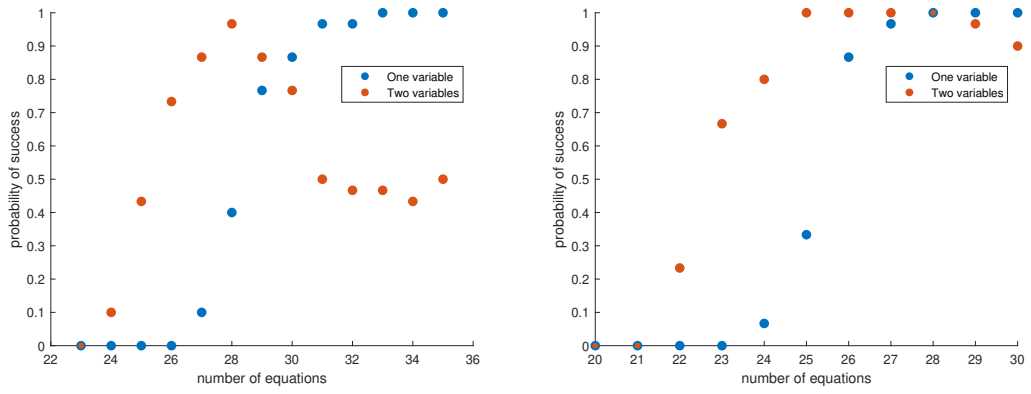

Figure 4.19: Comparison between the one and two variables lattice attack for a user with a random solution to the equation. The left figure has $l = 10$ and the right one has $l = 11$.

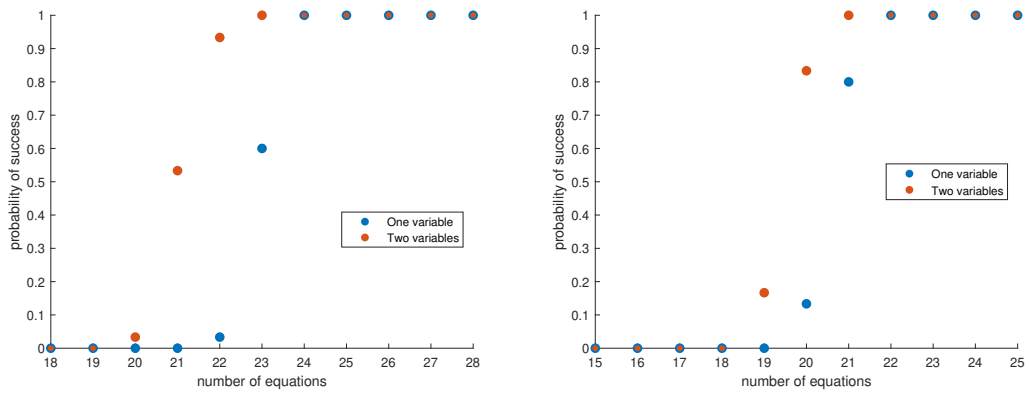

Figure 4.20: Comparison between the one and two variables lattice attack for a user with a random solution to the equation. The left figure has $l = 12$ and the right one has $l = 13$.

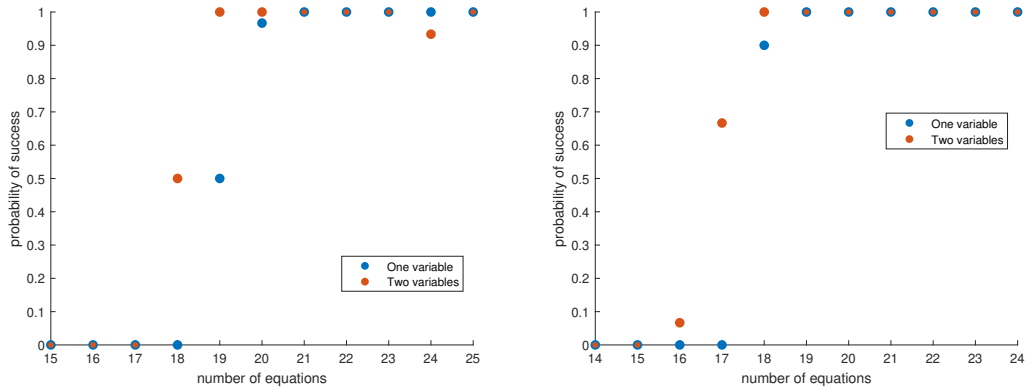

Figure 4.21: Comparison between the one and two variables lattice attack for a user with a random solution to the equation. The left figure has $l = 14$ and the right one has $l = 15$.

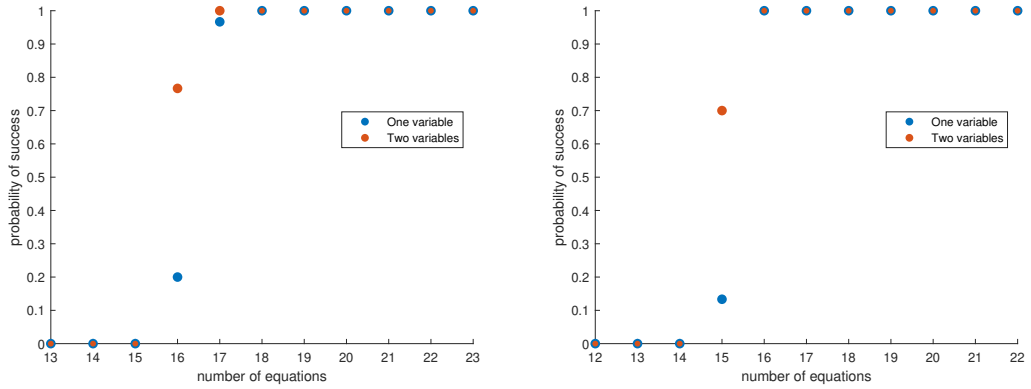

Figure 4.22: Comparison between the one and two variables lattice attack for a user with a random solution to the equation. The left figure has $l = 16$ and the right one has $l = 17$.

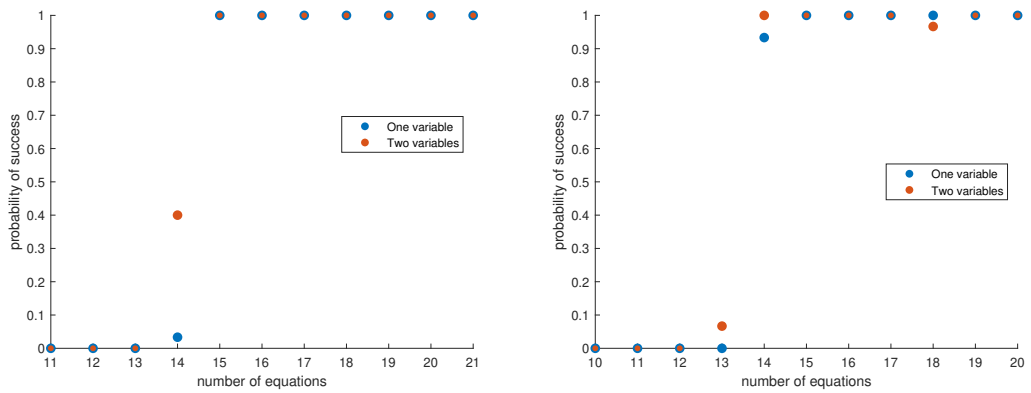

Figure 4.23: Comparison between the one and two variables lattice attack for a user with a random solution to the equation. The left figure has $l = 18$ and the right one has $l = 19$.

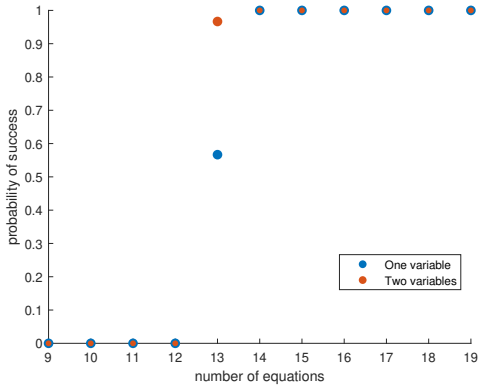

Figure 4.24: Comparison between the one and two variables lattice attack for a user with a random solution to the equation with $l = 20$.

For all values of l, there is at least one value n where the probability of success for the two variables lattice attack is higher than the probability of success for the one variable lattice attack. The smallest values of l are the ones where the most number of those values of n exist, they are also the values where the percentage points' increase from the one variable attack to the two variables attack is highest. Even though the two variables lattice attack has a better probability of success than the one variable lattice attack, the two variables lattice attack needs to have m inequalities $|b_i|_q \leq C$, while the one variable lattice attack only needs to have n inequalities $|b_i|_q \leq C$.

4.3.2 User whose solution to the equation is zeros

In order to compare the different attacks when we only use n inequalities in both methods let us assume that we have two vectors $t = (t_1, \ldots, t_n)$ and $a = (a_1, \ldots, a_n)$. Suppose that the system of equations

$$x_i - t_i y + a_i \equiv 0 \mod q, \quad \text{for } i = 1, \ldots, n, \tag{4.3}$$

is satisfied by $x_1 = b_1, \ldots, x_n = b_n, y = \alpha$, where $|b_i|_q \leq C$ for $i \in \{1, \ldots, n\}$. Let us define a vector b as $(b_1, \ldots, b_n)$. Similarly to what was done in the previous section, t is appended with itself $\lfloor \frac{m}{n} \rfloor$ and the

m first entries are taken and t is updated to be this vector. We do the same for the vectors a and b. For this $t = (t_1, \ldots, t_m)$ and $a = (a_1, \ldots, a_m)$ we have the system of equations

$$x_i - t_i y + a_i \equiv 0 \mod q, \quad \text{for } i = 1, \ldots, m \tag{4.4}$$

which has $x_1 = b_1, \ldots, x_m = b_m, y = \alpha$ as a solution.

To use the two variables method, let us start by randomly generating a vector $t' = (t_1^{'}, \ldots, t_m^{'})$ and an integer β. Then, we compute the vector $a^{'} = (a_1^{'}, \ldots, a_m^{'})$ in such a way that $x_1 = 0, \ldots, x_m = 0, z = \beta$ is a solution to

$$x_i - t_i^{'} z + a_i^{'} \equiv 0 \mod q, \quad \text{for } i = 1, \ldots, m \tag{4.5}$$

Adding (4.5) with (4.4) we have the system of equations

$$x_i - t_i y - t_i^{'} z + a_i^{'} + a_i \equiv 0 \mod q, \quad \text{for } i = 1, \ldots, m, \tag{4.6}$$

which is satisfied by $x_1 = b_1, \ldots, x_m = b_m, y = \alpha, z = \beta$.

This is now a lattice problem in two variables with a solution where $|b_i|_q \leq C$ for $i \in \{1, \ldots, m\}$. However, this problem is not equivalent to the one in 4.3.1, since this solution has repeated values of b_i and in that section the values of b_i were random in the interval $\{-C, \ldots, C\}$.

For each C, to test this attack, the algorithm starts by randomly generating two integers $1 \leq \alpha, \beta \leq q - 1$, the vectors $t = (t_1, \ldots, t_m)$, $t^{'} = (t_1^{'}, \ldots, t_n^{'})$ and the vector $b = (b_1, \ldots, b_n)$, where $-C \leq b_i \leq C$ for $i \in \{1, \ldots, n\}$. Then, it appends t' with itself $\lfloor \frac{m}{n} \rfloor$ times, and it takes its first m entries and updates t' to be equal to this vector. It also appends b with itself $\lfloor \frac{m}{n} \rfloor$ times, and it takes its first m entries and updates b to be equal to this vector. Next, the algorithm computes $a = (a_1, \ldots, a_m)$ in such way that $x_1 = b_1, \ldots, x_m = b_m, y = \alpha, z = \beta$ is a solution to (2.23). Lastly, the algorithm uses LLL on $\mathcal{L}(M)$ where M is the matrix (2.24), and it returns 1 if either $v = (b_1, \ldots, b_m, \alpha C/q, \beta C/q, -C)$ or $-v$ is a vector in the reduced basis and 0 otherwise. Each of these tests were run thirty times. The average of those thirty tests was taken, and that average is the experimental probability of success.

For each $\log C$ and m the values of n tested are the ones where the probability of success for two variables is higher than the probability of success for one variable in 4.3.1.

The values of $\log C$ and m used to test are the ones in Table 4.13 and Table 4.15. The results for this method are presented in Figures 4.25, 4.26,4.27,4.28,4.29,4.30.

The probability of success for this version of the two variables method is, as expected, not as high as the one in 4.3.1. However, surprisingly, for most values of l and n the method has a higher probability of success than the one variable method. The increase in performance is especially evident for $l \in \{10, 11, 12\}$. For those values of l, the probability of success for the two variables lattice attack is never lower than the probability of success for the one variable lattice attack. Furthermore, the only values of n where the probability of success for the two variables lattice attack is not higher than the probability of success for the one variable lattice attack is when the probabilities are both equal to zero. Despite the fact that for many values of C and n the probability of success for the two variables attack is better

than the probability of success for the one variable attack, whenever the probability of success is 0 for the one variable method the probability of success for the two variables method is low, it is never more than 10%. Therefore, unlike the method in 4.3.1, in this method if the private key can not be computed with the one variable lattice attack it can only be computed with a low probability with the two variables lattice attack. Notwithstanding, if the one variable method was not able to compute the private key it is possible that this attack can and, consequently, we should try to use this method for that user.

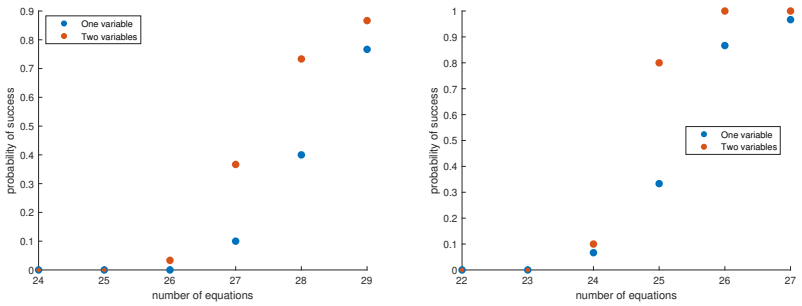

Figure 4.25: Comparison between the one and two variables lattice attack for a user with a zero solution to the equation. The left figure has $l = 10$ and the right one has $l = 11$.

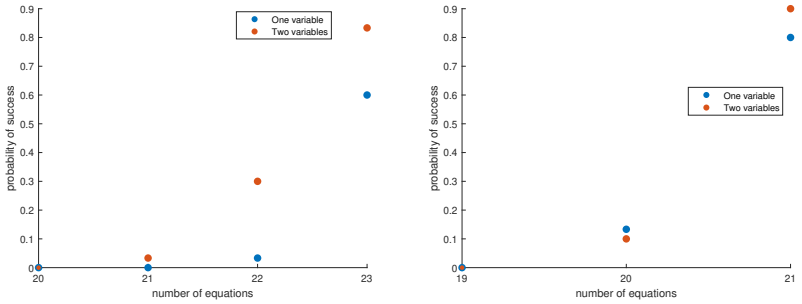

Figure 4.26: Comparison between the one and two variables lattice attack for a user with a zero solution to the equation. The left figure has $l = 12$ and the right one has $l = 13$.

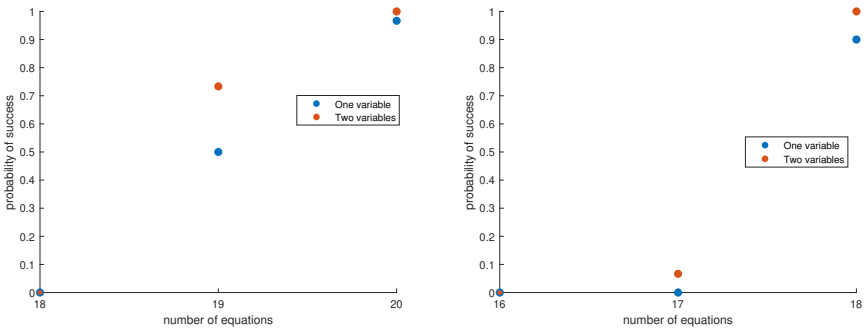

Figure 4.27: Comparison between the one and two variables lattice attack for a user with a zero solution to the equation. The left figure has $l = 14$ and the right one has $l = 15$.

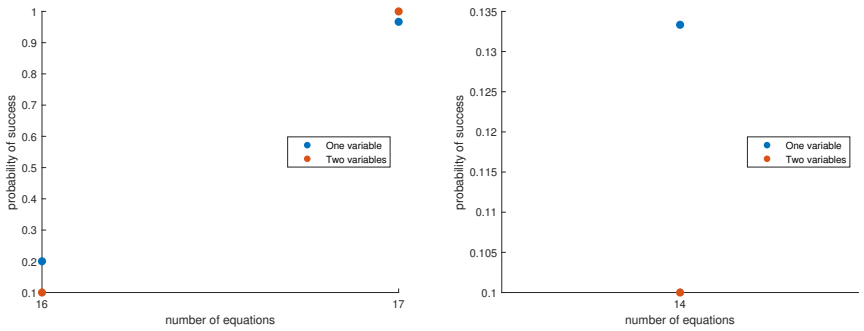

Figure 4.28: Comparison between the one and two variables lattice attack for a user with a zero solution to the equation. The left figure has $l = 16$ and the right one has $l = 17$.

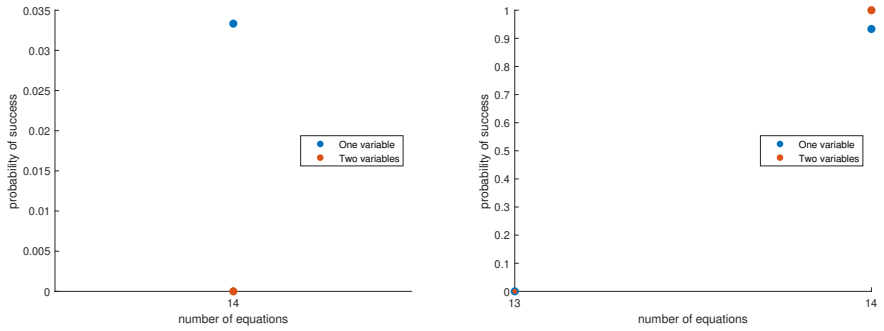

Figure 4.29: Comparison between the one and two variables lattice attack for a user with a zero solution to the equation. The left figure has $l = 18$ and the right one has $l = 19$.

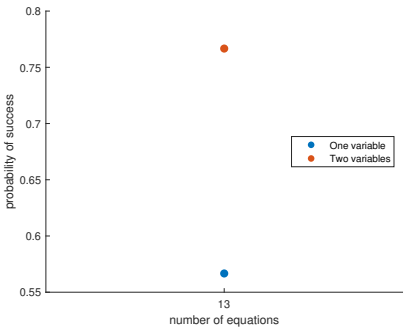

Figure 4.30: Comparison between the one and two variables lattice attack for a user with a zero solution to the equation with $l = 20$.

Chapter 5

Conclusions

The results show that for this solution to the HNP and using the LLL algorithm implemented in Mathematica, converting a one variable lattice attack to a two variables lattice attack can increase the probability of success. We also confirmed that the ECDSA is secure against all these attacks if the nonces are randomly generated. However, if the nonces are generated with some weak PNRG, it may be possible to design an attack which can recover the private key. Moreover, if one is able to design a side-channel attack which recovers a number of most, or least, significant bits, the private key can also be recovered with a high probability.

5.1 Future Work

For our tests we needed to have at least 7 bits leaked to have a good probability of success. It is possible that if we had tested for lattices with higher dimensions we would be able to recover the private key with only 6 leaks, but for higher dimensions than the ones tested the attacks took too much time and we were not able to compute them. So, one future test that could be done would be to run these tests with a faster programming languages. Another change that could be done is using the BKZ 2.0 [17] as the lattice reduction algorithm instead of the LLL, since the BKZ 2.0 is a better lattice reduction algorithm for high dimensions. It would also be interesting to compare the two variables lattice attack and the one variable lattice attack using a faster programming language and the BKZ 2.0 for larger values of C and verify if the two variables lattice attack still has a higher probability of success than the one variable lattice attack.

Another attack which could be implemented is one attack which uses a group attack and a lattice attack, it could start by using a group attack and if the group attack does not recover the private key, using that information to design the lattice attack. Other attacks which could be designed is to try to use a three or more variables lattice attack and ascertaining if that attack increases the probability of success, such as the two variables lattice attack did.

Bibliography

[1] ANSI X9.62:2005: Public Key Cryptography for the Financial Services Industry, The Elliptic Curve Digital Signature Algorithm (ECDSA), 2005.

[2] S. Nakamoto. Bitcoin: A peer-to-peer electronic cash system. *Cryptography Mailing list at https://metzdowd.com*, 03 2009.

[3] D. Boneh and R. Venkatesan. Hardness of Computing the Most Significant Bits of Secret Keys in Diffie-Hellman and Related Schemes. *In: Koblitz N. (eds) Advances in Cryptology — CRYPTO '96. CRYPTO 1996. Lecture Notes in Computer Science*, vol 1109. Springer, Berlin, Heidelberg, 1996.

[4] A. K. Lenstra, H. W. L. Jr., and L. Lovász. "Factoring Polynomials with Rational Coefficients". *Mathematische Annalen*, Vol. 261(No. 4):515–534, 1982. doi: 10.1007/BF01457454.

[5] C. P. Schnorr and M. Euchner. Lattice basis reduction: Improved practical algorithms and solving subset sum problems. *Math. Program.*, 66(2):181–199, Sept. 1994. doi: 10.1007/BF01581144.

[6] J. Breitner and N. Heninger. Biased nonce sense: Lattice attacks against weak ecdsa signatures in cryptocurrencies. Cryptology ePrint Archive, Report 2019/023, 2019. `https://eprint.iacr.org/2019/023`.

[7] L. Babai. On Lovász' lattice reduction and the nearest lattice point problem. *Combinatorica*, 6(No. 1), 1986. doi: 10.1007/BF02579403.

[8] W. Diffie and M. Hellman. New directions in cryptography. *IEEE Trans. Inform. Theory*, IT-22(No. 6):644–654, Nov 1976.

[9] N. A. Howgrave-Graham and N. Smart. Lattice Attacks on Digital Signature Schemes. *Designs, Codes and Cryptography*, 23:283–290, 2001. doi: 10.1023/A:1011214926272.

[10] P. Q. Nguyen and I. E. Shparlinski. The Insecurity of the Elliptic Curve Digital Signature Algorithm with Partially Known Nonces. *Designs, Codes and Cryptography*, 30:201–217, 2003. doi: 10.1023/A:1025436905711.

[11] N. Benger, J. van de Pol, N. P. Smart, and Y. Yarom. "Ooh Aah... Just a Little Bit" : A Small Amount of Side Channel Can Go a Long Way. In *Cryptographic Hardware and Embedded Systems – CHES 2014*, pages 75–92. Springer Berlin Heidelberg, 2014. ISBN 978-3-662-44709-3.

[12] E. D. Mulder, M. Hutter, M. E. Marson, and P. Pearson. Using Bleichenbacher's Solution to the HiddenNumber Problem to Attack Nonce Leaks in384-Bit ECDSA. *J Cryptogr Eng*, 4(33):201–217. doi: 10.1007/s13389-014-0072-z.

[13] D. Bleichenbacher. On The Generation of One-Time Keys in DL Signature Schemes. In *Presentation at IEEE P1363 Working Group meeting*, November 2000.

[14] D. Bleichenbacher. On the Generation of DSA One-Time Keys. In *Presentation at Cryptography Research, Inc., San Francisco, CA*, 2007.

[15] N. Koblitz. *Algebraic Aspects of Cryptography, Algorithms and Computation in Mathematics 3*, chapter 6. Springer, 2^{nd} edition, 1999. ISBN: 3-540-63446-0.

[16] P. Q. Nguyen and D. Stehlé. LLL on the Average. *In: Hess F., Pauli S., Pohst M. (eds) Algorithmic Number Theory. ANTS 2006. Lecture Notes in Computer Science*, vol 4076. Springer, Berlin, Heidelberg, 2006.

[17] Y. Chen and P. Q. . Nguyen. BKZ 2.0: Better Lattice Security Estimates. In *Proceedings of the 17th International Conference on The Theory and Application of Cryptology and Information Security*, ASIACRYPT'11, pages 1–20, Berlin, Heidelberg, 2011. Springer-Verlag. ISBN 978-3-642-25384-3. doi: 10.1007/978-3-642-25385-0_1.