INSTITUTO SUPERIOR TÉCNICO
Universidade Técnica de Lisboa

# A Survey of Cryptanalytic Attacks on RSA

**Filipe da Costa Boucinha**

A Dissertation presented
in partial fulfilment of the
Requirements for the Degree of Master in

**Mathematics and**
**Fundamental Applications**

**October 2011**

## Abstract

RSA was the first public key cryptosystem to be published and it is one of the most widely used. One of the reasons for this is its simple implementation, another one is the deep analysis it has been the subject of.

We begin this work with a brief introduction of the mathematical notions required for implementing RSA, along with a brief description of the evolution of cryptography.

In chapter 2 we describe some of the most important methods for primality checking and for integer factorization.

We start chapter 3 by listing some faulty uses of RSA, followed by some attacks on RSA sessions using low public or private exponent and recommendations on how to avoid them. This work can therefore be seen as a guide for choosing appropriate generating primes and keys for RSA.

Though there were no new findings obtained with the experimental results we got, we present them, along with the implementations of the respective attacks, hoping that they will captivate the readers to understand its implementation and motivate further improvements.

**Keywords:** RSA, cryptanalysis, primality, factorization

O RSA foi o primeiro criptosistema de chave pública a ser publicado e é um dos mais usados hoje em dia. Uma das razões para isso é a sua simples implementação, outra é a análise profunda a que já foi sujeito.

Começamos este trabalho com uma introdução das noções matemáticas necessárias para a implementação do RSA, juntamente com uma descrição de alguns dos mais importantes métodos de factorização e verificação de primalidade.

Finalmente listamos algumas más utilizações do RSA, seguidas de alguns ataques a sessões de RSA que usem pequenos expoentes privados ou públicos e algumas maneiras de os evitar. Este trabalho é portanto um guia para a escolha apropriada dos primos geradores e da chave do RSA.

Embora não tivessem sido obtidos novos resultados com os resultados experimentais obtidos, apresentamo-los juntamente com as implementações dos ataques respectivos, na esperança de conseguir cativar o leitor a perceber os ataques e a tentar melhorar os mesmos.

**Palavras Chave:** RSA, criptanálise, primalidade, factorização

**Acknowledgements**

I am deeply thankful to my supervisor Professor Vilius Stakenas. Not only for making me so interested in the RSA cryptosystem in particular and investigation in general, but also for his extreme kindness, sensitivity and helpfulness.

I would like to sincerely thank to my supervisor Professor Carlos Caleiro for his helpful remarks, suggestions and reflections about this work.

My family has been a great support since the beginning of my life, so I would like to thank them most of all, since it is because of them that I stand here today.

Finally, I would like to thank all my friends who helped me to endure this task both in Vilnius and in Lisbon. I am extremely grateful to them and I hope I have the opportunity to show it after a reasonable amount of time.

# Contents

# List of Tables

# 1 Introduction

## 1.1 Cryptography

Cryptography is a science practised for over four thousand years. It regards, in a rather simple way, the finding and improvement of methods for sending messages between two chosen parts without risk that an unauthorized third party can understand the sent messages.

While in the beginning this process was achieved by sending the messages personally, i.e., from mouth to hear, with the increasing of the communcations' distance this has become impossible. So cryptographers started developing methods of sending messages from a sender, **Alice**, to a receiver, **Bob**, without a malicious eavesdropper, **Marvin**, being able to understand the messages unless he knows exactly how they were encrypted. Such a method is called a cryptosystem. Its mathematical description is given below:

**Definition 1.** **Cryptosystem** is a tuple $< M, C, K, e, d >$ where:

$M$ is the set of plaintext messages

$C$ is the set of cypher text messages

$K$ is the set of keys

$e$ is the encryption function, $e : M \times K \to C$ such that $e(m|k) = c$

$d$ is the decryption function, $d : C \times K \to M$ such that $d(e(m|k)|k) = m$

To send a plaintext $m$, Alice encrypts it with an *a-priori* defined key $k$, obtaining the cypher text $c = e(m|k)$, which she sends to the Bob, who decrypts it using the same key $k$ obtaining $m = d(e(m|k)|k)$.

The key $k$ should be known only by Alice and Bob, otherwise Marvin could decrypt $c$ just by knowing the decryption function. For this reason, this kind of cryptosystems are called **Private Key Cryptosystems** or **Symmetric Key Cryptosystems**, as the same key is used for encryption and decryption.

Throughout the years, numerous private key cryptosystems were created, with ever increasing complexity. Eventually, they were considered insufficient. What made them insufficient was the requirement that the secret key had to be known both to Alice and Bob. This means that before exchanging the encrypted messages, Alice and Bob needed a secure channel (that is, a channel to send messages without the risk of Marvin intercepting any message) to exchange the secret key. But suppose now that Alice and Bob never met, will never have the chance to meet, and have at their disposition only an insecure channel to exchange messages. Can they exchange encrypted messages between them, with Marvin knowing all these cypher texts but without being able to decrypt them? Until we got to 1976, this question seemed to have no answer. It was then that Diffie and Hellman, predicting a coming revolution in cryptography[13], proposed a new concept of cryptosystem which would lead to the creation of RSA.

## 1.2 Public Key Cryptography

Public key cryptography consists in cryptosystems where there is no need for a secure channel to exchange any prior information, like the secret key used by private key cryptosystems. A common analogy to explain the concept of public key cryptography is that of a simple mailbox. Everyone can put a letter in Alice's mailbox but only Alice, who has the key for her mailbox, can open it and read her letters. The same is true for Bob and his mailbox. In a technical way, this can be described as follows: Alice creates a pair of public and private keys. Then she reveals her public key, allowing anyone to encrypt messages with this key and send them to her. When receiving them, she decrypts them using her private key. To clear any doubts, should Bob wish to receive encrypted messages he should also create a pair of public/private keys and follow Alice's procedure.

This way, there is no need for a safe channel to agree on a key, as there are no common keys which need to be changed between the several users. The conditions such a cryptosystem should satisfy, which were initially proposed by Diffie and Hellman in 1976, are presented below:

**Definition 2.** **[Diffie-Hellman concept of public key cryptosystem]** A cryptosystem where the key consists of a pair of public/private keys, where the encryption function uses the public key and the decryption function uses the private key, such that:

it should be easy to create pairs of public/private keys

it should be easy do encrypt messages knowing the public key

it should be easy do decrypt messages knowing the private key

it should be hard to compute the private key from the public key

For a public key cryptosystem to be safe, it should be hard to invert the encryption function without knowing the private key, and it should be hard to deduce the private key from the public key alone. This kind of functions, which are easy to compute but difficult to invert without knowing some extra parameters are called **trapdoor one-way functions**.

Regarding cryptosystems used nowadays this inversion is not impossible: rather it is extremely time consuming, making it useless for Marvin to try it when the information being transmitted is only relevant for a short period of time.

## 1.3 The RSA cryptosystem

The RSA cryptosystem is the first ever published public key cryptosystem, developed by Rivest, Shamir and Adleman, first presented in their 1978 article[38], and based on the Diffie-Hellman proposal. Its implementation depends on an *a priori* choice of two large prime numbers $p$ and $q$, that are multiplied to obtain the RSA modulus $N = pq$ and a subsequent choice of a public and a private integer parameters, $e$ and $d$, satisfying $ed = 1 + k(p-1)(q-1)$ for some integer $k$. These two computations are actually a trapdoor one way function of RSA: while it is easy to compute $N = pq$, we will show that it is hard to factor it. As for the equation which allows us to define the exponents, it will be shown that it is hard to deduce $d$ from $e$ and $N$ without knowing $p$ and $q$.

Since its first description by the three computer scientists in 1976, RSA has been thoroughly analysed and many attacks against it have been found. These attacks, which aim to recover an encrypted message or to deduce the private key, have lead to restrictions over the choice of the four parameters mentioned above and to limitations to the use of RSA. However, the set of these attacks has not led to contradictory restrictions on the parameters, so the system has not been proven to be unsafe. It is possible, so far, to choose the right parameters so that all the known attacks become infeasible. In fact, the deep knowledge we have about its weaknesses makes it more reliable, since less surprises are expected than they were 30 years ago.

Since it has not been proven unsafe and it is not generally believed that it will be so, RSA is presently the most used public-key cryptosystem in the world. For this reason, its safe implementation is a matter of extreme importance. As this depends mostly on the choice of the four parameters mentioned above, every survey and new result about this topic will lead to a greater safety on the transmission of information online, preventing cyber crime and providing greater confidentiality to internet users.

## 1.4 Objective

The reliability of any cryptosystem relies mainly on how much it has been analysed. RSA has been the subject of numerous analysis and that is one of its strengths: no devastating attack has been found yet.

This work results from an analysis on several known mathematical attacks on the RSA cryptosystem and their respective complexity. The objective is that someone who wishes to implement RSA has a brief and clear summary of the main precautions to have especially when it comes to choosing the generating primes and the encryption/decryption exponents. Besides the theoretical complexity of the attacks we present the mathematical basis for the results presented, experimental results and the implementation of some of the attacks, with the aim of motivating people to improve the known attacks and to discover new ones.

## 1.5 Mathematical Basis

In this section we start by presenting definitions for some of the symbols used throughout the work. Then we present the definitions necessary to understand what is the efficiency of an algorithm. Next we introduce the reader to modular arithmetic, the basis of RSA, along with some useful theorems and algorithms. Finally we present some advanced results used in the attacks against RSA with low exponents.

### 1.5.1 Notation

We will use the following symbols whenever $a, b, N$ are positive integers, $p$ a prime number, $m = p_1^{\alpha_1} p_2^{\alpha_2} ... p_n^{\alpha_n}$ is an odd integer with prime factors $p_1, ..., p_n$ and $(\frac{a}{b})$ is referred to as the Legendre symbol.

Table 1: Notation

| Symbol | Definition |
|---|---|
| (a,b) | greatest common divisor between a and b |
| lcm(a,b) | least common multiple of a and b |
| $\phi(N)$ | number of positive integers smaller than and co-prime to N |
| $\lambda(N)$ | smallest positive integer $m$ such that $a^m \cong 1 \pmod{N}\ \forall a \in \mathbb{Z}_N : (a, N) = 1$ |
| $\left[\frac{a}{p}\right]$ | $\begin{cases} 0 & \text{, if } p \mid a \\ 1 & \text{, if a is a quadratic residue modulo p} \\ -1 & \text{, if a is a quadratic nonresidue modulo p} \end{cases}$ |
| $(\frac{a}{m})$ | $= [\frac{a}{p_1}]^{\alpha_1} [\frac{a}{p_2}]^{\alpha_2} ... [\frac{a}{p_n}]^{\alpha_n}$ |

### 1.5.2 Time Complexity

In cryptography, rather than the value of an integer, it is common to work with its size in bits. Most of the algorithms presented in this work will depend on the size of integers, so it is convenient to present firstly the following definition:

**Definition 3. The size of** $N$ is the number of bits it takes to represent $N$ in base 2, namely $\log_2(N)$

Throughout the rest of this work, when referring to the size of a number, $\log(N)$ should be considered $\log_2(N)$. To describe an algorithm's efficiency we usually relate its running time to some class of functions, for which we use the following notation:

**Definition 4. Big-O notation:** Given two functions $f$ and $g$, we say that $f(n) \in O(g(n))$ iff:

$$\exists \text{ constants } c, n_0 > 0 : \forall n \geq n_0 \text{ we have } 0 \leq f(n) \leq cg(n) \tag{1}$$

This means that, for sufficiently large values of $n$, the function $f$ does not exceed $g$.

The efficiency of the algorithms will be described in terms of the number of basic operations it executes in function of the input given.

**Definition 5. The running time of an algorithm** $A$**,** $T_A(n)$ is the number of elementary instructions it executes when the input data is $n$. Occasionally, the running time can depend on more than one parameter. When the situation will be clear, we will simply say that **the running time of an algorithm** $A$ **is** $O(f)$, meaning that $T_A(n) \in O(f(n))$.

It is obvious that each function does not belong to a single class of complexity.For example, if $T(n) \in O(f(n))$, then obviously $T(n) \in O(2f(n))$. For this reason, when saying that $T(n) \in O(f(n))$ we mean that $f$ is the smallest known function that satisfies this, unless otherwise stated.

Here are some simple cases for the complexity of an algorithm $A$, $T_A(n)$:

1. $T_A(n)$ is $O(\log(n)^\alpha)$ means that the running time of A is polynomial in the size of its input.
2. $T_A(n)$ is $O(\beta^{\log(n)})$ means that the running time of A is exponential in the size of its input.

The first case refers to algorithms which we say **run in time polynomial in** $\log(N)$. Such an algorithm is called **efficient** as its complexity function grows in a rather controlled way. In opposition, algorithms belonging to the second case are called **inefficient**: they quickly become infeasible by any computer or even network of computers.

When referring to problems, we will call them **hard** if there is no known algorithm running in polynomial time that solves it. In opposition, a problem which can be solved for any input with an algorithm running in polynomial time is said to be an **easy** problem.

### 1.5.3 Modular Arithmetic

Modular Arithmetic is a fundamental basis for cryptography in general and for RSA in particular. For this reason we present a set of definitions and results which will enable the reader to fully understand the operations behind RSA. We will also present some of the most used algorithms within the RSA operations.

We start with the congruence relation:

**Definition 6.** Let $N$ be a positive integer. We say that **two integers** $a$ **and** $b$ **are congruent modulo** $N$ if there exists an integer $k$ such that $a - b = kN$. We represent this relation with the following notation:

$$a \cong b \pmod{N} \tag{2}$$

It should be noted that this is an equivalence relation. After choosing the modulus $N$, we get a partition of the integers into $N$ different classes since every integer $a$ must be congruent modulo $N$ to an integer in the set $\{0, 1, ..., N-1\}$. When the modulus $N$ is implicit, we define $[a] = \{b \in \mathbb{Z} : a \cong b \pmod{N}\}$, called the equivalence class of $a$. We introduce two operations with these equivalence classes, simple extensions of the usual addition and multiplication:

**Definition 7. (Modular Operations)**: Given two classes of equivalence $[x]$ and $[y]$ defined modulo $N$, we define the operations:

addition: $[x] + [y] = [x + y]$

multiplication: $[x][y] = [xy]$

Throughout the text we will stop referring to the equivalence class $[x]$ as $[x]$ and instead write simply $x$. The modulus regarding which the equivalence class is defined will also not be written, rather it will be explicit from its context.

**Definition 8.** Given an integer $N > 0$, the **modular ring** $\mathbb{Z}_N$ is the set:

$$\mathbb{Z}_N = \{[0], [1], ..., [N-1]\} \tag{3}$$

(which will also be defined as $\mathbb{Z}_N = \{0, 1, ..., N-1\}$)

along with the modular operations defined above. The division operation is an extension of the multiplication operation: to divide the equivalence class $x$ by the equivalence class $y$, we multiply $x$ by the inverse of $y \pmod N$, according to the next definition.

In the ring $\mathbb{Z}_N$ we can easily define inverses, which will be necessary to create RSA keys:

**Definition 9.** Let $a$ be an element of the modular ring $\mathbb{Z}_N$. **The inverse of** $a$ **modulo** $N$ is the integer $x$ satisfying:

$$ax \cong 1 \pmod N \tag{4}$$

which we will refer to as $a^{-1} \pmod N$.

It is important to note that $a^{-1} \pmod N$ exists if and only if $(a, N) = 1$. When this is the case, the Extended Euclidean Algorithm (explained in the next section) will provide us with the values $x, y$ such that $ax + yN = 1$. From the last equation we get:

$$ax \cong 1 \pmod N \Leftrightarrow ax - 1 = kN \Leftrightarrow ax - kN = 1 \tag{5}$$

We know $a$ and $N$ so $x$, the inverse of a, is given by the Extended Euclidean Algorithm (along with the value of $k$ , which can be discarded). If $(a, N) \neq 1$ then there are no integer solutions $x, k$ for this equation, and therefore there is no inverse of $a$ modulo $N$. However, if given a modulus $N$ and an integer $a$, we find out that $a$ does not have an inverse modulo $N$, then we can compute a factor of $N$ by simply computing $(a, N)$.

Another very important procedure for RSA regarding congruences is **Modular Exponentiation**. Given integers $b, e, N$, suppose we wish to calculate the integer $c$ satisfying:

$$c \cong b^e \pmod{N} \tag{6}$$

A straightforward method is to simply calculate $b^e$ and then its remainder when divided by $N$. This algorithm's complexity is $O(e)$ so it is infeasible for large values of $e$. An alternative method, more efficient, is described below:

**Algorithm 1. (Exponentiation by repeated squaring and multiplication).** Given integers $b, e, N$, to compute the modular exponentiation $c \cong b^e \pmod{N}$ we proceed as follows:

```
1. write $e$ in its binary representation: $(e_{m-1}, e_{m-2}, ..., e_1, e_0)_2$ such that $e = \sum_{i=0}^{i=m-1} e_i 2^i$.
2. set $c = 1$.
3. from $i = m - 1$ to $i = 0$:
    1. $c = c^2 \pmod{N}$.
    2. If $e_i = 1$, then $c = cb \pmod{N}$.
4. return $c$.
```

The complexity of this algorithm is $O(m) = O(\log(e))$. The operations in step 3 can be efficiently done implementing a technique called Montgomery Reduction[34].

We now introduce one definition used in the surprisingly efficient deterministic primality checking algorithm presented in Chapter 2.

**Definition 10.** Given an integer $a$ and an integer $N$ such that $(a, N) = 1$, **the multiplicative order of $a$ modulo $N$**, denoted $o_N(a)$, is the smallest positive integer $k$ such that

$$a^k \cong 1 \pmod{N} \tag{7}$$

And a definition used in the application of the Solovay-Strassen primality test.

**Definition 11.** Given an integer $q$ and an integer $n$, we say that $q$ **is a quadratic residue modulo $n$** if there exists an integer $x$ such that:

$$x^2 \cong q \pmod{n} \tag{8}$$

If we want to solve a system of linear modular equations there is one efficient theorem which provides the correct answer quickly:

**Theorem 1.** ***(Chinese Remainder Theorem):*** *Let $a_1, ..., .a_n$ be $n$ integers and $p_1, ..., p_n$ be relatively prime positive integers. Set $P = \prod_{i=1}^{n} p_i$ and, for $i = 1, ..., n$ define $y_i$ such that:*

$$y_i \frac{P}{p_i} \cong 1 \pmod{p_i} \tag{9}$$

*Then, one solution of the system*

$$\begin{cases} x \cong a_1 \pmod{p_1} \\ x \cong a_2 \pmod{p_2} \\ \vdots \\ x \cong a_n \pmod{p_n} \end{cases}$$

*is given by:*

$$x_0 = \sum_{i=1}^{n} a_i y_i \frac{P}{p_i} \tag{10}$$

*Any other integer solution, $x$, of the system of congruences satisfies:*

$$x \cong x_0 \pmod{P} \tag{11}$$

### 1.5.4 Useful Algorithms and Results

We now include some useful definitions and results used in the proofs of the attacks presented in chapter 3.

**Definition 12.** Let

$$p(x) = a_n x^n + a_{n-1} x^{n-1} + ... + a_1 x + a_0 \tag{12}$$

be a polynomial of degree $n$ with roots $\alpha_i$ for $i = 1, ..., n$ and

$$q(x) = b_m x^m + b_{m-1} x^{m-1} + ... + b_1 x + b_0 \tag{13}$$

be a polynomial of degree $m$ with roots $\beta_i$ for $i = 1, ..., n$.
**The Resultant of $p$ and $q$, denoted** *Resultant*$(p, q)$, is defined by

$$Resultant(p, q) = a_n^m b_m^n \prod_{i=1}^{n} \prod_{j=1}^{m} (\alpha_i - \beta_i) \tag{14}$$

There is a lot of efficient ways to compute the Resultant of two polynomials. One property of the resultant which will be useful for an attack in Chapter 3 is that its application allows for the elimination of one variable from a system of two polynomial equations[45].

To calculate $(a, b)$ we can use the Extended Euclidean Algorithm, which runs in time linear in the size of $a$ and $b$ and outputs also integers $x, y$ such that $xa + yb = (a, b)$. We present it now:

**Algorithm 2. (Extended Euclidean Algorithm):** Given two integers $a > b$, in step $k = 0$ we set $r_{-2} = a$, $r_{-1} = b$, $x_0 = y_0 = 0$, $x_1 = y_1 = 1$ and compute $r_0 : r_{-2} = q_1 * r_{-1} + r_0$, where $q_1 = \lfloor \frac{r_{-2}}{r_{-1}} \rfloor$. In step $k = i$, we compute $r_i : r_{i-2} = q_i * r_{i-1} + r_i$, where $q_i = \lfloor \frac{r_{i-2}}{r_{i-1}} \rfloor$. Then we compute $x_i = x_{i-2} - q_{i-1}x_{i-1}$ and $y_i = y_{i-2} - q_{i-1}y_{i-1}$ if $q_{i-1}$ is defined.

The algorithm stops at step $k = l$ if $r_l = 0$ and outputs $(a, b) = r_{l-1}$ and also $x_{l-2}$ and $y_{l-2}$ which satisfy $x_{l-2}a + y_{l-2}b = (a, b)$.

Some properties of **Euler's Totient function**, $\phi(N)$, and **Carmichael's Lambda Function**, $\lambda(N)$, are particularly relevant for RSA analysis:

**Theorem 2.** *If $p, q$ are two different prime integers and $N = pq$ we have:*

1. $\phi(p) = p - 1$
2. $\phi(N) = \phi(pq) = \phi(p)\phi(q) = (p - 1)(q - 1)$
3. $\lambda(N) = lcm(p - 1, q - 1) = \frac{(p-1)(q-1)}{(p-1,q-1)} = \frac{\phi(N)}{(p-1,q-1)}$

We are now in position to present Euler's Theorem.

**Theorem 3.** ***(Euler's Theorem)**: If $a$ and $N$ are co-prime positive integers then:*

$$a^{\phi(N)} \cong 1 \pmod{N} \tag{15}$$

*Proof.* Let $m_1, m_2, ..., m_{\phi(N)}$ be the positive integers co-prime to and less than $N$. These numbers are all distinct modulo $N$ and $a$ is co-prime to $N$, so each of the integers $am_1, am_2, ..., am_{\phi(N)}$ is congruent to one of $m_1, m_2, ..., m_{\phi(N)}$. Because congruences preserve multiplicity, we have:

$$am_1am_2...am_{\phi(N)} \cong \quad m_1m_2...m_{\phi(N)} \pmod{N} \Leftrightarrow \tag{16}$$

$$m_1m_2...m_{\phi(N)}a^{\phi(N)} \cong \quad m_1m_2...m_{\phi(N)} \pmod{N} \tag{17}$$

Diving both sides by $m_1m_2...m_{\phi(N)}$ we get the equality. $\square$

### 1.5.5 Continued Fractions

The continued fraction expansion of a real number $x$ is its representation as the tuple $[a_0; a_1, ..., a_n, \cdots]$, such that the following equality is satisfied:

$$x = a_0 + \cfrac{1}{a_1 + \cfrac{1}{a_2 + \cfrac{1}{\ddots + \cfrac{1}{a_n + \cfrac{1}{\ddots}}}}}$$

with the coefficient $a_0$ being any integer and the coefficients $a_1, \cdots, a_n, \cdots$ positive integers. If $x = \frac{p}{q}$ is a rational number then the number of coefficients in its continued fraction expansion is finite, namely $n = \log(q)$. This coefficients are the quotients computed by the Euclidean Algorithm applied to $p$ and $q$.

The value $c_i = [a_0; a_1 \cdots a_i$, for $i = 0, ..., n$, called the **i-th convergent of** $x$, can be seen as an approximation of the value of $x = [a_0; a_1, \cdots, a_n]$.

We now present one result about continued fractions which is the basis for Wiener's attack, presented in Chapter 3.

**Theorem 4.** *Let $\alpha \in \mathbb{Q}$ and $c, d \in \mathbb{Z}$ such that:*

$$|\alpha - \frac{c}{d}| < \frac{1}{2d^2}.$$

*Then $\frac{c}{d}$, as an irreducible fraction, is one of the convergents of the continued fraction expansion of $\alpha$.*

### 1.5.6 Results from Coppersmith

In an article written in 1997[9], Coppersmith shows how to solve a modular polynomial equation of degree $k$ in a single variable $x$, as long as the solutions are sufficiently small. The work of Coppersmith is based in the theory of Lattices, namely in one algorithm developed by Lenstra, Lenstra and Lovasz, the LLL-algorithm[26].

**Theorem 5.** *Let $p(x)$ be a monic integer polynomial of degree $k$ and $N$ a positive integer of unknown factorization. We can find all integer solutions $x_0$ of $p(x_0) \cong 0 \pmod{N}$, such that $|x_0| < N^{\frac{1}{k}}$, in time polynomial in $\log(N)$ and $k$.*

There is a generalized version of this result:

**Theorem 6.** *Let $N$ be an integer of unknown factorization which has a divisor $b \geq N^{\beta}$, for $0 < \beta \leq 1$, and $f(x)$ be a univariate monic polynomial of degree $k$. Then we can find all solutions $x_0$ of $f(x) \cong 0 \pmod{b}$, satisfying $|x_0| \leq cN^{\frac{\beta^2}{k}}$, in time polynomial in $\log(N), c$ and the number of roots.*

For these theorems to result in algorithms that run, in practice, in polynomial time in the size of the input, we generally need to make two assumptions:

**Conjecture 1. The Coppersmith's method assumptions** are:

1. The polynomials with a known small solution, either over $\mathbb{Z}$ or $\mathbb{Z}_N$, have only one small solution.

2. The polynomials obtained from the LLL-reduced basis vectors are all algebraically independent.

Though these assumptions usually hold, there is some reported cases where they do not. Deeper knowledge regarding these two assumptions would greatly improve the security of RSA.

## 1.6 RSA Definition

We are now in conditions to present the mathematical definition of RSA. We will consider an RSA cryptosystem to be a tuple $< N,M,C,K,E,D >$ where:

> $N = pq$ - the public modulus, the product of two different prime numbers $p, q$.
>
> $M$ - the set of plain text. $M = \mathbb{Z}_N$.
>
> $C$ - the set of cypher texts. $C = \mathbb{Z}_N$.
>
> $K$ - is a tuple $< p, q, e, d >$ where $(d, \phi(N)) = 1$ and $ed \cong 1 \pmod{\phi(N)}$
>
> > $K_r =< e, N >$ is the public key
> >
> > $K_p =< p, q, d, N >$ is the private key
>
> $E$ - the encryption function: $E : M \to C$, $c = E(m|K_r) \cong m^e \pmod N$
>
> $D$ - the decryption function: $D : C \to M$, $m = E(c|K_p) \cong c^d \pmod N$

$e$ and $d$ are called **public** and **private exponent** respectively. The exponents satisfy the equation $ed - 1 = k\phi(N)$, which is therefore called the **key equation**. It is also possible to define the exponents modulo $\lambda(N)$. Since this is a multiple of $\phi(N)$ the rest of the procedures described in this section are the same.

The transmission of messages is as follows: suppose Alice wishes to send a plain text message $m \in M$ to Bob. Alice encrypts $m$ using Bob's public key $< e, N >$ and obtains

$$c \cong m^e \pmod N \tag{18}$$

Then, she sends $c$ through an open channel to Bob. Now Bob gets $c$ and decrypts it using his private key $< d, N >$:

$$m' \cong c^d \cong m^{ed} \cong m \pmod N \tag{19}$$

The last equality is a result of Euler's Theorem:

$$m^{ed} \cong m^{1+k\phi(N)} \cong m(m^{\phi(N)})^k \cong m1^k \cong m \pmod N \tag{20}$$

The cryptosystem is easy to implement, which is one of the reasons why it is so popular.

**Algorithm 3. The implementation of a RSA session** with a $n$ bit modulus:

1. Generate two large different prime numbers $p, q$. This can be efficiently done by generating random positive integers of size $\frac{n}{2}$ and checking their primality (as described in chapter 2) until we have two primes[38].

2. Compute $N = pq$ and $\phi(N) = (p-1)(q-1)$.

3. Choose $e : (e, \phi(N)) = 1$ and compute $d \cong e^{-1} \pmod{\phi(N)}$. To choose $e$ in such a conditions, all one has to do is to choose a prime $e' > max\{p, q\}$ and reduce it modulo $N$ [38]. To compute $d$ we can then use the algorithm described in Chapter 1 for computing modular inverses.

4. Make available for everyone the public key $< e, N >$ and keep **completely** in secret the private key $< p, q, d, N >$.

All of the parameters of the RSA will influence its security level and the running time of the operations used by RSA. We usually require the size of $N$ to be large, usually 1024, 2048 or larger. Because we want that it will also be hard to factor, it is recommended to have $p$ and $q$ as large as possible given the size of the modulus. The usual procedure to ensure this is for a $n$-bits modulus is to generate $p$ and $q$ as $\frac{n}{2}$-bits primes, like described in step 1, which will be called **balanced primes** as they have the same bit size.

In step 3 we can compute the exponents in reverse order: we usually chose to compute first the exponent for which we need some restraints. These can be for example $e = 3$ in order to reduce encryption costs or $d > N^{\frac{1}{4}}$, which prevents some attacks like shown in Section 3.

## 1.7 RSA Safety

The security of RSA depends on two of its trapdoor one way functions. The encryption function is one of them, and the problem of inverting it is called the RSA problem:

**Definition 13. (The RSA Problem)** Given a cypher text $c = m^e \pmod{N}$ encrypted using the public key $< e, N >$, compute the plain text $m$ knowing only $< e, N >$.

This problem is, mathematically, to compute $e$-th roots modulo $N$. As it is an open question to know whether there is a polynomial time algorithm which calculates these roots (though, as we saw, there is an algorithm that encrypts the messages in polynomial time), it is assumed nowadays that it is hard to calculate such roots when $m \in \mathbb{Z}_N$ is randomly chosen and $N$ is generated by random large primes $p, q$.

Another trapdoor one way function in RSA is the modulus. Computing $N = pq$ is easy, but what about factoring $N$ knowing only $< e, N >$? The Problem of Factoring Large Integers is defined as follows:

**Definition 14. (The Problem of Factoring Large Integers)** Given a large integer $N$, compute its prime factorization.

Again this is a trapdoor one way function: it is easy to compute the product but difficult to compute the factors.

So lets look at the RSA Problem. If he wants to know $m$, Marvin can try to find out $d$ and then decrypt $m$. To compute $d$, Marvin must first know $\phi(N)$ and for this he needs to factor $N$. If Marvin can factor $N$, he can compute $\phi(N)$ and $d = e^{-1} \pmod N$. This means that once we solve the problem of factoring $N$, we can actually solve the RSA Problem for any $m$. So the RSA problem is at most as difficult as the Problem of Factoring Large Integers. It remains nowadays an open question to know whether both problems have the same complexity.

As the RSA modulus $N$ is a large number, factoring it is a rather hard task given that its prime factors are generated randomly and balanced. The study of factoring methods is an active field in Mathematics and, though it gained special relevance more than 30 years ago with the appearance of RSA, it is still unknown whether a polynomial time algorithm that solves this problem in a classic computer exists. For this reason, it is assumed that if Marvin wishes to factor an RSA modulus, in order to break the system, will need an amount of time far larger than the usual duration of an RSA session.

So the safety of RSA relies deeply on the assumption that both the RSA Problem and the Problem of Factoring Large Integers have no polynomial time algorithm that solves it. But more relevant in practical terms is the fact that, over 30 years of existence, no devastating attack on RSA has been publicized which cannot be easily avoided as we will show in this work.

There is some flaws in the definition of RSA presented in the previous section. In fact, it is not possible to safely implement RSA in such a way. We now present some reasons why such an implementation does not provide appropriate security.

If Marvin knows $\phi(N)$, then he can solve the system of equations

$$\Big(N = pq \ \wedge \phi(N) = (p-1)(q-1)\Big)$$

for $p$ and $q$ and therefore find the factorization of $N$. It is easy to check that $p$ and $q$ are the solutions of

$$x^2 - (N - \phi(N) + 1)x + N = 0 \tag{21}$$

which can be solved efficiently. For this reason, the value of $\phi(N)$ should always be kept secret.

Suppose Alice wishes to send to Bob a plain text $m = kp$ where $p$ is one of the factors of $N$. She will encrypt it with Bob's public key $< e, N >$ obtaining $c \cong m^e \pmod N$. If Marvin intercepts $c$, all he has to do is to compute $(c, N) = p$ to discover a factor of $N$ and consequently break the system. So, when choosing the plain texts, **we cannot choose plain texts which are not relatively prime to $N$**.

RSA has the **Homomorphic Property**, that is, the encryption of the product of two plain texts is equal to the product of the two encryptions. In RSA, this is equivalent to the following statement: given two plain texts $m_1, m_2$ and their cypher texts $c_1, c_2$, we have:

$$(m_1 m_2)^e \cong c_1 c_2 \pmod N \tag{22}$$

which is verified for RSA. This paves the way for an attack against RSA. Suppose $m$ is encrypted as $c \cong m^e \pmod N$. Marvin, knowing $c$, chooses a random $x \in C$ and asks for the plain text of $c_0 \cong cx^e \pmod N$. Notice that this plain text, $m_0$, satisfies:

$$m_0 \cong c_0^d \cong (cx^e)^d \cong c^d x^{ed} \cong mx \pmod N \tag{23}$$

So all he has to do is compute the plain text $m \cong m_0 x^{-1} \pmod N$.

Another fact about RSA is that it does not provide **Semantical Security**, which happens in a cryptosystem when, knowing only the cypher text and the public key, no information about the plain text can be recovered. This is clearly not the case regarding RSA as we have defined it. Knowing two different plain texts (say, "YES" and "NO") and one cypher text, Marvin can easily find out to what plain text the cypher text corresponds to by simply encrypting it with the public key. This means that there is no deterministic public key cryptosystem that is semantically secure.

Because of these properties, it becomes clear that, prior to encryption, a random padding scheme should be applied to the plain text.

To sum up, there is some mandatory implementation changes we can state at this moment:

1. The value $\phi(N)$ should be kept secret,
2. We cannot use plain texts $m : (m, N) \neq 1$,
3. A prior random padding scheme should always be applied to $m$ before encryption.

## 1.8 Variants of RSA

In this section we present some of the variants of RSA that have been created over the years. Although not analysed in this work, we will present some of their advantages comparing with the original RSA cryptosystem.

### 1.8.1 CRT-RSA

The encryption and decryption operations take time linear to the bit size of the encryption and decryption exponents respectively [20]. One simple variant of RSA consists of, in the decryption process, using all the information in the private key $< p, q, d, N >$, that is, to use also the factors $p$ and $q$. Knowing them, Bob can compute two partial decryptions $m_p \cong c^e \pmod{p}$ and $m_q \cong c^e \pmod{q}$ and then combine the results using the Chinese Remainder Theorem to obtain the plain text $m$. This is an usual procedure because it reduces the decryption costs by a factor of 4 [23]. In the literature it is usually refered to as **CRT-RSA**.

### 1.8.2 Multi-Prime RSA

Multi-Prime RSA, like the name says, uses $r < 1$ distinct primes $p_1, p_2, ..., p_r$ to form the modulus $N = p_1 p_2 ... p_r$. A Multi-Prime RSA instance using $r$-primes is called $r$-prime RSA. For greater safety, it is usual to use balanced primes. The proceedings like computing the exponents and the encryption method are done in exactly the same way as the original RSA (that is, 2-RSA).

The greatest advantage of Multi-Prime RSA is that it allows for a faster decryption process when Bob uses its knowledge of the factorization of $N$. When receiving a cypher text $c = m^e \pmod N$ he starts by computing the partial decryptions $x_i \cong c \pmod{p_i}$ for $i = 1, ..., r$. Then he applies the CRT theorem and recovers the plain text $m$. According to results by [19], the worst case cost of decryption for a 3-prime RSA is $\frac{9}{4}$ smaller than the worst case for RSA and this results gets even better with 4-prime RSA, which presents a worst case cost 4 times smaller than that of the RSA.

One interesting topic regarding Multi-Prime RSA is the number of primes one should use. It is obvious that the more primes we use the quicker the decryption will be, but using too many will result in $N$ having smaller factors. The Number Field Sieve is the most efficient generic factorization method known presently. Its running time depends solely on the size of $N$, the integer we want to factor. On the other side, the Elliptic Curve Method is an extremely efficient factorization algorithm for numbers $N$ with a small prime factor, as it depends on the size of $N$ but also on the size of $p$, its smallest prime factor. When we choose the size of the modulus, $N$, the runtime of the NFS is fixed. The runtime of the Elliptic Curve Method will then be proportional to $N$ but also to $p$, the size of the smallest prime factor $p$. So, the idea that occurred to Lenstra was to, for each size of a modulus $N$, calculate how many primes one can use in $r$-prime RSA so that the runtime for both algorithms is the same for standard RSA and for $r$-prime RSA. His work[28] resulted in recommendations for the number of primes that we can use safely for $r$-prime RSA with modulus of 1024,2048,4096 and 8192 bits. It should be noted that, besides taking in account the available computational power at the time of publication of the article, Lenstra also considered its probable evolution.

### 1.8.3 Common Prime RSA

For the Common Prime RSA the primes $p$ and $q$ are chosen so that $(p-1, q-1)$ has a large prime factor and the encryption and decryption exponents are defined as inverses modulo $\lambda(N)$ instead of modulo $\phi(N)$. The advantage of this variant is that it allows us to use smaller encrypting and decrypting exponents, therefore reducing greatly the encryption and decryption operations' costs, that are seemingly resistant to the existing attacks on RSA. The downside is that generating the primes $p$ and $q$ is more time consuming. For a detailed survey about this variant of RSA we suggest reading [20].

# 2 Primality Tests and Factorization Algorithms

## 2.1 Primality Tests

Primality tests are algorithms that receive as input a positive integer $N$ and output some statement about its primality. In the best scenario, they guarantee that either $N$ is prime or that $N$ is composite. Such tests are called deterministic. Other tests either guarantee that $N$ is composite, or that $N$ is prime with a certain probability. These tests are called probabilistic.

The most obvious primality test for an integer $N$, known for thousands of years, consists in simple trial division by all integers up to $\sqrt{N}$, since any composite number must have a divisor $p$ that satisfies $p \leq \sqrt{N}$. If a factor is found, we know that the number is composite, otherwise we prove that $N$ is prime. This deterministic algorithm has efficiency $O(\sqrt{N})$ and therefore is extremely inefficient for large integers, say with 1024 bits like often used in RSA.

The probabilistic primality tests we will explain, most of them, simply aim to prove that $N$ is a composite number. If they do not prove it, then there is a chance that they are prime. Repeating this test with different parameters will lead to a bigger certainty that we are in the presence of a prime number.

### 2.1.1 Fermat's Primality Test

The first primality test presented in this section derives from a theorem discovered by Fermat in 1640. The proof is due to Leibniz, who found it 40 years later. It states the following:

**Theorem 7.** *For any prime number $p$ and for any integer $a$ such that $0 < a < p$, we have:*

$$a^{(p-1)} \equiv 1 \pmod{p} \tag{24}$$

*Proof.* $a, 2a, 3a, ..., (p-1)a$ are all distinct mod p, therefore each of them is congruent to one of $1, 2, 3, ..., p-1$. Because congruences preserve multiplicity, we have:

$$\begin{aligned} a2a3a...(p-1)a \cong & \quad 1 * 2 * 3 * ... * (p-1) \pmod{p} \Leftrightarrow & (25) \\ (p-1)!a^{p-1} \cong & \quad (p-1)! \pmod{p} & (26) \end{aligned}$$

Diving both sides by $(p-1)!$ we get the equality. □

So if we use contrapositive of this theorem, we get a test to check whether a given number is composite:

**Theorem 8.** *An odd positive integer $N$ is composite if there exists a positive integer $a$ such that $(a, N) = 1$ and*

$$a^{N-1} \ncong 1 \pmod{N} \tag{27}$$

There are composite numbers $N$ that satisfy $a^{N-1} \equiv 1 \pmod{N}$ for some $a$ in the conditions of the theorem. In this case, $N$ is called a *base-a pseudoprime.* We call it pseudoprime because, if we run Fermat's Primality Test (FPT) for $N$ with such a base $a$, the test will not identify $N$ as composite. One way to round this problem is to simply run the test again with a different $a$ : *base-a pseudoprime* form quite different sets for different values of $a$, so we have the hope of identifying $N$ as a composite by simply trying out different values of $a$. There is, however, one problem: there are composite numbers $N$ which are *base-a pseudoprime* for all $a$ satisfying the conditions of the theorem. These numbers are called *Carmichael Numbers.*

**Definition 15.** A composite number $N$ is called a **Carmichael Number** if it satisfies:

$$a^{N-1} \cong 1 \pmod{N} \tag{28}$$

for every positive integer $a$ such that $(a, N) = 1$.

So a Carmichael number will never be identified as composite by FPT. Although they are highly infrequent, Carmichael numbers are an infinite set so they represent the biggest flaw of FPT. For this reason, FTP alone is not a deterministic test.

### 2.1.2 Solovay-Strassen Test

This is the primality test suggested by the RSA team for generating the primes $p$ and $q$. It is based on the following theorem, due to Euler[1]:

**Theorem 9.** *If $p$ is an odd prime number, and $b \in \{1, ..., p-1\}$ such that $(b,p) = 1$ then:*

$$\left(\frac{b}{p}\right) \cong b^{\frac{p-1}{2}} \pmod p \tag{29}$$

So the idea of the test goes as follows: given an integer $N$ whose primality we want to check, we choose a random integer $b$ such that $0 < b < N$ and check $(b, N)$. If the greatest common divisor is different from one, we found a factor of $N$ so $N$ is composite. If it is 1, we verify the congruence: if it fails, then we know that $N$ is composite. If it is true, then there is a positive probability that $N$ is prime.

Like the FPT, this test only gives a reliable output when it proves compositeness because like with the FPT, there are composite numbers which will satisfy the congruence for some bases. However, there are no composite numbers which will satisfy the congruences for all the bases like the Carmichael Numbers do for the FPT. In this way, the Solovay-Strassen test is a much better test than FPT. The following result states this:

**Theorem 10.** *Let $N$ be an odd composite. Then there is an element $b \in \mathbb{Z}_N : (N, b) = 1$ such that:*

$$\left(\frac{b}{N}\right) \ncong b^{\frac{N-1}{2}} \pmod N \tag{30}$$

So a composite number $N$ can be a pseudoprime for some bases $b$, but it will never be for all of them. Therefore, if we run the test using all integers up to $N$ as bases, we will be sure about $N$'s primality. But do we really need to use all of them? The next theorem tells us about the number of bases that, for a given composite $N$, actually satisfy the congruence:

**Theorem 11.** *Let $N$ be an odd composite. Then at least half of the integers $b$ co-prime to $N$ in $\{1, ..., N-1\}$ satisfy:*

$$\left(\frac{b}{N}\right) \ncong b^{\frac{N-1}{2}} \pmod N \tag{31}$$

[1]In this section $(\frac{a}{b})$ refers to the Legendre Symbol

The consequence of this theorem is that for each test we apply to verify $N$'s primality, the probability that the test is wrong is less than $\frac{1}{2}$. This means that, if we apply $k$ tests with different random bases, and $N$ passes all the tests, then there is a chance of less than $\frac{1}{2^k}$ that $N$ is composite.

### 2.1.3 Miller-Rabin Primality Test

The primality test presented in this section represents an evolution of FPT as for it there is no equivalent for the *Carmichael Numbers*, that is, there is no composite numbers which will pass the test for every different base. It is also more reliable than the Solovay-Strassen test as we will see. Again there is some composite numbers which fool the test. These are called the *strong pseudoprimes* but these numbers are, as we will see, slightly more understandable. The test is based on the following theorem, due to Miller:

**Theorem 12.** *Given an odd prime $N$, written as $N = 1 + 2^s d$, where $d$ is odd, then the sequence:*

$$a^d, a^{2d}, a^{4d}, ..., a^{2^{s-1}d}, a^{2^s d} \pmod N \tag{32}$$

*ends with 1. If*

$$a^d \not\equiv 1 \pmod N$$

*then the value preceding the first appearance of* $1$ *is* $N - 1$.

This condition, unfortunately, is also verified by some composite numbers, for some base $a$.

**Definition 16.** A composite number $N$ which satisfies the conditions described by the theorem above, for an integer $a$, is said to be a **strong pseudoprime to the base** $a$. If a number is either prime or strong pseudoprime, we call it **probably prime**.

To apply the Miller-Rabin test, we need the following theorem:

**Theorem 13.** *Given an integer $N$, let*

$$N - 1 = 2^s d$$

*with $d$ odd, and $s \geqslant 0$. Then $N$ is probably prime if:*

$$a^d \cong 1 \pmod{N} \tag{33}$$

*or*

$$(a^d)^{2^r} \cong (N-1) \pmod{N} \tag{34}$$

*for some $r$ less than $s$.*

The running time of this probabilistic algorithm is $O(k \log^3 N)$, where $k$ is the number of times we run the test with different bases $a$. Therefore the Miller-Rabin Primality test is a probabilistic primality test running in polynomial time. Could we make this a deterministic primality test? The answer is yes, we can, but we lose its efficiency.

**Theorem 14.** *Let $N > 1$ be an odd composite integer. Then $N$ passes the Miller-Rabin Primality Test for at most $\frac{N-1}{4}$ bases $a$ with $1 < a < N$.*

So here is yet another deterministic primality test: given that an integer $N$ passes $k > \frac{N-1}{4}$ tests, we are sure it is prime. However, the running time of this algorithm is $O(\frac{N}{4} \log^3(N))$, still infeasible. There is one interesting result that would permit this test to become both deterministic and efficient:

**Theorem 15.** *If the Generalized Riemann Hypothesis (GRH) is true and $N$ passes the Miller-Rabin Primality Test for all bases $a : 1 < a < 2(\log N)^2$, then $N$ is prime.*

So the proof of the GRH would lead to a deterministic version of the Miller-Rabin Primality Test with running time $O(\log^5 N)$. Though this is only a conjecture and therefore the test can be applied reliably only in its probabilistic version, it is one of the most used primality tests within programs such as Mathematica. The reason follows:

**Theorem 16.** *The probability that a composite $N$ passes the Miller-Rabin Primality Test for a random base $a$ is $\frac{1}{4}$ at most.*

This means that, if we test for 100 different random bases, the probability that all the tests are wrong is less than $\frac{1}{10^{40}}$!

### 2.1.4 AKS Test

In 2004, a major breakthrough was achieved by a team of computer scientists from the Indian Institute of Technology Kanpur. In an email sent worldwide, Manindra Agrawal, Neeraj Kayal and Nitin Saxena (AKS) announced the world they had found a deterministic polynomial time algorithm to verify an integer's primality.

Although the running time of the original algorithm was $O\ (\log^{\frac{21}{2}} N)$[29], some recent efforts by Hendrik Lenstra and Carl Pomerance have created a variation of the algorithm whose time complexity is $O\ (\log^6 N)$ [29] (which is still slower than the Miller-Rabin Deterministic Test under the GRH).

The algorithm is based on the following result:

**Theorem 17.** *Let $a \in \mathbb{Z}, N \in \mathbb{N}, N \geq 2$ and $(a, N) = 1$. Then $N$ is prime if and only if:*

$$(X + a)^N = X^N + a \pmod{N}. \tag{35}$$

This criteria, which is stronger than those presented before as it is both necessary and sufficient for an integer to be prime, cannot be applied straightforwardly to check an integer's primality because, for large $N$, there are too many coefficients to calculate in the binomial. So the great development in 2004 was to, based on this criteria, create a deterministic algorithm for checking primality. The original AKS algorithm is presented next:

**Algorithm 4. The AKS algorithm**. Given a positive integer $N > 1$:

1. Choose $a \in \mathbf{Z}$ and $b > 1$. If $N = a^b$, then STOP and output COMPOSITE.
2. Find the smallest $r$ such that $o_r(N) > \log^2(N)$.
3. If $1 < (a, N) < N$, for some $a \leq r$, output COMPOSITE.
4. If $N \leq r$, output PRIME.
5. For $a = 1, 2, ..., \lfloor \sqrt{\phi(r)} \log(N) \rfloor$, do:

   if $((X+a)^N \not\equiv X^N + a \pmod{X^r - 1, N}))$, output COMPOSITE. (36)

6. Output PRIME

It is straightforward that, when we input a prime number, the output will always be prime: in steps 1 and 3 it cannot be considered composite, and in step 6, because of the previous theorem, it will also not be considered composite in any of the iterations of the cycle. So it will be considered prime, either in step 5 or step 7. For a proof that a composite integer is never considered prime one can read the original AKS article [29]. In [41] an implementation of the algorithm is described.

There is one important conjecture that, if proven, would improve the efficiency of this algorithm.

**Conjecture 2.** Let $r$ be a prime that does not divide the positive integer $N$ and such that $(X-1)^N \cong X^N - 1 \pmod{n, X^r - 1}$. Then

$$(N \text{ is prime} \bigvee N^2 \cong 1 \pmod{r}) \tag{37}$$

If this conjecture could be proved, Lenstra and Pomerance's variant of the algorithm would have complexity $O(\log^3 n)$[41].

## 2.2 Factoring Algorithms

Like we said before, the security of RSA depends strongly on the difficulty of factoring the modulus $N$. For this reason, special attention should be given to the state of the art of integer factoring algorithms when implementing RSA. There are published standards with recommendations [2] for the size of $N$ and the primes' size depending mostly on how long we wish to keep the encrypted data safe. This recommendations depend on some of the algorithms presented in this section.

Factoring algorithms fall into two categories. One is the **General Purpose Factoring Algorithms** which behave in approximately the same way for integers with the same size. The second category is that of the **Special Purpose Factoring Algorithms** which behave better for integers $N$ with certain specific characteristics, like the size of the smallest factor of $N$ or the divisors of $(N-1)$ or $(N+1)$. To attack an RSA session with balanced primes, a general purpose algorithm is *a priori* more suitable, since the factors of $N$ are randomly chosen and of the same size.

[2]http://www.rsa.com/rsalabs/node.asp?id=2218

We will begin with an old method created by Fermat. This method is still relevant today, though inefficient, because it was the basis for many stronger algorithms that came later. Next we present Pollard's $\rho$ method, which, though much more recent, is also an obsolete algorithm that laid the basis for some more sophisticated algorithms that followed it. We finish by presenting the Elliptic Curve Method and the General Number Field Sieve, two powerful factoring algorithms, special and general purpose respectively.

### 2.2.1 Fermat's Factorization

A very old idea to find a factor of an integer comes from Fermat. Suppose $N = pq$ is a product of two primes. If we find two integers $x, y : N = x^2 - y^2$ and $x - y > 1$ then we have that $N = (x - y)(x + y)$, therefore we found the two factors of $N$. So Fermat's idea was to try several pairs of $x, y$ until the first equality is verified. He created an algorithm so as to choose these trials values. We describe below the algorithm that, for an input $N = pq$, returns its factors:

**Algorithm 5. Fermat's Factorization Algorithm**

Given a composite integer $N$,

```
1. Set $w = \lfloor\sqrt{N}\rfloor$ and $x = w$
2. set $y = \lfloor\sqrt{x^2 - n}\rfloor$
3. if $n = x^2 - y^2$, output $x - y$ and $x + y$
4. if $x < n$, replace $x$ by $x + 1$ and go to step 2.
```

Because we know that $N$ is composite, the algorithm will always return its prime factors. Looking at step 4, we realize that it will try all integers from $\sqrt{N}$ to $N$ and because of this it is actually a very slow algorithm in the worst case scenario, taking $O(\sqrt{N})$ steps, meaning it is even worst than trial division.

However, it should be noted that when applied to an RSA modulus $N$ whose two prime factors are close to each other, the method works quite fast. In the extreme case when the factors are consecutive primes, a few seconds will be enough to find them! So, when implementing RSA, one should be aware that choosing consecutive primes will make the system vulnerable to one of the oldest factorization methods!

### 2.2.2 Pollard's $\rho$ Algorithm

Pollard's $\rho$ Algorithm, described by Pollard in 1975[35], aims to find a small factor $p$ of a given integer $N$. Because of this focusing on a small factor of $N$, Pollard's $\rho$ is considered a special purpose factoring algorithm. We present a simplified version the algorithm:

**Algorithm 6. Pollard's $\rho$ Algorithm:** Given a composite $N$:

1. Set $a = 2, b = 2$.
2. Define the modular polynomial $f(x) = (x^2 + c) \pmod N$ with $c \neq 0, -2$
3. For i=1,2,.., do:
    - (a) Compute $a = f(a), b = f(f(b))$.
    - (b) Compute $d = (a - b, N)$.
    - (c) If $1 < d < N$ then return $d$ with success.
    - (d) If $d = N$ then terminate the algorithm with failure.

The function $f$ is used to create two pseudo random sequences on $\mathbb{Z}_N$. The reason for this is that, picking randomly two numbers $x, y \in \mathbb{Z}_N$, there is a probability of 0.5 that after $1.777\sqrt{N}$ tries one will be congruent modulo $N$[37]. If they are and $a \neq b$, then $(a - b, N)$ yields a factor of $N$.

The runtime of the algorithm is $O(\sqrt{p})$[24], where $p$ is $N$'s smallest prime factor. This means that against an RSA modulus $N$ with balanced primes the runtime of the algorithm is $O(N^{\frac{1}{4}})$, making it an inefficient method.

### 2.2.3 Elliptic Curve Method

In 1985, Hendrik Lenstra had a visionary idea. He thought of using Elliptic Curves to factor integers. It is said that that Lenstra's discovery was based on Pollard's $(\rho - 1)$ method[44], a variant of Pollard's $\rho$ method described above.

The Elliptic Curve Method's (ECM) running time depends on the smallest factor $p$ of the integer $N$ being factored, making it particularly powerful for finding "medium-sized" factors of large integers. As we will show in the next section, ECM is not used directly to factor RSA modulus, instead it is used as an auxiliary step in the General Number Field Sieve algorithm [44] described in the next section. The basis of the algorithm is the following result:

**Theorem 18.** *Let $N > 1$ be an integer with $(N, 6) = 1$. Let $E$ be an elliptic curve modulo $N$, and let $m$ and $s$ be positive integers such that $s$ divides $m$. Suppose there is a point $P \in E(\mathbb{Z}/n\mathbb{Z})$ satisfying:*

1. *$m \cdot P = 0$,*
2. *$(m/q) \cdot P$ is defined and different from $0$, for each prime $q$ dividing $s$.*

*Then $\sharp E(\mathbb{F}_p) \cong 0 \pmod{s}$ for every prime $p$ dividing $N$ and, if $s > (N^{\frac{1}{4}} + 1)^2$ then $N$ is prime.*

Using this result, Lenstra developed an algorithm to check whether a given composite $N$ is prime. We present a description of the algorithm taken from [1]

**Algorithm 7. Lenstra's Elliptic Curve Algorithm**: Given an integer $N$:

1. Check that $(6, N) = 1$ so that $N \neq m^r$ for any $r \geq 2$.
2. Choose integers $A, x_1, y_1$ such that $1 < A, x_1, y_1 < N$.
3. Let $E$ be the elliptic curve $E : y^2 = x^3 + Ax + B$ where $B = y_1^2 - x_1^2 - Ax_1$ and let $P = (x_1, y_1) \in E$.
4. Check that $a = (4A^3 + 27B^2, N) = 1$.

   If $a = N$, go back to step 2 and choose a new A.

   If $1 < a < N$, then $a$ is a factor of $N$.
5. Choose an integer $k$ such that

   k=$LCM\{1, 2, 3, ..., K\}$ for some $K \in \mathbb{N}$.
6. Compute $kP = \left(\frac{a_k}{d_k^2}, \frac{b_k}{d_k^3}\right) = 1P + 2P + 2^2P + 2^3P + ... + 2^rP$, for some $r \in \mathbb{N}$.

   $= P_0 + P_1 + P_2 + ... + P_r$, for some $r \in \mathbb{N}$.

   $= \sum_{k_i=1}^{r} P_i$.
7. Calculate $D = (d_k, N)$

   If $1 < D < N$, then D is a non-trivial factor of $N$. If $D = 1$, return to step 2 and choose a new A. If $D = N$, return to step 5 and decrease the value of $k$.

The running time of this algorithm is $E(N, p) = (\log_2 N)^2 e^{\sqrt{2}(\log p)^{\frac{1}{2}}(\log\log p)^{\frac{1}{2}}}$ [20], where $N$ is the integer to be factored and $p$ its smallest prime factor.

### 2.2.4 General Number Field Sieve

The General Number Field Sieve (GNFS) is the fastest known general purpose method for factoring large integers. For this reason, it is very suitable for attacking RSA, as the factors $p$ and $q$ are balanced and so $N$ does not have a small factor, which makes the special purpose algorithms weaker.

For a pleasant reading about the contributions several people gave to the development of this algorithm and a simple explanation of its procedure one can read [36].

The GNFS is an extremely complex algorithm using results from several fields of mathematics and for this reason it was not possible to present it in our work. For an extensive explanation of the mathematical basis behind this algorithm we suggest the reading of [6]. The running time of the GNFS for factoring an integer $N$ of size $n$ is $E(n) = exp(1.923n^{\frac{1}{3}} \log^{\frac{2}{3}} n)$[4][24].

As the state of the art in integer factoring, GNFS's complexity has a double importance: attacks on RSA which take more time than GNFS to factor the number are not interesting anymore.

GNFS actually holds the record for the largest general integer ever factored, the RSA-200, a number with 200 digits.

## 2.3 Overview of the Different Methods

Regarding the primality tests we described, clearly AKS is the most promising one. For this reason, developments regarding this algorithm should be followed closely. In what concerns the probabilistic tests described before, we think that the Miller-Rabin test is the most efficient and reliable test. It is of special interest that, if some advances will be made, it can become an extremely simple polynomial time deterministic test. Finally, FTP was included solely for demonstrative purposes and the Solovay-Strassen test was included because it was the (now obsolete) primality test proposed in the original RSA article[38].

Regarding the factorization algorithms presented in this work, Fermat's factorization method does not present a serious menace for a RSA session unless the primes chosen are (almost) consecutive. It is usually advised to use Pollard's $\rho$ method to find factors of up to 30 bits [11] and then switch to the Elliptic Curve Method if we are searching for factors larger than 30 bits. However, recent developments described in [11] suggest that this number of bits should actually be reduced. Regarding the GNFS, it should be noted that this is without doubt the fastest known algorithm for factoring hard integers like the RSA modulus $N$. One of the steps in the GNFS involves factoring some easier integers, that is, integers which are expected to have at least one small factor. In this step of GNFS, Lenstra's ECM represents a fundamental tool.

# 3 Cryptanalysis of RSA

In this section we present the reader to some of the known attacks against RSA. As we had a limited amount of time and a specific area of interest (mathematics), it was not possible to include all the existing attacks against RSA. Rather, we introduce attacks that explore uniquely the mathematical structure of RSA. For some other kind of attacks on RSA we suggest the reading of [4][22][8][48].

## 3.1 Kind of Attacks

There are several types of attacks on RSA. The obvious one is to factor the modulus $N$, which will create a total break of the system: the cryptanalyst will be able to decrypt all messages. This can be achieved with one of the methods presented in the last chapter. As the factoring methods described above still do not run in polynomial time, an appropriate choice of the size of $N$ makes this **factoring attack** infeasible. There are published standards with recommendations for the size of $N$ that should be chosen, depending mostly on the amount of time we wish to keep our data secret.

As we said before, the security of RSA relies mainly on the hardness of the RSA Problem and the Problem of Factoring Large Integers. There is however, like shown before, some implementation errors that can open breaches on RSA security. We start this chapter by presenting some basic errors an inexperienced user can commit when starting out with RSA and provide the reader the ways on how to avoid these errors. In the second and third section of this chapter, we illustrate the dangers of choosing small public and private exponents respectively, recommending (presently) safe bounds for both these values.

## 3.2 Some Misuses of RSA

The attacks presented in this section were found a long time ago. They showed some of the possible misuses of an RSA session.

### 3.2.1 Common Modulus Attack

The idea of the common modulus is that in a session of RSA with several users there is a trusted entity which defines a modulus $N$ and provides for each user a pair of public and private valid RSA keys defined modulo $\phi(n)$, but not the factorization of $N$. That is, each user $U_i$ gets the public key $< e_i, N >$ and the private key $< d_i, N >$. Simmons [42] showed that, without needing to factor the modulus, if the same plain text is encrypted and sent to two users with co-prime public exponents, any other user can decrypt the corresponding cypher text.

**Theorem 19.** *Let $N = pq$ be a RSA modulus and let $< e_1, N >, < e_2, N >$ be two public keys such that $(e_1, e_2) = 1$. Suppose a plain text $m$ is encrypted with both public keys. Knowing $c_1 = m^{e_1} \pmod N$, $c_2 = m^{e_2} \pmod N$ and the public keys, we can compute $m$ in time polynomial in $\log(N)$.*

*Proof.* Knowing $e_1$ and $e_2$, we compute integers $a_1, a_2$ such that $a_1e_1 + a_2e_2 = 1$ using the Extended Euclidean Algorithm. Now we compute

$$c_1^{a_1} c_2^{a_2} \cong m^{a_1 e_1} m^{a_2 e_2} \cong m^{a_1 e_1 + a_2 e_2} \cong m \pmod N \qquad (38)$$

Both the Extended Euclidean Algorithm and the final computation are done in time polynomial in $\log(N)$, so the result follows. □

So this means that anyone with access to the public keys and the cypher texts would be able to intercept all the plain texts which would be encrypted twice to different users.

We implemented this attack and tested it for various values of the size of $N$. The results are compiled in Table 2.

Table 2: Common Modulus Attack's Experimental Results

| size of $N$ (in bits) | time to compute $m$ (in seconds) |
|---|---|
| 8 | 0.000021 |
| 16 | 0.000022 |
| 32 | 0.000048 |
| 64 | 0.000059 |
| 128 | 0.000188 |
| 256 | 0.000478 |
| 512 | 0.001517 |
| 1024 | 0.007022 |
| 2048 | 0.036865 |
| 4096 | 0.231358 |

In 1984 an even stronger attack against Common Modulus RSA was discovered by DeLaurentis[12]. He proved that you don't even need two encryptions of a plain text to actually decrypt all the encrypted messages. The theorem states that any user of the system can actually create a new private key which will work with any other chosen public key. Here is the result:

**Theorem 20.** *Let $< e, N >$ be a valid RSA public key with corresponding private key $< d, N >$. Let $< e_1, N >$ be the public key from another user such that $e_1 \neq e$. Then a private key $< d_1, N >$ corresponding to $< e_1, N >$ can be computed by:*

$$d_1 = e_1^{-1} \pmod{\frac{ed-1}{(e_1, ed-1)}} \tag{39}$$

*in time polynomial in* $\log(N)$*.*

*Proof.* We rewrite the key equation

$$ed \cong 1 \pmod{\phi(N)} \Leftrightarrow ed - 1 = k\phi(N) \tag{40}$$

and notice also that, as $e_1$ is a public exponent it satisfies $(e_1, \phi(N)) = 1$ and therefore $(e_1, k\phi(N)) = k'$ for some $k'$ that divides $k$. Now let $k'' = \frac{k}{k'}$. The modulus in computation (39) can be written as:

$$\frac{ed-1}{(e_1, ed-1)} = \frac{k\phi(N)}{k'} = k''\phi(N) \tag{41}$$

So $e_1$ and $d_1$ satisfy:

$$d_1 \cong e_1^{-1} \pmod{k''\phi(N)} \Longrightarrow d_1 e_1 \cong 1 \pmod{\phi(N)} \tag{42}$$

Therefore $d_1$ is a valid private exponent corresponding to $e_1$. All computations can be done in time polynomial in $\log(N)$. □

The implementation of this attack is presented in the appendix. The experimental results obtained, again for different values of the size of $n$ are represented in Table 3.

The abnormal values for $n = 32$ and $n = 64$ are probably due to the fact that the only relevant operation depending on the size of $N$ timed was a modular exponentiation.

There is one approach even more simple. Using his pair of private/public keys, any user can compute a multiple of $\phi(N)$ through the key equation $ed - 1 = k\phi(N)$. Then, by the results of Miller [33], he can use a probabilistic

Table 3: DeLaurentis Attack's Experimental Results

| size of $N$ (in bits) | time to compute $d_1$ (seconds) |
|---|---|
| 8 | 0.000019 |
| 16 | 0.000012 |
| 32 | 0.000031 |
| 64 | 0.000031 |
| 128 | 0.000030 |
| 256 | 0.000051 |
| 512 | 0.000058 |
| 1024 | 0.000092 |
| 2048 | 0.000186 |
| 4096 | 0.000387 |

algorithm running in polynomial time to factor $N$. So, **when implementing RSA, it is not possible at all to use the same modulus for different users**.

### 3.2.2 Hastad's Broadcast Attack

The following attack is due to Hastad[18]. It is also known as Common Plain text Attack due to the fact that it needs, like the previous attack, that the same plain text be encrypted more than once. In the original attack, presented below, we actually need $k$ messages, $k \geq e$, where $e$ is a common public exponent used to encode the $k$ messages. The theorem supporting it follows:

**Theorem 21.** *Suppose a plain text $m$ is encrypted $k$ times with the public keys $< e, N_1 >, < e, N_2 >, ..., < e, N_k >$ where $k \geq e$ and the $N_1, N_2, ..., N_k$ are pairwise co-prime. Let $N_0 = min\{N_1, N_2, ..., N_k\}$ and $N = \prod_{i=1}^{k} N_i$. If the plain text $m$ satisfies $m < N_0$ then Marvin, knowing $c_i \cong m^e \pmod{N_i}$ and $< e, N_i >$ for $i = 1, 2, ..., k$, can compute the plain text $m$ in time polynomial in* $\log(N)$.

*Proof.* Given that the ($N_i$'s are co-prime, we can apply the CRT to compute $C \cong m^e \pmod{N}$. As $m < N_0$ we have that $m^e < N_1 N_2 ... N_k = N$ and so $C = m^e$. Therefore all we need to do is to compute the $e$-th root of $C$ over the integers to find out $m$. All computations can be done in time polynomial in $\log(N)$. $\square$

The implementation of this attack is presented in the appendix. Table 4 contains the experimental results:

Table 4: Common Plain text Attack's Experimental Results

| size of $N$ (in bits) | time to compute $m$ (seconds) |
|---|---|
| 16 | 0.000169 |
| 32 | 0.000051 |
| 64 | 0.000049 |
| 128 | 0.000049 |
| 256 | 0.000052 |
| 512 | 0.000062 |
| 1024 | 0.000066 |
| 2048 | 0.000095 |
| 4096 | 0.000097 |

Another attack, known as the Related Plain text Attack, allows for the encrypted messages to be different but related by known polynomials, and requires a larger number of messages to be encrypted. The result, due to Bleichenbacher [17], is as follows:

**Theorem 22.** *Given the public keys* $< e_1, N_1 >, < e_2, N_2 >, ..., < e_k, N_k >$ *where the modulus are pairwise co-prime, and* $f_1(x) \in \mathbb{Z}_{N_1}[x], ..., f_k(x) \in \mathbb{Z}_{N_k}[x]$, *set* $N_0 = min\{N_1, N_2, ..., N_k\}$ *and* $N = \prod_{i=1}^{k} N_i$. *For a plain text* $m < N_0$, *if* $k \geq max_i\{e_i deg(f_i(x))\}$ *then given* $c_i = f_i(m) \pmod{N_i}$ *and* $< e_i, N_i >$ *for* $i = 1, 2, ..., k$, *the plain text* $m$ *can be computed in time polynomial in* $\log N$ *and* $max_i\{e_i deg(f_i(x))\}$.

*Proof.* We can suppose all $f_i(x)$ are monic. If not, we just need to multiply them for the inverse of the leading coefficient. If this inverse does not exist for $f_j(x)$, we find a factor of $N_j$ and from $c_j$ we find $m$.

Set $\delta = max_i\{e_i deg(f_i(x))\}$ and also $h_i = \delta - deg(f_i(x)^{e_i})$ for $i = 1, ..., k$. Now we define the $k$ monic polynomials of degree $\delta$:

$$g_i(x) = x^{h_i}(f_i(x)^{e_i} - c_i) \in \mathbb{Z}_{N_i} \text{ , for } i = 1, ..., k \quad (43)$$

Notice that $g_i(m) \cong 0 \pmod{N_i}$ for $i = 1, ..., k$, so we can use the Chinese Remainder Theorem using the $g_i(x)$ and $N_i$ as inputs and compute a new degree $\delta$ monic polynomial $G(x) \in \mathbb{Z}_N[x]$ satisfying:

$$G(m) \cong 0 \pmod{N} \tag{44}$$

where $m < N_0 < N^{\frac{1}{l}} < N^{\frac{1}{D}}$. So we are in condition to use Coppersmith's result. Therefore, we can compute $m$ in time polynomial in $\log(N)$ and $\delta$. □

The results obtained with the algorithm presented in the appendix are shown in Table 5.

Table 5: Related Plain text Attack's Experimental Results

| size of $N$ (in bits) | time to compute the polynomial $G(x)$ (seconds) |
|---|---|
| 782 | 0.000289 |
| 1093 | 0.000308 |
| 1793 | 0.000390 |
| 3373 | 0.000524 |
| 6251 | 0.000736 |
| 12246 | 0.001269 |
| 23918 | 0.0026934 |

The reason for such fast results is that the final step of the attack, solving the univariate modular polynomial, was not done.

Actually, May and Ritzenhofen [30] have improved the bound for the number of cypher texts required. Setting $\delta_i = e_i deg(f_i(x))$, if the inequality

$$\sum_{i=1}^{k} \frac{1}{\delta_i} \geq 1 \tag{45}$$

is satisfied then the plain text $m$ can be recovered from the $k$ cypher texts.

To prevent this broadcast attacks, we need to ensure that this last inequality is not satisfied. A very straightforward way would be not to transmit the same (or related) message massively. If there is a need for this, then we should ensure that inequality (45) is not satisfied. For this we can transform the messages with polynomials of high degree or alternatively use high public exponents.

### 3.2.3 Cycling Attack

The Cycling Attack was one of the first attacks on RSA to be described [43]. As the name of this attack suggests, the way this attack works is by repeatedly encrypting the cypher text. When Marvin gets $c \cong m^e \pmod{N}$, he will encrypt the cypher text with the public key and this will lead him to, eventually, getting an encryption which will be the original cypher text. That is, after $l+1$ encryptions, he will have:

$$c^{e^{l+1}} \cong c \cong m^e \pmod{N} \tag{46}$$

so he will know that the previous encryption is the original plain text, that is:

$$c^{e^{l}} \cong m \pmod{N} \tag{47}$$

The implementation of this attack resulted in the experimental results shown in Table 6.

Table 6: Cycling Attack's Experimental Results

| size of $N$ | average running time (seconds) |
|---|---|
| 8 | 0.000012 |
| 16 | 0.000010 |
| 32 | 0.000008 |
| 64 | 0.000010 |
| 128 | 0.000011 |
| 256 | 0.000013 |
| 512 | 0.000011 |
| 1024 | 0.000011 |
| 2048 | 0.000016 |
| 4096 | 0.000015 |

These results reveal little dependence on $N$.

This means that, after $l$ encryptions of the cypher text, he finds the plain text. For this reason, the value $l$ is called the **recovery exponent for the plain text** $m$. So, given a plain text $m$, all Marvin needs to do is to find its recovery exponent $l$ to reveal the plain text. As this value can be found by an exhaustive search, we want it to be as large as possible. Immediately we see that the plain text message $m = 1$ is a message with recovery exponent equal to 1, that is, Marvin gets the same cypher text after only one encryption, therefore he concludes that the message was not encrypted at all. So such a message should never be sent. Thankfully, most of the plain texts present a much larger recovery exponent. Although it is not possible to easily calculate $l$ for any given plain text, one result is particularly helpful to avoid this attack:

**Theorem 23.** *Suppose a plain text $m$ is encrypted with the public key $< e, N >$. Then the recovery exponent of $m$ divides $\lambda(\lambda(N))$.*

For a proof of this theorem one can read [25]. So, in order to avoid an attack against our RSA system, we should ensure that $\lambda(\lambda(N))$ is large (as it is the maximum value of $l$) and has large prime divisors. It has been shown that, for a large RSA modulus with balanced, randomly generated primes, this is the usual case[22].

There is a generalized cycling attack on RSA. Given a cypher text $c$, it consists in finding the smallest integer $k$ such that $(c^{e^k} - c, N) > 1$. After this, if:

$$c^{e^k} \cong c \pmod p \text{ and } c^{e^k} \ncong c \pmod q \tag{48}$$

we have that $k = p$. Conversely, if

$$c^{e^k} \ncong c \pmod p \text{ and } c^{e^k} \cong c \pmod q \tag{49}$$

then we know that $k = q$. In any way, we found one of the prime factors of the modulus and therefore we broke the system completely. Again according to [22], this attack is infeasible for large values of the randomly chosen primes $p$ an $q$.

This attack provides us with one extra security measure when choosing the generating primes $p$ and $q$. We should ensure that the smallest prime factor of $\lambda(\lambda(pq))$ is a large number.

## 3.3 Recovering Plain texts Encrypted With Small Public Exponent

This section, which could actually include some of the attacks presented in the previous section, regards the safety of the RSA cryptosystem when a small public exponent, such as $e = 3$, is used. This may occur when the encrypting device has small computational powers, such as a cell phone. The attacks are far from breaking the system as they do not aim to factor the modulus but rather to recover specific plain texts or part of them.

### 3.3.1 Stereotyped Message Attack

When encrypting a plain text $m$, one should be careful about the size of it. As the last attack of the previous section shows, the plain text $m = 1$ is easily recovered. This, unfortunately, is not the only one, specially when we use a small public exponent. If the public exponent is sufficiently small, there is a risk that the cypher text will satisfy $c = m^e < N$. Knowing this, all Marvin has to do is to calculate the e-th root of $c$ over the integers. So plain texts $m$ such that $m < N^{\frac{1}{e}}$ cannot be used. This is relevant because RSA is often used to share a key to use in a symmetric key cryptosystem. As an example, if we use a 1024 bit modulus RSA and a public exponent $e = 3$, then the key to share should have at least 342 bits!

The success of the stereotyped message attack depends not on the size of the plain text, but rather on the fraction of its bits we know. To understand how this can be possible, we introduce a classical example: suppose that each day in the morning, a central authority encrypts and sends to another user the plain text "The secret for 1, September, 2011 is ?????". Marvin, knowing this procedure (suppose he worked at the company and now wishes to have its revenge on it) will know a part of the plain text and ignore only a small part of it. Coppersmith[9] showed that, if both the unknown part (that is, the "?????") and the public exponent are sufficiently small, then Marvin can recover the part of the plain text which he still doesn't know.

**Theorem 24.** *Suppose a plain text $m$ is encrypted with the public key $< e, N >$. Knowing $< e, N >$, $c \cong m^e \pmod{N}$ and all the plain text $m$ except a fraction smaller than $\frac{1}{e}$ of consecutive bits of $m$, we can calculate the unknown fraction of bits (and therefore $m$) in time polynomial in $\log(N)$ and $e$.*

*Proof.* Because there is only a fraction of less than $\frac{1}{e}$ consecutive bits of $m$ that we do not know, we can write $m = m_2 2^{k_2} + m_1 2^{k_1} + m_0$ where only the value of $m_1$ is unknown and $|m_1| < N^{\frac{1}{e}}$. Let $f_N(x) \in \mathbb{Z}_N[x]$ be the polynomial defined by

$$f_N(x) = 2^{-k_1 e}((m_2 2^{k_2} + x 2^{k_1} + m_0)^e - c) \pmod N. \tag{50}$$

Notice that $f_N(m_1) = 0 \pmod N$. So $m_1$ is a root of $f_N(x) \pmod N$ satisfying $|m_1| < N^{\frac{1}{e}}$ so we can apply Coppersmith's theorem to compute $m_1$ in time polynomial in $\log(N)$ and $e$ to find out $m$. $\square$

We implemented this attack and obtained the results shown in Table 7.

Table 7: Stereotyped Message Attack's Experimental Results

| size of $N$ | average running time (seconds) |
|---|---|
| 8 | 0.000057 |
| 16 | 0.000073 |
| 32 | 0.000131 |
| 64 | 0.000247 |
| 128 | 0.000550 |
| 256 | 0.001167 |
| 512 | 0.002209 |
| 1024 | 0.004626 |
| 2048 | 0.009513 |

The reason for such fast results is that the final step of the attack, solving the univariate modular polynomial calculated, was not done. This is because we could not implement Coppersmith's method.

There is ways of easily avoiding this attack. We can just choose $e$ such that $e > \log_2(N)$, which means that Marvin would need to actually know the whole plain text. An alternative defence consists in applying to the plain text a random padding which will represent a fraction of more than $\frac{1}{e}$ of the bits of the message.

There is an heuristic extension to this attack. If the unknown bits are not contiguous, we can still recover them as long as the fraction of the total amount of bits does not exceed $\frac{1}{e}$ of the bits of the message. This result depends on an heuristic method created by Coppersmith[9] to find small solutions of bivariate modular polynomials. If these methods can be proved to be right, a stronger attack would be proved to exist.

Therefore it is recommended that at least a fraction of $\frac{1}{e}$ of the bits of a plain text $m$ are randomized.

### 3.3.2 Related Message Attack

The attack presented in this section is considered a small public exponent attack because it is proved to work for $e = 3$. There is, however, some evidence that it should work also with greater exponents[22]. As this is not yet a fact, we present only the original version of the attack.

Suppose Alice sends the cypher text $c \cong m^e \pmod{N}$ to Bob. Marvin, pretending to be Bob, tells Alice that he did not receive the message and requests that it will be sent again. Now Alice will slightly alter the message, say, with a different time stamp, and send it again to Bob and also to Marvin. So Marvin will have two cypher texts, corresponding to two plain texts that, though not being the same, are related by a somewhat "simple" relation. Franklin and Reiter [47] showed that, knowing the two cypher texts and the nature of the relation between them, provided it is simple, Marvin can recover the plain text.

**Theorem 25.** *Let two plain texts $m_1, m_2$ satisfy $m_2 = am_1 + b$. Suppose this two plain texts are encrypted with the public key $< 3, N >$. Then, knowing the corresponding cypher texts $c_1, c_2$, $a, b, N$ and $e = 3$, it is possible to compute $m_1$ (and therefore $m_2$) in time polynomial in* $\log(N)$

*Proof.* Knowing $c_1, c_2, a, b, e, N$ we compute:

$$\frac{b(c_2 + 2a^3c_1 - b^3)}{a(c_2 - a^3c_1 + 2b^3)} \cong \frac{m_1(3a^3bm_1^2 + 3a^2b^2m_1 + 3ab^3)}{3a^3bm_1^2 + 3a^2b^2m_1 + 3ab^3} \cong m_1 \pmod N \quad (51)$$

All calculations are done in time polynomial in $\log(N)$, so we have the required result. $\square$

It becomes clear that, when using public exponent $e = 3$, we cannot send messages linearly related.

### 3.3.3 Random Padding Attack

As it was shown before, plain RSA without a prior padding scheme has been proven to be insecure. So, when implementing RSA, it is mandatory to pad the messages before encryption, that is, to transform the plain text $m$ in the plain text $m' = m+b$ where $b$ is usually a random number with some special structure (for example, number of bits). This procedure needs special attention. When the public exponent is $e = 3$ and the absolute value of $b$ sufficiently small, an attack by Coppersmith [9] allows for the recovery of the plain text.

**Theorem 26.** *Let $< 3, N >$ be a public key. Suppose two plain texts $m_1, m_2$ satisfying $m_2 = m_1 + b$ are encrypted with the public key $< 3, N >$. Knowing the two cypher texts $c_1 \cong m_1^3 \pmod N$ , $c_2 \cong m_2^3 \pmod N$ and the public key $< 3, N >$, if $|b| < N^{\frac{1}{9}}$, it is possible to compute $m_1$ and $m_2$ in time polynomial in* $\log(N)$.

*Proof.* We have that $m_1^3 - c_1 \cong 0 \pmod N$ and $(m_1 + b)^3 - c_2 \cong 0 \pmod N$. Now lets calculate the resultant, taking this two expressions as polynomials in b:

$$\begin{aligned}&Resultant_{m_1}(m_1^3 - c_1, m_1 + b)^3 - c_2) \cong \\ &\cong b^9 + (3c_1 - 3c_2)b^6 + (3c_1^2 + 21c_1c_2 + 3c_2^2)b^3 + (c_1 - c_2)^3 \pmod N \\ &\cong 0 \pmod N\end{aligned}$$

So the monic polynomial of degree 9 given by

$$f_N(x) = x^9 + (3c_1 - 3c_2)x^6 + (3c_1^2 + 21c_1c_2 + 3c_2^2)x^3 + (c_1 - c_2)^3 \pmod N \quad (52)$$

has a root $x_0 = b$. As $b < N^{\frac{1}{9}}$ , we can use Coppersmith's theorem to compute $b$. Once we know the value of $b$, we simply need to apply the attack described in the previous section for related messages. All computations are done in time polynomial in $\log(N)$. □

So it is essential to use a sufficiently large random padding.

### 3.3.4 Leaking Information

When we implement an RSA cryptosystem, the encryption and decryption exponents satisfy the so called key equation $ed = 1 + k\phi(N)$, where $k$ is a constant which should be kept secret. One of the reasons for this is presented below.

Boneh, Durfee and Frankel [10] [15] showed that, knowing $k$, Marvin can find some of the most significant bits of the decryption exponent:

**Theorem 27.** *Let $< e, N >$ and $< d, p, q, N >$ be a pair of public/private keys satisfying the key equation $ed = 1 + k\phi(N)$. Then, knowing $< e, N >$ and $k$, we can compute $d_1$ such that $|d_1 - d| < p + q$ in time polynomial in $\log(N)$.*

*Proof.* Set $d_1 = \lceil \frac{1}{e}(1 + kN) \rfloor$. So $d_1 = \frac{1}{e}(1 + kN) + \alpha$ for some $\alpha$ such that $|\alpha| < 1$. Rewriting the key equation:

$$ed = 1 + k\phi(N) \Leftrightarrow ed = 1 + k(N - s) \quad (53)$$

where s=p+q-1, we can compute:

$$|d_1 - d| = |\frac{1 + kN}{e} + \alpha - \frac{1 + k(N - s)}{e}| = |\frac{ks}{e} + \alpha| < s + 1 = p + q. \quad (54)$$

□

When we use balanced primes to generate the modulus, the inequality $p + q > \frac{3}{2}N^{\frac{1}{2}}$ holds. This means that given only the public key and $k$, we can always compute $d_1 : |d_1 - d| < \frac{3}{2}N^{\frac{1}{2}}$. This means that we know half of the most significant bits of the private exponent.

We still need to explain why this attack is considered a small public exponent attack. The reason for this is that we usually have no information at all about the constant $k$ in the key equation but, when $e = 3$, we need to have $k = 2$ The following result is taken from [4].

**Theorem 28.** *Given an RSA modulus $N = pq$ with $p, q > 3$ and the public key $< 3, N >$, then the constant $k$ in the key equation $ed = 1 + k\phi(N)$ satisfies $k = 2$.*

*Proof.* From the key equation $ed - 1 = k\phi(N)$ we know that $0 < k < e$. So, in our case we have $k = 1$ or $k = 2$. Because $(3, p-1) = 1$ we have $p - 1 \ncong 0 \pmod 3$ and, as $p > 3$, we have $(3, p) = 1$ and therefore $p - 1 \ncong 2 \pmod 3$ and so we get that

$$\begin{cases} p - 1 \cong 1 \pmod 3 \\ q - 1 \cong 1 \pmod 3 \end{cases}$$

So $\phi(N) = (p-1)(q-1) \cong 1 \pmod 3$. Taking the key equation modulo $e = 3$, we get:

$$3d \cong 1 + k\phi(N) \pmod 3 \Longrightarrow k \cong 2 \pmod 3 \tag{55}$$

Because $k = 1$ or $k = 2$, we have $k = 2$ and this concludes our proof. □

So every time we use $e = 3$ we are giving away half of the bits of the private exponent. However, this has not been proven to be enough to factor the modulus[22], so this attack does not represent a total break of the RSA.

## 3.4 Factoring the modulus of RSA with Small Private Exponent $d$

After the previous attacks, we now study attacks on RSA sessions where the private exponent, $d$, is low. These attacks are much more destructive than the ones presented in the previous section: they aim to factor the modulus, thus breaking the whole system. The theorems presented on this section suppose the encrypting and decrypting exponents are defined modulo $\lambda(N)$. The results, however, would apply if these exponents were instead defined modulo $\phi(N)$ like in the previous sections.

We present one first result, which will prove useful for the first attack of this section. Given the key equation $ed = 1 + k\lambda(N)$, we have

$$0 < k = \frac{(ed-1)}{\lambda(N)} < \frac{ed}{\lambda(N)} < min\{e, d\} \tag{56}$$

In particular we have that $k < d$.

### 3.4.1 Wiener's Continuous Fractions Attack

This attack, extremely simple to implement, is due to Wiener[46]. It factors the RSA modulus provided that the private exponent $d$ is sufficiently small.

**Theorem 29.** *Given an RSA modulus $N = pq$ and a public key $< e, N >$, let $< d, p, q, N >$ be its corresponding private key, where $ed = 1 + k\lambda(N)$. Let $g = (p-1, q-1), g_0 = \frac{g}{(g,k)}$ and $k_0 = \frac{k}{(g,k)}$. If $d < \frac{pq}{2(p+q-1)g_0k_0} = \frac{N}{2sg_0k_0}$ then $N$ can be factored in time polynomial in $\log(N)$ and $\frac{g}{k}$.*

*Proof.* Given that

$$\lambda(N) = lcm(p-1, q-1) = \frac{\phi(N)}{(p-1, q-1)} = \frac{N-s}{g} \tag{57}$$

we can rewrite the key equation as:

$$ed = 1 + k\lambda(N) = 1 + \frac{k}{g}\phi(N) = 1 + \frac{k_0}{g_0}(N-s) \tag{58}$$

where $k_0 = \frac{k}{(k,g)}$ and $g_0 = \frac{g}{(k,g)}$. Suppose we divide both sides of this equation by $dN$:

$$ed = 1 + \frac{k_0}{g_0}(N-s) \Leftrightarrow \frac{e}{N} = \frac{1}{dN} + \frac{k_0}{g_0dN}(N-s) = \frac{1}{dN} + \frac{k_0}{g_0d} - \frac{k_0s}{g_0dN} \tag{59}$$

Now we can majorate:

$$|\frac{e}{N} - \frac{k_0}{g_0d}| = |\frac{1}{dN} - \frac{k_0s}{dg_0N}| < \frac{k_0s}{dg_0N} = \frac{1}{2(dg_0)^2} \tag{60}$$

So, from the theorem presented in the Continuous Fractions section, we know that $\frac{k_0}{dg_0}$ is one of the convergents in the continuous fraction expansion of $\frac{e}{N}$. Let $c_i = \frac{a_i}{b_i}$ be the i-th convergent of $\frac{e}{N}$. Then for some $j$ we have $\frac{k_0}{dg_0} = \frac{a_j}{b_j}$. Now we can notice that the equation can be written as:

$$ed = 1 + \frac{k_0}{g_0}\phi(N) \Leftrightarrow \phi(N) = e\frac{dg_0}{k_0} - \frac{g_0}{k_0} = \lfloor e\frac{b_j}{a_j}\rfloor - \lfloor\frac{g_0}{k_0}\rfloor \tag{61}$$

So, if we know the correct convergent $c_j$ and guess the value of $\lfloor\frac{g_0}{k_0}\rfloor$, we can compute $\phi(N)$. To find the convergent and compute $\phi(N)$ we proceed as follows: for each convergent, we compute the corresponding candidate to $\phi(N)$: $\phi_c = \lfloor\frac{e}{c_i}\rfloor + m$. For each candidate, we try to factor the modulus, that is, solving the system $N = pq$ and $\phi_c = (p-1)(q-1)$. If a factorization is reached, then we have the right convergent. If none of the candidates is the right one, we increment $m$ by one and start again. This way, we are sure that $m$ will be equal to $\lfloor \frac{g_0}{k_0} \rfloor$ at some point. As $m < \lfloor \frac{g_0}{k_0} \rfloor$, we try a maximum of $\lfloor \frac{g_0}{k_0} \rfloor$ iterations for each convergents, whose total number is polynomial in $\log(N)$. $\square$

The implementation of this attack is presented in the appendix. The results obtained are represented in Table 8.

Table 8: Wiener's Attack's Experimental Results

| size of $N$ | average running time (seconds) |
|---|---|
| 32 | 0.007722 |
| 64 | 0.012929 |
| 128 | 0.046937 |
| 256 | 0.151693 |
| 512 | 0.497492 |
| 1024 | 1.78601 |

The most obvious way to avoid this attack is do simply use $d > N^{0.25}$. In its work, however, Wiener presented yet another solution to avoid this attack: using a larger public exponent $e$. This is not a problem: after choosing $e$, one simply adds to $e$ successively multiples of $\phi(N)$ until $e$ satisfies $e > N^{\frac{3}{2}}$. Therefore, it is still possible to avoid this attack and at the same time use a small private exponent as long as this last inequality is satisfied.

### 3.4.2 Improving Wiener's Attack

In its Ph.D. dissertation[16], presented in 2002, Durfee, working with Boneh, discovered a new way to attack RSA when a small secret exponent is used. Its attack is an improvement over Wiener's attack as it aims to show that a private exponent $d < N^{0.292}$ renders the system totally insecure, improving Wiener's bound of $d < N^{0.25}$. However, proof of this was not found by Durfee, thought the attacks work in practice. We present a sketch of his attack.

If we take the key equation we can rewrite it in the following way:

$$ed \cong 1 \pmod{\phi(N)} \Leftrightarrow ed + k(N + 1 - (p + q)) = 1 \qquad (62)$$

If we set $s = -(p+q)$, $A = N+1$ and take the equation $\pmod e$, the equation becomes:

$$k(A+s) \cong 1 \pmod e \tag{63}$$

The following notation will be used for this attack and the next: we will write $e = N^{\alpha}$ for some $\alpha$. As $e$ is usually the same size of $N$, $\alpha$ is usually close to 1. As for $d$, as we are talking about small private exponent attacks, we will assume it satisfies the inequality $d < N^{\delta}$ for some $\delta$. Therefore the goal of a low private key attack is to work for as high a $\delta$ as possible. Now lets look at the existing constraints over $k$ and $s$:

from the key equation we get:

$$|k| < \frac{de}{\phi(N)} \leq \frac{3de}{2N} < \frac{3}{2}e^{1+\frac{\delta-1}{\alpha}}. \tag{64}$$

as we know that $p$ and $q$ are less than $2\sqrt[2]{N}$ we have:

$$|s| < 3N^{0.5} = 3e^{\frac{1}{2\alpha}} \tag{65}$$

As in the usual case we have $\alpha \approx 1$, if we ignore the small constants we have the following problem, known as the **small inverse problem**: finding integers $k$ and $s$ that satisfy:

$$k(A+s) \cong 1 \pmod e \text{ such that } |s| < e^{0.5} \text{ and } |k| < e^{\delta} \tag{66}$$

The reason for the name of the problem is that, in some way, we are given an integer A and we are looking for a number close to it whose inverse is small $\pmod e$. If, for a given $\delta$ we can list all the solutions of this problem, then one of them will reveal the true value of $s$ and therefore allow us to factor $N$ This means that the RSA system is insecure when one uses a private exponent satisfying $d < N^{\delta}$ for a value of $\delta$ for which we can solve the Small Inverse Problem efficiently.

Thought it was Boneh and Durfee's conviction that this problem can be solved for $\delta < 0.292$, they could not prove this fact. The reason for this is that their solution to the small inverse problem, which is based on the LLL algorithm and Coppersmith's results, requires Conjecture 1 . However, their attack does result in practice and for this reason a decryption exponent $\delta < 0.292$ should be avoided. It should be noted that, on their work, Boneh and Durfee claim their conviction that the small inverse problem can solved even for $\delta < 0.5$.

Blomer and May[3], based on the Boneh-Durfee attack, presented a new heuristic attack for solving the small inverse problem for $\delta < 0.290$. Though they do not improve the previous existing bound, their attack is simpler to analyse.

Their main result, again dependant on Conjecture 1, is presented now. This is the general result, where the encryption exponent $e$ is not assumed to be of the same magnitude of $N$, that is, $\alpha \neq 1$.

**Theorem 30.** *For every $\epsilon > 0$, there exists an $N_0$ such that for every $N > N_0$ the following holds: Let $N = pq$ be an RSA modulus and $p, q$ balanced primes, $e = N^\alpha$ the encryption exponent and $d = N^\delta$ its corresponding decryption exponent defined modulo $\phi(N)$ such that*

$$\delta < \frac{2}{5} - \frac{3}{5}\alpha + \frac{1}{5}\sqrt{4\alpha^2 - 2\alpha + 4} - \epsilon.$$

*Then $N$ can be factored in time polynomial in $\log(N)$, provided that Conjecture 1 holds.*

We can see that setting $\alpha = 1$ yields the bound $\delta < 0.29$.

# 4 Conclusions

The several attacks presented in this work, or any other known attack, have not rendered the RSA cryptosystem unsafe. There is good prospects that some attacks, specially the ones by Boneh and Durfee and Blomer and May will become more powerful as soon as we get more knowledge on what we called *The Coppersmith Method Conjectures*. These conjectures seem to be closely related with the security of RSA and proving them would be definitely useful.

However, none of these heuristic attacks seems to threat RSA security completely. New bounds will be set and new precautions will be needed. But the most important criteria for the RSA safety is still the Large Integer Factorization Problem. Though there have been several advances in this area, an algorithm running in polynomial time to factor large integers is still not known.

The experiments on this work have not led to any new results but we hope that, along with the simple implementations presented, they can motivate the reader to further explore the existing attacks on RSA and to think how one can improve them.

One topic that deserves more study is the variants of RSA. They present exciting new possibilities but their security has not yet been as analysed as thoroughly as that of standard RSA. Its probably in them that the future of cryptography lies, so the sooner we understand them, the sooner it will be safer for us to rely on them.

# 5 Bibliography

[1] Kamilah Abdullah, *Comparison between the RSA Cryptosystem and Elliptic Curve Cryptography*, Master's thesis, University of Waikato, 1998.

[2] Manindra Agrawal, *Primality Tests Based on Fermat's Little Theorem*, ICDCN, 2006, pp. 288–293.

[3] J Blomer and A. May, *Low Secret Exponent RSA Revisited*, Lecture Notes in Computer Science (2001).

[4] Dan Boneh, *Twenty Years of Attacks on the RSA Cryptosystem*, Notices of the American Mathematical Society (1999).

[5] Richard P. Brent, *Recent Progress and Prospects for Integer Factorisation Algorithms*, Proceeding COCOON '00 Proceedings of the 6th Annual International Conference on Computing and Combinatorics (2000).

[6] Matthew E. Briggs, *An Introduction to the General Number Field Sieve*, Master's thesis, Virginia Polytechnic Institute and State University, 1998.

[7] Daniel R. L. Brown, *A Weak-Randomizer Attack on RSA-OAEP with* $e = 3$, 2005.

[8] Carlos Frederico Cid, *Cryptanalysis of RSA: A Survey*, Information Security Reading Room (1988).

[9] D. Coppersmith, *Small Solutions to Polynomial Equations, and Low Exponent RSA Vulnerabilities*, Journal of Cryptology (1997).

[10] G Durfee D Boneh and Y Frankel, *An Attack on RSA Given a Small Fraction of the Private Key Bits*, Lecture Notes in Computer Science (1998).

[11] Tania Lange Daniel J. Bernstein, Peter Birkner and Christiane Peters, *ECM using Edwards Curves*, Journal of Cryptology (2010).

[12] J. M. DeLaurentis, *A Further Weakness in the Common Modulus Protocol for the RSA Cryptoalgorithm*, Cryptologia (1984).

[13] Whitfield Diffie and Martin E. Hellman, *New Directions in Cryptography*, IEEE Transactions on Information Theory (1976).

[14] ANDREJ DUJELLA, *Continued Fractions and RSA with Small Secret Exponent*, Tatra Mt. Math. Publ (2004).

[15] D Boneh G Durfee and Y Frankel, *Exposing an RSA Private Key Given a Small Fraction of its Bits*, Journal of Cryptology (2001).

[16] Glenn Durfee, *Cryptanalysis of RSA Using Algebraic and Lattice Methods*, Ph.D. thesis, Stanford University, 2002.

[17] M. K. Franklin and M. K. Reiter, *A Linear Protocol Failure for RSA with Exponent Three*, CRYPTO '95 rump session (1995).

[18] J. Hastad, *On using RSA with Low Exponent in a Public Key Network*, Lecture Notes in Computer Science (1985).

[19] M Jason Hinek, *Low Public Exponent Partial Key and Low Private Exponent Attacks on Multi-prime RSA*, Master's thesis, University of Waterloo, 2002.

[20] M. Jason Hinek, *Another Look at Small RSA Exponents*, 2006.

[21] M Jason Hinek, *On the Security of Some Variants of RSA*, Ph.D. thesis, University of Waterloo, 2007.

[22] M. Jason Hinek, *Cryptanalysis of RSA and its Variants*, Chapman Hall, 2010.

[23] M. Jason Hinek and Charles C. Y. Lam, *Common Modulus Attacks on Small Private Exponent RSA and Some Fast Variants (in Practice)*, IACR Cryptology ePrint Archive **2009** (2009), 37.

[24] Abdullah Darwish Imad Khaled Salah and Saleh Oqeili, *Mathematical Attacks on RSA Cryptosystem*, Journal of Computer Science (2006).

[25] S. Katzenbeisser, *Recent Advances in RSA Cryptography*, Klewer Academic Publishers (2001).

[26] A.K. Lenstra, H.W.jun. Lenstra, and Lászlo Lovász, *Factoring Polynomials With Rational Coefficients*, Math. Ann. **261** (1982), 515–534.

[27] Arjen K. Lenstra, *Primality Testing*, Proceedings of Symposia in Applied Mathematics, Mathematics and Computer Science.

[28] Arjen K Lenstra, *Unbelievable Security Matching AES Security Using Public Key Systems*, Proceedings Asiacrypt 2001, LNCS 2248, Springer-Verlag 2001, 6786, Springer-Verlag, 2001, pp. 67–86.

[29] Neeraj Kayal Manindra Agrawal and Nitin Saxena, *PRIMES is in P*, Ann. of Math (2004).

[30] A. May and M. Ritzenhofen, *Solving Systems of Modular Equations in One Variable: How Many RSA-Encrypted Messages Does Eve Need to Know?*, Public Key Cryptography (2008).

[31] Alexander May, *Using LLL-Reduction for Solving RSA and Factorization Problems: A Survey*, Journal of Cryptology (2010).

[32] S.M. Mashihure Romman Md. Ali-Al-Mamun, Mohammad Motaharul Islam, *Performance Evaluation of Several Efficient RSA Variants*, International Journal of Computer Science and Network Security (2008).

[33] G. L. Miller, *Riemann's Hypothesis and Tests for Primality*, Journal of Computer and System Sciences (1976).

[34] P.L. Montgomery, *Modular Multiplication Without Trial Division*, Math. Computation (1985).

[35] J. M. Pollard, *A Monte Carlo Method for Factorization*, BIT Numerical Mathematics (1975).

[36] Carl Pomerance, *A Tale of Two Sieves*, Notices of the AMS (1996).

[37] H. Riesel, *Prime Numbers and Computer Methods for Factorization*, Birkhauser, 1994.

[38] A. Shamir R.L. Rivest and L. Adleman, *A Method for Obtaining Digital Signatures and Public-Key Cryptosystems*, Magazine Communications of the ACM (1978).

[39] Josef Pieprzyk Huaxiong Wang Ron Steinfeld, Scott Contini, *Converse Results to the Wiener Attack on RSA*, International Workshop on Practice and Theory in Public Key Cryptography (2005).

[40] Burt Rosenberg, *The Solovay-Strassen Primality Test*, October 1993.

[41] R G Salembier and Paul Southerington, *An Implementation of the AKS Primality Test*, Computer Engineering (2005).

[42] G. J. Simmons, *A "Weak" Privacy Protocol using the RSA Crypto Algorithm*, Cryptologia (1983).

[43] G. J. Simmons and M. J. Norris, *Preliminary Comments on the MIT Public-key Cryptosystem*, Cryptologia (1977).

[44] William Stein, *Elementary number theory. Primes, congruences, and secrets. A computational approach.*, New York, NY: Springer, 2009 (English).

[45] M Trott, *The Mathematica Guidebook for Symbolics*, Springer-Verlag, 2006.

[46] M J Wiener, *Cryptanalysis of Short RSA Secret Exponents*, IEEE Transactions on Information Theory (1990).

[47] Oded Yacobi and Yacov Yacobi, *A New Related Message Attack on RSA*, Essays in Memory of Shimon Even, 2006, pp. 187–195.

[48] Song Y. Yan, *Cryptanalytic Attacks on RSA*, Springer, 2008.

# A Implementations of the attacks from section 3.2

## A.1 Common Modulus Attack

```
rcm = {};
i = 3;
While[i < 13,
nbits = 2^i;
p = RandomPrime[{2^(nbits/2 − 1), 2^(nbits/2) − 1}];
q = RandomPrime[{2^(nbits/2 − 1), 2^(nbits/2) − 1}];
While[q == p, q = RandomPrime[{2^(nbits/2 − 1), 2^(nbits/2) − 1}]];
n = p ∗ q;
e1 = n;
While[e1 == n, e1 = RandomPrime[n]; ]
e2 = n;
While[e2 == n||e2 == e1, e2 = RandomPrime[n]; ]
m = RandomPrime[n];
c1 = PowerMod[m, e1, n];
c2 = PowerMod[m, e2, n];


r = Append[rcm, {nbits, Timing[
l = ExtendedGCD[e1, e2];
a1 = l[[2, 1]];
a2 = l[[2, 2]];
dec = Mod[PowerMod[c1, a1, n] ∗ PowerMod[c2, a2, n], n]
][[1]]}];
Print[rcm];
i++;
]
```

## A.2 DeLaurentis Attack

```
rla = {};
i = 3;
While[i < 13,
nbits = 2^i;
p = RandomPrime[{2^(nbits/2 - 1), 2^(nbits/2) - 1}];
q = RandomPrime[{2^(nbits/2 - 1), 2^(nbits/2) - 1}];
While[q == p, q = RandomPrime[{2^(nbits/2 - 1), 2^(nbits/2) - 1}]];
n = p * q;
e1 = n;
While[e1 == n|| e1 == p||e2 == q, e1 = RandomPrime[n]; ];
d1 = PowerMod[e1, -1, n];
e2 = n;
While[e2 == n||e2 == e1, e2 = RandomPrime[n]; ];

rla = Append[rla, {nbits, Timing[
naux = (e1 * d1 - 1)/GCD[e2, e1 * d1 - 1];
d2crypt = PowerMod[e2, -1, naux]
][[1]]}];
Print[r1];
i++;
]
```

## A.3 Hastad's Common Plaintext Attack

```
rhc = {};
e = 3;
i = 3;
While[i < 12,
nbits = 2^i;
nlist = {};
b = 1;
primos = {};
While[Length[nlist] < e + 2,
p = RandomPrime[{2^(nbits - 1), 2^nbits - 1}];
q = RandomPrime[{2^(nbits − 1), 2^nbits − 1}];
While[Intersection[{p}, primos] ≠ {},
p = RandomPrime[{2^(nbits − 1), 2^nbits − 1}]; ];
While[Intersection[{q}, primos] ≠ {}||p == q,
q = RandomPrime[{2^(nbits − 1), 2^nbits − 1}]; ];
n = p ∗ q;
neuler = (p − 1)(q − 1);
If[GCD[e, neuler] == 1,
nlist = Append[nlist, {e, n}];
primos = Append[primos, p];
primos = Append[primos, q];
];
];
n1 = nlist[[4]];
n2 = nlist[[5]];
n3 = nlist[[3]];
m = RandomInteger[{1, Min[n1, n2, n3] − 1}];
c1 = PowerMod[m, e, n1];
c2 = PowerMod[m, e, n2];
c3 = PowerMod[m, e, n3];

rhc = Append[rhc, {BitLength[n], Timing[
n0 = Min[n1, n2, n3];
n = n1 ∗ n2 ∗ n3;
c = ChineseRemainder[{c1, c2, c3}, {n1, n2, n3}];
dec = c^(1/e); ][[1]]}
];
i++;
];
rhc
```

## A.4 Related Plaintext Attack

```
rrp = {};
b = 3;
While[b < 10,
nbits = 2^b;
i = 3;
g = 50;
nlist = {};
primos = {};
While[Length[nlist] < g,
e = Prime[RandomInteger[{2, 5}]];
p = RandomPrime[{2^(nbits - 1), 2^nbits - 1}];
q = RandomPrime[{2^(nbits - 1), 2^nbits - 1}];
While[Intersection[{p}, primos] ≠ {},
p = RandomPrime[{2^(nbits - 1), 2^nbits - 1}]; ];
While[Intersection[{q}, primos] ≠ {}||p == q,
q = RandomPrime[{2^(nbits - 1), 2^nbits - 1}]; ];
n = p * q;
neuler = (p - 1)(q - 1);
If[GCD[e, neuler] == 1,
nlist = Append[nlist, {e, n}];
primos = Append[primos, p];
primos = Append[primos, q],
nbits = nbits + 1;
];
];
l = Max[Table[nlist[[i, 1]] * 2, {i, 1, g}]];
coef = Table[{RandomInteger[{0, 2}], i}, {i, 1, l}];
poli = Function[{i, m},
Mod[m^coef[[i, 1]] + coef[[i, 2]], nlist[[i, 2]]]
];
deg = Function[i, coef[[i, 1]]];
max = Max[Table[nlist[[i, 1]] * deg[i], {i, 1, l}]];
i = 2;
n0 = nlist[[1, 2]];
```

```
While[i ≤ 23,
n0 = Min[n0, nlist[[i, 2]]];
i++;
];
i = 2;
n = nlist[[1, 2]];
While[i ≤ 23,
n = n * nlist[[i, 2]];
i++;
];
m = RandomInteger[n0 − 1];
clist = Table[PowerMod[poli[i, m], nlist[[i, 1]], nlist[[i, 2]]],
{i, 1, l}];

rrp = Append[rrp, {BitLength[n], Timing[
sigma = Max[Table[nlist[[i, 1]] * deg[i], {i, 1, l}]];
h = Table[sigma − deg[i] * nlist[[i, 1]], {i, 1, l}];
gpoli = Function[{i, m},
Mod[PowerMod[m, h[[i]], nlist[[i, 2]]]*
(PowerMod[poli[i, m], nlist[[i, 1]], nlist[[i, 2]]] − clist[[i]]),
nlist[[i, 2]]]
];
mlist = Table[n/nlist[[i, 2]], {i, 1, l}];
minv = Table[PowerMod[mlist[[i]], −1, nlist[[i, 2]]], {i, 1, l}];
gcrt = Function[m,
r = 0;
i = 1;
While[i ≤ l,
r = r + gpoli[i, m]mlist[[i]]minv[[i]];
i++;
];
Mod[r, n]
];
][[1]]}];
b++;
```

];

**rrp**

# B Implementations of the attacks from section 3.3

## B.1 Stereotyped Message Attack

```
rsm = {};
i = 3;
While[i < 14,
nbits = 2^i;
p = RandomPrime[{2^(nbits/2 − 1), 2^(nbits/2) − 1}];
q = RandomPrime[{2^(nbits/2 − 1), 2^(nbits/2) − 1}];
While[q == p, q = RandomPrime[{2^(nbits/2 − 1), 2^(nbits/2) − 1}]];
n = p ∗ q;
e = 3;
While[GCD[e, (p − 1)(q − 1)] ≠ 1, e = NextPrime[e]; ];
Print["RSA gerado: ", i];
bits = Table[RandomInteger[{0, 1}], {i, 1, nbits}];
l = Length[bits];
f = Floor[(1/e) ∗ l] − 1;
p = RandomInteger[{1, l − f}];
u = 1;
m = 0;
While[u<=l,
m = m + bits[[u]] ∗ 2^(l − u);
u++;
];
c = PowerMod[m, e, n];

rsm = Append[rsm, {nbits, Timing[
k = 1;
m2 = 0;
While[k < p,
m2 = m2 + 2^(32 − k) ∗ bits[[k]];
k++;
];
j = p + f;
m0 = 0;
While[j<=l,
m0 = m0 + 2^(l − j) ∗ bits[[j]];
j++;
];
k1 = l − (p + f − 1); ][[1]]}];
i++;
];
rsm
```

## B.2 Related Message Attack

```
rrm = {};
i = 3;
While[i < 13,
nbits = 2^i;
p = RandomPrime[{2^(nbits/2 − 1), 2^(nbits/2) − 1}];
q = RandomPrime[{2^(nbits/2 − 1), 2^(nbits/2) − 1}];
While[q == p, q = RandomPrime[{2^(nbits/2 − 1), 2^(nbits/2) − 1}]];
n = p ∗ q;
e = 3;
m1 = RandomPrime[{5, n}];
a = RandomInteger[n];
b = RandomInteger[n];
m2 = am1 + b;
c1 = PowerMod[m1, e, n];
c2 = PowerMod[m2, e, n];

rrm = Append[rrm, {nbits, Timing[
Mod[(b(c2 + 2a^3c1 − b^3))/(a(c2 − a^3c1 + 2b^3)), n]][[1]]}];
i++;
]
rrm
```

## C Implementation of Wiener's attack

```
rwi = {};
i = 5;
While[i < 14,
nbits = 2^i;
p = RandomPrime[{2^(nbits/2 − 1), 2^(nbits/2) − 1}];
q = RandomPrime[{2^(nbits/2 − 1), 2^(nbits/2) − 1}];
While[q == p, q = RandomPrime[{2^(nbits/2 − 1), 2^(nbits/2) − 1}]];
n = p ∗ q;
lambdan = LCM[(p − 1), (q − 1)];
d = RandomPrime[Floor[n^(1/4)]];
While[GCD[d, lambdan] ≠ 1, d = RandomPrime[Floor[n^(1/4)]]; ];
g0 = n;
k0 = 1;
l = 1;
While[d > p ∗ q/(2(p + q − 1)g0 ∗ k0),
p = RandomPrime[{2^(nbits/2 − 1), 2^(nbits/2) − 1}];
q = RandomPrime[{2^(nbits/2 − 1), 2^(nbits/2) − 1}];
While[q == p, q = RandomPrime[{2^(nbits/2 − 1), 2^(nbits/2) − 1}]];
n = p ∗ q;
lambdan = LCM[(p − 1), (q − 1)];
d = RandomPrime[Floor[n^(1/4)/l]];
While[GCD[d, lambdan] ≠ 1, d = RandomPrime[Floor[n^(1/4)/l]]; ];
e = PowerMod[d, −1, lambdan];
k = (e ∗ d − 1)/lambdan;
g = GCD[p − 1, q − 1];
g0 = g/GCD[g, k];
k0 = k/GCD[k, g];
l++;
];
rwi = Append[rwi, {nbits, Timing[
conv = Convergents[e/n];
candidates = Table[Floor[e/conv[[i]]], {i, 2, Length[conv]}];
m = 0;
fact = 0;
While[fact == 0,
j = 1;
While[j<=Length[candidates]&&fact == 0,
fitemp = candidates[[j]] + m;
primos = Solve[x^2 - (n - fitemp + 1)x + n==0, x];
ptemp = x/.primos[[1]];
qtemp = x/.primos[[2]];
If[PrimeQ[ptemp]&&PrimeQ[qtemp], fact = 1, j++];
];
m++;
];
][[1]]}];
i++;
]
rwi
```